Facilitating Teacher Participation in Intelligent Computer Tutor Design: Tools and Design Methods

Tom Murray
Ph.D. Dissertation

Computer and Information Science Department
University of Massachusetts

COINS Technical Report 91-95
January 1992

Facilitating Teacher Participation in Intelligent Computer Tutor Design: Tools and Design Methods

A Dissertation Presented

by

Thomas J. Murray

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

Doctor of Education

February 1992

School of Education

Facilitating Teacher Participation in Intelligent Computer Tutor Design: Tools and Design Methods

A Dissertation Presented

by

Thomas J. Murray

Approved as to style and content by:

Klaus Schultz, Chair

Beverly Park Woolf, Member

Howard A. Peelle, Member

Bailey Jackson
Dean of the School of Education

Send communications to:

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003
EMAIL: tmurray@cs.umass.edu

This research was supported in part by:

Apple Computer Inc., Cupertino, CA.

The National Science Foundation under grant number MDR 875162.

The Office of Naval Research under a University Research Initiative Grant, no. N00014-86-K-0764.

ACKNOWLEDGMENTS

It has been a long, productive, and satisfying journey, marked by the guidance and support of many. Foremost, I would like to thank my committee members who each, in his or her own way, inspired me by modeling how to be successful academicians without being plagued by the imposing furrowed brow and pale ivory-tower glow; by the graceful management of their priorities they blend work, family, friendships, and exercise into balanced full lives.

Klaus Schultz has been a good friend and strong ally, and a careful reader of the mountains of material I've asked his advice on over the years, pointing out many connections in my work that I had missed. Beverly Woolf's Strunk-&-White-style editing, sometimes ruthless but always from the heart, have been essential in transforming this and many other documents from rhetorical gobbledygook to a state of greater elegance; her support has been multi-faceted, generous, and consistent. Howard (Hap) Peelle, whose insightful and divergent thinking has lead me to broaden my perspective on many occasions, pointed the way in my early tenure as a graduate student, and he has shown confidence in my intelligence and judgment throughout. I would also like to acknowledge the important mentorship of John Clement in several projects during my tenure as a graduate student.

For the execution of this research there were many helping hands. One could not hope for a better working partner than Charlie Camp, who, with determination, patience, and good cheer underwent a host of trials and tribulations wearing the triple-hats of subject matter expert, experimental subject, and co-researcher. Thanks to Frank Linton and Kim Gonzalez for their hard work as knowledge base managers and for important feedback about software usability.

This journey would have been even longer and much less pleasant had it not been for the programming assistance and advice of: Miguel Cardos (porting code from HP to Mac);

Craig Fournier, Raul Bose, and the SFSU crew (crane boom simulation); Marcus Weinhardt (graphical network editor); Jim Salehi (help system interface); and finally, Dan Suthers (Lisp wisdom and magic!). Additional gratitude is owed to Dan Suthers and Miguel Cardos for loyal friendship and key discussions on early versions of the software and to Frank Linton for useful feedback on this document.

Generous financial and moral support for myself and the UMass KCSG lab was received through Barbara Bowen of Apple Computer's External Research Group, Mark Miller of Apple Computer's Business Learning Research group, and Andrew Molnar of the National Science Foundation.

When I began this project in earnest I set a challenging goal for myself to maintain peace of mind, physical health, and enjoyment as top priorities. Thus, "the real dissertation," as I called it, was the "struggle" to live my life as a university researcher and doctoral candidate without struggling. I have succeeded (to my amazement) in large part due to having the people mentioned below in my life.

Thanks to my parents Joe Murray and Dot Murray for steadfast belief in my abilities and graceful tolerance of my decisions in my transition from engineer-making-big-bucks into realms in which they had little experience (academia, meditation, dance, martial arts, etc.). Their consistent faith in me since childhood was internalized and has been at the core of all I have ever accomplished.

I would like to thank Jeanne Edelen, Caroline Ayres, and Charlie Morrison for emotional support, and Felix Bizaoui for being my pal and helping me remember the real dissertation. Special thanks and hugs to Victoria Yoshen for making the final eight months of this project one of the most wonderful periods of my life.

Finally I would like to express my special gratitude to Keith Otis in writing (if I did so in spoken words he would pretend not to understand and ask me to race him around the back yard or build a dinosaur with LEGOs). He grew from infancy to boyhood as this project grew from inception to completion, and has been my greatest teacher and dearest companion.

ABSTRACT

Facilitating Teacher Participation in Intelligent Computer Tutor Design: Tools and Design Methods

February 1992

THOMAS J. MURRAY, B.S. PHYSICS, WORCESTER POLYTECHNIC INSTITUTE
M.ED., EDUCATION, UNIVERSITY OF MASSACHUSETTS
M.S., COMPUTER SCIENCE, UNIVERSITY OF MASSACHUSETTS
ED.D., EDUCATION, UNIVERSITY OF MASSACHUSETTS

Directed by: Professor Klaus Schultz

This work addresses the widening gap between research in intelligent tutoring systems (ITSs) and practical use of this technology by the educational community. In order to ensure that ITSs are effective, teachers must be involved in their design and evaluation. We have followed a user participatory design process to build a set of ITS knowledge acquisition tools that facilitate rapid prototyping and testing of curriculum, and are tailored for usability by teachers. The system (called KAFITS) also serves as a test-bed for experimentation with multiple tutoring strategies. The design includes novel methodologies for tutoring strategy representation (Parameterized Action Networks) and overlay student modeling (a "layered" student model), and incorporates considerations from instructional design theory. It also allows for considerable student control over the content and style of the information presented. Highly interactive graphics-based tools were built to facilitate design, inspection, and modification of curriculum and tutoring strategies, and to monitor the progress of the tutoring session. Evaluation of the system includes a sixteen-month case study of three educators (one being the domain expert) using the system to build a tutor for statics

(forty topics representing about four hours of on-line instruction), testing the tutor on a dozen students, and using test results to iteratively improve the tutor. Detailed throughput analysis indicates that the amount of effort to build the statics tutor was, surprisingly, comparable to similar figures for building (non-intelligent) conventional computer aided instructional systems. Few ITS projects focus on educator participation and this work is the first to empirically study knowledge acquisition for ITSs. Results of the study also include: a recommended "design process" for building ITSs with educator participation; guidelines for training educators; recommendations for conducting knowledge acquisition sessions; and design tradeoffs for knowledge representation architectures and knowledge acquisition interfaces.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This document describes the design and formative evaluation of a computer system, or set of "tools," that allow educators (teachers, curriculum designers, and instructional theorists) with no experience in computer programming to participate fully in the design, building, evaluation, and modification of intelligent computer tutors. The system facilitates the process of encoding an instructor's knowledge about *what* to teach and *how* to teach in his area of expertise. A computer tutor built using these tools engages the student in a highly interactive tutorial session based on multiple strategies defined by an instructor, and the tutoring is flexible in responding to students' needs and to the curriculum context. In this document we describe (1) the computer system, (2) our methods for using the system to acquire instructional knowledge, and (3) a case study of three educators using the system to build a tutor for part of a high school physics curriculum.

1.1 Background and Motivation

1.1.1 Intelligent Tutoring Systems

Computer scientists and educators have had great hope that artificial intelligence technology could be used to design flexible, effective, and powerful learning environments. Intelligent tutoring systems[1] (ITSs) are computer programs that incorporate artificial intelligence (AI) knowledge representation and control paradigms. These paradigms include

[1]Intelligent tutoring systems, intelligent learning environments, knowledge based tutoring systems, and intelligent computer aided instruction (ICAI) systems are different terms with similar meaning, i.e computer

frame-based and rule-based representations of knowledge (as in expert systems) and explicit models of domain expertise,[2] the student's knowledge and preferences, and the instructional process. One way to describe ITSs is to compare them with traditional computer aided instruction (CAI) (see Wenger [1987] for a more detailed introduction to ITSs). In CAI each tutorial decision is explicitly encoded, for example, "if the student answers 'YES' to question #32 give explanation #45." CAI systems do have limited ability to personalize the tutorial, since the material presented depends on the student's behavior (i.e. the student's answers to questions). However CAI has many problems, including limited flexibility because the designer must enumerate every possible situation and combination of situations the program will respond to, and building and modifying the programs is difficult because all important decisions are implicit. Specifically, CAI systems contain knowledge about the following *implicitly*:

1. structure of the knowledge being taught,
2. reasons or strategies used to make instructional decisions, and
3. assumptions about the student's knowledge or mental state.

ITS systems add increased sophistication and flexibility by explicitly encoding models of the processes relevant to instruction (see Figure 1.1.1), including:

1. a model of expertise in the domain,
2. a model of tutoring expertise, and
3. a model of the student.

---

assisted learning programs that incorporate artificial intelligence technology and paradigms. These terms can imply a focus on different issues for different authors, but for the purposes of this paper we treat them as equivalent, unless otherwise noted.

[2]We use the word "domain" to mean the content area or subject matter area. Domain expertise is expertise in solving problems in the domain. Though the meanings are not synonymous, we use the words teacher, tutor, domain expert, and instructional designer interchangeably to refer to the hypothetical user of the KAFITS tool, or, in some contexts, the teacher who participated in this study.

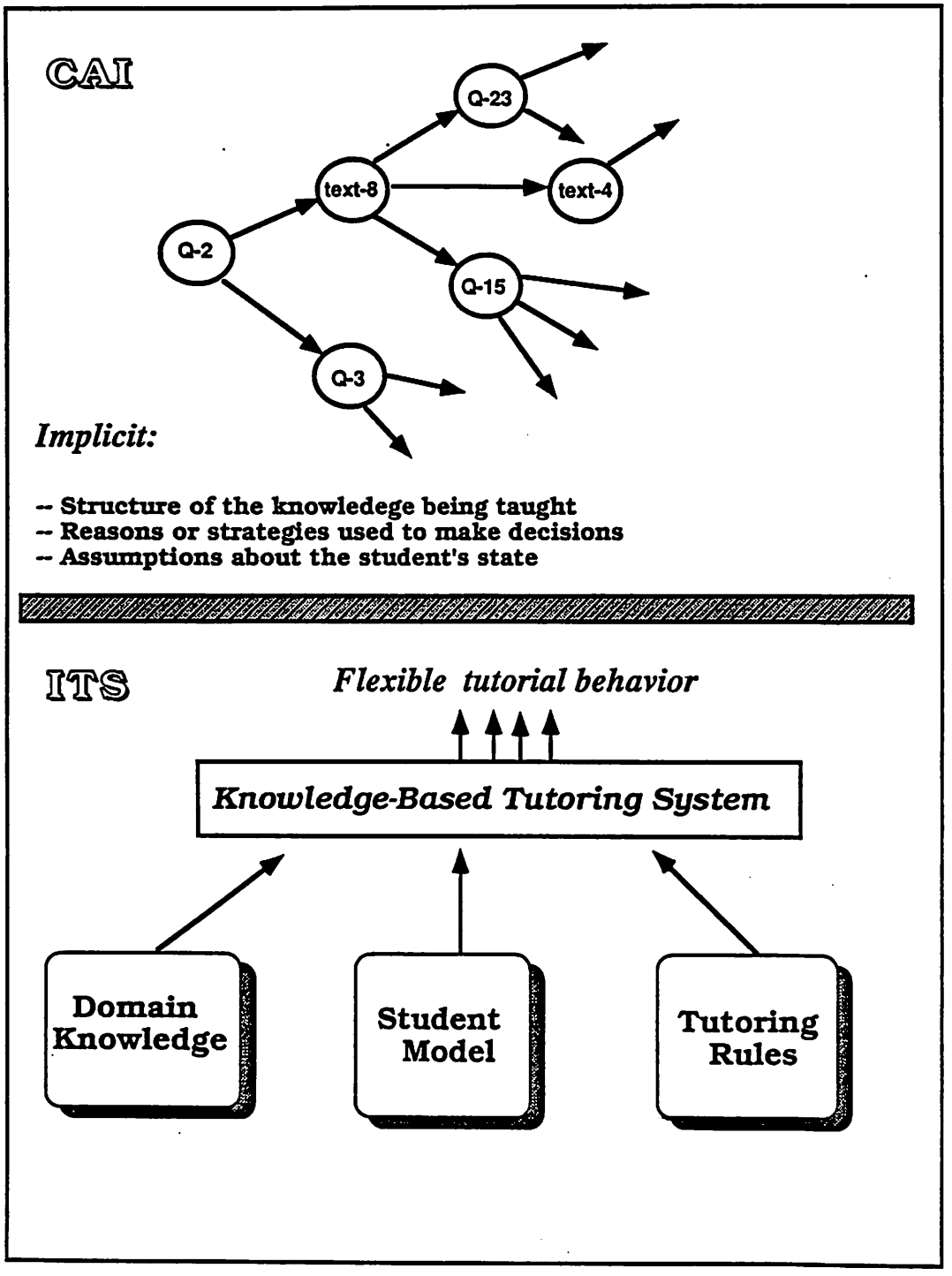


Figure 1.1 Comparing CAI with ITS

These three models are constructed and/or updated by the ITS as it creates a tutorial session on the fly. Modeling domain expertise enables flexibility in generating problems and explanations; modeling tutoring expertise enables multiple tutoring styles to be represented explicitly, as in the tutoring rule "if student-is-confused then give-more-hints," and modeling the student's knowledge enables instruction to be tailored to the needs of each student.

1.1.2 ITS is "AI Complete"

Intelligent tutoring systems are among the most complex and challenging areas where artificial intelligence is being applied. Ultimately, ITS designers strive to endow their systems with sophisticated "intelligent" capabilities, some of which involve: modeling expert level problem solving skills and information organization; constructing a dynamic model of the learner's knowledge, mental state, and beliefs; engaging in meaningful and individualized dialog with the learner; responding to the learner's requests for information and explanation; incorporating principles and strategies of good tutoring based on learning theories and pragmatic constraints; providing a rich and exciting, yet accessible, environment for learning and applying skills; and even incrementally improving their performance in all of the above areas as the tutor encounters novel student behavior. The problems of ITS construction are said to be "AI complete," i.e., contained within their solution are the solutions to many of the difficult problems facing the entire field of artificial intelligence (for example: natural language generation and understanding, intelligent interfaces, planning, knowledge representation, and expert systems).

1.1.3 Tutoring Systems are Needed

If ITSs present such difficult challenges why attempt to build them? Research in ITS continues because the potential gains are as great as the known difficulties. Education is not just an "applications area" in which to test expert systems technology,[3] but is an end

[3] An expert system is an AI system that simulates expert-level skills using a set of rules that capture the nature of the expertise.

in itself. Documented deficiencies of our current educational system for both children and adults [US Department of Education 1983, National Science Foundation 1983] underscore the need for new learning paradigms in schools, home education, and industrial training. Personalized tutoring (by human tutors) has been shown to be highly effective [Bloom 1984] compared to classroom teaching and individualized (or mastery) learning. But society does not have the resources to provide each learner with a skilled tutor in every area. Computer programs will not (in the foreseeable future) be as "intelligent" as good human tutors in understanding domains or in being sensitive to the learner's needs, but they *can* improve education by assisting overburdened teachers and instructing in areas where human experts are not available. Shute [1990] summarizes several studies that demonstrate that in some situations students using ITSs learn more efficiently and effectively than students covering the same subject matter in classrooms. Computer tutors can also include learning environments, such as simulations of physical systems and interactive databases, which would be categorically impossible without the use of computers.

The success of ITSs does not require all of the sophisticated capabilities and "intelligence" mentioned in Section 1.1.2. Rather, they only have to be sufficiently better than the (static) text book or the (impersonal) lecture to make a positive impact on education.

1.1.4 Problem Areas

Significant progress has been made in ITS research [Kearsley 1987, Psotka et al. 1988] but there are still many unexplored areas and most of the key issues identified have not been resolved. Below we describe the issues needing attention that this study addresses:

ITS research vs. application gap. ITSs are often designed by scientists with little experience teaching in the domain of the ITS (except perhaps for computer programming domains). It is important for educators (and students as well) to participate in the ITS design process. However, as research in ITS continues to produce more sophisticated systems and theories, the gap between the research community and the educational community continues to widen, because educators' understanding, acceptance, and use of this research

has been much slower than research progress. As this theory-application gap widens it becomes more difficult for educators to participate in ITS research and application, and as a result research is becoming increasingly academic and unconnected to the pragmatic aspects of teaching and learning. Clancey & Joerger [1988] state that "...the reality today is that the endeavor is one that only experienced programmers (or experts trained to be programmers) can accomplish. Indeed, research of the past decade has only further increased our standards of knowledge representation desirable to teaching, while the tools for constructing such programs lag far behind or are not generally available."

General frameworks. It is often difficult to apply the ideas generated by one ITS research project to another, or to compare two ITS systems [Ohlsson 1986]. This lack of generality is due to many factors, including insufficient evaluation, the limited number of ITSs designed to answer general theoretical questions, and the lack of a shared conceptual vocabulary. Domain independent frameworks, of which few exist, facilitate more and clearer collaboration and critique.

Explicit representation of tutoring strategies. One might think that the major focus of ITS research, a field that deals with automating tutoring, would be to simulate tutoring and/or teaching expertise. Yet surprisingly little research deals directly with representing pedagogical knowledge (i.e. knowledge related to teaching and learning in a specific domain) or with modeling general pedagogical expertise. ITSs have been designed to tutor in many domains, and each designer has his/her own ideas for promoting learning. A few systems are based on specific theories of cognition and/or instruction (such as in Anderson et al. [1985a]), but in general there is little agreement, and often controversy, about the best methods for encouraging learning (in general, and in specific domains). The instructional principles or rules underlying ITS design are often ad-hoc and/or not represented explicitly. Without explicit representations of the rules or strategies used it is difficult to evaluate or build upon ITS designs.

Multiple tutoring strategies. In addition, very little ITS research incorporates *multiple* tutoring strategies. As educators become more involved in the ITS design process, and

as ITSs become more educationally realistic, the breadth of content and teaching styles in these systems will need to expand. Most intelligent tutoring systems focus on a limited instructional domain and embody a single theory of instruction or learning. However, a system designed to teach various types of knowledge in multiple domains will need to be sensitive to the pedagogical properties of the information being taught and be able to switch between multiple tutoring strategies. Systems are needed which facilitate experimentation with various tutoring styles, rather than committing to a particular learning theory or tutoring style.[4]

Experimental ITS workbenches It is not enough to explicitly represent and select among multiple tutoring strategies—cognitive and educational *research* is needed to determine the most effective strategies for various instructional contexts. Few theory-based general paradigms for instruction via intelligent tutoring systems have been put forth, and few specific strategies have been tested (Anderson et al. [1984] being a notable exception). Most relevant (non-computer based) instructional and cognitive theories are not operationalized to a level easily implemented in a computer, nor do they anticipate many practical factors and domain related idiosyncrasies. More research is needed on computer-based instructional strategies, and practicing educators should be involved in this research. But since the experience of learning via an intelligent tutoring system is so novel that neither teachers *nor* theorists can foresee many crucial issues, much of this research needs to be done *on-line.* The insights, principles, and rules used in ITSs should originate from a rich synthesis of learning and instructional theories, insights from practicing teachers, and on-line experimentation, and this synthesis can be greatly enhanced by appropriate computer tools. Halff [1988, pg. 99] emphasizes: "...laboratories for systematic manipulation of alternative tutoring methods are needed."

Instructional design theory. The volumes of work, both experimental and prescriptive, generated by the instructional design research community have been largely ignored

[4]Ohlsson [1986, pg. 220] says: "In order to provide adaptive instruction, a tutor must have a wide range of instructional actions to choose from."

by the ITS research community[5] [Halff 1988]. Yet for decades instructional systems design (ISD) has been formulating answers and hypotheses to questions of vital importance to ITS. Instructional design theories, although not as well-founded on cognitive theory as many in the ITS field would like, address the breadth of situations and practical realities of instruction and training. Rather than ignore ISD, ITS researchers should become familiar with it and use those ISD principles that are cognitively feasible.

ITS evaluation. ITS research suffers from a lack of principled evaluation of its artifacts (as does AI research in general [Buchanan 1987, Rosenberg 1987]). This is partly due to a lack of agreement about which experimental and evaluation methods are appropriate, and partly because the field is in a formative stage. Consequently, papers containing descriptions of non-substantiated ideas, un-implemented systems, and implemented but untested systems are often published. It seems clear [Cohen & Howe, 1988] that traditional quantitative scientific analysis is inappropriate in most situations, yet some form of rigor must be adhered to if progress in the field is to continue.

Next we describe the research goals motivated by the above concerns.

1.2 Description of the Study

1.2.1 Research Questions and Goals

The overriding question we address in this study is:

> What are the *key issues* in the design and use of a tool that allows *educators* to *participate intimately* in building and evaluating *ITSs*?

This terse summary of our goal is elaborated below where we expand upon each of the emphasized terms in the overriding question.

[5] Both historical and other reasons are responsible for this lack of collaboration. We do not speculate to any length about the reasons.

- This is an exploratory study in an area where there has been little previous investigation. Therefore the goal is to *identify "key issues"* rather than test clearly stated hypotheses.

- *"Educators"* includes teachers, instructional designers, learning theorists, etc. (especially those without programming, computer science, or ITS backgrounds). Since the effort and expertise required to build an ITS is at least as much as that required to write a test book, we do not expect all teachers to participate in ITS construction; on the contrary, we assume that practicing classroom teachers and industry trainers who help design ITSs will be few in number and "above average" in their teaching abilities and knowledge of the domain. However *any* instructor should be able to *use* the resulting computer tutor in his/her classroom.

- *"Participating intimately"* means that the educator is able to design the knowledge base, enter information within the knowledge base, run the tutor to test various aspects of it, and modify the knowledge base for debugging or customization. It is understood that a knowledge engineer[6] (KE) is needed to assist in this process. The KE needs to train the user in the practical and conceptual aspects of the tools and be available for consultation, but the goal is to have the educator use the tools relatively independently.

- *"ITSs"* are, for our purposes, computer tutors which behave differently for different curriculum characteristics and student characteristics, and in which this flexibility is represented *explicitly*, i.e. not hard-wired. Some characterize "intelligence" in ITSs in terms of how well they approximate human tutoring. However, our vision is more modest: to build artifacts that are better learning aides than textbooks, classroom lectures, or traditional CAI. In addition, the state of the art is far from being able to simulate human intelligence, and more importantly, ITSs, in our view, should not (and could not) replace human teachers.

---

[6] A knowledge engineer is a scientist or engineer who works with a domain expert to encode domain expertise in an AI system.

Another framing of this study's overriding question is: *What are the factors involved in enabling an instructor to create and evaluate a computer tutor with a high level of complexity and flexibility* (as described in the description of ITSs above)?

This research directly addresses the "problem areas" described earlier in this chapter by including the following **broad goals**:

1. Design an architecture supporting opportunistic invocation of **multiple tutoring strategies.**
2. Define a **conceptual vocabulary** for representing the objects, events, and relationships involved in tutoring.
3. Make the architecture and vocabulary **non-technical**, i.e., usable by teachers.
4. Design a **knowledge acquisition interface**[7] which facilitates rapid prototyping and easy creation, modification, and testing of both instructional content and tutoring strategies.
5. Incorporate selected findings from **instructional design theory** and **cognitive psychology**.

Our method of addressing these goals was to build the KAFITS (Knowledge Acquisition Framework for ITS) system[8] and test it on typical users. We refined the broad goals and the overarching question discussed above by positing a series of more **specific research questions**, listed below. We did not intend to produce definitive answers to these questions, but used them as context to guide the design of the research study.

1. What are important features for ITS knowledge acquisition systems?

[7]Knowledge acquisition is the process of acquiring an expert's knowledge for representation in an AI system.

[8]Usually "KAFITS" refers to both the framework and the interface, unless the distinction is relevant and so noted.

2. What aspects of knowledge acquisition systems (in general, and KAFITS in particular) are difficult for educators to grasp or use, and how long does it take to learn how to use it?

3. How does KAFITS facilitate content analysis of the subject matter?

4. How much independence can an instructor have in using KAFITS? Where in the design process is the knowledge engineer most needed (and why)?

5. How difficult it is for instructors to articulate what and how they teach?

6. How does an instructor's pre-existing knowledge help or hinder his use of KAFITS to design a tutor?

7. How domain independent is KAFITS?

8. How much time and effort does it take to build an ITS using KAFITS?

9. What kind of assistance/reference help is useful, both on-line and off-line?

10. What is the difference in training needed for different levels of users?

11. Does the teacher learn anything from the ITS design process that can be used in classroom instruction?

1.2.2 Description of the KAFITS System

In pursuing the above goals we have designed, implemented, and tested an ITS knowledge acquisition (KA) framework (a representational system) and a knowledge acquisition interface (a computer program) which reifies this conceptual framework for instructors. The framework and implemented system, called KAFITS, incorporates instructional design paradigms and facilitates rapid creation and manipulation of multiple tutoring strategies. The representational framework comprises a general system or language for describing what to teach and how to teach it. The knowledge acquisition interface is a tool (actually a set

of tools) which allows a teacher, sometimes with the assistance of a knowledge engineer, to design, test, and modify intelligent computer-based instruction.[9]

Figure 1.2 shows a high level diagram of the system components. Domain knowledge (examples, questions, topics, etc.) and strategic knowledge (tutoring strategies for how to use the domain knowledge) are stored in separate knowledge bases. The Browser is the user's interface to the domain knowledge base, and the Strategy Editor is the user's interface to the strategic knowledge base. The strategies can be thought of as rules which specify how to use the domain knowledge. An interpreter (or Tutoring Engine) uses the domain knowledge and strategic knowledge (along with information from the student model) to create tutoring sessions. The educator/user can test the tutor[10] (run it as if he/she were a student) and easily modify the domain and strategic knowledge to debug and improve it. In this study the Browser was built and tested with educators, and a prototype of the Strategy Editor was built, but not tested with educators. The system also has a student model (a model of the student's correct and buggy knowledge) and a student interface (which allows the student to take initiative during the tutoring session).

1.2.3 Overview of the Methodology

Exploratory nature of the study. Although research in the areas of multiple tutoring strategies for ITSs and knowledge acquisition interfaces for ITSs is quite limited, several research teams are building systems motivated by goals which overlap those expressed here, i.e. generic architectures and ITS authoring tools usable by educators,[11] and

[9]Note that the "knowledge acquisition" discussed here is acquisition of pedagogical and curriculum knowledge from the instructor. It is not acquisition of domain expertise, i.e., the system does not facilitate the acquisition of physics problem solving expertise from a physics teacher. Also note that KAFITS is a representational *framework* and an *interface* allowing an instructor to represent his/her knowledge in terms of that framework. Is is *not* a tool for *automatic* acquisition of knowledge from instructors or for learning from its own experience.

[10]In this document when we refer to a computer "tutor" we usually mean a tutor built using KAFITS, i.e. the KAFITS system combined with the knowledge base of a particular domain (unless the context clearly implies a different sense). Similarly "the tutor" will usually refer to the physics tutor built during this study.

[11]We describe several of these systems, and how this project relates to them in Section 2.1.

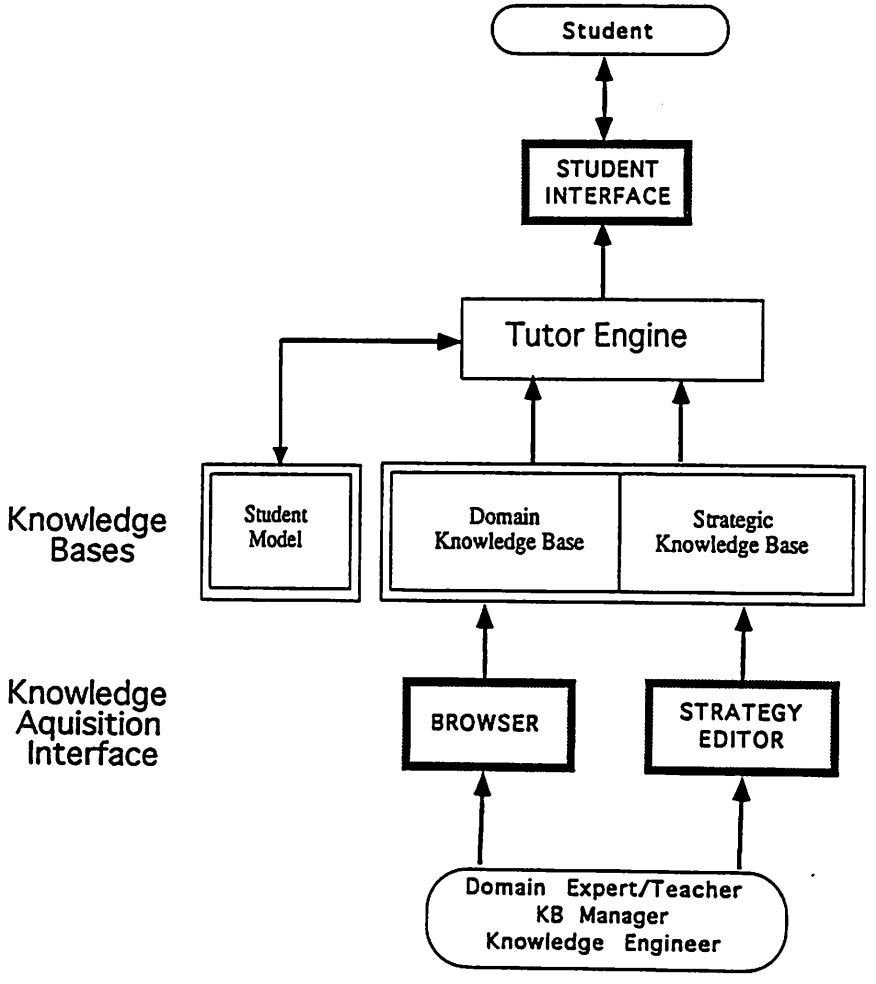


Figure 1.2 The KAFITS System

we have borrowed design ideas from these projects. Unfortunately there are no accepted or experimentally established guidelines for designing such systems, and little rationale for design decisions is given in the literature. Also, no current ITS project studies empirically how easy, powerful, or flexible the systems are for teachers or instructional designers to use. For all of the above reasons this research is "exploratory" and constitutes a plausibility study. The evaluation is formative and qualitative, as described below. We investigate the feasibility and practicality of a knowledge acquisition tool being used by educators to build an ITS, and report on the issues identified.

User participatory design. Development of the KAFITS system was based on a user participatory process [Bromberg & Henderson 1990], i.e. design and implementation are iterative and concurrent with use by a domain expert[12] (and two "knowledge base managers"[13]), allowing maximum design input from the user's perspective.

Formative evaluation. A user participatory design becomes a formative evaluation when the researcher keeps records of the difficulties and successes encountered, and the effects of modifications made to the system. Starting with a basic prototype system, we recorded observations of the instructor using the system and incrementally improved the system. In this document we report on aspects of the system that worked as well as those that did not, and give suggestions for future modifications.

Case study method. Since we are studying the construction of a single tutor as designed by a single instructor, the study is primarily a case study. The generalization possible from studying several domains or instructors is traded for a deeper analysis of a single case.

Outline of the study. The sixteen month study went through several phases, including: familiarizing the domain expert with the KAFITS framework, curriculum design,

[12]We use the term "domain experts" to refer to teachers or instructional experts using the KAFITS system. The term "user" refers to anyone using the KAFITS system for knowledge engineering (primarily domain experts and knowledge base managers), *not* students using the tutor.

[13]The knowledge base manager is a member of the ITS design team whose task it is to enter the knowledge as specified by the domain expert into the knowledge base, and test the curriculum for obvious errors (i.e. errors not related to the domain content).

implementation and debugging of the statics knowledge base, testing the statics tutor on 19 subjects, and refining and expanding the knowledge base. These phases are described in detail in Chapter 4.

Data collection Several types of data were collected, including: notes from structured and unstructured interviews, "edit records" recording the details of usage of the knowledge acquisition interface, "trace files" recording the details of student runs of the tutor, and taped debriefing sessions with students. These are described in Section 3.4.

The instructional domain. This study includes a case study of a high school physics teacher[14] using the KAFITS system to build a tutor for statics.[15] The topic network designed by the domain expert, see Figure 1.3, shows the scope of the curriculum. The domain expert's main goal was to give students a qualitative, intuitive understanding of the relationships between forces in static (non-motion) situations, and he intended the curriculum to be used after students had some initial exposure to the important concepts from a classroom or textbook. The curriculum focuses on developing a qualitative understanding of the following topics: Newton's Third Law, linear equilibrium, the properties of different types of forces (tension, gravity, and contact forces), and how to evaluate free body diagrams. Topics are classified according to knowledge type, such as fact, procedure, Mis-KU (misconception), etc. (see the key in the lower left of the figure). Links between nodes indicate relationships between the topics, such as part-of, critical-misconception, and various types of prerequisite links (familiarity, easy, typical, and difficult, etc.).[16] Tutoring strategies use the node and link types to determine the order to present topics. The most general topics in the network are: static forces, FBD (free-body diagram) solution analysis, and linear equilibrium.

Figures 1.4 and 1.5 are screen dumps from a tutoring session that illustrate typical qualitative questions asked of the student, and Appendix G shows a tutorial dialog from a

[14]Dr. Charles Camp is a physics teacher at the Amherst Regional High School, Amherst, Massachusetts. We refer to him as the "domain expert."

[15]The framework has been applied in two other domains in limited ways, but these are not discussed.

[16]The node types and link types are described in Section 3.1.5.

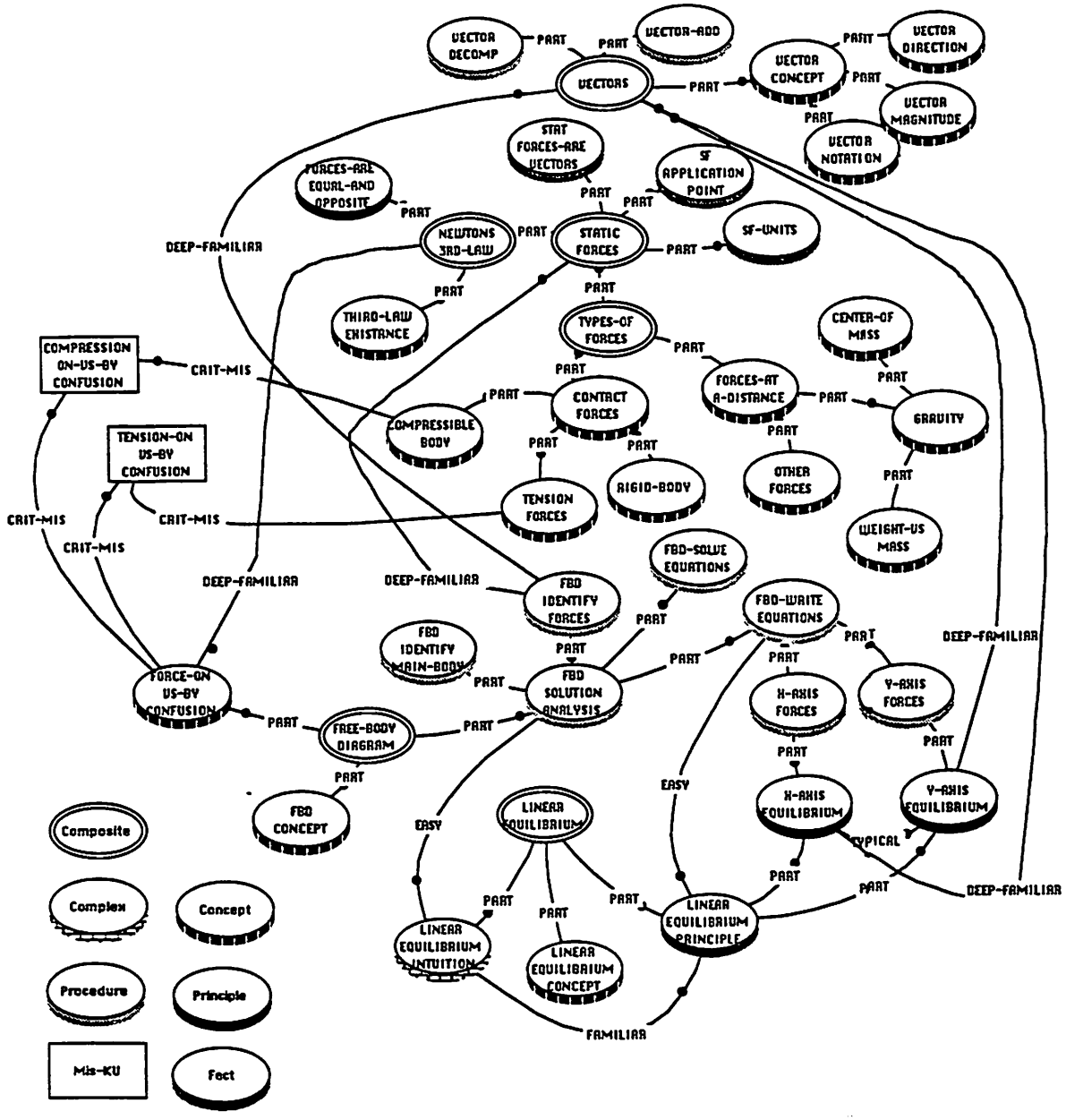


Figure 1.3 Topic Network

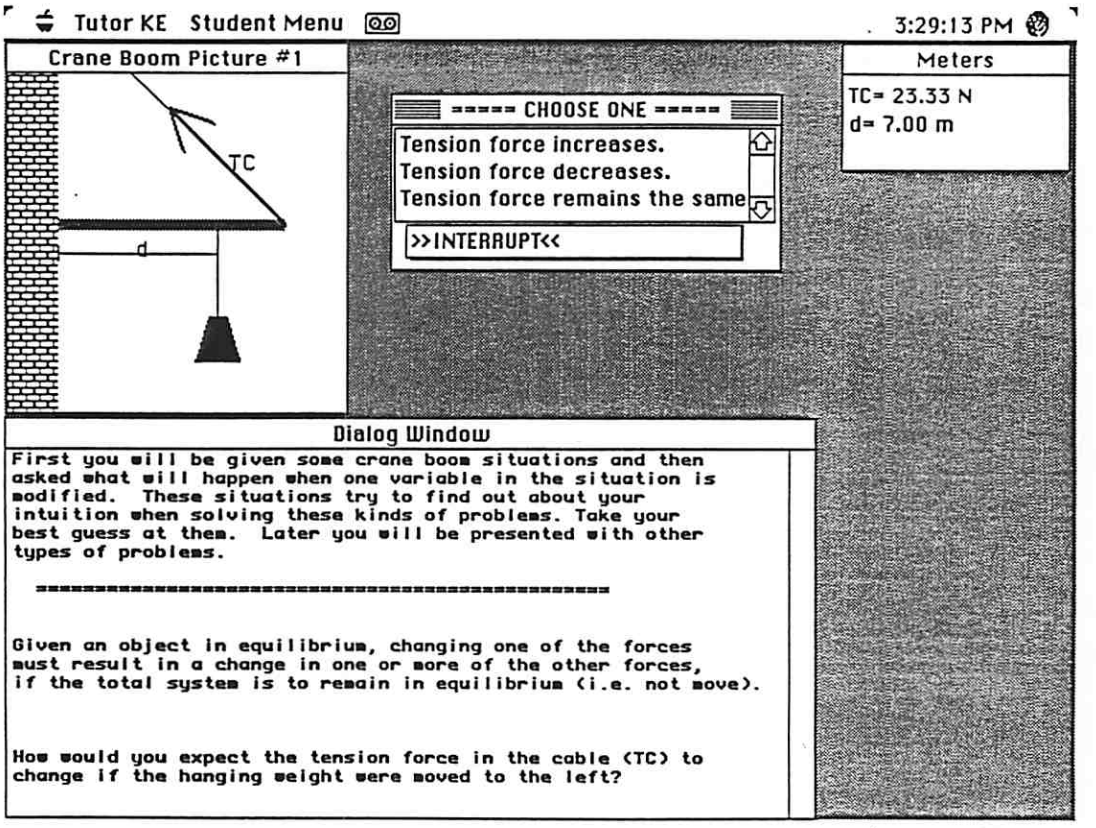


Figure 1.4 Sample Screen with Crane Boom

typical statics tutor session. Part of the curriculum centers around a learning environment called the "crane boom" in which the student can manipulate a simulated physical system and observe the resulting forces and force components [Duckworth et al. 1987, Woolf et al. 1988]. Figure 1.4 shows a typical question about the crane boom. In this case, the crane boom is brought up as a static picture. In other cases it is brought up as an interactive simulation for the student to manipulate and/or measure.

1.3 Scope of the Study

Though research in ITS has been ongoing for over two decades, due to the complexity of the problems needing to be addressed, it is still in a formative stage. Researchers in the field

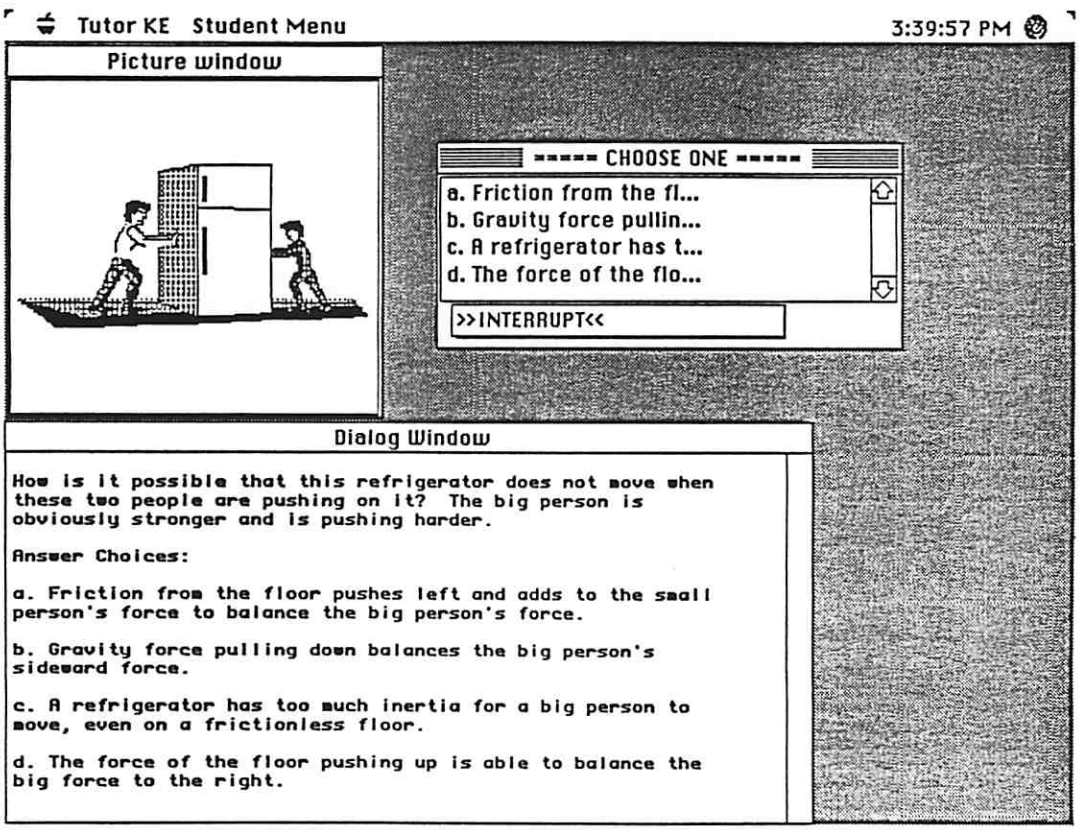


Figure 1.5 Sample Screen with Picture

are still defining the major research questions, while making modest gains in understanding and solving isolated issues. Much of the work done to date involves building systems to teach a particular subject, a post-hoc credit-blame analysis of the system's performance, and then generalizing the results. The generality of such results is limited. In contrast, this research does not aim to improve instruction in a particular subject, nor does it involve building a "performance system," i.e. one intended to be robust and complete enough to teach students in a stand-alone fashion. We address a small handful of the issues facing ITS designers, as listed in Section 1.1. Since it is just as important to delineate what a research project is *not* as it is to describe what it *is*, we list limitations to the scope of the study below.

Instructional principles and cognitive models. The study does not involve a descriptive explanation or inquiry into how teachers teach. It incorporates some results of instructional and cognitive theory, and tests whether these are acceptable and useful to instructors, but no cognitive model is proposed for the organization of knowledge in students or instructors.

Domain expert systems. The study is not an attempt to completely represent an expert's knowledge about the domain to be taught. Rather, it is concerned with representing pedagogical knowledge *about* the domain knowledge. There is no domain expert system component that can solve the types of problems given to the student.

Simplicity and modularity of domain knowledge. Our goal is to develop a framework that is general and easily used and modified. Sophisticated student modeling (such as model-tracing, as used with expert domain models [Anderson et a. 1985a]) and highly interactive simulations [Woolf et al. 1986] will not be included.[17] The tutor built for this

[17]Research focusing on the representation of complex problem solving behavior (i.e. computational modeling of performance in the domain being taught) requires detailed cognitive task analysis. It is also limited to procedural skills. Research that focuses on tutoring with sophisticated learning environments requires knowledge representation schemes and control methods tailored specifically to the learning environment. Learning environments tend to be focused on teaching domain-specific problem solving using non-directive or coaching strategies.

study does incorporate a ("black box") simulation environment, and there is only limited assessment of the student's behavior while using the simulation.

Representational adequacy vs. inferencing power. The system is "knowledge intensive," deriving its flexibility from the variety and large amount of information available to the tutoring strategies. Our near term goal is for *representational adequacy*—i.e. we are interested in determining what types of objects, attributes, and relationships are sufficient for a domain independent tutoring system with multiple strategies. The sophisticated AI inferencing needed to generate and recognize natural language, or compute questions and explanations from first principles, can be added at some future date.

Basic knowledge types. Because the tutoring is driven by topic network traversal, rather than by a computational model of problem solving, the tutoring styles supported tend to be more directive than in "coaching" tutors. The KAFITS framework supports the teaching of "basic" types of knowledge more readily than "complex knowledge and skills." Basic knowledge types include concepts, facts, simple procedures, and principles. These are usually *prerequisite* to more complex skills such as heuristic problem solving and metacognitive skills [Gagne, 1985]. The teaching of complex skills seems best accomplished with predominantly student-controlled, problem-driven, and remediation-driven tutorial environments, while teaching basic types of knowledge is best accomplished by more directive, tutor controlled environments. Nevertheless, the designer can create strategies that simulate a wide range of styles: from more directive to more student controlled, from minimum to maximum feedback, from error driven to curriculum driven, etc.

Student learning. This research is focused on knowledge acquisition, i.e. the process by which the teacher's information is transferred to the tutoring system. We have not measured whether students learn better (vs. any other method) with this system, or how various aspects of the system affect the learning process. Student learning is more a function of what the domain expert enters in the knowledge base than it is of the KAFITS representational framework. We are, however, interested in how the teacher uses the system

in the overall design process, i.e. how he/she uses feedback from test runs with students to modify the knowledge base.

Student initiative. Regardless of the teaching strategy, some degree of student control is crucial in all tutoring systems. The system allows students to interrupt the tutoring session to ask for information, ask to be taught a new topic, change the teaching style, skip forward or back up in the lesson, etc. We are primarily interested in how the design of the framework facilitates or inhibits students' ability to determine the amount, content, and sequencing of the course material. We did not investigate students using this capability or how this capability relates to learning.

1.4 Contributions

This study makes several original contributions to research in the field of Intelligent Tutoring Systems as described below.[18]

Tools for Rapid prototyping and tailoring of ITSs. KAFITS allows an instructor to easily and rapidly design, test, and modify an intelligent tutoring system, and it allows an to instructor customize an existing ITS for his/her content preferences and teaching style.

User participatory design of a general ITS framework. Though other general ITS frameworks have been designed (see Section 2.1) none (that we are familiar with) have been designed in a user participatory way. Therefore we believe that the KAFITS representational framework and user interface are more comprehensible and relevant to practicing teachers. We also identify general design issues and tradeoffs for building ITS knowledge acquisition interfaces.

Case study of the use of knowledge acquisition tools. This study is unique in that we report on the problems, issues, and tradeoffs encountered when educators are

[18]Determining which aspects of the research are original contributions is based on our survey of the literature, as summarized in Chapter 2.

involved in hands-on ITS design and construction, and we base our analysis on empirical evidence. Reports of other general ITS frameworks or shells do not discuss issues encountered when the systems are used by educators.

Multiple strategies. Other generic ITSs do not include a general control mechanism that allows multiple tutoring strategies, or a knowledge acquisition interface for creating and modifying multiple tutoring strategies.

1.5 Limitations to Conclusions

There are several factors which limit the generality of our contributions and analysis (also see Section 1.3 above for areas in which we do not plan to draw conclusions or make substantive contributions).

Domain independence. Since this is a case study involving the design of a single computer tutor, possible claims about the domain independence of the KAFITS framework and interface are limited.

Low number of subjects. Our conclusions about the usefulness of the KAFITS tools and our knowledge acquisition method are limited because we observed only three educators using the system (the domain expert and two knowledge base managers).[19] However, as explained in Section 5.4.1, the original prototype KAFITS system, and many of our conclusions, are based on experiences working with educators outside the scope of this study.

Role of the experimenter. Since the author designed and implemented the KAFITS framework and also acted as the knowledge engineer in this study, our ability to make objective observations is limited. Possible claims about the usefulness of KAFITS with an arbitrary teacher and knowledge engineer are similarly limited.

[19] But see Section 2.5.2 for a discussion of the benefits of case study research.

Moving targets. Since we have followed a formative evaluation process and modified the system incrementally, there is no base-line system which has been constant over the entire experiment. This may make it difficult to draw conclusions from observations made at different stages of the design process.

1.6 Guide to the Reader

This document is primarily addressed at those involved in ITS research or construction. However, we have made every attempt to make it readable by anyone having a modicum of familiarity with basic ITS concepts. A glossary of terms and important concepts can be found in Appendix A.

Chapter 2 is a review and synthesis of various sub-fields of the literature, including sections on generic intelligent tutoring systems, instructional design theory, knowledge acquisition, AI systems design and research methodology, and qualitative evaluation methods. The section on generic ITSs will familiarize the reader with the important issues in ITS design. At the end of each section is a summary, followed by a discussion of how this study and the KAFITS system relate to issues identified for that literature sub-field.

Chapter 3 is a detailed description of the KAFITS system, including the representational framework, the knowledge acquisition interface, the student interface, and implementation and extendibility issues. Only a cursory reading of Chapter 3 is needed before reading the analysis and summary chapters (Chapters 5 and 6). Readers not familiar with the terms "instance," "object," "method," and "slot" may want to read the descriptions of these terms in Appendix A before reading Chapter 3.

Chapter 4 describes our research method. We describe the subjects and domain chosen for the case study, and argue that the participants are typical users of the software. We also give a time line of the study and discuss data collection techniques.

Chapter 5 gives results and analysis of the data collected. We discuss results, issues, and tradeoffs for knowledge engineering, knowledge representation, and interface design. The reader who wants to skip details of how conclusions were reached can read the summaries that follow each section (and some sub-sections).

Chapter 6 summarizes the results and contributions of the study and proposes several generalizations to our results, including: a high level design specification for generic ITS shells, an ITS knowledge type classification scheme, and ITS student interfaces. We also suggest a number of research projects that build upon this study, and make recommendations to the educational community regarding the use of ITS shells in public education and industry.

CHAPTER 2

LITERATURE REVIEW AND THEORETICAL BACKGROUND

This review and synthesis of various sub-fields of the literature serves two purposes: it gives the reader a general understanding of relevant concepts, issues, and trends in the literature, and establishes a theoretical foundation upon which to base this study. The main areas investigated are: intelligent tutoring systems, instructional design theory, AI knowledge acquisition, AI systems design, AI research methodology, and qualitative evaluation methodologies for AI tutoring systems. Also included are references from learning theory (including cognitive science), and human computer interactions (usability and interface design).

The first section is an overview of generic knowledge-based tutoring systems and shells, including a brief general introduction to intelligent tutoring systems.[1] This section discusses implemented systems that are designed for generality and domain independence and provide ITS construction tools usable by non-programmers. We reference authors that fall within the ITS literature and authors that are more closely associated with instructional science.

The second section is a survey of aspects of the instructional design literature related to ITS. Instructional design (or instructional theory, or instructional science) aims at prescribing optimal methods of instruction. Some references from learning theory (including educational psychology and cognitive psychology) are included, but learning theory is not the main focus (as it is in many other ITS investigations). This is because learning theory

[1] It is suggested that readers not familiar with the ITS field look elsewhere [Wenger 1987, Ohlsson 1986] for a more complete overview of the field.

is involved in describing how the mind works, giving general suggestions or paradigms for instruction, but for a theory to be most useful to ITS designers it must go a step further and prescribe specific *instructional methods*—and this is the domain of instructional theory. We look to the instructional literature for basic components or structures necessary or desirable for a generic computer tutoring framework. We do not focus on specific instructional theories except for Component Display Theory (CDT). We describe CDT because it is a good example of an instructional theory that is at the same time general, concrete, and simple, and because we borrowed from CDT in designing the KAFITS framework.

The third section is a discussion of ITS research and design methodologies. We discuss how engineering and research goals are intertwined, and how software design and evaluation methods are intertwined. We include discussions of iterative design, user participatory design, and interface design. The fourth section is an overview of AI knowledge acquisition methods and issues. We describe and compare a number of methods applicable to ITSs. The fifth section is a discussion of ITS evaluation methodologies. We describe how evaluation methods taken from the fields of AI, education, and psychology, can, and should, be used to evaluate intelligent tutors. The section focuses on qualitative and formative evaluation methods.

2.1 Generic Intelligent Tutoring Systems and Shells

In this section we review AI instructional or tutoring systems designed to be domain independent and discuss design issues for generic ITSs. Many of the systems mentioned lean toward what might be called "authoring systems" or "shells" for ITSs.

2.1.1 A Definition of Intelligent Tutoring Systems

There are divergent opinions in the field about what an intelligent tutoring system is, or what constitutes research in the field of ITS;[2] therefore we describe some definitions of ITS found in the literature and then give our own definition. Also, the reader will also be afforded a brief overview of intelligent tutoring systems.

A prime motivation for using computers to teach is their potential to dynamically tailor instruction to the needs of individual learners. "The computer can, in principle, be programmed to adapt both the *content* and the *form* of instruction to the student's understanding of the subject matter" and to other parameters of the instructional setting [Ohlsson 1987]. Computer aided instruction (CAI) was one of the first applications of computers, but CAI (in its traditional form) has fallen far short of the above vision. More recently, education was one of the first application areas of artificial intelligence (AI), and AI was seen as the path that could restore and realize the vision of the individualized computer tutor.

Carbonell, working on SCHOLAR, one of the earliest intelligent tutoring projects [Carbonell 1970], proposed a paradigm shift in computer aided instruction. In traditional CAI, which Carbonell called a "frame oriented" approach, segments of curriculum were represented in pre-stored units (often called frames) which were presented to the student in a fixed, pre-defined sequence. Carbonell proposed an "information-structure-oriented" approach in which the domain knowledge was represented explicitly. Such a system, he argued, would have greatly increased flexibility in responding to the student by, for example, answering student questions and generating tutorial dialogues. Carbonell's paradigm shift emphasizes that intelligent tutors are "generative," and is, to this day, still a fair description of the difference between CAI and ITS. Similarly, Wenger [1987, pg. 7] (in a fairly recent overview of the field) says that the main feature of ITS which distinguishes it from

[2]One approach to this controversy is not to use the term "ITS" at all, i.e. to avoid preconceptions by using such terms as "knowledge-based tutoring system," or "interactive learning environment." This results in the area of fuzziness being shifted a bit to one side or the other, but questions about what systems or research fit into that category remain.

CAI is "the shift from the programming of decisions to the programming of knowledge." He uses the term "knowledge communication systems" for ITSs.

One of the first general paradigms for an ITS framework, and perhaps the only one that has ever been generally accepted, is the three-module division of expertise, proposed in the early 1970s, and often cited today [Woolf & McDonald 1984]. The three components are:[3]

- a *student model* component that stores information and makes inferences concerning the student's knowledge, knowledge "bugs", and learning preferences,
- an *expert system* component that models expertise in the domain being taught, and
- a *tutoring* component, embodying knowledge about how to teach.

Clancey [1986a] views AI as a science dealing with the construction of computational qualitative process models.[4] AI-based instructional programs can represent three kinds of processes qualitatively: human reasoning (from the perspective of either the expert or the student), real world processes, and the communication process (which includes teaching and diagnosing the student's knowledge). He notes that depending on the domain, the instruction may focus on conveying a reasoning process (as in mathematics or computer programming) or a physical process (as in geology or mechanics). The domain content model (or "subject material" model) of an ITS contains "correct" models of the real world and/or reasoning process. The student model contains dynamically updated "novice models" of the real world and/or a reasoning process. Clancey's definition of an ITS is: "[an] AI-based instructional program [that] represents at least one component in the form of a qualitative model."

In contrast, Ford [1988] suggests appraising the level of "intelligence" of an ITS system according to how it stands up to fifteen criteria phrased as questions (developed by Self [1985b]), for example: "can the system answer arbitrary questions from the user about the

[3]Variations on this taxonomy exist, such as including a communication component that contains natural language and discourse expertise, or a component that deals with the tutor's interface to the student.

[4]I.E. models that describe objects and processes in terms of spatial, temporal, and causal relationships.

subject?" and "can the system give alternative explanations?" This constitutes defining or appraising ITSs according to how closely they measure up to the capabilities of a (good) human tutor. This approach has its uses, but it also has significant limitations. It may be appropriate to use this ITS definition if one's goal is to improve education for a specific topic. However, in ITS research, where we try to gain a deeper understanding of ITS design issues, it is better to focus deeply on specific research areas than spread the effort over all desirable ITS features. Also, though Ford's approach is useful in helping us visualize the long term goals for the field, performing a "check-mark" evaluation of an ITS system offers little substantive information to other researchers to guide their endeavors.[5]

In summary, we gave given several common criteria for determining whether a computer tutor is an ITS: a generative knowledge view, a three-component view, a qualitative processes view, and a performance criterion view. Our definition harks back to Carbonell's emphasis on the generativity of ITSs. We characterize the "intelligence" of the system in terms of the flexibility of its response, or of the space of possible responses. For the purposes of our discussion, **an intelligent computer tutor is** *a computer tutor which can behave differently for different students, discourse situations, and/or subject matter characteristics, and (most importantly) where this flexibility is represented explicitly* (i.e. all potential curriculum paths and discourse paths are generated, not stored). In accordance with Wenger (above), rather than storing each decision, the knowledge enabling those decisions is stored.

2.1.2 Early Systems

Though the designers of many tutoring systems have laid claim to the generality of their methods or design philosophy, until recently very few ITSs were general in more than principle or extrapolation. Before discussing current generic ITS shells, we review how several early systems or proposals contributed to our current understanding of generic ITSs.

[5]Unless one's research goal was to evaluate the interaction of ITS sub-systems, in which case focusing on many desirable ITS features is more appropriate.

The three-component paradigm for ITS systems. The three-component model mentioned above (composed of the domain module, student model, and the tutoring module) is a workable paradigm for expressing the capabilities that a "truly intelligent" tutor must have, i.e. capabilities that good human teachers have that require sophisticated reasoning (and therefore, supposedly, AI technology). However, it has not been a useful way to model the subsystems of ITS software. Few systems (and no generic systems) following this architecture completely and cleanly. One reason is that an ITS with robust functioning in all three areas is beyond the state of the art in AI, and research projects have focused on specific issues within the paradigm. A more important reason that the three-component paradigm is a "paradigm," and not a design framework, is that in practical (and perhaps also theoretical) terms it is extremely difficult to cleanly break a tutoring system's functionality into the three independent modules. There is much overlap between the domain (inputs) and range (effects) of the inference rules for the three sub-systems. The interdependence of the three main functions of an ITS has led designers to use more cohesive architectures, such as object-oriented programming (discussed below). Though the three-component paradigm has not worked well to model the subsystems of an ITS, it *is* often and successfully used as a model of the *knowledge bases* needed for tutoring.

Other issues in early systems. In order to evaluate generic ITS frameworks on theoretical or empirical bases they must have explicit knowledge representation schemes.[6] Ideally all software modules will have explicit representational languages or frameworks. Several early ITS systems approached this goal. For example, Clancey's [1982] pioneering work on the GUIDON system emphasized the separation of knowledge structures (the knowledge base) from the procedures that interpret knowledge (the inference engine), and emphasized the importance of separating the domain knowledge from the tutoring knowledge. The GUIDON system represented both domain and teaching expertise in production rules; O'Shea's tutor for quadratic equations [O'Shea 1982] was another early system that

[6]Being explicit about the structure and allowed relationships between entities in the knowledge base is important. It is *not* as crucial to be explicit about how these schemes are *implemented* in the computer.

used a production rule representation of tutoring expertise.[7] Production rule formalisms have some disadvantages [Clancey 1985, Lesser 1984], leading Woolf [1984] to use a three-layered transition network and meta-rules to represent tutoring and discourse knowledge in her MENO Tutor. The tutor was able to simulate segments of flexible tutorial discourse. It is also noteworthy that the system was tested for two domains (unfortunately, it is rare that generic ITS frameworks are applied to more than one domain).

Much was learned from the early attempts at general ITS architectures. None of these systems, however, have been taken on by other research teams as truly general mechanisms or "shells" from which to build tutors (indeed, most were intended to be only research vehicles). The last few years have seen a resurgence in efforts to design generic shells, authoring systems, and theoretical frameworks for intelligent tutors.

2.1.3 An Overview of Current Generic Systems.

We will look at seven tutoring system research projects which have generality as a main design goal: Training Express, PTA, Bite-Sized Tutor, MAIS, IDE, ID Expert, and Expert CML. The first three systems were designed as generic shells based on extensions to the traditional ITS paradigms. The other four were created as tools for instructional theory and curriculum development, but all incorporate aspects of traditional ITS (including a student model and AI knowledge representation techniques). Though other generic ITS projects are described in the literature, we chose to look at only functioning systems. All seven systems are at least partially implemented and most have had initial trial runs. However, research on ITS shells is in its infancy—none of the systems are being used routinely and none have undergone evaluation. We describe the seven projects and discuss their unique aspects in light of these **desirable characteristics for generic ITSs or ITS shells** (each characteristic is described in detail later):

[7] A unique feature of the quadratic tutor was that it was "self improving." Based on statistical information from past tutoring sessions, the program deduced rule changes likely to cause improvement for certain high level goals, such as decreasing student time on task, decreasing computation time, and increasing student scores.

1. domain independence and sufficient scope,
2. usability by educators,
3. theoretical basis for instructional decisions,
4. explicit representation of domain knowledge,
5. explicit representation of strategic knowledge,
6. implementation and evaluation for practicality and usability.

Training Express. Training Express [Clancey & Joerger 1988] runs in conjunction with a traditional expert systems, and, using some pedagogical information from an instructional designer, produces a tutor that teaches the expert knowledge. Clancey's main goal was to create a *practical*, low overhead, conceptually simple tutoring mechanism. An interface is provided which allows the instructor to adapt the rules in a traditional expert system shell (the M1 shell) by inserting "break points" and elaborations into the rule base. The instructional designer defines problems (cases) and key concepts, and associates the concepts with specific expert rules. The resulting tutoring system is an apprenticeship style tutor. The tutor can ask the student to solve problem cases and can interrupt the student to ask probing questions about important concepts. The student can ask the tutor for help solving problems and ask how the tutor reached its conclusions. The system has no explicit representation of the student model or tutoring strategies. Its main feature is its practicality—Clancey has demonstrated (subject to further testing with students) that a workable generic intelligent tutor based on an expert system domain model can be realized within a simple framework. The system's greatest source of power is also its biggest limitation: it can only teach domains that can be represented using production rules, i.e. domains with simple procedural skills.

PTA. The PUPS Tutoring Architecture [Anderson & Skwarecki 1986] is a generic shell for tutors that incorporate the model tracing paradigm used in Anderson's previous tutoring systems (including the Geometry Tutor [Anderson & Reiser 1985] and the LISP

Tutor [Anderson et al. 1985]). Like Clancey's GUIDON system, both domain expertise and tutoring expertise are represented in production rules ("goal-driven" production rules in this case). Unlike any of the other systems we discussed, the domain model has cognitive fidelity, and the student model is runable. The student model is a rule-based system itself, containing a subset of the expert rules, plus some "buggy" rules, which can be used to predict or explain student behavior. The knowledge representation and design philosophy of Anderson's tutors are based on his ACT* theory of human cognition [Anderson 1983]. The tutoring strategy, also inspired by ACT* is, as in Training Express, fairly simple and straight-forward. Since representing expertise in terms of production rules and cataloging potential buggy rules is a difficult task, it is unlikely that PTA can be used by teachers or domain experts without significant training.

Bite-Sized Tutor. The Bite-Sized Tutoring Architecture [Bonar, Cunningham, & Schultz 1986] is an ITS authoring language that uses an object-oriented representational paradigm. The goal of the system is to allow rapid prototyping, testing, and modification of tutoring systems, with the involvement of domain experts who are not programmers. Traditional tutoring systems are organized around functional components such as "diagnoser," "explainer," "tutor," etc. In designing instructional systems one finds that the information relevant to a given curriculum item (such as "Kirchoff's Second Law" or "How to use a graph") is distributed over several of the functional components. Therefore, in practice, it is hard to design these subsystems to be independent of each other. Also, the domain specific information is hard to modify and not very modular. In the object-oriented paradigm the information needed to explain, diagnose, and teach each bit of domain knowledge is organized around that piece of domain knowledge, making the knowledge base very modular. Also, objects can inherit properties from similar or more general objects, decreasing representational redundancy and potential inconsistencies in the knowledge base. One distinguishing characteristic of this research is that tutors in several domains are being implemented using this system. This research group (University of Pittsburgh's Learning Research and Development Center) has developed tutors for programming, economics, electricity, and hydrostatics that use (or have inspired the development of) the Bite-Sized

paradigm. The Bite-sized paradigm facilitates ITS construction by AI programmers and but there it does not address the issues of tools or methods which allow educators to understand and participate in building ITSs.

MAIS. The Minnesota Adaptive Instructional System [Tennyson 1986, Tennyson & Christenson 1988] is based on a specific instructional theory. It is unique among the systems described here in two ways. First, it is self improving ("adaptive")—its instructional strategy is based on nine instructional variables[8] that are adjusted continuously based on a statistical analysis of the system's performance (i.e. the success and efficiency of the student's learning). Its second unique feature is that it incorporates affective variables (motivation, perseverance, personality) in its student model. Its instructional strategy is a Bayesian conditional probability model embedded in computer code which adapts and personalizes instruction. Though unique in these ways, MAIS is limited in that it does not have an explicit representation of its strategies or a user interface for inspecting and changing its knowledge.

ID Expert. The Instructional Design Expert [Merrill 1987, Merrill 1989] is an expert system that assists instructional designers in course specification. Like MAIS, Merrill's system is founded on a specific instructional theory, his Component Display Theory. Component Display Theory (described in more detail in a later section) prescribes a mapping from the characteristics of what is taught (e.g., whether it is a skill or a concept) to methods for teaching (for example how many and what kind of examples to give). Unlike most other systems described here, in which the user creates and modifies objects in a knowledge base in an un-constrained manner, knowledge acquisition in ID Expert takes the form of a long series of specific questions to the instructional designer. The system's input comes from an interactive session in which the user is prompted to specify and refine taxonomies of concepts, common errors, and behavioral objectives. The major drawback of this system is that the output is a written course specification (detailing the ordering and manner of

[8]The variable are: number of examples, amount of information, sequencing of information, format of information, learning time, corrective error analysis, mixed initiative level, amount of advisement, and amount of refreshment and remediation.

presentations and tasks to be given to the student), and the capability to produce on-line instruction from the instructional specification has not been built yet (see [Merrill et al. 1990] for a description).

IDE. The Instructional Design Environment [Russel, Moran, & Jordan 1988] was developed at Xerox Park and built using the Notecard system [Halasz et al. 1986]. It is a tool which assists in the design and development of curriculum, content, and delivery of instructional material. Unlike the above systems, IDE is designed to be used by multiple experts designing large curricula, and (since is is based on the Notecard system) it has a user friendly interface for inspecting and modifying knowledge. The output of IDE includes: a Knowledge Structure (or concept network) representing the knowledge to be taught; a set of Instructional Units (text, pictures, simulations, etc., that the student can interact with); and a set of Course Control Rules which guide the sequencing of Instructional Units. This information is passed to the "IDE Interpreter" [Russel 1988] which generates an on-line tutorial using the output of IDE. Alternatively, IDE's output can be used by an instructor as a specification for a non-computer-based course. Instructional goals are represented in an AND/OR tree, constructed by interpreting rules in a forward chaining manner, with backtracking and replanning as needed.

Another distinguishing characteristic of the system is that the instructional designer can record justifications for design decisions using "rationalization" links between objects in the knowledge base. Justifications are a form of documentation which makes it possible to build large tutorial data bases with multiple designers.[9] However, the justification arguments consist of canned text typed in by the user, so there is no automatic inferencing from evidence to yield instructional decisions.

Though the IDE framework is potentially powerful (though this has not been demonstrated empirically) its complexity may be prohibitive for practicing educators to use, and it is limited in its ability to allow teachers to define tutoring strategies.

[9]Thus justifications (arguments based on evidence) can be recorded for all rules, objects, and information in the system. For instance, general Tutoring Principles are justified by Literature References. Cognitive Principles for teaching a particular domain are justified by Tutoring Principles and Course Objectives.

Expert CML. The Expert Computer-Managed Learning system [Jones & Wipond 1989] was developed with the help of a team of instructional experts. Like ID Expert, it contains an expert system in the domain of instructional systems design. Unlike ID Expert, it synthesizes rules from several instructional experts and instructional theories. Its interaction with the user is open ended, compared to the interactive dialogue of ID Expert. Like IDE, the structure of the objects in the system encourages top-down design, and it supports multiple authors working on large courses. It also has a fairly sophisticated user interface. The user is guided through the design process by templates, instantiating courses, topics, lessons, objectives, learning activities, etc. Unlike IDE, it uses its instructional design expertise to check consistency and completeness within the knowledge base. Examples of the types of warnings or advice generated by the expert rules are: "topic XX is of type 'concept', but none of its sub-topics are concepts," and "the sub-topics of topic XX will probably take longer than the learning time allocated for topic XX. You may want to split topic XX into parts." The output of Expert CML is a runable (on-line) course with student monitoring capability. Since little information is given about Expert CML in use in realistic situations, its practicality, usability, flexibility, and expressiveness are unknown. Though its curriculum representation is powerful, its tutoring strategy representation is limited.

Other Systems. Following is a list of other instructional systems that have design goals that overlap with the systems described above (such as generality and usability by non-programmers), and have been at least partially implemented. These systems were not included above because only brief descriptions were found, because there was no indication that they were used by anyone other than the designers, or because they focus on a limited instructional area.

- BB-IP. A blackboard-based dynamic instructional planner [W. Murray 1990].
- COACH. A shell for intelligent help systems [Winkels et al. 1988, and Breuker et al. 1987].
- DOCET. Didactics On Computer: an Environment Tutor [Bonarini, Filippi, & Muti 1988].

- DOCENT. An intelligent system for assisting teachers in the planning and analysis of instruction [Winne & Kramer 1988].
- Micro-search. A "shell" for building systems to help students solve non-deterministic tasks [Sleeman 1987b].
- PIXIE/LMS. A shell for developing intelligent tutoring systems [Sleeman 1987a].
- SCALD. Scriptal Computer Aided Learning Designer [Nicolson 1988].
- SCENT. A Student Computing Environment for LISP programming that incorporates a black-board based instructional planner. [McCalla & Greer 1988].
- SIIP. Self Improving Instructional Planner [Macmillan et al.1988].
- TOTS. Task-Oriented Tutoring System [Rickel 1988].
- The Teacher's Apprentice. An intelligent authoring system for ITS mathematics [Lewis et.a l 1987].

Proposed functional or theoretical ITS frameworks which have *not* been implemented can be found in: Bumbaca [1988], Begg & Hogg [1987], Derry et al.[1988], and Cerri [1988].

2.1.4 Design Issues for ITS Shells

Here we discuss the desirable characteristics of ITS shells in terms of design issues, comparing and contrasting the seven systems described above.

Domain independence and sufficient scope. *ITS shells should be able to encode domain and tutoring knowledge from a variety of domains and should be able to cover a realistic segment of curriculum* (between a dozen concepts and an entire course).

All of the systems have described domain independence as a major design goal (as compared with most ITS systems, which were designed for a specific domain, and for which domain independence is argued for post-hoc). When we consider the variety of domains

dealt with by ITSs (including programming, geometry, industrial training, economics, structural analysis of airplanes, grammar, etc.) and the tutoring styles used (gaming, scientific inquiry, apprenticeship learning, example-based concept learning, constrained-path problem solving, etc.) it is hard to imagine *any* system being general enough to cover all of these possibilities. In fact, though each system is domain independent, their frameworks for representing teaching knowledge limit their scope, in *practical* terms, to certain kinds of knowledge or interaction styles (some of the authors acknowledge this limitation explicitly). This is only to be expected, given the magnitude of the ITS problem; however, it is desirable for designers and authors to be *explicit* about these limitations (which, if made explicit would be design constraints rather that limitations). Unfortunately, stating these limitations clearly will be difficult until the field has a descriptive taxonomy for the many types of domains and interactions.

Usability. *ITS shells should be usable by educators who are not programmers or AI scientists.*

All of the systems claim to be usable by instructional personnel who are not familiar with AI concepts or programming. For many of the systems this has yet to be demonstrated, and IDE is best suited for domain experts who are also experts in instructional design theory. PTA and Training Express both require that domain expertise be modeled computationally using a rule-based formalism. This is prohibitive for most teachers, though Training Express is designed for users with "minimal training or knowledge engineering experience." Whether or not these systems can be used by teachers "off the street," there is still a need for tools that instructional research and development teams can use to build tutors without starting from scratch each time.

There are also differences among the systems described in terms of the complexity of the representational frameworks, and in the amount of training needed to use them. ID Expert and Expert CML are designed for instructors with minimal training in instructional theory or the system's framework. The amount of on-line assistance provided to the user (discussed below) significantly influences the training needed. Expert CML and IDE are

designed to allow several curriculum developers to work together designing a course over a long time period. For all of the systems (except ID Expert) a fair amount of training in the conceptual framework of the system seems necessary to design a course or tutor from scratch. However, much less training should be needed for an instructor to modify an existing knowledge base to correct bugs or tailor it to her needs. The difference between these two levels of use is not elaborated on in the literature, so one can't say how easy it is to make small knowledge base modifications using these systems.

Overall, there is a tradeoff between power/flexibility and ease of use. Frameworks with simple knowledge structures or tutoring strategies can be used with less training, but the tutorial behavior is comensurally limited.

Assistance provided. *On-line assistance is an important component of usability.*[10] As mentioned, IDS Expert guides the user through the knowledge acquisition process with a detailed interactive dialog. If the dialog is specific enough, the resulting knowledge base can be guaranteed to be well-formed. However the user is constrained to wade through dozens (or hundreds) of questions, and cannot design and modify the knowledge base in an opportunistic manner. This may be appropriate for a novice instructional designer, but an expert or seasoned user may feel overly constrained. Expert CML allows for a flexible design process, and carries out consistency and completeness checking on the curriculum knowledge. It is designed to be used by both experienced instructional designers and novices, including even student teachers (though this has yet to be demonstrated). Training Express helps the user analyze the rule base to determine where it is best to set "break points."

Theoretical basis for instructional decisions. The teaching methods used (implicitly or explicitly) by tutoring systems should be founded upon learning and/or instructional theories suitable to the subject matter area. However, *tutoring shells should allow*

[10]Usability should be designed into many aspects of an ITS shell, including the conceptual vocabulary, the representational framework, and the knowledge base tools. Unfortunately the only clear evidence of usability features in the descriptions of most of the systems is related to help, assistance, and knowledge base consistency checking.

for a variety of strategies, and should be somewhat strategy-independent as well as domain independent.

The systems we are comparing differ greatly with respect to their theoretical basis: PTA is based on a tested theory of human cognition; instructional decisions in MAIS are based on empirical testing of instructional design variables; ID Expert is based on Component Display Theory, a well established instructional theory which is supported by cognitive principles; IDE is based on the experience and knowledge of a team of instructional design experts; Bite-Sized Tutor and Expert CML allow different strategies to be represented, but are not committed to any instructional style.

Explicit representation of domain knowledge. What separates these systems from CAI authoring shells is the *flexibility and power gained from representing knowledge explicitly using AI knowledge representation paradigms.*

Most of the systems incorporate network representations of topics (domain knowledge). Most use frame-based representations of the objects in the system. Many of the systems use rule-based representations of the domain knowledge and/or teaching strategies. *All* of the systems use at least one of these three representational methods, and many use networks, frames, *and* rules.

Training Express and PTA use rule-based (expert system) representations of domain expertise. All of the other systems (except MAIS) use frame-based representations of curriculum information, usually in the form of topic units (or "modules") arranged in a curriculum network or taxonomy with instructional units that specify specific tutorial interactions. ID Expert, Expert CML, and IDE have explicit representations of instructional goals. ID Expert, Expert CML, and MAIS incorporate expert systems in instructional design to aid in the development of the knowledge base.

Explicit representation of strategic knowledge. *Tutoring strategies should be represented explicitly, and in a form that can be manipulated. In addition, domain knowledge and tutoring strategies should be represented separately.*

There is a wide variation in how the systems implement instructional expertise.[11] PTA uses rule-based tutoring strategies. Tutoring rules are represented explicitly in IDE and Expert CML. Both of these systems, as well the the Byte-sized tutor, allow encoding of arbitrary strategies. It is not clear what instructional expertise looks like in the Byte-sized Tutor, or in ID Expert. In MAIS the teaching strategy is represented in computer code, incorporating many numerical/statistical values. Though the strategy is opaque, MAIS is unique in that instructional *variables* are clearly indicated. Training express, like PTA, has fairly simplistic and fixed (though perhaps effective) instructional strategies, so there is less need for explicit representation of strategic knowledge.

Implementation and evaluation for practicality and usability. It is no longer very useful, as it was when ITS was in its infancy, to report on ITS systems or shells that have not left the drawing board. In addition, when systems are implemented there is a danger that they will not be practical or usable if they are developed in the isolation of the lab. *Evaluation should occur in concert with development, and arguments for generality and usefulness should be supported with data.* Ideally, this data should come from:

- incorporating realistically large knowledge bases,
- encoding multiple non-similar domains and teaching styles,
- use by many instructional designers with varying experience, and
- demonstrating that students learn from the resulting curriculum, and are comfortable with the resulting learning environment.

Unfortunately, given the practical constraints and priorities under which research and development groups operate, such complete evaluations have been infrequent. However, systems *will* be judged by the diversity, depth, and instructional success of their use in realistic situations.

[11] Unfortunately many of the references are not specific about the control structures used, and do not give examples of typical instructional rules, so only a limited comparison could be made.

All of the systems described (except for Training Express) are very large projects that have been designed over years by a number of people, and have been used to represent at least one small instructional domain. So in a sense all authors are speaking from some experience in having their systems used. However, none are being used "out in the field" by instructional designers not closely associated with the original design teams. We can assume, due to the omission of evaluative data in written descriptions of the projects, that there are many unresolved issues, as will as untested features and capabilities of the systems. It is hard to tell from the reports which of the many features described have been used in a non-trivial way. Later (in Section 2.5) we give a detailed discussion of ITS evaluation methods and advocate the inclusion of discussions of limits, unsuccessful features, and design tradeoffs in the descriptions ITSs.

Student modeling and diagnosis. One ITS component that is weak in most of the systems described is student modeling, and especially diagnosis. PTA and Training express have runable student models with some diagnostic capabilities. However, they are useful only in domains were the model tracing paradigm is applicable, i.e. domains where the knowledge is procedural in nature, and can be realistically captured in an expert system. Bite-Sized Tutor, IDE, and Expert CML have overlay student models. The MAIS student model is numerical (which is usually seen as a detractor), and incorporates affective variables. Though many of the systems incorporate some type of student model, little or nothing is said about strategies for constructing or updating these models. This is probably because diagnosis and the construction of student models is one of the least understood ITS research areas—little can be said about it "in general," which limits the incorporation of "generic" student models into generic ITSs for the time being.

2.1.5 Summary of Generic Intelligent Tutoring Systems and Shells

In this section we gave several alternative definition of ITSs and gave the definitions of an ITS emphasizing flexibility and generativity that we use in our work; tracked the history of generic tutoring systems in the ITS literature, discussing key issues; presented an overview

of seven current generic tutoring systems; highlighted seven important characteristics of such systems; and discussed design issues (including domain independence, scope, usability, knowledge representation, student modeling, and evaluation) in light of the seven systems described.

It is a major undertaking to build a general ITS framework or shell, because many of the capabilities of an "ideal" ITS are needed, at least in primitive form, in order to create a running system (for example, a student model and a student interface), and although it is often desirable to focus research on a specific capability or issue, one has to have a "critical mass" of functionality to test it with students or teachers. All of the systems we reviewed are prototypes, in the midst of many-year development and evaluation cycles. At least for restricted types of domains and tutoring styles, these systems do produce reasonable curriculum specifications or computer tutors. But the jury on ITS shells is still out. The systems use a diversity of frameworks and theoretical bases, with little design-level commonality among them, and there is little evidence to allow us to evaluate their relative successfulness or usability. Next we will describe how our research on the KAFITS system relates to the issues and systems discussed in this section on ITS shells.

2.1.6 KAFITS and Generic Tutoring Systems

Here we relate the above discussion of generic tutoring systems to the KAFITS system. KAFITS is designed to be powerful yet easily used and learned, and this is achieved by: 1. an interface which visually portrays the conceptual structure of the framework; 2. a framework and conceptual vocabulary designed with users in mind (and with feedback from users), 3. many powerful features which the beginning user can easily ignore; and 4. on-line help and assistance features. KAFITS adheres to the design guidelines given in Section 2.1.3 for generic ITS shells:

1. it is domain independent, and facilitates the creation of curricula covering a large number of diverse topics;

2. a user participatory design process was employed to develop a system that can be used by instructors with no particular background;[12]

3. its design incorporates some aspects of Instructional Design Theory, and it facilitates the implementation and testing of arbitrary tutoring strategies based on instructional theories;

4. domain knowledge is represented explicitly in a domain knowledge base;

5. strategic knowledge is represented explicitly, and separately from the domain knowledge, in a strategic knowledge base;

6. issues of evaluation and design are addressed directly, and KAFITS use by typical users is studied.

Like the other ITS shells described in this section we have used AI representational methods to design a "generative" tutor. Like IDE we have a sophisticated knowledge acquisition interface. Like the Bite-Sized Tutor we use an object-oriented representational paradigm that allows for a flexible and modular representation of domain knowledge.

KAFITS is unique among the systems described in several ways:

- KAFITS **explicitly represents strategic knowledge** in a form that can be easily modified, monitored and tested (see the description of parameterized action networks in Section 3.1.7);
- KAFITS supports **multiple tutoring strategies** invoked opportunistically in response to the tutorial situation;
- the KAFITS **overlay student model has several unique features**, including multi-layered inferencing, reasoning with uncertainty, and nonmonotonic reasoning, as described in Section 3.3.

[12]They must be familiarized with the conceptual framework and the operation of the system, which takes about one day for users testing and modifying the knowledge base, and about a week for users creating a knowledge base from scratch (see Section 5.3.2).

- KAFITS emphasizes **usability** by domain experts or instructional experts who are not programmers. Other systems are *intended* to be usable, but their designs seem more focus on generality than usability. Other studies do not clearly address the issues encountered when teachers try to use the systems.

- Similarly, there is little evidence of evaluation of these other generic tutoring systems. We are conducting **a formative evaluation in the form of a case study** of three users. Our primary goal is to study the issues that arise in the endeavor of doing ITS knowledge engineering with teachers; it is not our goal to develop the most powerful or general ITS shell from first principles.

This work is also more limited than some of the projects described in two respects: we do not incorporate expert system models of domain expertise or rule-based representations of student knowledge; we do not model expertise in instructional design to guide the design of the knowledge base; KAFITS has not yet been used in multiple domains or with multiple experts; and it does not incorporate a runable student model for procedural skills.

2.2 Instructional Theories and Intelligent Tutoring

Though we do not focus on evaluating or implementing specific instructional theories, consideration of existing learning and teaching theories is essential for several reasons. First, we borrow from several instructional theories in designing the KAFITS framework. Second, in designing a representational framework that is general enough to encode diverse strategies or rules, the ITS designer needs to be aware of the range of forms and content of existing theories. In this section we will review aspects of instructional design theory relevant to ITS design, including those used in the KAFITS system.

2.2.1 Instructional Design Theory

Instructional Design Theory (IDT, sometimes called Instructional Science, or Instructional Design) is "a discipline that is concerned with improving one aspect of education: the process of instruction... [It is] concerned primarily with prescribing optimal methods of instruction to bring about desired changes in student knowledge and skills" [Reigeluth 1983a, pg. 4]. In describing what IDT, is it is useful to relate it to other areas of inquiry. Reigeluth [1983a, pg. 6] writes: "the field of education can be viewed as being comprised of knowledge about curriculum, counseling, administration, evaluation, [and] instruction...Instruction can be viewed as being comprised of five major activities: design, development, implementation, management, and evaluation." So instructional design is a sub-field of instruction, which is in turn a sub-field of education.[13] It is "concerned with understanding, improving, and applying methods of instruction" [pg. 7].

The major difference between instructional science and learning science (which includes instructional psychology and aspects of cognitive psychology) is that the former is primarily a *prescriptive* science, while the later is primarily *descriptive*. The primary purpose of IDT is to prescribe optimal methods of instruction, while the primary purpose of learning theory is to describe human learning. IDT can (and should) *base* its prescriptions on learning theory.

Though learning theory is primarily descriptive, it is often stated in prescriptive terms, such as "to increase long-term retention, ensure that knowledge is organized into stable cognitive structures;" but such principles are not concrete enough to be classified as instructional design principles. Instructional design principles must include specific instructional methods, for example: "to increase long-term retention, begin instruction with an overview...then gradually elaborate on each aspect" (both of the above quotes are from Reigeluth [1983a, pg. 23]).

[13] Reigeluth [1983a] also discusses how the various sub-fields interrelate and overlap.

2.2.2 In Defense of IDT

ITS designers have, for the most part, ignored the instructional design field and (in those ITS projects that have any theoretical basis at all) focused on principles from cognitive psychology.[14] There are several likely reasons for this. One is that AI has strong historical links to cognitive psychology (and cognitive science). The two fields share much terminology and some of the same interests, such as discovering how the mind works. The second reason is that ITS and IDT historically have some fundamentally different perspectives on human learning. IDT has Skinnerian roots, usually employs stimulus response theories, emphasizes the measurement of learning, and often tightly constrains the learner's experience. Teaching systems (both on line and non-computer-based) designed with IDT can seem restrictive and regimented. In contrast, cognitive scientists have championed research on problem solving, learning by doing, and mental models—all of which usually imply more open-ended teaching situations and less emphasis on measuring the student's learning. Still another reason ITS has leaned more toward cognitive psychology than IDT is that IDT tends to focus on (relatively) well understood types of knowledge, such as facts, procedures, and simple skills. ITS researchers often emphasize the learning of more complex (and more interesting) forms of knowledge such as metacognitive skills, problem solving heuristics, and mental models, and there is a belief that these more complex types of knowledge are more important yet are not being taught in schools. ITS researchers also have a tendency to have a somewhat anarchistic view of education, viewing the methods of the current educational system as dull, mechanical, and ineffective—and IDT is often associated with the existing educational system (and also with industrial training).

Regardless of the reasons most ITS researchers have ignored IDT, IDT has much to offer. For one thing, even though complex knowledge is both crucial and taught poorly in schools, students still need to learn the more "basic" forms of knowledge—for their own sake, and as a foundation for more complex knowledge. Also, more is understood about the

[14]This situation is gradually changing, and lately there have been several ITS papers incorporating IDT and several IDT papers incorporating cognitive psychology results [e.g. Tennyson & Rusch 1988, Merrill 1983, Gagne 1985].

basic types of knowledge; for example, hundreds of papers have been written on concept learning, many involving empirical research. In contrast, teaching "mental models" is not as thoroughly studied—in fact there is not even an agreed upon *definition* of it. IDT has more concrete strategies and rules to offer than cognitive science. For example, IDT findings suggest the number and type of examples to give when teaching a concept. Compare this with the idea (found in the cognitive science literature) of "apprenticeship learning" [Collins et al.1986], which, though significant, is much less concrete or refined. Therefore, a tradeoff exists between focusing on types of knowledge for which we have more complete and detailed theories, yet may be less important or less interesting to researchers (such as concepts and procedural skills), vs. focusing on the (perhaps) more important and interesting types of knowledge for which we have a less complete understanding (such as mental models and metacognitive skills).

Actually, the important distinctions are not between IDT and cognitive science, but: 1. whether basic or complex knowledge should be focused on, and 2. whether the learning situation should be structured or more open ended. IDT has, for historical reasons (which become less valid as time goes on), been associated with basic types of knowledge and structured learning situations, and, until recently, has not based its principles on deep causal theories of how the mind/brain work, as cognitive science tries to do. However, IDT is in fact concerned with all forms of knowledge, including complex knowledge, and all methods of instruction, including unstructured ones, and instructional theorists have, for the past five or so years, been incorporating cognitive science into their theories. The the main issue is, as mentioned above, that in order for a principle to be useful as an instructional design principle, it must be fairly concrete and specific, incorporating specific instructional methods. We simply do not know enough yet about more complex forms of knowledge to be able to prescribe many instructional principles for them, at least as successfully as has been done for basic types of knowledge.

2.2.3 What Can Be Gleaned From IDT

Instructional theory prescribes instructional methods for instructional situations. Reigeluth [1988a] calls these prescriptions instructional design *principles*. They are written in terms of instructional design *concepts*. The principles describe relationships (causal or otherwise) between the concepts. According to Reigeluth, there are three kinds of instructional design concepts which make up the language within which the principles are expressed: conditions, methods, and outcomes. Most instructional design principles can be expressed in descriptive or prescriptive ways. The descriptive way looks like this: IF <*condition* X exists> AND <you perform the actions of *method* Y> THEN <the result is *outcomes* Z>. However, the prescriptive formulation of principles are more useful: IF <your goal is *outcomes* Z> AND <*condition* X exists> THEN <perform the actions of *method* Y>.[15]

One of the design goals of the KAFITS system is to provide a semantics (vocabulary) and syntax (structure) for representing instructional strategies. These strategies can come from educators, domain experts, or from the literature. In exploring the literature we are looking for representational components that are: 1. general enough to be used for many strategies, 2. specific and concrete enough to be implemented on a computer, and yet 3. simple or intuitive enough to be understood and used by educators without a great deal of training. Unfortunately, since the instructional design field is (like all "soft" sciences) quite diverse and without a common underlying framework, there is no clear indication of any generally accepted representational components in the literature. However we have found three areas where generalizations can be made that are useful for ITS designers. The first is in classifying instructional strategies as macro-strategies vs. micro-strategies. The second common element is the use of knowledge-type classification schemes. The third is a description of several general areas of instruction that must be included in any complete instructional theory. We will discuss each of these below.

[15]This reformulation works only if the outcomes are desirable, otherwise one would not want to use them as goals.

Micro vs. Macro Levels

Instructional design theories or strategies usually describe components of instruction at either of two levels: the micro level or the macro level. The micro level deals with how to teach single ideas (concepts, facts, principles, etc.). Merrill's [1983] Component Display theory and Gagne's [1985] theory of instruction are micro-level theories. The macro level deals with sequencing, summarizing, and synthesizing a number of ideas, and involves curriculum design. Examples are Reigeluth's Elaboration Theory [1983b], and Ausubel's Advanced Organizer theory [1963].

Knowledge Classification Schemes

A common thread for many writers in both instructional theory and learning theory is the classification of different types of knowledge. Bloom's Taxonomy of Educational Objectives [1956], listing over 30 categories of educational objectives (things one might want to teach) is one of the earliest and best known of these.[16] This early attempt was a synthesis of the thinking of a panel of educational theorists. Unfortunately, it did not clearly distinguish (subjective) knowledge from (observable) behavioral objectives. Gagne [1985] gives a more modern classification in terms of the changes in behavior that are used to infer that learning has occurred. His theory is based on five types of learned capabilities: intellectual skills, cognitive skills, verbal information, motor skills, and attitudes (and his theory has many sub-classifications of these five). Gagne is one of the field's main proponents of the idea that different types of learning require different types of instruction. Merrill's Component Display Theory [1983] also relies heavily on a knowledge classification scheme, as does Reigeluth's Elaboration Theory of Instruction [1983b]. There is an extensive literature dealing just with concepts, distinguishing many types of concepts and concept learning [Tennyson & Park 1980, Hunt 1962].

[16]The knowledge is arranged in six main categories: knowledge, comprehension, application, analysis, synthesis, and evaluation.

Cognitive scientists tend to use classifications that are less elaborate, yet derived from tested theories of how the mind works. Anderson's ACT* theory [1983] distinguishes declarative from procedural knowledge. Many AI-based theories of learning and instruction, and many AI-base tutoring systems also use a procedural-declarative distinction, though they do not all define these categories in the same way. Half [1988] classifies existing AI tutors as being either expository (teaching declarative knowledge, and using primarily dialogue as the instructional method), or procedural (teaching skills, and using primarily a coaching-like environment as the instructional method.).

Basic Areas of Instruction

Here we list several areas of instruction that cover most of the range of instructional methods found in the literature. In the literature one finds different instructional and psychological justifications, and different approaches to implementing, each of these "areas." We will not discuss these justifications and approaches.

1. **Pre-instruction.** Before introducing an idea do one or more of the following: relate it to other things that have been learned, motivate the student to want to learn the thing, or give an overview of what is to be learned.

2. **Post-instruction.** After something has been taught, do one or more of the following: summarize it, or relate or synthesize it with other ideas that have been presented.

3. **Exposition.** Giving examples, explanations, definitions, etc. Issues include how, when, and what kind of exposition to give.

4. **Practice.** Allow the student to use the new information. Issues include: How much should the student practice, and with what tasks? Should she get more examples, or be asked more questions? What style of practice should be sued: rote, learn by doing?

5. **Feedback.** Give appropriate feedback after student actions. Issues include: When and how to respond to the student's behavior. Should students be told whether they were correct, given hints, or told the correct answer?

6. **Remediation.** Give additional material if the student has a misconception, does not learn with the standard instructional method, or shows improper understanding of something previously taught and presumed to have been learned.

7. **Assessment.** Determine when a student has learned something, or when any of the tutor's objectives have been met.

These areas of instructuion are not all needed for every instrucitonal sitiation, but an ITS shell be able to represent and use them all. In sum, we suggest that generic ITS frameworks have mechanisms for: 1. distinguishing macro and micro levels of instruction, 2. distinguishing knowledge types, and 3. incorporating all of the *basic areas of instruction* listed above, and 4. allowing for substantial student control (see Section 6.3.4).

2.2.4 Component Display Theory

Component Display Theory (CDT) [Merrill 1983] is an instructional theory dealing with the micro level of instruction. We describe it because we borrow from CDT in the design on the KAFITS conceptual vocabulary, and because it provides a good example of an instructional design theory that is conceptually clear, powerful, and operationally concrete. Unlike most other instructional design theories, the concepts and prescriptions in CDT are described at a detailed operational level, making parts of it amenable to computer implementation.

CDT defines a *descriptive* language for the concepts, conditions, methods, and outcomes of instruction. The language consists of three taxonomies: one for describing student performance/behavior (learning or behavioral objectives), one for describing subject matter (a knowledge type classification), and one for describing instructional behavior or actions. We have borrowed primarily from the vocabulary of subject matter, called the performance-content matrix (PC-matrix). CDT also describes a *prescriptive* model for teaching. This model includes a mapping from behavioral objectives to expository and inquisitory presentations (with the PC-matrix serving as an intermediary in this mapping). In the original

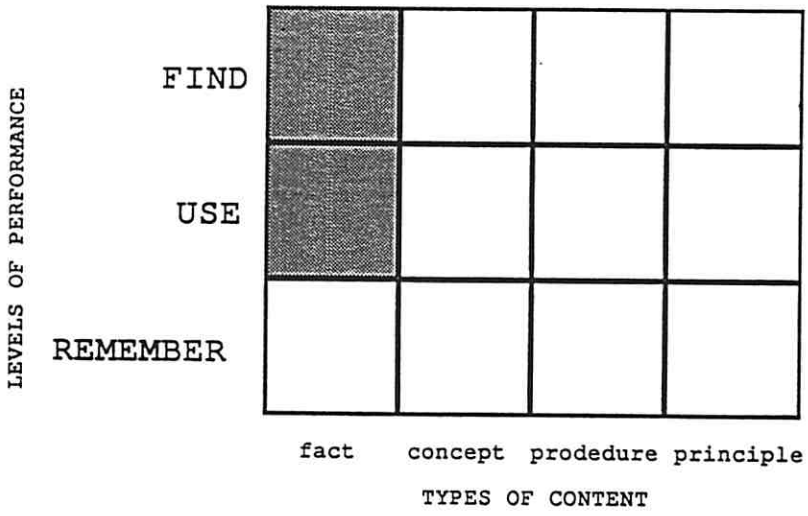


Figure 2.1 Merrill's Performance-Content Matrix

design for this study we had planned to incorporate aspects of this mapping, but did not, for reasons described in Section 2.4.

The Performance-Content Matrix

CDT assumes Gagne's [1985] hypothesis that different types of learning require different types of instruction (and different ways to evaluate the learning).[17] The Performance-Content (PC) matrix (see Figure 2.1) is a key contribution of CDT. Previous theories have given classifications of types of knowledge or instructional objectives which are hierarchical in nature [eg. Bloom 1956]. There are many attributes of instructional content which are relevant to the instructional strategy used, and the space of types of knowledge is multidimensional. A hierarchical classification fails to capture some important structure of knowledge, making knowledge classifying ambiguous. Merrill's two dimensional classification is more expressive, usable, and intuitively clear than hierarchical schemes.

Since one of the goals of CDT is to have a prescriptive system that is understandable by teachers and instructional designers (as distinct from researchers and theorists), the matrix

[17]Gagne and Merrill do not seem to make a clear distinction between types of learning and types of knowledge, and I think they mean the same thing in this context.

is relatively small (three by four), even though finer distinctions are possible. For the same reason (ease of use or felicity) Merrill does not attempt to devise a many-dimensional model of knowledge types, which might be more theoretically appealing or complete. The insight that led to the PC-matrix was seeing the need to classify knowledge (or learning) according to both performance level and content type. CDT includes three performance levels: remember, use, and find (create); and four content types: facts, concepts, procedures, and principles. The matrix cells for "use fact" and "find fact" are blanked out in the matrix, since, by (Merrill's) definition, facts can only be remembered and recalled (i.e. to use a fact is to remember it). Therefore there are ten fundamental knowledge types in CDT.[18]

Mapping from Objective to Knowledge Type to Instruction

CDT describes a method for mapping behavioral objectives to knowledge types, and prescribes a method for mapping knowledge types to instructional methods. To map from behavioral objectives to [content types] CDT uses a template representation of objectives similar to: GIVEN <stimulus representation> OF <stimulus constraint> EXPECT <behavioral constraint> BY <behavioral appearance> WITH <criterion>. For example: (1) GIVEN pictures OF new-examples EXPECT [the student to be able to] classify BY sorting; and (2) GIVEN a-description OF an-event EXPECT [the-student-to] discover-relationship BY experimentation. To determine the [content type] a lookup table is provided that maps from descriptions like the one above to one of the [content types] shown in the PC-matrix.

Merrill's mapping from knowledge types to instructional presentations uses interrogatory and inquisitory "presentation forms" such as: asking for definitions, asking for classifications of items, giving generalities (definitions), giving instances (examples), giving hints, and giving contextual information. CDT provides a mapping from knowledge types to sequences

[18]Merrill makes finer distinctions to the performance levels, such as distinguishing between recalling verbatim vs. paraphrased, in some situations, and Reigeluth [1983b] suggests some refinements to the content types.

of presentation forms.[19] Merrill's theory states that some of the presentation forms are necessary and sufficient to learning the subject matter (called "primary" presentation forms) and others are not necessary but are used to enhance the efficiency or effectiveness of the learning (called "secondary" presentation forms). The reader is referred to Merrill [1983] for a detailed description of the rules for determining the presentation forms.

2.2.5 The Merging Paths of ITS and IDT

As we alluded to earlier, ITS researchers have ignored IDT because of a mismatch in the goals, methods, and historical roots of the two fields. However, these fields are becoming decreasingly dissimilar. Increasingly, instructional theories are addressing higher order skills and basing their theories on cognitive principles. As ITSs move into more varied and realistic instructional settings some issues that IDT has been studying for decades become more salient. Even though most instructional theories or models are too general or vague to be immediately incorporated in computer systems, IDT has a wealth of empirical and theoretical information ripe for use by the ITS community.

Our overview of IDT has been quite narrow. To give the reader a flavor for the range of relevant issues covered by IDT, and how IDT could guide some aspect of ITS design in the future, we list several ITS issues, and note one IDT theory that is relevant for each.[20]

- **Representation of pedagogical information, motivation.** Ausubel's Advance Organizer model [Ausubel 1963] structures learning around a hierarchical organization of the concepts. To insure that instruction is both meaningful and relevant, he suggests first presenting an overview (at an appropriate level of abstraction) of the subject matter, relating it to other concepts near it in the hierarchical structure, and then using successive differentiation, presenting more general or inclusive ideas first.

[19] One drawback of CDT is that the prescribed presentation forms are given in a fixed linear ordering.

[20] We do not necessarlity agree with the the content of any of these theories, we only wish to demonstrate the range of relevant issues covered in IDT.

- **Student modeling, longitudinal effects.** Though not an instructional theory, Piaget's developmental model [Varma & Williams 1976], which proposes that a student's position within a sequence of stages of cognitive growth determines the types of knowledge that can be learned, can be used to design instruction according to a model of the student.

- **Tutoring strategies, primitive tutorial actions.** Skinner's Operant Conditioning theory [1957] focuses on the use of (and types of) reinforcement given to the student.

- **Tutorial session management, assessment.** Bloom's Mastery Learning model [Bloom 1984] proposes frequent assessment of student performance and provides instructional structures that try to insure that all (or most) learners demonstrate a predetermined mastery performance level before moving forward in the curriculum.

- **Learner control, types of knowledge.** Flavell [1981] and Schoenfeld [1985] emphasize metacognitive skills—mental processes that monitor and manage one's current cognitive processes.

- **Affective factors, group learning.** Thelen's Group Investigation Model of teaching [Joyce & Weil 1986, Chapter 2] focuses on how to integrate independent and group investigation within a democratic process that respects the input of all of the students.

2.2.6 Summary of Instructional Theories and Intelligent Tutoring

In this section we defined instructional design (or instructional theory) and explained why we focused more on instructional design than learning theory. The need for instructional strategies that mention specific instructional methods was emphasized. We argued that ITS designers should incorporate more instructional design theory into their systems, and hypothesized reasons why they have not in the past. From the instructional design literature, three recommendations for generic ITS's were given: distinguishing micro from macro instructional levels, incorporating knowledge classification schemes, and incorporating seven basic areas of instruction (pre-instruction, post-instruction, exposition, practice,

feedback, remediation, and assessment). We then described Merrill's Component Display Theory, because we have borrowed from it in designing our conceptual framework, and because it illustrates an expressive, usable, and implementable instructional design theory. Finally, we alluded to the breadth of issues covered by IDT that are relevant to ITS design by giving examples of IDT theories and pointing to ITS issues they address.

2.2.7 KAFITS and Instructional Theory

The KAFITS framework incorporates the following principles and features inspired by instructional design theory:

- KAFITS's primitive tutorial actions and default tutoring strategies incorporate the seven "basic areas of instruction" identified in Section 2.2.3.
- KAFITS's overall control structure, the Four-level Decision Model, is based on micro and macro levels of instruction (see Section 3.1.3).
- KAFITS incorporates a knowledge-type classification scheme based on Merrill's PC-Matrix (described below), which helps the expert articulate and distinguish instructional objectives, and facilitates the creation of tutoring strategies that distinguish between different types of knowledge.
- the KAFITS topic Part relationship allows subsumption networks of concepts as in Ausubel's theory of menaingful learning. [Ausubel 1963].
- the KAFITS performance/mastery topic levels allows stragegies to duplicate mastery learning, as discussed in [Bloom 1984].

A modified PC-Matrix. Gagne's [1985] hypothesis that different categories of knowledge (learned capability) require different methods for promoting the learning of that capability,[21] though not proved in a strict sense, it is generally accepted in instructional

[21] Gagne actually refers to "outcomes" (i.e. behaviors) rather than "knowledge."

theory. In ITSs, the Gagne hypothesis calls for multiple tutoring strategies that are sensitive to knowledge categories. We have chosen to base our knowledge type classification on Merrill's Performance-Content (PC) Matrix [1983] and its conceptual simplicity compared with other classification schemes.

Our representational framework (described in Section 3.1.6) includes several modifications to Merrill's PC matrix. First, we make the distinction between Basic knowledge types (the ones in the PC Matrix) and Complex types of knowledge types (including metacognitive skills, scientific inquiry skills, mental models, and creativity). We found this necessary in order to include many higher level abilities that are not in Merrill's PC Matrix.[22] Also, we have added a meta-knowledge[23] level to Merrill's matrix and divided the use level into apply-use and apply-problem-solve (see figure 2.2). Apply-use refers to the ability to employ knowledge in a context where it is clear that the knowledge is needed. Apply-problem-solve refers to using a piece of knowledge with the additional ability to recognize the need to use it in an open problem solving context.

KAFITS incorporates content types as node types in the topic network and incorporates performance levels as levels within each topic. For example the topic "gravity" is of type *concept* and the designer can specify presentations for the remember, apply-use, and problem-solve levels within this topic.

The KAFITS system could be used to represent and experiment with multiple alternative instructional theories. We originally intended to encode several instruction theories, including Merrill's, using the KAFITS tools, but this was not done because we realized that it was enough for the domain expert to encode his own ideas about how to teach his domain without adding the additional, and obfuscating, task of learning and/or being constrained to specific instructional models.

---

[22]Merrill's Component Display Theory prescribes very concrete and specific instructional actions based on the characteristics of the target behavior. Complex knowledge types are not included in Merrill's theory because they are quite difficult to define or prescribe how to teach at the necessary level of precision. [true acc. to new stuff?]

[23]Meta-knowledge is knowledge *about* knowledge, such as when to use it, why it is usefull, whether it is hard to learn, etc.

Complex Knowledge / Skills

General prob. solv. skills
Metacognitive skills
Scientific inquiry skills
Formal logical skills
Mental models
Creativity
etc.

Basic Knowledge / Skills

MODIFIED PERFORMACE-CONTENT MATRIX

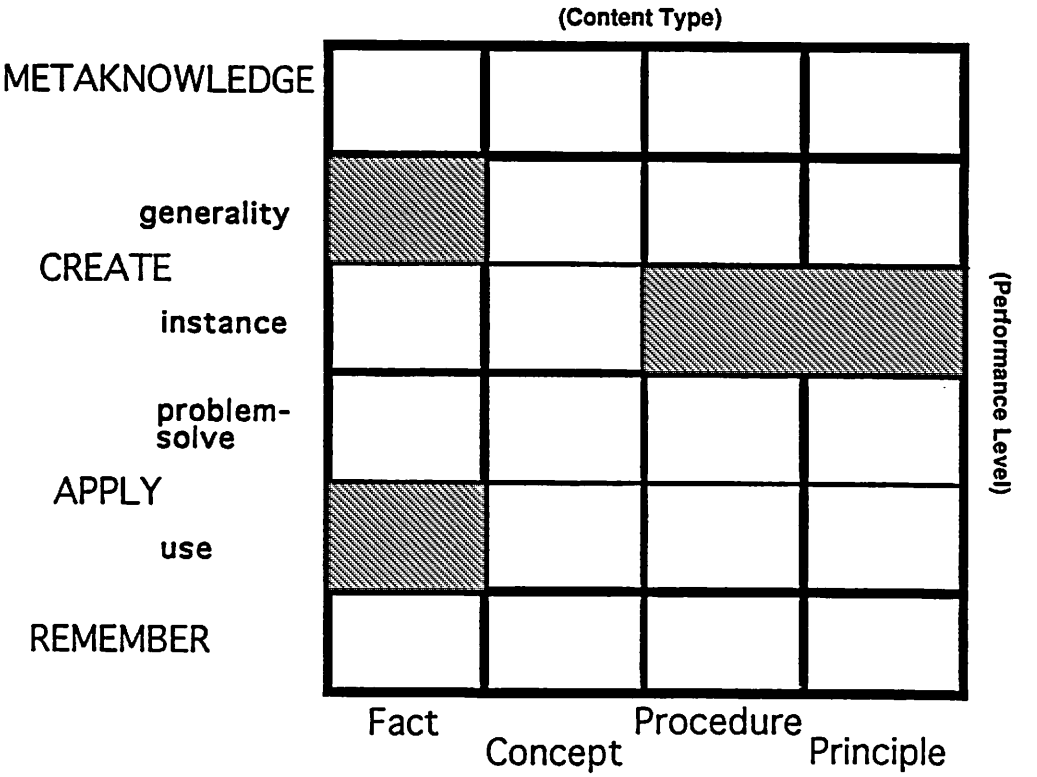


Figure 2.2 A Modified Performance-Content Matrix

2.3 Empirical Research and Iterative Design

In ITS research, and in most sub-fields of AI, there is an intimate melding of the theoretical and the practical—those of us doing research and building systems are both scientists and engineers. We must balance the desire to build systems that works with the desire to discover new (or challenge existing) theories and models. In this section we discuss how research methodology and systems design methodology can be combined in the context of ITS research. (In later sections we the related topics of discuss knowledge acquisition methods (Section 2.4) and evaluation methods (Section 2.5).)

2.3.1 AI Research

Research methods and systems design methods used in main stream AI are applicable to the ITS sub-field. Buchanan [1987] describes AI research as a cyclic process with theoretical, engineering, and analytical phases. He suggests that research includes all of these steps at least once:

- **Theoretical Steps:**

1. Identify the problem.
2. Design a method for solving it.

- **Engineering steps:**

3. Implement the method in a computer program.
4. Demonstrate the power of the program (and thus the method).

- **Analytical steps:**

5. Analyze data collected in the demonstration.
6. Generalize the results of the analysis.

These research steps are typically repeated in a cycle from theorizing to engineering to analysis and back to theorizing. Though the steps may seem too obvious or general to use as a basis on which to design a study, Buchanan enumerates them to point out that much of the research reported in the field is lacking in one or more key areas, including: clearly identifying the goals of a study; stating the theoretical underpinnings; describing the scientific and/or engineering methodology; building a system which implements the theory; and evaluating/generalizing data. [24]

Cohen & Howe [1988a] distinguish three types of AI research: theoretical AI, empirical AI, and applied AI. Their discussion focuses on empirical AI, which they describe in comparison to theoretical and applied AI. Theoretical AI is purely analytical—its hypotheses can be proved without implementation. Empirical AI, on the other hand "tells us things about the behavior of AI systems—the interactions of knowledge representations, inference methods, algorithms...that we could not anticipate from purely theoretical AI" (page 3). Applied AI deals with the practical issues of implementing existing theories, models, or systems in realistic situations. Empirical AI, on the other hand, aims at developing and experimenting with new methods.

Empirical AI science has important distinctions from other sciences. Cohen & Howe [1988a] point out that empirical AI research focuses on investigating human artifacts, in contrast to behavioral (or physical) science research which is about investigating naturally occurring phenomena. Behavioral science research is about "teasing apart the components of behavior and their causal relationships," asking "why does [some natural system] behave this way," and typically uses methods such as searching for factors and statistical hypothesis testing. In contrast, empirical AI is about "putting all the components together in one box to *produce* behavior," asking "what knowledge representation and algorithms do we need to make a system that performs [in some desired way]," and the most commonly used method is a cyclic design process [Cohen & Howe 1988a, pg. 18].

---

[24] For example, Buchanan [1988, pg. 16] says that "with AI programs, designers often advance many interdependent claims at once [and] do not state explicitly what those claims are ... this is reprehensible scientific behavior."

ITS Design Methods vs. Research Methods

ITS research involves empirical AI,[25] and Buchanan addresses empirical AI (in Cohen & Howe's classification) in his six research steps. Empirical AI has elements of theory and application. Buchanan's steps emphasize the need for ITS designers/researcher's to clearly state research goals, evaluate their systems, and generalize findings. There are also many applied, pragmatic aspects to ITS research, as ITS researchers are increasingly aware of the importance of human factors and interface issues [Bonar 1991, Frye et al. 1980], and, by its very nature, ITS must be tested in the field before researchers can be confident of their theories. Pragmatic concerns for moving ITSs out of the lab and into the classroom, home, or workplace are discussed in Section 6.5 and in [Baker 1991, Johnson 1988, and Woolf 1990].

In this study we combine an ITS design methodology with an ITS evaluation methodology. We re-interpret Buchanan's research steps as follows for ITS design/research:

1. Identify research questions, posit hypotheses, describe the underlying pedagogical/psychological theory;
2. Choose an instructional domain, a domain expert, a system architecture, a software design method, and an evaluation method that address the questions and hypotheses of step 1;
3. Build an intelligent tutor (this step includes software design, knowledge acquisition and human factors considerations);
4. Test the tutor (in dry runs and/or student trials);
5. Analyze the test data;
6. Generalize the results, make conclusions related to the questions and hypotheses, and (usually) start over again with step 1 to refine the program and/or the theory.

[25] It could be argued that some ITS research, including our present study, is empirical *research* but not empirical *AI* research, since it does not contribute significantly to the general body of AI knowledge and theory.

In this study we:

- discuss ITS *evaluation methods* (part of steps 2 and 5 above) in general (Section 2.5.3), and describe our evaluation which incorporates formative evaluation and case study methodologies;
- discuss ITS *usability* design issues (Section 2.3.2), and give our own results on usability (Section 5.5.4);
- discuss *knowledge acquisition* (part of step 3) issues (Section 2.4); and give results from our study of ITS knowledge acquisition tools and methods (Section 5.4);
- expand on steps 3 and 4 in our description of a user participatory *ITS design process* (below).

2.3.2 Iterative and Participatory Design

Both Buchanan and Cohen & Howe emphasize the cyclic nature of AI research.[26] Iterative design is also strongly recommended *throughout* the literature in computer usability and human-computer interaction (HCI).[27] In studies of software intended to be used by people there is a high degree of uncertainty as to the outcomes. Baker [1991, pg. 152], in discussing computer based training, notes that "what technology is almost guaranteed to do is to generate, by its very existence, outcomes and applications that were not previously considered...nor imagined by the...designer." Similarly Gould [1988, pg. 7] offers these general observations on designing systems based on studies of users:

- Nobody can get it right the first time,
- Development is full of surprises,

[26] Whiteside et al.[1987, pg. 12] refers to cyclic design as "incremental, evolutionary, and conscious iteration."

[27] HCI studies are often from the perspective of software development and application, rather than the perspective of pure research, but the principles given in the HCI literature hold for any user-focused research even if the computer system is not intended for the marketplace or workplace.

- Developing user-oriented systems requires living in a sea of changes.

Therefore continuous testing and refinement, i.e. *iterative design*, is needed to ensure usability. In the context of research (as opposed to software design) this suggests a *formative evaluation* methodology (which we discuss in more detail in Section 2.5.2).[28] However, the use of iterative design along with formative evaluation still does not address some important issues—*users* must be in the design/evaluation loop. Gould [1988, pg. 4], in addition to recommending iterative design, gives these principles for system design:

- Early and continual focus on users—i.e. direct contact with users to understand cognitive, behavioral, and social characteristics of their task and task environment.
- Integrated design. Design, build, and test all components of the system in parallel.
- Early and continual user testing. Users, under observation or some other form of measurement, do real work with prototype systems.

Participatory Design

A recent trend in HCI and usability research, called "user participatory design" (or "participative design" [Blomberg & Henderson 1990]), involves a shift from user-as-subject to user-as-co-researcher. Ascertaining how users conceptualize their domains and how they conceptualize the computer system is an essential part of studying usability. Researchers must account for the conceptual models the user brings to the task, and determine how these models might help or hinder the model of the system that the designer wants to portray. For example, people have experience with desktops, file cabinets, and street maps, and these conceptions may help or interfere with their understanding of a computer tool.

Whiteside et al.[1987] discusse the importance of "contextual research" (studying users during real work) in usability studies, saying: "human action as observed derives its mean-

[28] Iterative design becomes a formative evaluation if one keeps data on the issues encountered, the changes made to the system, and the results of these changes. Systems building becomes research when the goal is to explore the important issues and report to others, rather than to build a system that works.

ing from the context in which it occurs" (pg. 19). Designing for usability means not only satisfying external functionality and performance criterion, but ensuring that the user is motivated and informed. Therefore, determining the user's *experience* as well as her observable behavior is important.

In HCI research (and in knowledge acquisition research as well) the researcher is often inextricably entangled with what is observed. He influences the data by the choice of questions asked, by instructional remarks and other informative interventions, and simply by being present as the user uses the system (Whiteside et. al 1987). Therefore, as mentioned previously, traditional experimental methods are often inappropriate; and when the goal is to uncover the users experience of a system, which is un-observable, traditional experimental methods have no bearing.

The above mentioned goals of (1) determining users' conceptual models, and (2) determining users' experience of the system, are facilitated by having the user be a co-researcher (i.e. participatory design). Just as the researcher is inextricably involved with the subject, the subject shares responsibility in discovering the experience of working with the system. The researcher "explains what is of interest; there is no thought of concealing the conditions for fear of contaminating the data...[and the user, as co-researcher, helps] direct the discussion, indicates relevant areas for exploration, and responds to the [researcher's] interpretations" [Whiteside et. al, pg. 26].

We have discussed AI research and how research methodology and system design methodology can be combined. This study is an investigation of 1. an ITS knowledge acquisition process and 2. ITS knowledge acquisition tools. Below we discuss usability design principles we needed to be aware of in designing the knowledge acquisition tools (we discuss knowledge acquisition methodology in Section 2.4).

2.3.3 Usability and Interface Design

ITS knowledge acquisition tools should adhere to established *human factors design principles*, especially if they are designed to be used by instructors. Nickerson & Pew [1990, pg. 42] give several:

1. Make it hard for the user to inadvertently cause a disaster.
2. Make the user feel that s/he are doing the task rather than instructing the computer to do it (for example, by using direct manipulation of icons).
3. Modularize applications in terms the user's tasks, rather than programming convenience.
4. Use simple metaphors to explain task organization (for example, the "desktop").
5. Limit interactive access to when it is needed.
6. Help the user "navigate" within the information available.

Nelson [1990] gives a further discussion of *navigation issues* in the context of using Hypertext. His studies have indicated that a majority of users experience confusion and disorientation in trying to use a large Hypertext document. This problem applies to any large knowledge base where there are many links between the objects. Designers of interfaces should make moving between linked items intuitive, and provide methods for the user to know his context ("you are here") within the knowledge base.

Miller [1988] notes that in designing an ITS interface one must consider users' *cognitive capabilities and limitations*, and the knowledge and cognitive structures they are likely to bring to the task.[29] "People combine [existing] knowledge with their observations of the structure and behavior of the interface to construct a conceptual model of the system" (pg. 145). He gives three characteristics of a good conceptual model: clarity, coverage,

[29]Miller's article discusses ITS interface issues in the context of the *student*, but the principles he gives are general human factors principles, equally applicable to an interface for knowledge acquisition.

and sound level of abstraction in suggesting that interfaces be designed to offer a good conceptual model to the user. Other discussions of ITS interface design can be found in [Bonar 1991, Frye et al. 1988].

2.3.4 Summary of Empirical Research and Iterative Design

In this section we pointed out that ITS research (like most AI research) blends aspects of both research (discovering knowledge) and engineering (building artifacts) and must combine scientific methodology with design methodology. We discussed "empirical AI" (which covers almost all ITS studies) as compared with "theoretical" and "applied" AI [Cohen & Howe 1988] and as compared with research in other fields, and we re-interpreted Buchanan's [1987] six steps for AI research in terms of ITS design/research. We discussed how an *iterative design* method fits the needs of both human factors (usability) concerns and the cyclic evolutionary nature of AI research. We then discussed how a *user participatory* design process also supports usability goals. Finally we presented interface and human factors design principles which should guide the design of ITS interfaces (student interfaces as well as knowledge acquisition interfaces).

2.3.5 KAFITS Research and Design Issues

Since in this study we are both evaluating the artifacts we build and improving their performance, an *iterative design process*, incorporating evaluation at many stages, has been used. Since usability of our system is important, we have followed a *user participatory design process* as well (in Section 2.3.1 we summarize results of our study that argue the need for user participation).

Each of Nickerson & Pew's six human factors design principles (see above), as well as the issues raised by Miller about the need for ITS interfaces to support cognitive models, are addressed in the KAFITS system and/or discussed in our analysis of the study (in Section 2.3.2). Our "empirical AI research" combines several evaluation methodologies, including

qualitative evaluation methods, formative evaluation, and a case study, as discussed in Section 2.5.5.

2.4 Knowledge Acquisition and Intelligent Tutoring

Here we discuss the most important and difficult aspect of building ITS knowledge bases—acquiring the domain expert's knowledge. A wide range of expertise is needed to build an intelligent tutor. ITS designers, in the tradition of AI, have long recognized the need for domain experts and cognitive scientists, and recently there is an increased recognition of the importance of expertise in instructional science and human factors. In addition to these forms of expertise, we advocate for the participation of practicing educators to insure that intelligent tutors are realistic and anticipate pragmatic issues. Two kinds of knowledge must be acquired, domain knowledge and pedagogical knowledge,[30] and there are two issues in getting this knowledge into a computer tutor: knowledge acquisition and knowledge representation. We are interested in methods for *eliciting* (acquiring) knowledge from experts and methods for *representing* what experts communicate. Acquisition and representation are closely related, since the framework used to represent knowledge will constrain and/or support the elicitation process.

ITS researchers have addressed knowledge representation, but, for the most part, have not dealt explicitly with knowledge acquisition issues (except for task analysis studies).[31] However, knowledge acquisition has been studied in AI, and we can relate this research to ITSs. Therefore, in this section we present a fairly complete (though in some cases shallow) review of AI knowledge acquisition methods and tradeoffs, both for the edification of the interested reader, and to describe and justify the methods we used for this study.

---

[30] "Pedagogical knowledge" includes tutoring strategies and domain-specific information about how to teach the curriculum.

[31] Much has been written about the process of acquiring knowledge for CAI (a process called authoring or courseware development). These inquiries have some useful advice to offer the ITS designer in areas such as organizing working sessions with educators, choice of media, and screen and graphics design. However, for the most part, design guidelines given for courseware development have little to offer ITS designers since they produce fixed instructional scripts with no explicit models of teaching, domain knowledge, or the student.

2.4.1 Knowledge Acquisition Issues

Knowledge acquisition is the transfer and transformation of expertise from people into a form that can be used by machines (Buchanan et. al 1983).[32] It is one of the first steps in the design of expert systems (including ITSs) and is the biggest bottleneck in the knowledge engineering process. (Hoffman [1987, pg. 53] states that "the identification and encoding of knowledge is one of the most complex and arduous tasks encountered in the construction of [an expert system].") Though expert systems have existed for some time, there is as yet no consensus on the best way to go about the knowledge acquisition process.[33] However, a corpus of knowledge acquisition methods, all of them appropriate to building ITSs, have been identified and partially standardized, and advantages and disadvantages of some of these methods have been documented. Before describing specific knowledge acquisition methods we summarize several of the most important issues facing AI and ITS knowledge engineers: mappings from domain tasks (and skills) to knowledge acquisition methods for eliciting descriptions of those tasks; automation of the knowledge acquisition process; important characteristics of domain experts, knowledge engineers, and domains; and secondary goals of knowledge engineers.

Mapping tasks to methods. Gaines & Bose [1988, pg. xviii] refer to the "piecemeal nature of techniques and tools" in calling for a more systematic approach to determining the best knowledge acquisition methods for each application. The issues receiving the most attention in mainstream knowledge acquisition research are: 1. the delineation and analysis of methods, and 2. the analysis of domain (or task) characteristics for determining the best

[32]Related terms: "knowledge engineering" is the process or methodology for acquiring, representing, and using qualitative models of systems (i.e. building and expert system) [Gaines & Bose 1988]. "Knowledge elicitation," "the process by which facts, rules, patterns, heuristics, and operations used by humans to solve problems, in a particular domain, are elicited" [Garg-Janardan & Salvendy 1988] (also called "knowledge extraction"), is a part of knowledge engineering. For the purpose of this paper knowledge acquisition, knowledge engineering, and knowledge elicitation are synonymous, unless otherwise stated. Note also that expert systems are often called "knowledge-based systems."

[33]Hoffman [1987, pg. 54]: "No systematic research has been conducted on the question of how to elicit an expert's knowledge and inference strategies;" Gruber [1988 pg. 3]: "there is little consensus on a methodology for knowledge acquisition."

methods to use. AI researchers are slowly converging on a mapping from domain or task characteristics to knowledge acquisition methods [Bose 1988].[34]

Automating knowledge acquisition. Though most knowledge acquisition is currently done by hand,[35] there is a movement away from protocol analysis and interviews toward computer-based interactive methods using elicitation tools. Many of the knowledge acquisition methods we list later can be automated. Also, computer-based methods are used to check the consistency and completeness of knowledge bases. In addition, *machine learning* can be considered a form of knowledge acquisition—analogy-based learning, explanation-based learning, concept learning, and other automatic inductive and deductive learning methods are sometimes included in a broad definition of knowledge acquisition.[36] Gilmore & Self [1988] give an overview of applications of machine learning to ITS. Studies have been conducted using machine learning to infer student models and to incrementally improve teaching strategies in ITS systems [Dillenbourg 1988, Kimball 1982].

Characteristics of domain experts and knowledge engineers. There is wide agreement about desirable characteristics in selecting domain experts (DEs, or "subject matter experts") (see Prereau [1987], McCaslin & Boord [1990], McGraw [1989], Slagle & Wick [1988]). These characteristics include:

- knowledgeability, self-assuredness, and credibility (the DE's expertise should have been acquired by successful task performance over a long period of time);
- communication and cooperation skills (the DE must be a "team player" and be capable of communicating personal knowledge, judgment, and experience);
- commitment, availability, and managerial support (the domain expert and his/her organizational superiors must support a substantial time commitment).

[34] Some of their concerns are related to our discussions about knowledge types (Section 6.3.3).

[35] Hoffman [1987, pg. 62] says that "in general, it takes two years to develop a prototype and about five years to develop a full-scale system."

[36] See Gruber [1988] for a discussion of machine learning techniques applied to knowledge acquisition.

Less is said in the literature about important characteristics of knowledge engineers, but McGraw [1989] mentions that the ITS knowledge engineer wears many hats, including that of knowledge analyst, communications facilitator, and instructional designer. [37]

Secondary goals of knowledge engineers. Knowledge engineers usually have secondary goals, other than eliciting knowledge, including:

- learning about the domain from the expert,
- establishing rapport with the expert,
- ascertaining the needs and preferences of users,
- and instructing the domain expert in the concepts of AI.

These goals must be considered when planning the knowledge acquisition process and choosing the participants.

Other issues. There are important knowledge acquisition issues discussed in the literature which we have not mentioned, but which we will provide references for: acquiring knowledge from multiple experts [Shaw & Gaines 1986, and LeClair 1988], meeting the needs of both novice and experienced users of knowledge acquisition tools [LeFrance 1989, Kopec & Latour 1989], and how to maintain an evolving knowledge base over time [Barker & O'Conner 1989, and K. Murray 1988].

ITS vs. Other Expert System Domains

The vast majority of knowledge acquisition research is done in the context of traditional expert systems, and intelligent tutoring systems differ from traditional expert systems in several significant ways. In some respects intelligent tutoring is such a difficult area that new and challenging issues are encountered that do not exist for prototypical expert system

[37] In addition, Slagle & Wick [1988] discuss how to select a good applications domain for expert systems, and in Section 6.3.2 we discuss how tutoring systems measure up in their scheme).

applications. In other respects the ITS state of the art is far behind the state of the art for prototypical expert system domains, so that some important knowledge acquisition issues have barely been addressed in ITS research.

As mentioned in a previous chapter, ITS is more than just an applications area for expert systems. In fact, if we use Slagle & Wick's [1988] "method for evaluating candidate expert system applications," ITS would seem a rather poor expert system application. Slagle & Wick list 24 "essential features" and 16 "desirable features" for expert system domains. ITS rates high on some features, including "task is knowledge intensive," "task not essential to deadline," "hard to transfer expertise," and "task identified as a problem area," but ITS fares poorly with many more features, such as "task is not language intensive," "task requires no common sense," "expert is articulate," "experts agree on good solutions," "solutions are explainable," and "task does not require real-time response." However, as mentioned previously, ITS research is justified for many reasons even if it is a non-optimal applications area for AI technology.

One of the main differences between traditional expert system applications and intelligent tutoring is that ITS research does not (except in rare cases) try to simulate human tutoring expertise. It uses human tutoring expertise to guide the design of tutoring rules, but the end goal is not to simulate human behavior. There are two reasons for this. First, we can only hope to approximate the simplest aspects of human tutoring, being constrained by the limitations of the computer. The types of information that are essential to a human teacher are much more subtle, unconscious, and immeasurable (facial expressions, students' personalities, etc.) than the information used by ITSs. And, perhaps more importantly, even if we *could* simulate human expertise, it may not be *appropriate* to do so. Learning via a computer and learning via human discourse are very different, and the most powerful and efficient strategies for teaching in these environments are likely to be equally different.

Another difference between tutoring systems and traditional expert system domains is that cognitive fidelity is much more important in tutoring system knowledge bases. This

makes the knowledge more difficult to acquire and much more difficult to evaluate (since we can never know with certainty what happens in an expert's head).

In summary, tutoring is not an easy domain in which to build AI programs, compared to most other expert systems domains. The acquisition of domain knowledge is made more difficult by the need for cognitive fidelity, and by the need to represent (even if shallowly) complex types of knowledge (such as intuitions and mental models) as well as procedures and rules. The acquisition of strategic knowledge is difficult because we do not have clear models of pedagogy for computer tutors (nor do we have operational models of human tutoring), making tutoring knowledge difficult to acquire and represent. Therefore, although knowledge acquisition methods used in mainstream AI are applicable to ITS, the methods or when they are best used may differ for ITS.

Current ITS Knowledge Acquisition Research

Perhaps for reasons mentioned above, almost all ITS studies that deal with knowledge acquisition have dealt with only a small number of the important knowledge acquisition issues, falling into one of three categories:

- Generality. The systems mentioned in Section 2.1.3 on generic ITS frameworks address knowledge acquisition in that they constitute frameworks and languages for representing knowledge. These projects, however, do not address the methods or issues involved in *acquiring* the knowledge, except (in several cases) to specify that it be done in a top-down or bottom-up fashion.

- Machine learning. One way to acquire knowledge is to let the computer program learn by itself (leaving the human out of the loop). ITS efforts in machine learning were mentioned earlier in this Section..

- Cognitive and procedural task analysis. The knowledge acquisition area that has received the *most* attention to date in ITS is task analysis [Means & Gott 1988, Lajoie 1986, Cerri 1988]. Procedural task analysis involves creating a procedural

representation of expert problem solving behavior. Cognitive task analysis infers underlying cognitive structures and processes, and is used to study both expert and novice behavior. (A further discussion of task analysis is found in Section 2.4.2).

Though the three areas above are important, we believe that ITS researchers should, on the whole, pay more attention to other traditional AI knowledge acquisition issues such as: What are good methods for acquiring a teacher's knowledge and building a knowledge base (perhaps as a function of the domain or task)? What signposts, issues, tradeoffs, and pitfalls does one encounter in trying to represent that knowledge? This study addresses these questions in the context of building a physics tutor with the KAFITS system. We also summarize a wide range of knowledge acquisition methods (below), noting how they apply to ITSs construction.

2.4.2 Knowledge Acquisition Methods

In order to provide ITS designers with an overview of commonly used knowledge acquisition methods, we mention the entire range of knowledge acquisition methods used in AI, describing how they apply to eliciting knowledge from domain experts and educators. Knowledge acquisition methods and tools can be categorized along many dimensions, for example: whether they are manual or computer-based [Boose 1988]; whether they involve natural or contrived situations for the expert [Shadbolt & Burton 1989]; the amount of domain understanding required of the knowledge engineer; and the knowledge engineer's level of involvement in knowledge engineering sessions (passive vs. active). Our organization of knowledge acquisition methods is synthesized from Hoffman [1987 and 1989], Bose [1988], Garg-Janardan & Salvendy [1988], and Shadbolt & Burton [1989], and includes the following methods (each described later in detail): "analysis of formal documents" and "tutorial sessions" are often used to bootstrap the process, giving the knowledge engineer enough information to propose an initial system architecture and to plan how he will work with the domain expert; "judgment tasks" can be used to develop an initial conceptual framework for a knowledge base; "analysis of off-line tasks" and "interviews" are typically

the work horses of knowledge acquisition, and are the main methods used for task analysis; and "on-line test-and-refine tasks" are used to test, modify, and extend a knowledge base. These methods are not mutually exclusive and are often combined.

Analysis of formal documents. Formal documents (a term borrowed from McGraw [1989]) such as texts, manuals, data bases, and other references, can provide a starting place for acquiring domain knowledge, and tutoring knowledge can be gleaned from instructional literature on theory and practice. But this is only a beginning, and it is essential to work closely with domain experts and educators to check and modify the information gained from other sources. Sub-categories include:

- Manual acquisition. Text can be an initial source of the knowledge engineer's familiarity with the domain. A prototype knowledge base can even be generated from text sources before consulting with the domain expert.
- Automated acquisition. Text can be automatically analyzed (the frequency and juxtaposition of words, for example) to yield the concepts and associations that might form the baseline for a domain [Shaw & Gains 1986]. Such methods tend to result in a large number of of spurious concepts, so they are not good for consistency or relevance, but they may be good for completeness, reminding the expert of areas of the domain they have not thought to mentioned.

Tutorial sessions. The expert can explain the concepts, procedures, tutoring strategies, etc., of the domain to the knowledge engineer (who might be taking the role of an apprentice), perhaps walking him through tasks or problem solving. This yields a useful first pass at the domain knowledge, and gives the knowledge engineer enough information to plan future knowledge acquisition sessions. Again caution is warranted: an expert's verbal knowledge of his problem solving or teaching skills will probably be incomplete and in places erroneous.

Judgment tasks. Originally taken from psychological and social-science experimental methods, these tasks involve rating, ranking, creating taxonomies, or sorting pre-defined

stimuli [Hoffman 1989]. They are a type of (very) "structured interview" (see below). Several systems have been built to automate these tasks and perform a variety of types of analysis on the data [Shaw & Gaines 1986].[38] Judgment tasks are especially useful in domains where the rules or strategies are fuzzy or not well articulated, yet performance is repeatable. Sub-categories include:

- Grid generation, hierarchy classification, and concept maps. The expert is guided though the generation of tables or graphs which diagramitically show relationships between ideas.
- Card sorts. Objects represented on cards are sorted in various ways, according to whatever characteristics the expert deems important. Card sorts are useful for discovering structure in knowledge. For example, having a teacher sort descriptions of word problems could yield a useful classification scheme, such as the categories: qualitative/intuitive, equation generation, and quantitative.
- Repertory grids. Repertory grids [Shaw 1981] organize numerical ratings of instances along a variety of bipolar dimensions. The grid is analyzed statistically to determine correlations or causal rules relating the dimensions.

Analysis of (off line) tasks. The expert is observed in action, solving a domain problem or tutoring (and see Section 2.5.3 for a discussion of cognitive task analysis vs. procedural task analysis). Parameters of this method include: whether to ask the expert for explanations as he works; whether to tape the session and/or perform a protocol analysis; whether to observe the expert at a real task during part of his normal work day, recreate a historically archived task situation, or invent a hypothetical task situation. Sub-categories include:

- Familiar tasks. The expert is observed performing a familiar everyday task.

[38] Rules (relationships between features) which best account for the data can be generated automatically, and automatic analysis can reveal structure in knowledge by constructing useful classifications and taxonomies.

- Tough or unusual cases. The expert is given a difficult or extreme case problem. This can reveal more refined reasoning.

- Constrained tasks. The expert performs a familiar task with contrived constraints on time or some other resources, or is not given some of the information he is normally given. He is forced to make approximations or extrapolations, use heuristics, etc. that might not occur in a normal situation. For example, a student works at a computer and the expert monitors and assists the student from another computer. Here the communication band-width is quite limited compared to face-to-face tutoring.

- Generating analogies. The expert analyzes or solves problems (cases) by comparing them with other cases (historical or standard cases). Similarities and differences in the cases and in the actions the expert would take are noted.

Interviews. The knowledge engineer questions the expert. This can be done in conjunction with a task (see above).[39] Sub-categories include:

- Unstructured interview. Opportunistic questioning is an acceptable way to acquire the initial scope and goals of a domain. It is also useful to repeat unstructured interviews at various stages in the knowledge acquisition process to generalize, refine, and reorganize the knowledge as it is being acquired.

- Structured interview (also called "focused" interview). The knowledge engineer plans what questions or tasks will be given to the expert prior to the knowledge acquisition session. To do this the knowledge engineer needs some understanding of the domain. The knowledge engineer also pre-determines whether there are constraints on what he can say. Specific guidelines for conducting interviews with the intention of protocol analysis can be found in Erickson & Simon [1984].

- Brainstorming. Creative problem solving, or idea generation (done without criticism or refinement of solutions) can be useful to get a feel for the scope of the domain, and

[39] Very loosely, we can see the discussion of analysis of tasks as relating to what to observe, and the discussion of interviews as relating to how to observe it.

can be useful when the problem solving strategies of the domain have not previousl been well articulated (in texts, manuals, etc.). Brainstorming is also useful when interviewing a group of domain experts.

- Decision analysis. Decision analysis techniques [Brown 1989, Hart 1986] can be used to pinpoint and analyze critical decision points in problem solving or teaching.

On-line testing and refining of the knowledge base. Using any one or a combination of the above methods a first-pass knowledge base can be created. Then the expert can directly inspect the contents of the knowledge base (using an interface), or run the expert system or tutor and observe its behavior. She can add, delete, modify, qualify, reorganize, or generalize the facts, concepts, rules, strategies, etc. for verity, completeness, efficiency, and coherence.

Comparisons of Knowledge Acquisition Methods

In the discussion of knowledge acquisition methods above we have indicated how each could be used in ITS design. Below we summarize comparisons of the methods that were found in the literature.

Interviews vs. other methods. Hoffman [1988] says that, although unstructured interviews are by far the most often used knowledge acquisition method, they have been shown to be one of the most inefficient. Also, experiments by Shadbolt & Burton [1989] indicate that protocol analysis of interviews performed significantly worse than other methods they analyzed, both in the resulting domain coverage and the effort it required. Hoffman warns that transcription and analysis of interviews takes at least 10 times as long as the interview (and often much longer). He points out that much information is lost in a taped interview, so protocol analysis is not only difficult but often not as useful as one expects. Yet it is difficult and disrupting to interrupt an expert very often while problem solving, so it is still useful to tape problem solving sessions and go over them later with the expert.

"The moral here is that the interviewer should take copious notes and not rely passively on audio tapes" [Hoffman 1987, pg. 56].

Structured vs. unstructured interviews. From his experiments, Hoffman concludes that structured interviews are more efficient than unstructured ones. He found that special tasks (i.e. contrived tasks) were more useful than natural tasks and familiar cases for getting tacit or non-conscious knowledge, and for focusing on specific areas of the knowledge. However, constrained or contrived tasks can make the expert uncomfortable, so they should not be over-done.

Similarly, Shadbolt concludes that contrived techniques did at least as well as a structured interviews of a familiar task in effort expanded and domain coverage. He found that contrived tasks (such as card sorts and grid constructions) tend to yield declarative knowledge, while natural tasks (such as familiar tasks and structured interviews) tend to yield more procedural knowledge.

Multiple experts. Lastly, Shadbolt concludes that using several experts with any single knowledge acquisition method yields more coverage than using several methods with the same expert.

2.4.3 Summary of Knowledge Acquisition and Intelligent Tutoring

In this section we first discussed the following knowledge acquisition issues identified in the AI literature: mapping tasks and domains to knowledge acquisition methods; automating knowledge acquisition; important characteristics of domain experts, knowledge engineers, and domains; and secondary goals of knowledge engineers (such as establishing rapport with the expert). We then compared ITS with other expert system domains, and discussed current ITS studies that deal with knowledge acquisition. Finally, we catalogued a number of knowledge acquisition methods applicable to various stages of building an ITS knowledge base, as summarized in Figure 2.3, and summarized some studies that have compared knowledge acquisition methods.

- Analysis of formal documents
  - Manual acquisition
  - Automated acquisition
- Tutorial sessions
- Judgment tasks
  - Grid generation, hierarchy classification, concept maps
  - Card sorts
  - Repertory grids
- Analysis of (off line) tasks
  - Familiar tasks
  - Tough or unusual cases
  - Constrained tasks
  - Generating analogies
- Interviews
  - Unstructured interviews
  - Structured interviews
  - Brainstorming
  - Decision analysis
- On-line testing and refining of the knowledge base

Figure 2.3 Knowledge Acquisition Methods

2.4.4 KAFITS and Knowledge Acquisition

Most ITS studies that deal with knowledge acquisition fall into one of three categories: architectures for generic shells, applications of machine learning, and task analysis of domain expertise. In contrast this study deals more directly with the issues of mainstream AI knowledge acquisition (though we also designed a generic shell architecture), and we discuss methods, tradeoffs, and tools for working with instructors/domain experts to acquire their pedagogical knowledge.

Knowledge Acquisition Methods Used

We used several knowledge acquisition methods to elicit pedagogical knowledge from the domain expert, the main methods being the *structured interview* and *on-line test-and-refine tasks*. We used several types of structured interview methods, including brainstorming, concept mapping, and script generation (see Section 5.1) to define the curriculum and tutoring strategies. In addition, we allowed the KAFITS representational framework to provide a structure for many of the sessions (for example, conceptualizing the domain content in terms of topics, misconceptions, hints, etc., guided our planning of knowledge elicitation sessions). In the knowledge acquisition literature structured interviews are recommended over unstructured interviews, and protocol analysis of taped interviews is discouraged. Therefore we used structured interviews when possible (especially during the design phases of the project), did not tape interview sessions, and followed Hoffman's [1988] advice to take copious notes during interviews.

After the domain content was encoded in the knowledge base, on-line test-and-refine tasks were used to debug, modify, and restructure the knowledge base. By using different teaching strategies (such as "skim" and "detailed"), and testing multiple alternative paths through the curriculum, the expert was able to test the knowledge base for correctness, completeness, and coherence. Unlike most knowledge acquisition interviews, in which the domain expert is interviewed so that the knowledge engineer can represent and enter the expert's knowledge, many interviews in this study served to help the domain expert repre-

sent *his own* knowledge in terms of a specific framework (KAFITS). Another fairly unique aspect of our process was the inclusion of design team members called "knowledge base managers" who did the data entry and syntactic debugging of the knowledge base, allowing the domain expert and knowledge engineer's time to be used efficiently.

In addition to structured interviews and on-line test-and-refine tasks, several other knowledge acquisition methods were used, including analysis of formal documents and unstructured interviews. Domain tutorial sessions were not needed because the knowledge engineer was familiar with the domain being taught (mechanics). Task analysis was not done during this study because: (1) we did not observe the domain expert solving physics problems (since we were not building an expert system in the domai)n, (2) we did not study students solving problems because we relied on the domain expert's knowledge of previous studies of misconceptions in the domain, and in addition we were not building a runable (cognitively valid) student model, and (3) we did not observe the expert tutoring because we did not focus on the acquisition of tutoring strategies (though taped tutoring studies that we administered prior to this project influenced the KAFITS design, as described in Section 5.4.1).

Knowledge Acquisition Issues Addressed

Knowledge acquisition issues we discussed in this section are addressed in our study as follows:

- **Mapping tasks to knowledge acquisition methods.** In Section 5.1.1 we describe the ten-step process we followed to design the knowledge base for the statics tutor. One unique aspect of our method is that the domain expert's representation and conception of curriculum knowledge evolved through classroom-like, CAI-like, and finally ITS-like representations.

- **Automating knowledge acquisition.** KAFITS does not incorporate automated knowledge acquisition techniques or machine learning methods, but does automatically check the consistency of some aspects of the knowledge base.

- **Important characteristics of domain experts, knowledge engineers, and domains.** In Section 5.1.2 we discuss important characteristics of domain experts and knowledge engineers which we abstracted from our observations. In Section 4.1.2 we discuss how our choice of domain limits our conclusions.

- **Secondary goals of knowledge engineers.** We found all four of the secondary goals mentioned mentioned—learning about the domain, establishing rapport, ascertaining the expert's needs, and instructing the expert in AI concepts—to be important in building the statics tutor, and we discuss these goals in Section 5.1.

- **Other issues.** KAFITS was designed to meet the needs of both experienced and novice users, as discussed in Section 3.5.1. Though we did not elicit expertise from multiple experts, we had two people testing and modifying the knowledge base simultaneously (and one of the knowledge base managers, Kim, was knowledgeable in physics), so some issues related to having multiple experts building a knowledge base are relevant .

2.5 ITS Evaluation Methods

AI, and therefore ITS, are relatively new fields having unique characteristics: their research and engineering goals are closely coupled, as are evaluation and design methodologies. As mentioned in Section 2.3, these fields have not matured to a point were principled scientific methodology is commonplace. Therefore in this section we offer a fairly complete compendium and discussion of evaluation issues, paradigms, and methods applicable to ITS research. For completeness we mention a wide range of methods, but focus mostly on methods applicable to formative qualitative studies, because the results of this study rely primary on formative qualitative evaluation. Since few general discussions of ITS research

methodology exist, one purpose of this section is to inform the reader of the range of methods available and the tradeoffs involved in selecting them. Another purpose is to justify our choice of evaluation methods, because some of the methods we employ are fairly uncommon (or not commonly understood) within the AI and ITS fields.

We begin our exposition by describing the current state of evaluation in ITS research. Then we describe three complementary overarching research paradigms. Then we describe a number of more specific common and uncommon evaluation methods. Finally we note how the evaluation methods used in our study of the KAFITS system relate to the evaluation methods listed.

2.5.1 The State of ITS Evaluation

It is generally accepted that a good evaluation methodology is lacking in most AI research ; the situation may be even worse in the ITS sub-field. Here are two harsh but typical critiques: "[Claims of ITSs] are based on testing that typically is poorly reported, inconclusive, and in some cases totally lacking" [Rosenberg 1987, pg. 7], "existing ITS literature is a barren source for good examples of outcome measurement" [Baker 1991, pg. 252]. [40] What factors contribute to the lack of sound evaluation—why is it so difficult or rare? Of the many possible reasons we will cite only a few. As a research field ITS is relatively new and "evaluation is not standard practice in part because we don't have formal research methods, standard experimental designs, and analytic tools" (Cohen & Howe [1988, pg. 1]; said of AI research, but applicable to ITS research). Also, research in ITS (in contrast to most AI expert systems research) is lacking an objective model of the expected behavior, i.e. the ITS field does not have concrete models of tutoring expertise (see discussion of ITS vs. traditional expert systems in Section 2.3.1). In addition, evaluating ITSs in realistic situations has been difficult because of the logistics of interfacing with classrooms and (until recently) the complexity and cost of the hardware on which ITSs are typically built [Woolf

[40] For other discussions of ITS evaluation, see Park et al.[1987], Koedinger & Anderson [1990], Shute [1990], Baker [1991], Littman & Soloway [1988].

(in press)]. In sum, implementation is just beginning to be common, evaluation is typically ad-hoc and shallow, and testing in realistic instructional settings is rare.[41] This is not surprising considering the reasons given above, and the fact that intelligent tutoring combines difficult issues from many disciplines: cognitive psychology, human factors, instructional theory, knowledge representation, etc.

Difficulties like those mentioned above do not prohibit ITS evaluation—it is essential—but we have to be creative about how we do it. It is not the case that in order to make a substantial contribution to the field ITS researchers have to evaluate the instructional effectiveness of their systems in controlled experiments.[42] "...As a field, we are not obliged to adopt 'scientific' evaluation criteria, but should design our own" [Cohen & Howe 1988a, pg. 3] (said of AI, but applicable to ITS). There is a wide range of evaluation methodologies applicable to ITS research, suited to many types of research goals and constraints, and we catalogue a number of them below as grist for the mill for ITS designers. First we describe general categories of evaluation methods or evaluation "paradigms," and then we describe specific evaluation methods.

2.5.2 Research and Evaluation Paradigms

In choosing an evaluation method one must determine: 1. the research goals, questions, or hypotheses addressed, 2. the phenomena to be observed or parameters to be measured, 3. the experimental procedure and data collection method, and 4. the data analysis method. In order to justify the methods we employ in this study, and also to suggest a partial mapping from goals/hypothesis/phenomena/parameters to data collection and analysis methods, we describe three categories of research/evaluation methods which emobdy different (but not

[41] The situation with regard to ITS evaluation is slowly improving as the field matures. Five years ago reports of un-implemented systems were common; today implementation is almost expected, yet evaluation is uncommon and usually ad-hock. If this trend continues, within five years some form of evaluation might be normal, and within ten years the field may boast of principled evaluation based on accepted methods and metrics.

[42] Cohen & Howe [1988b, pg. 19] note that "because we are not trying to reduce complex phenomena to their causal antecedents, we do not need to run large groups of subjects in experimental and control conditions...analytic techniques...are fundamentally reductionistic, and so are not much use unless one's goal is to identify the components of complex [naturally occurring] behavior."

mutually exclusive) research paradigms: formative evaluation, qualitative evaluation, and case study evaluation. These paradigms are not exclusive, in fact we combine all three in this study.

Qualitative Research

Though the distinction between qualitative and quantitative seems fairly obvious, it can be non-trivial to determine whether a study constitutes qualitative research or not. Bogdan & Biklen [1982, pg. 27] give five features of qualitative research:[43] [44]

- The researcher is the key instrument—subjective judgement is involved in collecting the data, be it note taking, photographs, videotapes, etc.;
- It is descriptive—the data collected is primarily in the form of words or pictures rather than numbers, and anecdotal evidence is common;
- It is concerned with process rather than only outcomes or products;
- Data tend to be analyzed inductively—abstractions and theory are built "bottom up"; and
- "Meaning" is of essential concern—the researcher is interested in the beliefs, attitudes, and perceptions of the participants/subjects.

Quantitative research, on the other hand, is usually characterized by the traditional "scientific" research paradigm, including data collection, control of variables, and statistical or numerical analysis, and is often deductive, intending to prove or disprove a hypothesis. The traditional scientific research paradigm is a *positivist* approach, with an emphasis on facts and the causes of behavior, assuming that one reality or one correct answer exists, and research is only a matter of measuring it. In contrast, qualitative research assumes

[43] Not all qualitative research satisfies all of these criteria—it is a matter of degree.

[44] See [Patton 1980, pg. 88] for a "check-list of evaluation situations for which qualitative methods are appropriate."

a *phenomenological* approach, "that the world is not an objective thing out there, but a function of personal interaction and perception" [Merriam 1989, pg. 17].[45] [46]

Formative Evaluation

Formative evaluation is a process in which empirical data is collected *during* the development process, and incremental improvements are made to a system based on this data [Patterson & Block 1987, pg. 26]. Formative methods are often used to "define and refine...goals and methods during the design process" [Littman & Soloway, 1988, pg. 210], or to get more clarity on what the salient issues are in a field of inquiry. Summative evaluations, on the other hand, test the effectiveness or correctness of a final product, and the researcher must be fairly clear about the research goals and the range of possible outcomes at start of the study.

Focus on Qualitative Formative Methods

The discussions above (and in Section 2.3.1) suggest that the traditional scientific research paradigm is not appropriate for many ITS studies. If it is most important to document or prove that an ITS system or method "works," (which is often particularly important when when the indended audience is a funding agent or the world outside one's field of research) then summative and quantitative methods may be best. But for ones research colleagues, who are less concerned with whether "it works" than they are with which aspects work, which don't work, and (most importantly) *how or why*, qualitative and formative evaluation methods are often most appropriate. Rosenberg [1987, pg. 7] says: "ITS implementors use primarily quantitative methods. In an area such as tutoring, this is probably a mistake." Buchanan [1988, pg. 1] agrees: "...at present time, AI has to be more concerned with *qualitative* statements of regularities than statistical statements because the

[45] Bogden & Biklen [1982 pg. 31] point out, however, that "the irony of it is that scientists in the hard sciences...do not define science as narrowly as some of those who emulate them."

[46] See Glaser & Strauss [1967, especially Chapter 5] for a method for analyzing large corpora of qualitative data.

framework for being more precise does not yet exist." In addition, Littman & Soloway [1988, pg. 210] suggest that "[since] building ITSs' is still somewhat of an art...for the time begin *formative* evaluation seems more appropriate for designers of ITSs.' Since we are mainly concerned with exploring key issues in ITS knowledge acquisition rather than demonstrating that a particular piece of software or computer curriculum is more effedctive than brand or method "X," we use several evalution methods, most of which are qualitative and formative.

Exploratory Research and Case Studies

Designing a research study requires familiarity with previous related work, accounting for the results of past studies, and usually using or extending previously applied methods or metrics. However, in some areas there has been little groundwork and it is difficult to design research studies. We call such areas "exploratory" areas of inquiry, and call research in these areas exploratory research. Exploratory areas of inquiry have the following characteristics:[47]

- there are no generally acknowledged experimental or evaluation methods, standards, or metrics;
- important issues, tradeoffs, and problem areas are not well documented; and
- little evaluative data is available from previous studies.

Based on these criteria, our study of ITS knowledge acquisition with educators is exploratory research. The goal of exploratory research is to better understand the problem, to discover what the important issues *are*, and *begin* to form theory where little or none exists. The *case study* method is well suited to exploratory research since it "aims to uncover the interaction of significant factors characteristic of the phenomena [being studied]," and is "particularly suited to situations where is impossible to separate the phenomenon's vari-

[47]Before committing to do research in such an area the scientist must first determine that the area has not been ignored because it is trivial, intractable, or unimportant.

ables from their context" [Merriam 1988, pg. 10].[48] The case study is a research paradigm, not a specific method, and "what makes [an] inquiry a case study is the decision to focus the inquiry around one instance" [Merriam 1988, pg. 44]. In so doing one can collect data of a more detailed and subtle nature than is possible if one spends the same resources on many instances. The case study researcher typically immerses himself or herself in a complex situation and tries to make sense of it—to highlight phenomena and distill categories and principles from complex experience, and assumes that "nothing is trivial, that everything has the potential of being a clue which might unlock a more comprehensive understanding of what is being studied" [Bogdan & Biklen 1982, pg. 28]. This contrasts with traditional science in which variables are controlled to minimize uncertainty and obfuscating complexity.

2.5.3 Evaluation Methods

Below we outline an array of evaluation methods, descriptions of which were found scattered throughout the AI, psychology, and education literature. We give examples of each method from the ITS field, describing commonly used methods first, then uncommon ones. The methods are not mutually exclusive, and are often used in combination. Most of the methods are applicable to qualitative research and formative research, but many could be performed in either qualitative or quantitative contexts, and many could be used as preliminary tests (which lead to the design of a system), as formative studies (done during the design of a system), or summative studies (done with a completed system).

An Overview of Common ITS Evaluation Methods

First we describe the four most common evaluation methods used in ITS research: existence proofs, quantitative studies, semi-quantitative studies, and cognitive studies.[49]

[48] Case studies can be quantitative, but are usually qualitative studies, and Merriam [1988, pg 57] notes that "a case study design can be used to test theory, but a *qualitative* case study usually builds theory."

[49] Some of the methods are described as being less than adequate when used alone. Note that this is not necessarily a reflection on the ITS study associated with that method because: 1. some studies employ

Existence proofs. "Existence proof" is the term used for a study which bases its conclusions on the successful performance of a system, and is the most common evaluation method used by ITS (and AI) researchers. Usually generalizations from existence proofs are limited because they are not based on an underlying theory (they don't prove or disprove anything), don't mention design tradeoffs or ideas that did not work, and don't provide reasons why various aspects of the system *did* work. Existence proofs usually propose new architectures or design methods, and if combined with other evaluation methods (such as cognitive studies, comparison studies, or inductive evaluation, each described below) can be informative. Sub-categories include:

- Description of a prototype system. Showing that a program achieves its goal behavior is usually the weakest form of existence proof, and is stronger if the goal behavior is stated clearly and precisely before the program is built and/or tested. For example, Burke & Ohmaye [1990] show that using a case-based approach to tutoring is possible.

- Turing test. By showing that a program simulates human behavior researchers can argue for the cognitive validity of their systems or show that a program has adequate knowledge to simulate expertise. Documented human behavior can be used as a standard, or the system's output can be judged by outside evaluators to be indistinguishable (within some parameters) from expert performance. For example, Woolf [1984] showed that her architecture for planning tutorial discourse could simulate documented instances of tutoring from three domains.

- Toy domains. A theory or computational model is shown to work in a very constrained situation. The prototypical example in AI is the "blocks world." An example from ITS is Burton & Brown's WEST tutor [1982] which aimed to teach discovery skills in the context of a simple computer game. Toy domains are used to eliminate the large number of variables (some of them uncontrollable) inherent in realistic domains, so that a general theory can be demonstrated. But generalizations from toy domains

---

more that one method, and 2. earlier studies were more exploratory in nature and are not expected to be as methodologically rigorous as more recent ones.

should be made with caution because experience has shown that "scaling up" AI systems is fraught with difficulties [Buchanan 1987].

- Inductive proof. A system is used with multiple (usually a few) domains or users and an argument is made for its generality. For example, the Byte-sized architecture [Bonar et al.1986] has been used to build tutors in electricity, PASCAL programming, economics, and hydrostatics. The strength of an inductive argument rests on the size and diversity of the successful instances.

Quantitative studies. Though we focus mainly on qualitative (and semi-quantitative) methods, for completeness we mention some exemplary quantitative summative evaluations in the traditional scientific paradigm.[50] Quantitative studies, thus far rare in the ITS field, are characterized by statistically significant student populations, experimental and control groups, and/or pre- and post-testing.

Excellent examples of quantitative evaluation can be found in many papers by Anderson and colleagues, and Shute and colleagues. Both of these researchers have shown that learning with intelligent tutors can result in significant improvements over classroom learning, and, equally important from a methodological standpoint, have documented surprises and "negative" results, as described below.

Typical evaluation metrics include: mastery levels (amount learned), learning rates (efficiency), retention, transfer and generalization of skills, and the range of learned abilities across a population. For example Anderson et al. [1985] compared students using the LISP tutor with students completing similar exercises on their own (both groups received the same lecture and reading material) and found the the LISP group learned in 33% less time, and scored 43% higher on final exams. Research does not have to be completely rigorous to be worthwhile: Bonar et al.'s [1988] study of the PASCAL ITS did not include a control group, but they compared students using the tutor with records of standard classroom

[50]The reader is referred to standard text books on statistical and quantitative analysis for descriptions of quantitative methods.

courses and found that it took about three times as long to learn the same material in the classroom vs. using the computer tutor.

Quantitative studies can also be used to explain individual differences in student behavior and demonstrate predictive correlations between learning variables. For example, Shute & Glaser [1990], in a study conducted with the Smithtown economics tutor, showed that scientific inquiry behaviors (such as hypothesis generation and testing) were significantly more predictive of successful learning than standard measures of general intelligence. In another study of the LISP tutor, Anderson [1990] found that students' acquisition and retention abilities explained variance in computer tutoring situations and predicted performance on paper-and-pencil exams.

Extremely few ITS papers report surprising results or results that contradict the original hypothesis.[51] But sharing such information is crucial if researchers are to learn from each others' mistakes. One example a report of unexpected results is Shute's [1990] study of two computer learning environments for electricity, one environment supporting inductive learning and the other supporting deductive learning. Her hypothesis that learning efficiency would be enhanced in the deductive mode was disproved—she found that there were no main effects of the learning environment on either learning outcome or learning efficiency. But she did discover that students with high working-memory capacity performed better in the deductive environment. In a study of the LISP tutor Schooler & Anderson [1990] discovered that there was an advantage to delayed feedback in terms of errors, time on task, and the percentage of errors that subjects corrected. In another study of the LISP tutor Corbett & Anderson compared four types of feedback and discovered that feedback did not affect the mean learning rate or post-test performance. The findings of these two studies were unexpected and in apparent disagreement with Anderson's ACT* theory of cognition

[51] Cohen [1991, pg. 26] found only 12 out of the 150 papers in the 1990 AAAI Proceedings discussed unexpected or negative results. He lists a number of methodological problems and indicates that they can be traced to the fact that systems are rarely based on *models*. "Lacking models of how systems are expected to behave, we will see no predictions, no hypothesis, no unexpected results or negative results, only assertions that a system works."

[Anderson 1983], leading the authors to re-interpret how the theory applied to the learning situations in the experiments.

Semi-quantitative studies of system components (sometimes called "quasi-experimental" studies). As alluded to previously, ITS researchers do not need to complete summative tests of the educational impact of full-functioning tutoring systems in order to contribute to research in the field. Here we list studies that use quantitative metrics to explore individual factors or features, but do not involve controlled studies, and use data pools too small to allow statistically significant conclusions.

ITS systems are complex and people are (clearly) more complex, therefore determining how ITSs will behave in realistic situations can only be accomplished through empirical methods. For example Murray et al. [1990], in a study of an analogy-based tutor for remediating physics misconceptions, mapped out locations in an analogy network where subjects experienced cognitive dissonance and changes in understanding. They found that most subjects experienced a significant change in understanding when presented with carefully sequenced analogies, and that many did not change until they were given a causal (molecular) model of the underlying physics of situations.

In some cases the success of ITS components can be evaluated without student pre- and post-testing. For instance, when the goal of a component is to diagnose student errors or generate new problems, the "hit and miss rates" can be tested. For example, in an evaluation of the PROUST system, Johnson [1988] reports that of 206 solutions to the rainfall programming problem, PROUST analyzed 81% completely (i.e. was able to come up with a consistent model of the student's underlying intentions in constructing the problem solution). He goes on to specify which aspects of student programs the system has difficulty diagnosing, and to suggest future improvements to the system. Similarly Sleeman [1982] reports that, in one test, his LMS system's "mal-rules" were able to account for 12 out of 27 incorrect student solutions to algebra problems.

Also, researchers can analyze how users utilize various features of a tutor. Winkels et al. [1986] report on a detailed analysis of the frequency with which students use various

Unix-mail commands from one coached session to the next. Similarly, Kimball [1982], in an evaluation of a tutor for symbolic integration, reports on student use of help and assistance features, and on the program's ability to recognize points in the tutorial where the student learned something new.

Cognitive studies and task analysis. Perhaps the most commonly used research method in ITS is observing the behavior of domain teachers (or experts) and students (or novices). Usually such studies are used to inform the design of the tutor rather than to test a theory of expertise, instruction, or learning, so they are not evaluation methods per se. (See our discussion of cognitive and task analysis methods for *knowledge acquisition* in Section 2.4.2.) Evaluation of a computer tutor occurs when the tutor is tested and on-line results are compared with studies of human behavior. Two types of task analysis are discussed in the literature: procedural task analysis and cognitive task analysis.

Procedural task analysis (usually called "task analysis" or "rational task analysis") involves studying the behavior of an expert to generate a formal procedural representation of some skill, and has long been used to identify and classify elements of expertise for the purpose of instruction [Gagne 1974]. More recently, cognitive task analysis is being used to infer the goals, attitudes, rules, and underlying cognitive structures or models of experts [Bonar & Soloway 1985, Means & Gott 1988, Lajoie 1986, Payne 1988].

Cognitive analysis has also been widely used to describe the mental structures and misconceptions that students or novices bring to the learning task [Clement 1982, Koedinger & Anderson 1990, Cerri 1988, VanLehn 1988, Littman & Soloway 1988]. Procedural and cognitive task analysis have been used to study instructional expertise as well as domain expertise [Winkels et al. 1986, Lewis et. al 1990, Collins & Stevens 1990, Lepper & Chabay 1988, Leinhardt & Greeno 1986].

The most common methods for task analysis are audio or video taped problem solving sessions, clinical interviews, and protocol analysis.

A Compendium of Less Common ITS Evaluation Methods

Below we describe less common evaluation methods that are applicable to ITS research.

Outside Assessment. Though outside assessment is not usually a proof of anything, the opinions of experts, or of a large number of users or potential users of a system, does carry some weight, and it can be significant if there is agreement. Sub-categories include:

- On-site expert evaluation. Experts observe and assess the behavior of the system, for example, several teachers observe students using an ITS and rate its teaching effectiveness. Both structured and informal means of gathering data are possible.
- Panel of experts (sometimes called a "blue ribbon panel"). A select group of experts is questioned by convening them or though correspondence. This method can be used in the proposal stage of a research project to substantiate the importance of a problem statement or the feasibility of an experimental approach, or can be used after a study to document the acceptance or feasibility of one's conclusions. A hyperbolic example is the following: documenting that 80 % of the editorial boards of the Artificial Intelligence and Cognitive Science journals believe that your research problem is "highly significant" and your approach is "feasible," which would argue heavily for the importance of a study, and thus the importance of the conclusions.

Comparison studies. "Comparison study" refers to a broad class of methods which compare (note similarities and differences between) the behavior or design of a system vs. some standard or other system. It is important to compare one's work with previous work, and comparisons with other systems are often invoked in ITS papers. However comparisons usually only serve to put one's study in context with previous work and do not add new knowledge to the field. Comparisons are strengthened if they include qualitative or quantitative data and detailed comparisons of program functionality, behavior, or educational effects. Sub-categories include:

- Gold standard.[52] Judge system performance against a well known successful case or standard. For example, Ford [1988] evaluates his tutor in the Highway Code domain by determining how well it lives up to a fifteen-criterion definition of "intelligence" in ITSs suggested by Self [1985b].

- Theoretical corroboration. Arguing or proving at a theoretical level that other systems can be simulated using a new (usually more general) system argues for the generality and extendibility of an approach. For example, Dillenbourg [1988] proposes a three dimensional system for classifying ITSs, and describes how his PRO-TEG system and six other ITS's compare using this scheme.

- Empirical Corroboration. In a field as complex as ITS empirical evidence of one's claims is usually needed. Empirically demonstrating that the features or tutoring behavior of an ITS accounts for that of another system establishes a provable base line.

- Duplication. Simulating the behavior of another system (for example, showing that an ITS can be configured to simulate Anderson's LISP tutor) can establish a base line from which to extend the findings of previous work.

Internal Evaluation. Internal evaluations [Littman & Soloway 1988] study the relationships between a program's architecture and its behavior, and often involve in-depth analysis of program traces and data structures. Sub-categories include:

- Knowledge level analysis. Addresses the question: what could a program infer from its *knowledge structures* assuming unlimited inferencing capability (i.e. arbitrary algorithms)? For example, in his GUIDON project Clancey [1982] analyzed the structure of the domain and control knowledge in the MYCIN expert medical diagnosis system. He determined how limitations in the knowledge representation would limit its effectiveness in computer tutoring. As a result of his analysis he re-designed the system (to make NEOMYCIN) so that it would support tutorial reasoning.

[52] Term borrowed from Cohen & Howe [1988a].

- Process analysis. Addresses the question: how do a program's *algorithms* determine and limit its behavior? For example, Littman & Soloway [1988] analyze the limitations of what the PROUST system can infer about the student, based on how it represents and diagnoses the programming task (and the student's model of programming). They analyze cases were PROUST could not completely diagnose students' programs and give recommendations for how the system could be improved to accommodate these failures.

- Ablation and substitution experiments [Cohen & Howe 1988a]. A part of a system (a component, feature, rule, etc.) is removed (or replaced with a more primitive version) and one observes how the system's performance is affected. It is usually clear that performance will suffer, but exactly how and how much it suffers can be surprising, and yield information about the knowledge or inferencing capabilities of the components. For example, Corbett & Anderson [1990] conducted a study in which the feedback mechanism of the LISP Tutor is altered or removed. They report on the effects to student mastery, learning time, and attitudes (self-perception, confidence, and enjoyment).

Miscellaneous off-line testing. "Miscellaneous off-line testing" refers to methods (not mentioned in other categories) that do not directly use the computer or the program under study.

- Debriefing interviews, user questionnaires, and surveys. Questionnaires and surveys can be used in interview form or via a written questionnaire (useful when collecting data from many people). They are often used to collect data about the reactions and beliefs of users before and/or after a study; for example, Schofield and Verban [1988] interviewed students and teachers involved in a study of Anderson's LISP tutor to discover barriers and incentives to using computers in schools.

- Wizard of Oz experiments. Using a person to simulate the behavior of a *proposed* system[53] can test aspects of a program design before the effort is expended to implement it. For example, Sandberg et. al. [1988] collected data to inform the design of an intelligent help system for a text editor by having an expert assist a user (novice) by communicating via a low bandwidth terminal setup that attempted to constrain the expert to act as the proposed computer tutor would.

- Matching, ranking and sorting tasks. Subjects are presented with stimulus (often on cards, unless the task is automated) and asked to match, rank, or sort them. Such tests are useful to ascertain categories or relationships that a person uses implicitly, but does not have explicit knowledge of.

2.5.4 Summary of ITS Evaluation Methods

ITS research takes place in the midst of great excitement and promise, yet also in the midst of great complexity and uncertainty. Principled evaluation of ITS systems, though important, is unfortunately uncommon. Discussions of evaluation methodologies and descriptions of tradeoffs among evaluation methods are also rare in the ITS field—which motivated us to compile descriptions of a number of paradigms and methods applicable to ITS evaluation. In this Section we first described several research/evaluation paradigms: qualitative (vs. quantitative), formative (vs. summative), and case study. We suggested that formative and qualitative methods are most appropriate for most ITS research and we described how case study methods fit the needs of "exploratory research." Then we described a compendium of evaluation methods, summarized in Figure 2.4, giving examples from ITS research. It is hoped that our presentation will 1. stimulate additional discussion about ITS evaluation, and 2. inform interested readers of the wide variety of methods they can use.

Though we have recommended that formative and qualitative studies be used when the audience is one's research colleagues, since evaluation is so uncommon it is more important

[53] Interestingly, computers are usually used to simulate human performance, but here one does the opposite.

- Existence proofs
  - Description of a prototype system
  - Turing test
  - Toy domains
  - Inductive proof
- Quantitative studies
- Semi-quantitative studies of system components
- Cognitive studies and task analysis
- Outside assessment
  - On-site evaluation
  - Panel of experts
- Comparison studies
  - Gold standards
  - Empirical corroboration
  - Theoretical corroboration
  - Duplication
- Internal evaluation
  - Knowledge level analysis
  - Process analysis
  - Ablation and substitution experiments
- Miscellaneous off-line testing
  - Wizard of Oz experiments
  - Matching, ranking, and sorting tasks
  - Questionnaires and surveys

Figure 2.4 Evaluation Methods

to pick *any* evaluation method suted to the research questions that one is comfortable with and use it.

The key to turning ITS systems building into research can be summarized as follows:

1. be clear about why you are building a system and relate your work to previous work;
2. record design decisions and what works and doesn't work as you build a system;
3. use a system in ways that challenge its performance;[54]
4. report unexpected and "negative" results ("air the dirty laundry") as well as successes, and describe *why* a system works (if it does at all) and why it doesn't work in some situations; and
5. generalize results beyond the specific application.

2.5.5 KAFITS Evaluation

As mentioned above, our study combines several evaluation paradigms: it is a *case study* and a *formative evaluation*, and it incorporates primarily *qualitative evaluation* methods, though some quantitative analysis is done. Our major source of data was (field) notes (as in most case studies). We also collected data from computer traces of student runs, traces of domain expert editing sessions, and paper worksheets used by the domain expert. Though most of the analysis is of the field notes and program traces, we employ a number of other evaluation techniques to get a bearing on important issues from several directions, as described below.

The study also constitutes an *existence proof* that a knowledge acquisition interface can be used by a teacher to build an intelligent tutor, and it constitutes a weak *inductive proof* of the usability of the interface, since three users were involved. We performed some

[54]Cohen & Howe [1988b, pg. 9]: "by testing a program at its known limits we can better understand its behavior...we can push its limits by providing imperfect data...restricted rescources...and perverse test cases."

quantitative analysis involving a number of variables, such as the size and complexity of the knowledge base, the time it took to complete various design steps, and the total development time per hour of on-line instruction (see Section 5.3.2). In addition, we included a *comparative analysis* of KAFITS vs. other generic tutorig systems in Section 2.1. Also, we conducted an audio tapped *debriefing interview* at the end of the study (described in Appendix C) and *tested the statics tutor* on about 20 subjects as part of the evaluation, as described in Section 5.2.

CHAPTER 3

DESCRIPTION OF THE KAFITS SYSTEM

In this chapter the KAFITS system, including the domain and strategic knowledge bases, the underlying knowledge representation and control mechanisms, the user interfaces, and the student model, are described. Though the system evolved over the duration of the case study, we describe it here in its final state, and in Chapter 5 we discuss how the system changed in response to feedback from users. We describe the KAFITS system first, *then* the research method (in the next chapter) because some familiarity of the system is needed to understand the description with the research methodology. Note that included in this chapter is a description of the interface for creating and modifying tutoring strategies, a prototype of which was built, but not tested with the domain expert. All other components described were used by the domain expert.

3.1 Description of the Representational Framework

3.1.1 Domain and Strategic Knowledge Bases

The KAFITS system is designed to represent *pedagogical knowledge*—i.e. knowledge related to teaching and learning in domain. Pedagogical knowledge includes how to teach the content, examples of concepts, questions that determine subject mastery, hints for questions, knowledge about prerequisites, common errors, etc. KAFITS is designed to represent expertise in teaching a domain—not expertise in solving problems in a domain

(i.e. pedagogical knowledge is distinct from performance knowledge).[1] As in many tutoring systems we impose a clear separation between the domain-dependent knowledge (the domain knowledge base) and the engine that interprets this knowledge [Clancey 1982]. *Unlike* most tutoring systems we represent strategic knowledge (i.e. tutoring strategies and rules) explicitly and declaratively (i.e. in data structures rather than Lisp code) in a strategic knowledge base (see Figure 1.2).

Strategies specify how to use the domain knowledge base content. For example, the domain knowledge base contains information about topics and about relationships between them (the topic network) and strategies in the strategic knowledge base define how to traverse this network when tutoring (see Figure 1.2); the domain knowledge base contains links between topics and numerous "presentations" having different functions (such as giving examples, introducing a topic, etc.) and the strategies determine which of these presentations will be used, and in what order; the domain knowledge base contains information for giving hints, revealing the correct answer, and strategies determine how and when to use this information. The strategic knowledge base is quasi-domain independent, i.e. some strategies are intended for any instructional domain (i.e. any domain knowledge base content) while others are designed to work for specific curricula.

3.1.2 Object-Oriented Representation

The representation of the domain knowledge is object-oriented. The object-oriented paradigm allows knowledge to be used for multiple purposes, facilitating modularity and a reduction in information redundancy [Bonar et al. 1986]. For example, the knowledge in KAFITS needed to define, summarize, test, remediate, give an example of, and/or teach a KAFITS topic is stored in close association with the topic. As in the Byte-sized Tutor [Bonar et al. 1986], teaching strategies and student modeling methods can be inherited and associated locally with the topic and presentation objects. We describe the objects in the KAFITS system below, in order of importance.

[1] However, it is possible to combine a domain expert system with a KAFITS-based tutor.

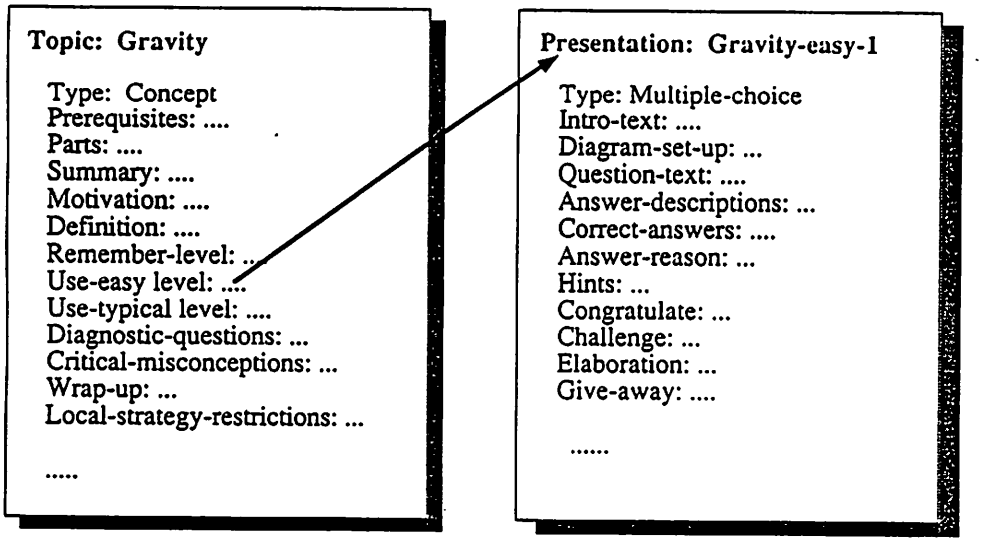


Figure 3.1 Topic and Presentation Attributes

Topic. Topics represent units of knowledge that can be taught, remediated, summarized, etc. (see Figure 3.1 for the attributes of topic objects). They are categorized according to content type, for example concept, fact, or procedure. Each topic has several performance (and mastery) levels associated with it (adapted from Merrill's [1983] Performance-Content matrix, as described in Section 2.2.4). Topics have pointers (including various types of prerequisite, part-of, and related-misconception links) to other topics, forming a topic network (as in Figure 1.3). Topics have pedagogical information such as summary, motivation, examples, etc., that reference presentation objects (see the arrow in the figure).

Presentation. Presentations specify expository or inquisitory interactions with the student (see Figure 3.1 for the attributes of presentation objects). They are composed of a task (such as a multiple choice question or problem solving exercise) and an environment for doing the task (such as a picture or a simulation of a physical system). Presentations also contain the breadth of possibilities for responding to the student, such as hints, congratulations, elaborations of answers, etc.

Mis-KU. Buggy student knowledge is represented in **Mis-KU** objects (mis-knowledge units). Mis-KUs represent misinformation (wrong facts), misconceptions, and buggy rules

or procedures. Mis-KUs contain information pertaining to diagnosing themselves, and remediating themselves.

Lesson. Lesson objects are pedagogically motivated groupings of topics reflecting a particular pedagogical view or goal. They are used to define high level goals for a tutorial session. Typically, a lesson specifies a small number of goal topics, and a default strategy to be used for the session. Lessons provide a level of abstraction or large granularity for incorporating the "glue" between topics [Lesgold 1988].

Example and Question. Examples and Questions are objects that allow for more flexible Presentations. Examples set up an environment or situation for the student, such as a simulation, a picture, a set of tools, etc. Questions give the student a task, such as a multiple choice question, a problem to solve, or simulation manipulation to carry out. Normally, presentation objects automatically incorporate the slots of an example and a question, and example and question objects are not needed. However in some cases the situation or task is re-used. For instance, there are many possible tasks that could be given to a student for a single situation (such as a specific blocks world set up). Conversely, a single task (such as the question "Which force is greater?") could be presented in a variety of situations. To reduce redundancy, each task or situation can be defined once, and presentations can consist of a paired example and question.

Storage-unit. Storage-units are objects used to store static global information that otherwise would reside in computer code and be inaccessible to the user. {Random-text} is one object of type storage-unit. A random phrase from a set of similar text items is returned by a computer function that that takes a keyword as input (for example, given :GREETING it returns a random item from "hello," "hi there," "hi," or "greetings"). These random text strings are incorporated into some strategy actions (such as Tell-correctness, which can say "you are right," "that is correct," etc.). The user can modify these items or create new random text items by using the Browser to edit the {Random-text} instance.[2]

[2]There is a slot for each keyword, and the contents of the slot is a list of text items.

With storage-units the user can have ready access to a wide variety of system parameters that would otherwise be inaccessible parts of computer code.

Other objects. The KAFITS framework incorporates three other types of objects: pictures and sounds, which contain information allowing the tutor to incorporate graphics and sound, and crane-booms-setups, which specify configurations of the simulation.

3.1.3 Layered Decision Model

Various schemes for organizing tutorial expertise in computer tutors have been proposed. Clancey [1982] used networks and a production rule formalism to identify admissible instructional actions. Woolf [1984] used a three layer "transition network" which defined states ("pedagogic states" "strategic states" and "tactical states") and default state transitions. Similarly, Breuker et al. [1987] used a three layer control structure having goals, strategies, and tactics. Others have described the decisions as falling into the categories "what to say," "when to say it," and "how to say it."[3] All of these schemes include some form of layered system going from abstract to more specific decisions until a concrete decision is made. Our scheme is a decision model related to the above schemes and to instructional design theory. It is designed to be understandable to instructors and to be easily assimilated into their model of pedagogical decision making.

Instructional theory typically divides instructional methods into the micro level (how to teach a single instructional unit) and the macro level (selection, sequencing, and synthesis of instructional units) [Reigeluth 1983a]. We use a refinement of this perspective, consisting of four decision levels. The macro level is refined into a *lesson level* and a *topic level* (see Figure 3.2). The micro level is refined into a *presentation level* and a *response level.* In the control structure of the program, these decision levels are nested control loops. One or more topics contained in each lesson, one or more presentations in each topic, and zero or more tutor responses to the student's response to each presentation.

[3]Since we are not concerned here with issues of discourse or natural language generation, text is "canned" or template text, and "how to say it" issues are bypassed.

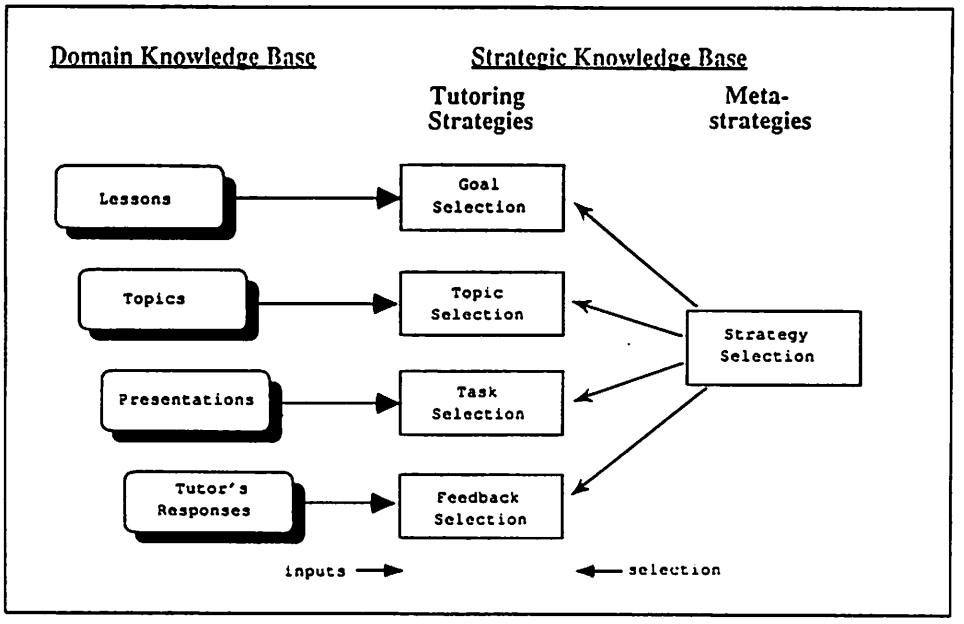


Figure 3.2 Four-Level Decision Model

Lesson level. At this level decisions are made concerning the high level goals of the instructional session. The need for a lesson level is supported by Reigeluth's [1983b] description of "sets" of topics taught together according to a specific type of knowledge type relationship, introduced with an "epitome", and followed up by a "within-set-synthesize," which relates the topics in a lesson.

Topic level. The topic decision level deals with choosing which topic to teach next. For instance, teaching strategies at the topic level determine which path is taken to traverse the topic network. Decisions about comparing, contrasting, synthesizing, introducing, and summarizing topics are made at the topic level. Decisions about remediating misconceptions are also made at this level.

Presentation level. The presentation decision level deals with the selection and sequencing of presentations such as analogies, definitions, examples, graphics, and simulations.

Response level. The response decision level deals with sequencing low level tutorial transactions, and with how to respond to the learner's actions, including the quantity and type of feedback to be provided.

The four-layer decision model determines the default behavior of the tutor, but the tutor is not constrained to follow this model, there are several ways to override it. First, the student can *interrupt* the session to go back, skip forward, or jump to another part of the curriculum. Second, the designer can specify a *local branch* for remediation (to another part of the curriculum) associated with a particular student response. Third, evidence for misconceptions can accumulate and the tutor periodically diverts the session to *remediate* the pending or suspected misconceptions. Lastly, *meta-strategies* determine at a global level when to switch from one tutoring strategy to another.

3.1.4 Object Mixins

KAFITS uses a "flat" hierarchy and "mixins" to organize its object types. This method contrasts with a strict hierarchical organization of objects in an inheritance network, a method which was implemented in our early design of KAFITS and is often seen in frame and object-based AI systems. Before describing the method we eventually used, we describe the more standard hierarchical approach, and why it was rejected.

In the hierarchical classification paradigm each object type has a parent (or class or super-type) and zero or more children (or sub-classes or sub-types), forming a tree structure (see Figure 3.3). The leaves of the tree contain instances. Each object is a specialization of its parent, inheriting the parent's properties (slots, slot values, methods) and having a few properties specific to itself. For instance, a top level class could be "topic," with sub-types of topics called concepts, skills, facts, etc. (see Figure 3.4). Each sub-type could in turn have its own sub-types, such as procedural-skills and problem-solving-skills, and so on, making further distinctions, such as the subtraction-procedure and the long-division-procedure. Such an organization has the advantages of procedural inheritance. For instance, default Summarize and Diagnose procedures could be defined for all topics, and each sub-class could specialize this procedure to its own needs. However, this way of implementing object inheritance was replaced with a flat hierarchical "mixins" scheme because the flat

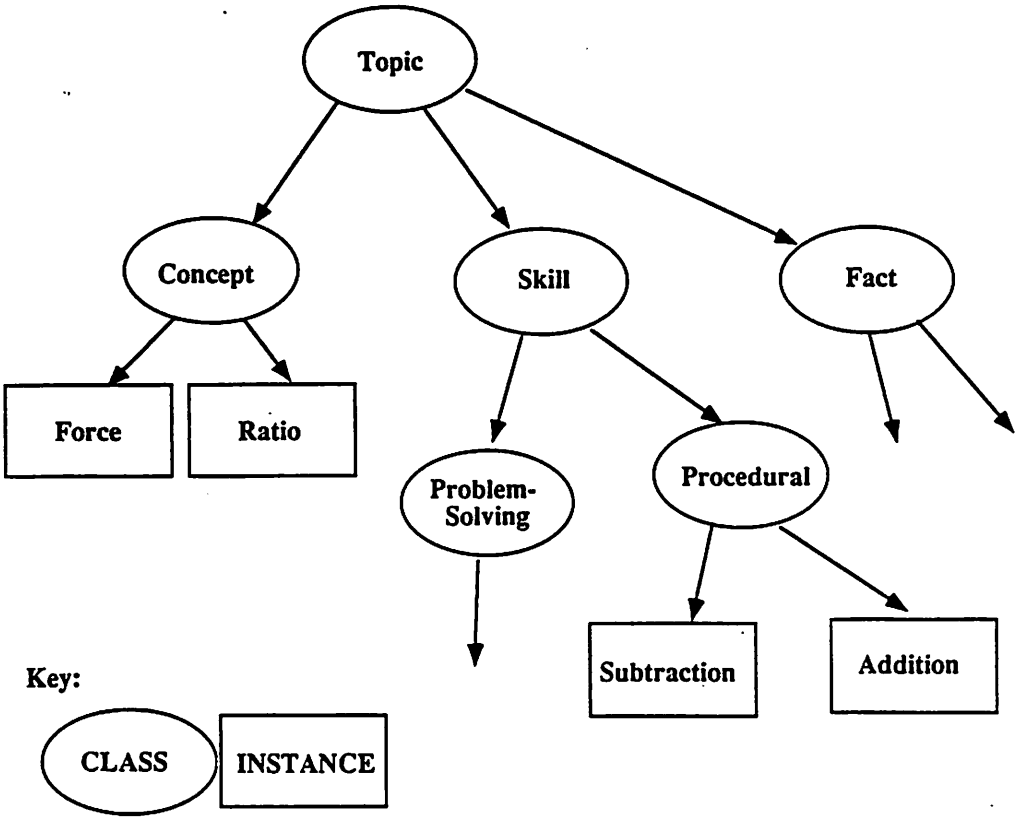


Figure 3.3 Hierarchical Object Organization
(Not implemented in the final system)

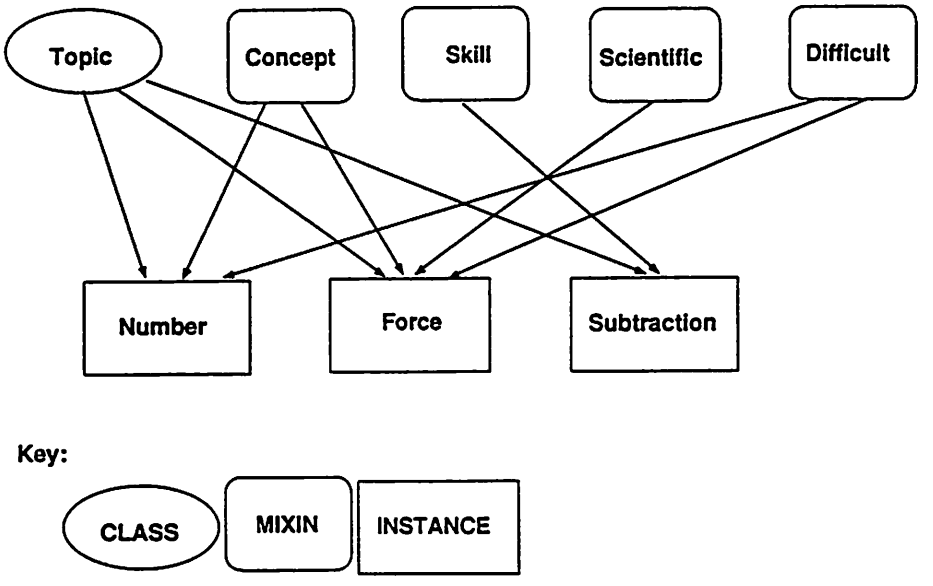


Figure 3.4 Flat Hierarchy for Mixins

hierarchy is less complex, more easily implemented, and more understandable to teachers (and we did not need the additional power afforded by the strict hierarchical scheme).

In a flat hierarchy there are only two levels. The top level contains the main object types and mixins for the types; and the lower level contains the instances. Mixin objects are like other objects, except their sole purpose is to be combined with other objects (i.e. the slots and methods of the mixin are added to the object). Each instance has one main parent (an object type) and specializes the attributes of this parent by adding the attributes of mixin objects designed to be combined with the object type. For example, in Figure 3.4 the topic object type has mixins that modify the basic topic object with attributes for concepts, difficult topics, etc. The Force topic in the figure is a difficult scientific concept having three mixins.[4] The user specifies a set of mixins when a new instance is created.

[4]Note that Figure 3.4 is only an illustration of how mixins work, and does not reflect the mixins in KAFITS. In fact the topic types in KAFITS are indicated by a "topic-type" slot.

3.1.5 Representing Curriculum Structure

Our representation of curriculum includes topics categorized according to knowledge type, several types of topic relationships, and performance and mastery levels within the topics. Borrowing from Merrill [1983] we distinguish content types from performance levels (see Section 2.2.4). We implement content types as node types in the topic network and implement performance levels as levels within each topic. For example the topic "gravity" is of type *concept* and the designer can specify presentations for the remember, apply-use, and problem-solve levels within it.

Topic Types. Our knowledge type categorization distinguishes "basic" vs. "complex" knowledge. Basic knowledge types correspond to the content types in the Modified PC-matrix described in Section 2.2.4 (facts, concepts, procedures, and principles). Complex knowledge types, such as mental models and physical intuition, (see Figure 2.2) are loosely defined as any type of knowledge not accounted for in the Modified PC-Matrix. Nodes in the topic network are classified as: fact, concept, procedure, principle, complex, composite, synthesizer (there are no synthesizers in the statics topic net), or Mis-KU[5] (misconception) (see the key in the lower left of Figure 1.3). Facts, concepts, procedures, and principles are defined in Appendix B. *Composite nodes* represent a collection of topics of different types.[6] For example, Linear Equilibrium is a topic of type composite; it is composed of three parts, LE-intuition, LE-concept, and LE-principle. *Synthesizer nodes* are used to represent a relationship between two topics (such as a comparison of two topics—see description in Section 5.4.3).

Topic Relationships. Links between nodes in the topic network indicate relationships between topics. There are three basic types of topic relationships: critical-mis-ku, part, and prerequisite. We found it necessary (as explained in Section 2.2.7) to further refine the prerequisite relationship into five types of prerequisite relationships, depending

[5]Mis-KUs are not a topic per-say, since they are represented using a their own object type.

[6]Normally a topic has parts of its own type, for example, a concept has sub-concepts as its parts, and a procedure has sub-procedures as its parts.

on how shallowly or fully a topic's understanding is needed: familiar, deep-familiar, easy, typical, and difficult. The first prerequisite level, familiarity, indicates that the student must only have been introduced to (given a summary, definition, or example of) the prerequisite topic. We also found the need for a "deep familiarity" prerequisite, meaning familiarity with a topic and all of its parts. There are prerequisite relationships for each mastery level (eg. {Linear-equilibrium-intuition} is an easy level prerequisite of {Free-body-problem-solution}). Tutoring strategies (at the topic level) use the node and link types in determining the order in which to present topics.

Topic Levels. Our modified PC-matrix distinguishes these "performance levels:" meta-knowledge, remember, use-apply, use-problem-solve, and create. In designing the topic network for statics we found that the knowledge referred to by some topics was limited to a single performance level ("use-apply"), and in these cases we needed another way to distinguish levels of performance. As a solution we include "mastery levels" for each performance level. We allow three mastery levels (easy, typical, and advanced) for each performance level. Mastery levels make the student model more precise and have allowed us to simulate "spiral teaching," as described (in Section 5.4.3). The performance and mastery levels are implemented as slots within each topic (see Appendix H). Though the statics tutor uses only five performance/mastery levels (remember, use-easy, use-typical, use-difficult, and meta-knowledge) many more are possible, up to a maximum of three mastery levels for each performance level in Figure 2.2.

3.1.6 Conceptual Vocabulary

Representing Tutoring Expertise

We have been involved in an ongoing effort (see Section 5.4.1, and Murray [1987]) to define a set of representational primitives (a conceptual vocabulary) for describing the objects, attributes, and events of tutoring. Tutoring strategies (regardless of whether they are represented as rules, decision networks, strategies, etc.) involve conditional actions

(IF/THENs).[7] We are converging on a limited vocabulary for the *antecedents* (i.e the "IF" parts, sometimes called predicates or situations) and the *consequences* (i.e. the "THEN" parts, sometimes called actions or results) of strategic decisions. Our goal is to develop a vocabulary that is *complete and expressive* enough to adequately represent the majority of the domain expert's strategies, and *conceptually simple* enough to allow domain experts to formalize their knowledge in a language that is intuitive and comfortable (see Section 5.4.2 for a discussion of the tradeoffs in designing expressive yet simple vocabularies).[8]

Antecedents. There are four types of antecedents: *curriculum-characteristics* (eg. factual-topic, difficult-question), *discourse-parameters* (eg. many-examples-were-given, last-question-was-wrong), *student-model-parameters* (eg. knows-topic, has-misconception, prefers-explanations), and *internal system parameters* (such as the PAN switches described later) which are variables that keep track of internal system states for control purposes. Therefore the conditional parts of rules are written in terms of characteristics of the content, the state of the discourse, the state of the student model, and internal system parameters.

Consequences. There are two types of consequences: *assertions* that assert a fact or value (IF xx THEN ASSERT yy, or IF xx THEN SET yy to zz), and *actions* that result in observable tutorial behavior (IF xx THEN DO yy). The conceptual vocabulary includes a set of *primitive tutorial actions*, such as Hint, Define, Remediate, etc., some of which are shown in Figure 3.5.

Elements of the Conceptual Vocabulary

The KAFITS framework was designed through a combination of experience with educators, taped interview studies (Section 5.4.1), and considerations from instructional design

[7] In the KAFITS system tutoring strategies are represented in action networks, as described in the next section, but this discussion is intended to apply to any scheme for representing tutorial expertise.

[8] At this time tutorial goals are not explicitly included in our strategies because this additional complexity was not needed. However, some studies have indicated that complex tutorial decisions are goal-oriented [Stevens & Collins 1977], therefore goals and plans may be included at some future date.

Topic level actions:

- Teach
- Motivate
- Summarize
- Define
- Teach-prerequisites
- Teach-parts
- Diagnose-critical-misconceptions
- Diagnose-topics-knowledge
- Give-examples
- Wrap-up

Response level actions:

- Set-up-environment
- Introduce-task
- Ask-question
- Give-reaction
- Congratulate
- Challenge
- Give-hint
- Reveal-answer
- Give-answer-reason
- Elaborate-on-answer
- Encourage
- Tell-whether-correct

Figure 3.5 KAFITS Primitive Discourse Actions

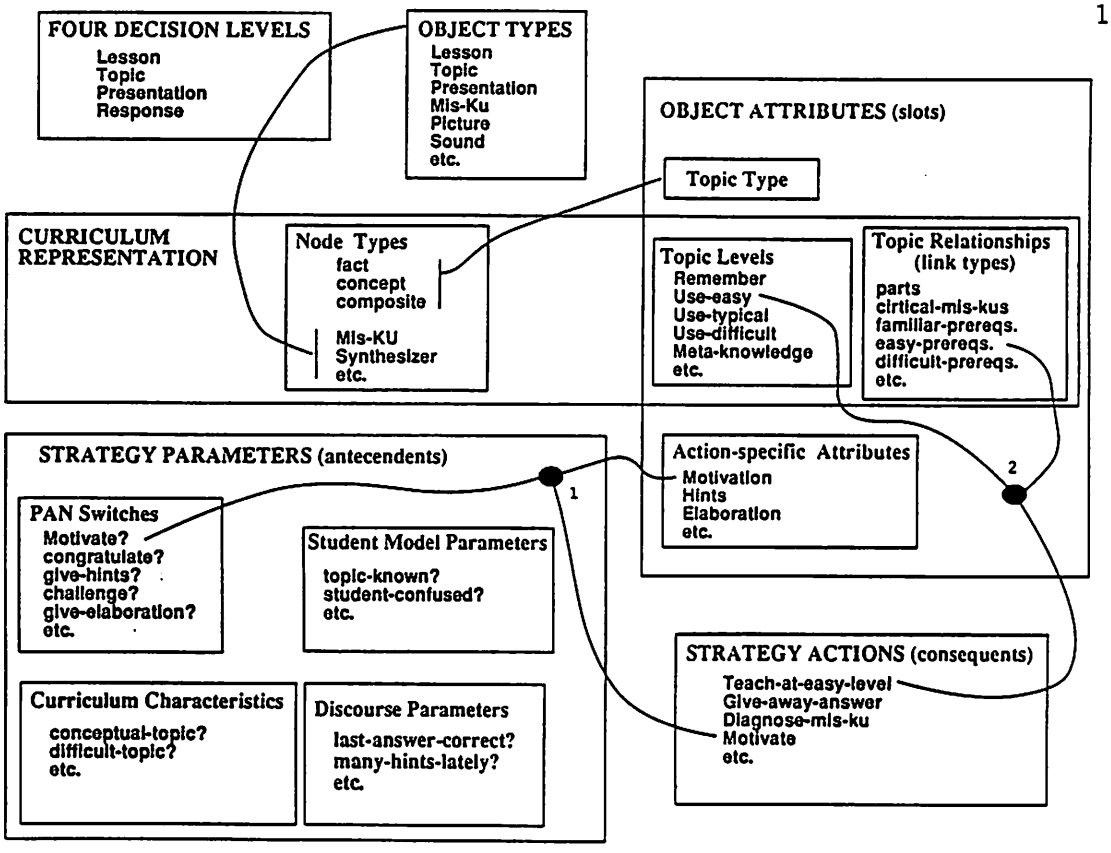


Figure 3.6 Elements of the KAFITS Conceptual Vocabulary

theory (Section 2.2.7).[9] The KAFITS conceptual vocabulary consists of the names of the entities, properties, and relationships used in the system, and is the language in which knowledge about curriculum and teaching is expressed. Technically, the KAFITS "framework" consists of the conceptual *vocabulary* plus a *structure* specifying how the things in the vocabulary are related or structured. For example "topic level" and "presentation level" are part of the vocabulary, and the Four-level Decision Model which defines how these levels are related is part of the structural framework; "topic" objects and "presentation" objects are part of the vocabulary, and the fact that topics reference presentations to specify interactions with the student is part of the structural framework.

The conceptual vocabulary consists these categories of entities: decision levels, object types, object's attributes (slots), strategy parameters, and strategy actions, as illustrated in

[9]See Section 3.6 for a discussion of extending the framework and conceptual vocabulary.

Figure 3.1.6.[10] [11] The names in several of the categories overlap (which can cause confusion when first learning about KAFITS), as illustrated by the numbered dots in the figure. For example, dot "1" illustrates that "Motivation" is a topic slot which contains canned text, "Motivate" is a primitive action which uses this canned text, and "Motivate?"is a PAN switch used in strategies to indicate whether motivation should be given. Figure 3.1 shows the attributes of topic and presentation objects.

3.1.7 Strategy Representation

Strategic information in KAFITS is represented with parameterized action networks (PANs). We first describe action networks, then *parameterized* action networks.

Our work on action networks is an extension of earlier work on discourse action networks [McDonald et al. 1986] and tutoring action networks [Woolf & Murray 1987]. Action networks have the look of conventional transition networks with "nodes" (states) and "arcs" (predicates) as one would find in ATNs [Woods 1970]. However, "actions" are used in the place of states, and "situations" (groupings of predicates) are used in the place of predicates. The motivation to modify the ATN architecture (as employed in Woolf [1984]) by replacing states with actions was based on the observation that ATNs were designed for natural language parsing and are non-deterministic.[12] Non-deterministic of uncertainty has no counterpart in discourse generation, which requires a *planning* rather than a *parsing* formalism [McDonald et al. 1986]. In ATNs, arcs represent predicates which can have

[10] "Curriculum Representation" elements are also shown, which include node types, topic relationships, and topic levels. Topic relationships and topic levels are topic attributes. Most node types correspond to the topic-type slot and some correspond to object types (such as Mis-KUs and Synthesizers).

[11] A more complete conceptual vocabulary for ITSs could include these additional categories: explanation types (eg. behavioral, causal, component, attribute, etc. [Stevens & Steinberg 1987]); types of student inquiries (eg. what/how/why, action/event/state [Gilbert 1987]); and types of examples (eg. extreme case, near miss, anchors, analogies [Murray 1988]).

[12] Nodes in the ATN formalism represent accepted definitions for incoming tokens and arcs represent tests made on those incoming tokens. Non-determinism was motivated by uncertainty or the need to wait for an accumulated global interpretation before the system could be confident about the local interpretation of each token being scanned.

side effects and can expand recursively (invoking other ATNs). Both of these features were needed for language recognition, but are not necessary for discourse planning.

In contrast to ATNs, in action networks the arcs (situations) represent Boolean combinations of predicates, and are not allowed to have side effects. Also, in action networks the *nodes* can expand recursively (invoking other networks) as in classical and hierarchical planners [Sacerdoti 1974].[13]

The left side of Figure 3.7 shows the Give-Feedback PAN. Lozenge-shaped nodes represent actions and arcs represent situations. After an action is completed all arcs emanating from that node are evaluated to determine which arc "passes" (using a conflict resolution scheme if several arcs pass), and program execution continues at the action pointed to by the arc that "passes." Oval nodes, such as the Congratulate node, represent calls to other action networks. Small circular nodes are either Empty nodes or Exit nodes. When control is passed to an Exit node (or when no arc emanating from the current action passes) the PAN is exited and control is returned to the PAN that called it (if there is one).

Arcs can be of several types: Always, Else, predicates, or Boolean combinations of predicates (using And, Or, and Not). The predicates refer to (1) the student model (for example, student-knows-topic or response-ok), (2) characteristics of the domain (for example task-is-difficult, or topic-is-conceptual), or (3) "switches." Switches define the parameterization of the network and constrain the possible paths through the PAN. A set of switches called "Helpful" is shown in the middle of Figure 3.7. Switches are either on or off. A set of switches with a particular setting is called a "switch register." For example, the Helpful switch register is defined to congratulate, but not challenge, the student. Notice that some of these switches correspond to predicates in the Give-Feedback PAN. When the Give-Feedback PAN is combined with the Helpful switch register, the action network shown

[13]We have chosen not to implement the classical top-down goal reduction type of planner for two reasons. The first is Occam's Razor. We have not yet encountered the need to incorporate all the power of classical planners, such as backtracking, constraints on actions, and reasoning about subgoal interaction. The second reason stems from our goal of conceptual simplicity and usability. Action networks, with their intuitive flow-chart like appearance, are easier for the domain expert to create, modify, monitor, and conceptualize than the plan operators of classical planners.

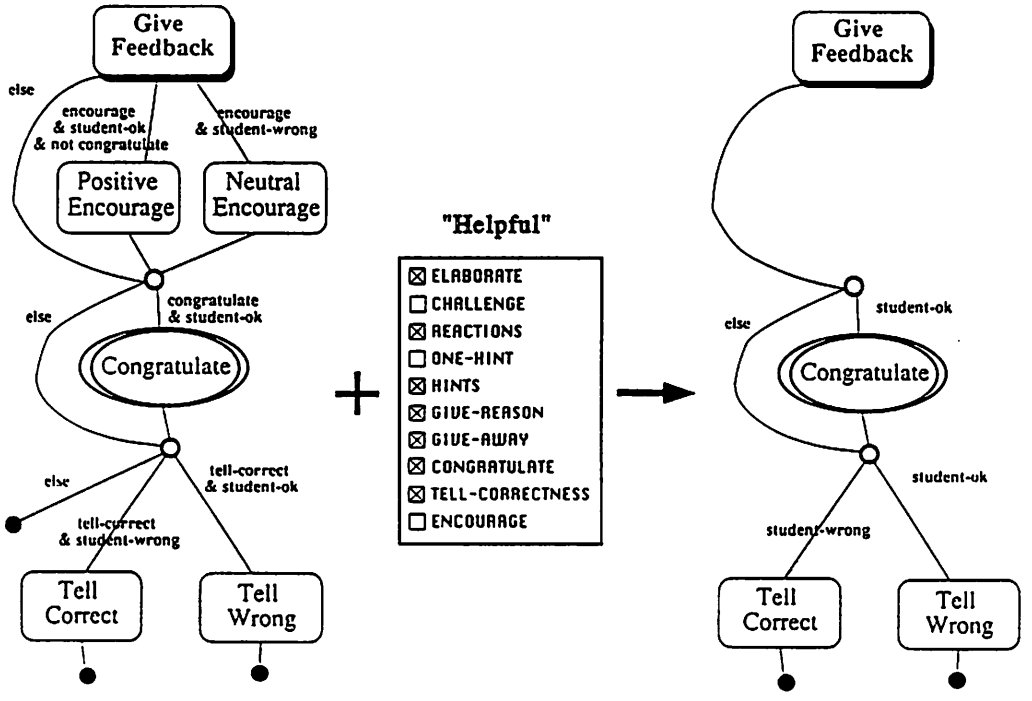


Figure 3.7 PAN + Switch Register = Strategy

to the right in the Figure results.[14] Thus, a "Strategy" is defined as a PAN combined with (constrained by) a switch register.

Parameterized Action Networks

Parameterizing action networks allows many alternative strategies to be represented using a single PAN. Rather than defining new action networks for each tutoring strategy, generic action networks are instantiated in multiple ways depending on the context at run time. For example, three of the arc tests in the Give-Feedback network correspond to switches: encourage, congratulate, and tell-correct. There are six different possible on/off combinations of these switches. Rather than defining six separate action networks, one for each case, we can define one PAN and six switch registers to cover all of the possibilities. The

[14]The resulting action network to the right in figure 3.7 is an illustration of how the Give-Feedback PAN behaves when the Helpful switch register is active. The resulting action network is not actually instantiated by the system. The Give-Feedback PAN is traversed by the system according to the values of its predicates, some of which depend on the current state of the student model, and some of which (the switches) depend on active switch registers.

domain expert can more easily create a new switch register than another action network. Also, having fewer action networks cuts down on the size of the strategic knowledge base and commensurate knowledge management problems.

The above explanation has included a simplification that we now remove. Actually, a switch register is associated with a *set* of hierarchically connected PANs, called a PAN* ("pan star"). The PAN* associated with a PAN includes the PANs it calls, plus all the PANs they call, and so on, recursively. For example, there are eight PANs in the PAN* for the Default Response Strategy (one of which is the Give-Feedback PAN), and four PANs in the PAN* for the Default Topic Strategy.[15] A switch register is defined for each PAN*. This explains why there are more switches in Figure 3.7 than are needed for the Give-Feedback PAN. A Strategy, then, is a PAN* combined with a switch register.

PANs vs. Production Rules. Action networks borrow features from production system formalisms as well as from network formalisms. Situations (multiple predicates) are associated with actions (nodes), in the manner of a production system. Discourse context is encoded in the structure of the network. Every situation (arc) implicitly includes as one of its constituent predicates the actions (nodes) from which it came. As in production systems, if multiple situations are true, a conflict resolution scheme is used to determine the next action. The single notational framework has the flexibility of a production system and the contextual record-keeping ability of a network formalism.

Though production rules are well suited for making assertions (IF xx THEN ASSERT yy), they can be awkward or unwieldy for representing control information (IF xx THEN DO yy) [Clancey 1986, Lesser 1984]. They obscure the difference between control and strategic information, and hide the structure of the strategic knowledge. Control information elicited from human experts often has a clearly defined structure. This structure is lost in the extremely modular format of production rules. Also, context is often important in rules elicited from humans. Structure and context are incorporated into production rules by

[15]The PANs for the default strategies are not shown here. Recall that there are strategies for each of the four Decision Levels. Unless otherwise specified, a "default strategy" is used at each level.

using ad-hoc antecedents or goals, but such information corresponds to the programming or implementation level of design, and is irrelevant to the cognitive level of design. Users should have a conceptually sound view of a system's control information and should not have to deal with information or decisions at the implementation level [Gruber 1987]. Structure and context are represented explicitly in PANs, i.e. the possible actions before and after an action are clearly indicated.

As an illustration, the arc that goes from the Positive-Encourage node to the Congratulate node of the Give-Feedback PAN (left side of Figure 3.7) could have been written in a production rule formalism as follows:

IF: 1. current goal is to Give-Feedback, and
2. last-action (i.e. the context) was Positive-Encourage, and
3. student-response-ok, and
4. Congratulate switch is on
THEN Congratulate

One such rule would have to be written for each arc in the network. The graphical PAN representation of Give-Feedback is more appropriate for ITS knowledge engineering than 13 textually represented rules (there are 13 arcs in the network) such as the one above because there are less entities to manage in the knowledge base, and because PANs provide a visual model which supports a robust cognitive model of the control process.

Changing strategies

The domain expert can define many alternative strategies—strategies for general tutoring styles and strategies tailored to specific segments of the curriculum. The KAFITS system can change the active strategies (there is one active or current strategy for each of the four decision levels) in the midst of a tutoring session. A strategy is changed by either changing the current PAN* or changing the current switch register (for one of the four decision levels).

Strategies can change dynamically during the tutoring session in three ways: at the local level, at the global level, and via student control. At the local level, an individual lesson, topic, or presentation can specify that a specific strategy be used. This local decision is stored in a slot of the instance and is editable like any other slot. At the global level, "meta-strategies" select the current strategies (see Figure 3.2). Meta-strategies are represented as IF/THEN rules. They probe the conditions of the student model and the domain model, and are checked periodically during program execution.[16] For example, IF student-is-floundering THEN USE Helpful-switch-register.

Lastly, the student can interrupt the tutoring session to change the strategy. The student is given a menu of choices, such as "more feedback," "less feedback," "more information," "less information," etc., which map into a change in the settings of the switches.[17]

3.2 Description of the Interfaces

The Knowledge Acquisition Interface is designed to reify the KAFITS representational framework, facilitating the organization and encoding of the domain expert's knowledge. The Knowledge Acquisition Interface has two components: an interface to the domain knowledge base (the Browser) and an interface to the strategic knowledge base (the Strategy Editor), which we describe in this Section. In this section we also describe the session monitors (tools that allow the domain expert to access information about the tutoring session) and the interface to the student. See Appendix D for a listing of all the menu operations available to the KAFITS user.

3.2.1 The Browser

[16] Meta-strategies and the meta-strategy interface have not been implemented yet.

[17] This feature has been implemented, but we have little data as yet on whether students can or choose to make use of it.

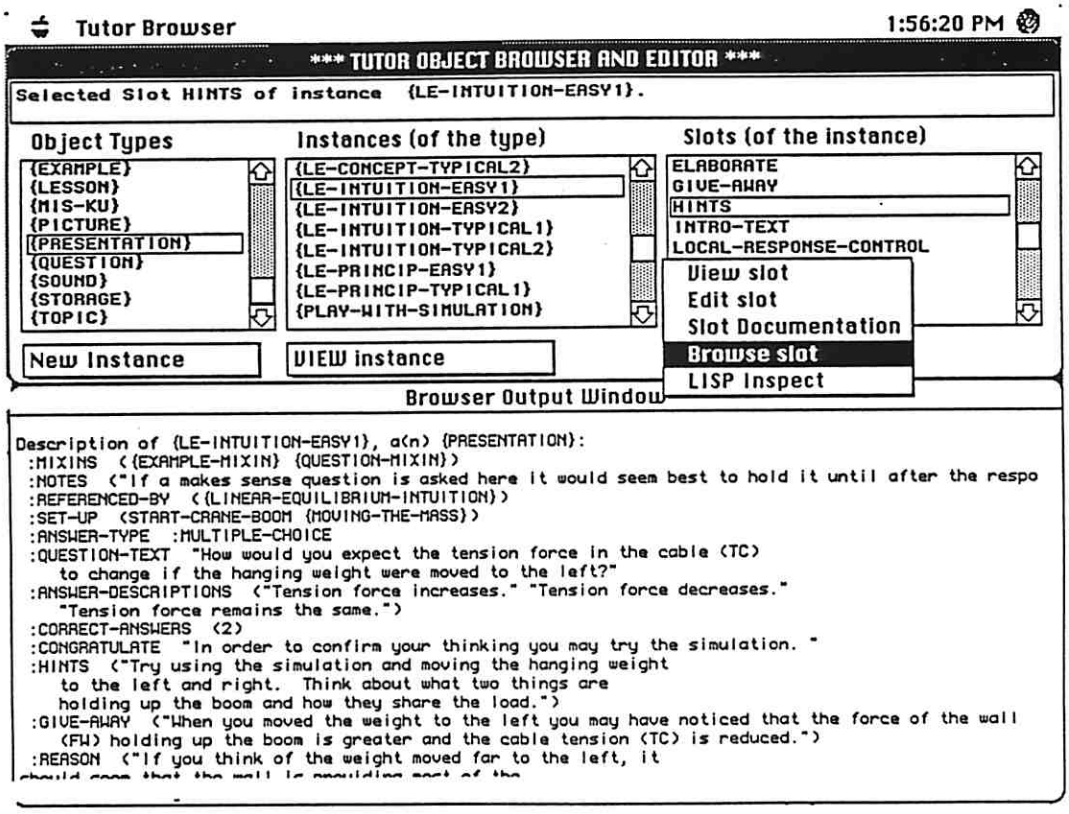


Figure 3.8 The Browser

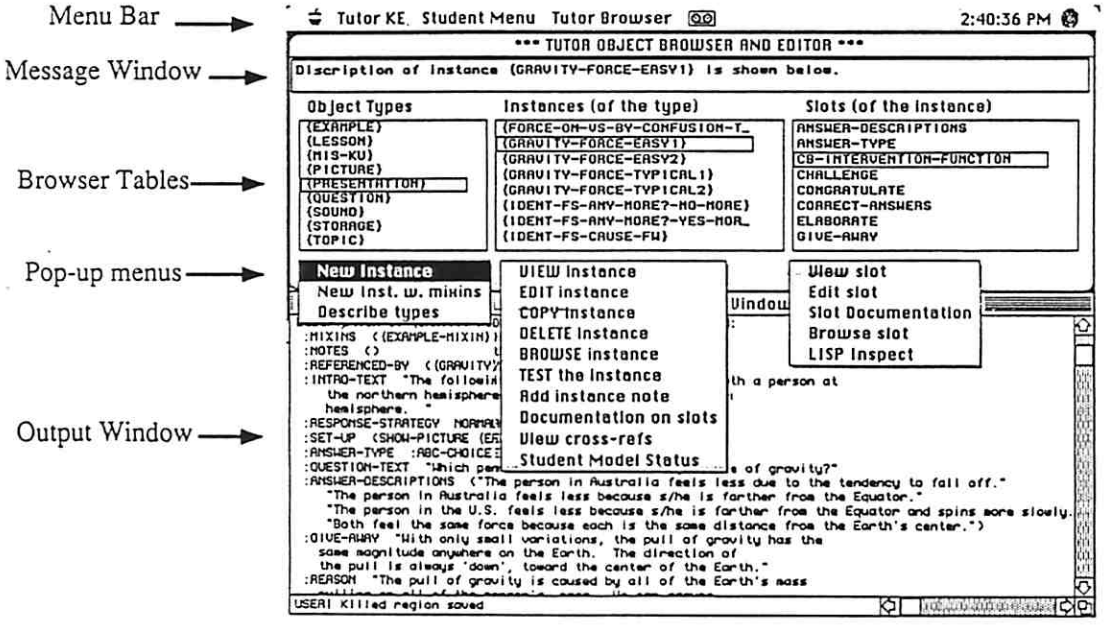


Figure 3.9 Components of the Browser

The Browser allows the domain expert to inspect, modify, and test the objects in the domain knowledge base. Figure 3.8 shows the Browser about to be used to Browse the slot Hints of the presentation LE-Intuition-Easy1 (the user had previously Viewed LE-Intuition-Easy1, and its description is shown in the Browser output window at the bottom of the figure).[18] At the top of the Browser a message window informs the user of warnings, gives brief instructions, and orients the user by giving information about the operation just completed. Below the message window are three tables from which the user can select (from left to right) an object type, and instance of that type, and a slot of the instance. Below the tables are three pop-up operations menus, one for each table (only the last operation performed is shown when the menu is not popped up). Figure 3.8 shows the user clicking on the slot operations pop-up menu and selecting "Browse slot". Figure 3.9 shows the operations for all the tables (the user can only see one pop-up menu at a time).

The Browser Output Window is a scrollable window in which information and listings are printed. When the user chooses to edit an instance or a slot an Editor Window appears, wherein the user can modify current values.

The Browser has the standard editing operations, viewing, copying, creating, editing, deleting, and also has these operations on instances: testing, browsing (viewing all objects that reference or are referenced by an object), adding documentation notes, viewing examples of and restrictions on slots, and inspecting the student model values associated with an instances. See Section 5.5.4 for a discussion of how the design of the Browser evolved in response to user input.

3.2.2 The Strategy Editor

The interface for creating and modifying PANs is shown in Figure 3.10. PANs are represented as editable graphic networks. Creating, deleting, repositioning, and testing

[18]The domain expert invents the names of the instances that he creates. LE-Intuition-Easy1 is the first presentation for the easy level of topic Linear Equilibrium Intuition. The Instances table of Figure 3.8 scrolls through an alphabetical listing of instances (presentations in this case, of which there are 81 defined for the statics domain).

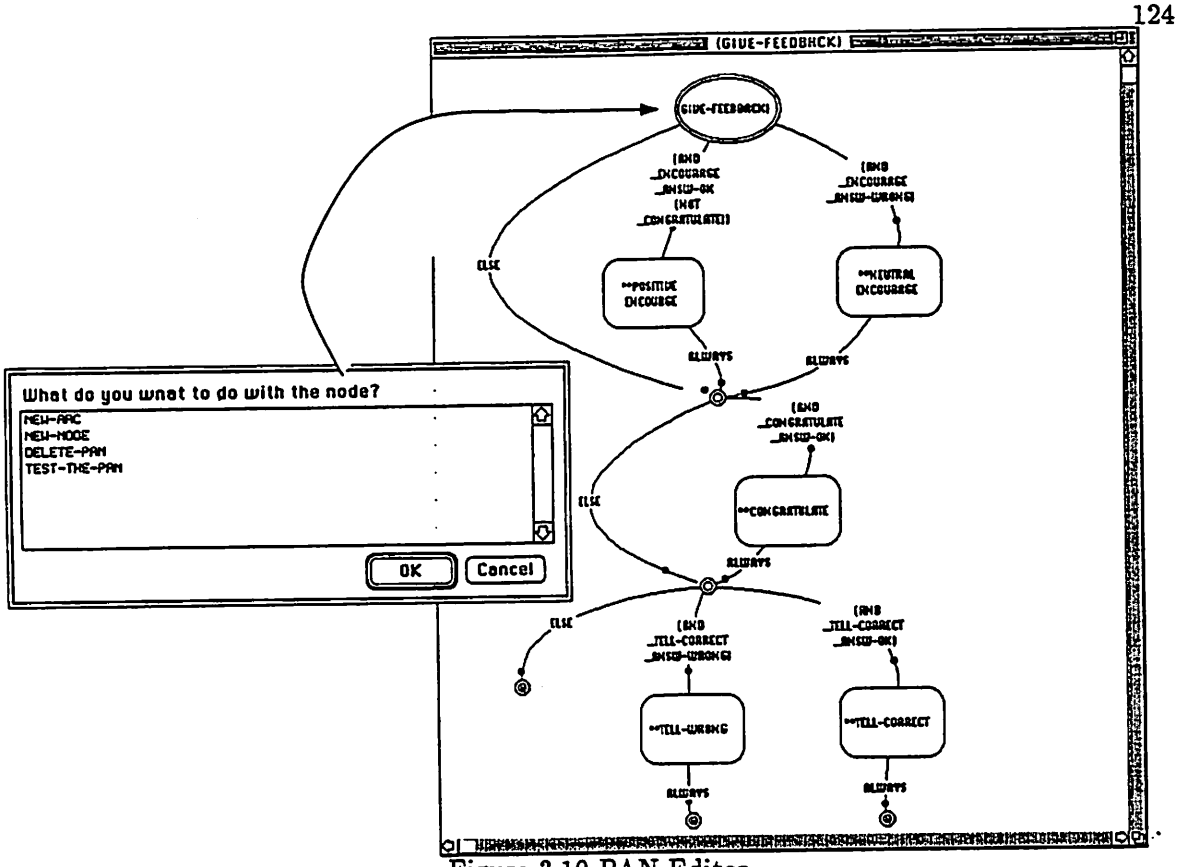


Figure 3.10 PAN Editor

nodes and arcs is done by clicking (or double-clicking) on a node or arc. The figure shows a menu of operations shown when the user clicks on a node.

Creating or editing a switch register involves clicking buttons to toggle switches on and off. The editor for switch registers is shown in Figure 3.11.

3.2.3 Session Monitoring Tools

The system runs on two physical monitors (screens): the "student screen" (a color monitor) shows either the tutorial session (Figure 1.5) or the Browser (Figure 3.8); the "knowledge engineering screen" (a large black and white monitor) is used for development and knowledge engineering and shows the monitoring tools. The system has four monitoring tools. Three of them are shown in Figure 3.12. The monitoring tools are invaluable in helping the teacher understand the progress of a tutorial session in terms of the curriculum, the flow of control through tutoring strategies, and the student model. These tools also help

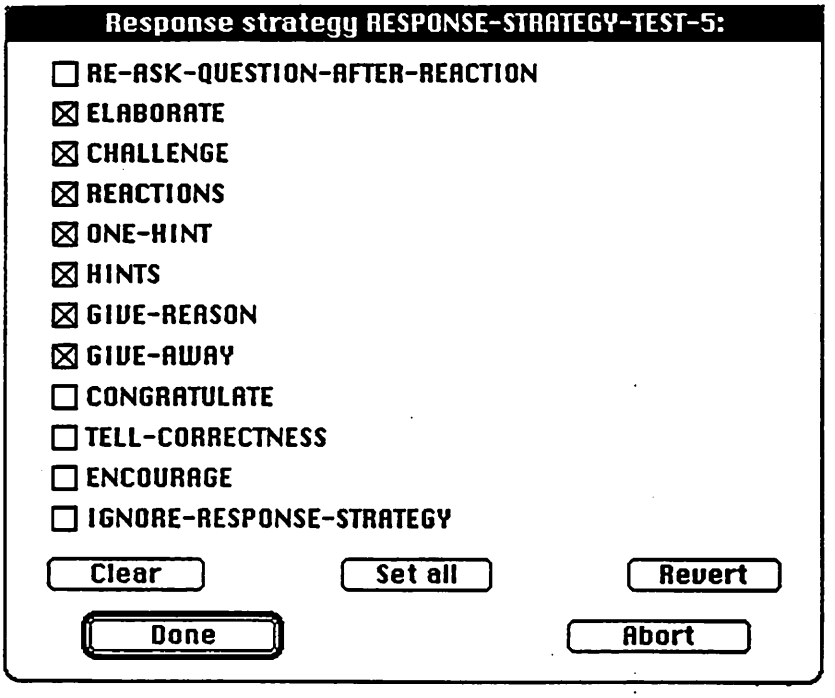


Figure 3.11 The Strategy Switch Editor

teach and reinforce the syntax and semantics of the KAFITS representational framework by providing visual models of concepts and structures of the framework. Each is described below.

Topic net display. The first monitoring tool is a graphic display of the topic network (see Figures 1.3, and 3.12), in which topics are highlighted to follow the tutor's traversal of the curriculum during a trial tutorial session.[19] The user can also click on the topic nodes to edit or inspect topics.

The size, shape, and scale of the topic net window can be adjusted, allowing the user to zoom in or back to see specific areas of the curriculum.

Event Log. The second tool is an Event Log which gives a detailed trace of the decisions and inferences made by the tutor, including the evaluation of each student response. This trace is written to a text file to allow analysis of real or dry-run tutorial sessions.

[19] We have considered giving the student access to the visual topic network, but have not tried it yet.

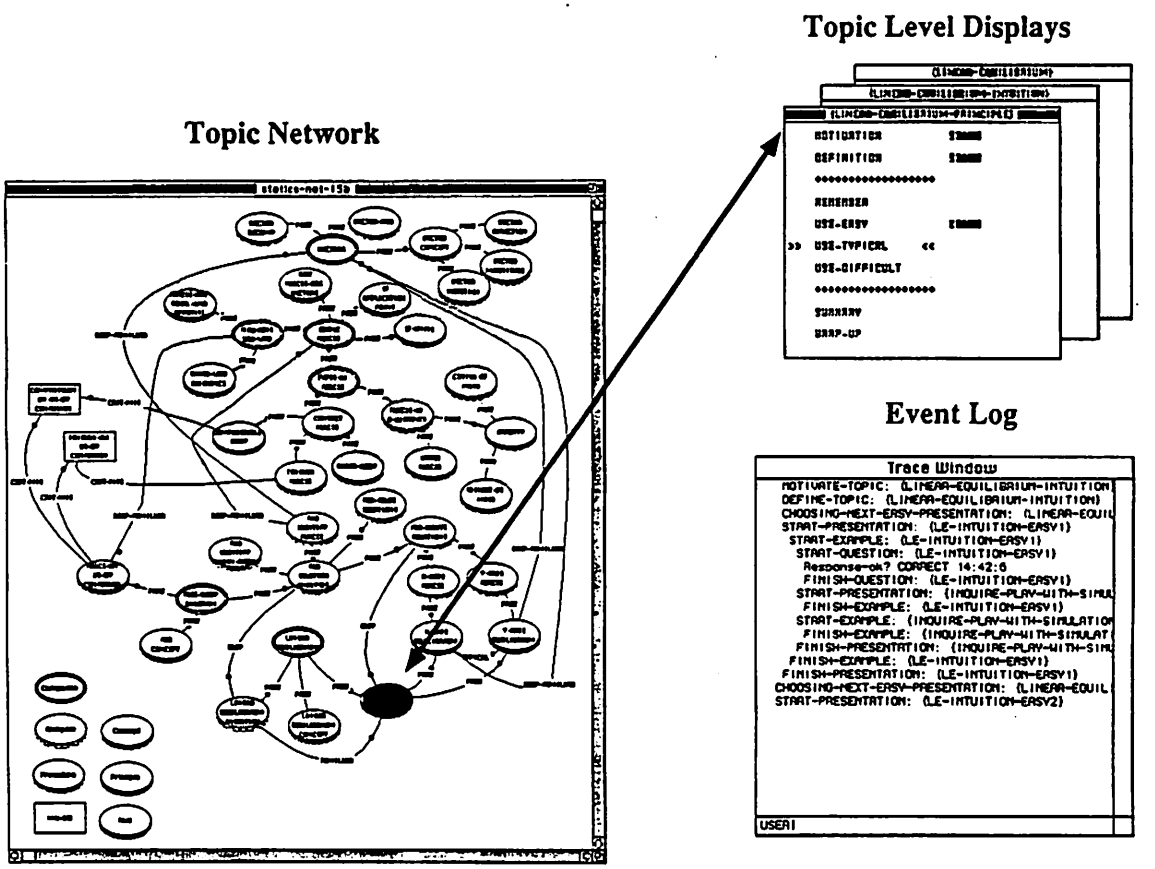


Figure 3.12 Monitoring Tools

PAN monitors. The third tool is a graphical representation of the PANs, which trace the flow of control through the action networks invoked by highlighting PAN nodes (see left side of Figure 3.7).[20] The domain expert can see control flow from one PAN to another and from action to action within a PAN. This visual tracing facilitates testing and modification of the strategic knowledge base, and provides the domain expert with a concrete visualization of what can, at times, be a complex control structure.

Topic Level Display. The fourth monitoring tool is the Topic Level Display, which details the current topic and pending topics (see Figure 3.13). Pending topics are stacked underneath the current one. For example, three Topic Level Displays are shown, with Linear-equilibrium-principle being the current topic. That topic is being shown to the student to satisfy a "familiarity" level prerequisite of Linear-equilibrium-intuition (the middle Display), which was in turn presented as a "part" of Linear-equilibrium (the bottom Display). The text on the left side of each Topic Level Display shows the levels at which the topic can be taught, with an arrow indicating the current level being presented to the student. On the right are the Student Model values of each level (the student model is described in Section 3.3).

An important feature of the knowledge acquisition interface is that it is easy for the domain expert to move back and forth between testing the curriculum (running it as a student) and modifying it with the Browser. The teacher can interrupt the session at any time to display information about objects, display the status of the student model, change strategies, or edit objects. He then returns to the tutorial session were he left off.

3.2.4 Student Interaction

The interface between the tutor and the student is a critical component of any computer tutor, but it is not being evaluated as part of this study. The character of the student interface and the types of computer/student interactions allowed depend critically on the

[20]The PAN monitor is not shown in Figure 3.12, and when used it takes the place of the topic net on the knowledge engineering monitor.

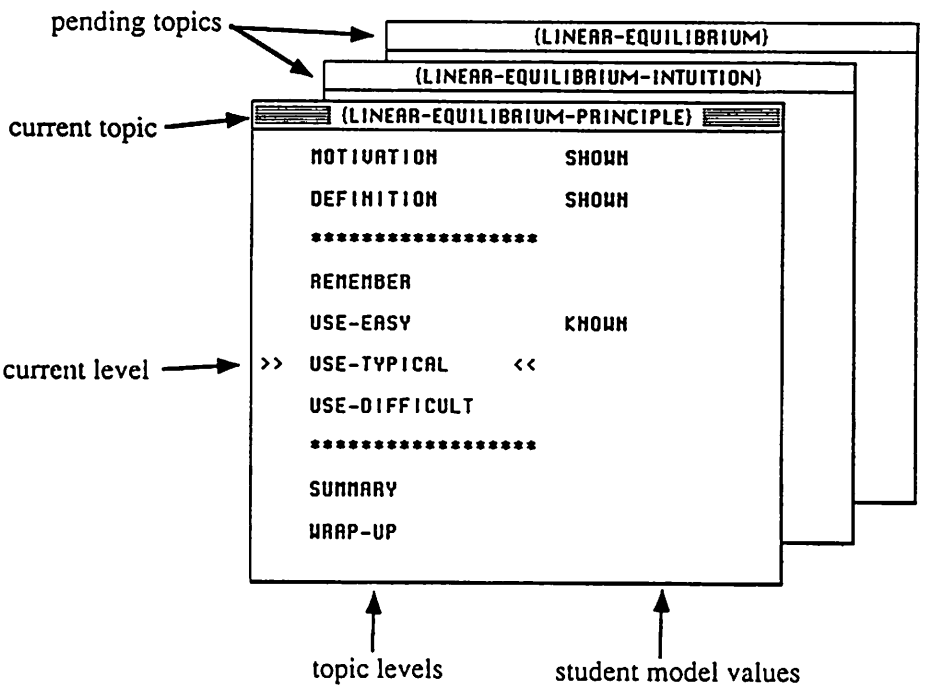


Figure 3.13 Topic Level Display

instructional domain (i.e. are not domain-independent). As mentioned previously, presentation objects are used to define interactions with the student. The are composed of an environment (or task-situation) and a task (or question) for the student to accomplish (or answer) within that environment. The standard media which KAFITS provides are picture objects, sound objects, and text. In the statics domain, the crane boom simulation and static crane boom pictures were added. KAFITS comes with three standard types of student interactions, and is designed to make it easy to add others.

Below we describe the student interaction types and the crane boom simulation (the learning environment for the statics domain) in particular. We also describe the "student initiative" capability, which allows the student to control the tutoring session.

Interaction types. Student behavior is processed according to the type of interaction. Currently there are three types of domain-independent interactions provided: multiple choice, type-in, and numeric; and three interaction types defined for the statics domain, crane-boom-value, crane-boom-point, and crane-boom-vector. Crane-boom-value interac-

tions ask the student to manipulate the simulation so that one or more of the 30 parameters is in a specific range. Crane-boom-point interactions ask the student to point to (click on) a place on the crane boom, and crane boom vector interactions ask the student to draw (click and drag) a vector on the diagram. For each type of interaction two functions are defined: one to set up the environment and record the student's response or behavior, and the other to process (evaluate) the student's response or behavior.

New interaction types are easily incorporated into the tutor by defining new get-student-response and process-student-response functions. Each type of interaction defines its own language for specifying the possible student behaviors. The teacher uses this language to fill in the Possible-answers slot of presentations. For example: for multiple choice the teacher needs only enter the text of the choice interactions, such as "high," "medium," and "low." For crane-boom-vector interactions, the size (in Newtons), angle, and location is specified (for example: SIZE 8-12 ANGLE 40-50 START left-end-of-beam). It is straightforward to implement new interaction types, such as parse-sentence, or keyword-match, to the KAFITS system (see Section 3.6).

The interactive simulation. The crane boom simulation can be shown non-interactively, as in Figure 1.4, or it can be brought up in interactive mode for student experimentation. Also, the student can call up the simulation at any time to freely explore in it, to manipulate the configuration of the boom, cable, and weight to measure forces, angles, and distances. Meters display any combination of 30 variables (Figure 1.4 shows two variables displayed) and as many as 18 different force vectors and vector components can be made visible (the figure shows one vector). The instructor determines which labels, meters, force vectors, etc. will be displayed for a given tutorial presentation. In free exploration mode the student can make these choices.

Student initiative capability. Giving the student control in a computer tutoring session is very important. Students should be able to choose the content and style of the information and tasks presented to them, and should be able to explore and inquire freely. In KAFITS the student can interrupt the tutorial session at any time and choose among

many commands. For instance, Figure 3.14 shows the pull-down menu options that are provided when the student is asked to answer a multiple choice question, and instead clicks the button to interrupt the tutor to execute a command. The student can ask for hints and explanations, and interrupt the presentation to visit another part of the curriculum. As well as having control over the course of a tutorial session, the student can get information about what the tutor is doing, and can inspect some aspects of the student model and the teaching strategies.[21]

Determining which student options are most important, how to encourage students to exercise these options, and the effect that numerous student-initiated options have on student learning are important issues outside the scope of this research. The options we have implemented are meant to increase the flexibility of the tutor and suggest the range of possible student initiatives, but we have not conducted empirical tests on this aspect of the tutor.

3.3 Layered Overlay Student Model

3.3.1 Purposes for the Student Model

One of the most important features of intelligent tutors is their ability to tailor instruction individually for each student. The *student model* is the dynamically updated knowledge base that records the tutor's inferences about the student's current state of knowledge.[22] VanLehn [1988] lists four common uses for student modeling: measuring the level of mastery to enable advancement, offering unsolicited advice when the student needs it, generating problems at the right level of difficulty, and adapting explanations to what the student already knows. Our design philosophy has been to postpone implementation of features until

[21] This is consistent with the concept of "collaborative tutors," as discussed in Section 6.3.4.

[22] Learning styles and preferences, historical data about the student, etc. are also stored in some student models. It is also possible to have separate dynamic knowledge bases for the student model and "discourse model" which keeps track of the current and past states of student-tutor interactions (a separate discourse model was included in early implementations of the KAFITS system).

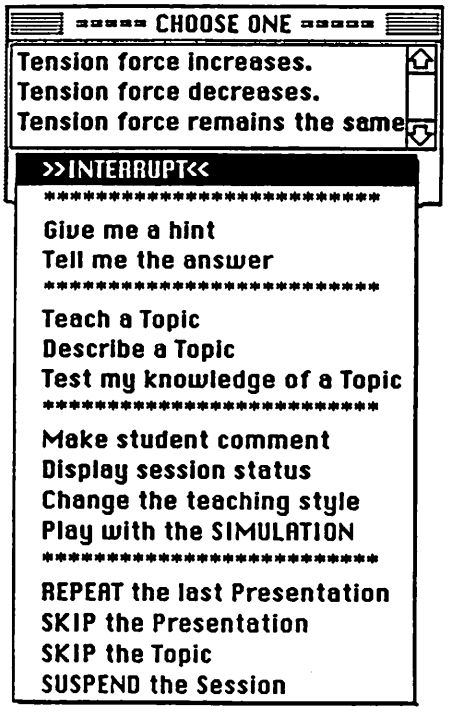


Figure 3.14 The Student Initiative Menu

a need has been established through working with the domain expert, by student trials, or from information needed by some other part of the tutoring system (see Section 5.5.3). Our tutor has minimal functionality in many respects, and the focus of the software development has been on the knowledge acquisition interface. Therefore VanLehn's list is useful to us in allowing for future extensions to the system, but is not useful in deciding how to design our student model or diagnosis mechanism. The KAFITS student model was designed, not from first principles about how to model or communicate knowledge, but *in response to the structure of the representational framework*

In designing our student model and diagnostic mechanism we used a scheme for considering the **purposes for student models** that is based more on practical than theoretical concerns. Below we list these purposes in order from most essential to least essential:

1. **Record keeping.** Minimally, the student model keeps records that prevent instructional material from being given repeatedly (unless the repetition is intentional). It should be able to answer the question "Has X been given to the student yet?"

2. **Appropriate information level.** Student models usually are able to answer the question "Does the student *know* X?" ITSs use this information to give instruction that is neither too boring (easy) nor too complex (difficult).

3. **Remediation.** Some student models can also diagnose common bugs or misconceptions in student knowledge. The purpose of diagnosis is to enable remediation of the bug or misconception or to postpone teaching something that a misconception impedes the learning of.

4. **Parameterizing tutorial behavior.** Finally the information in a student model can be used to alter the way a tutor presents information. This corresponds to VanLehn's "generating problems at the right level of difficulty" and "adapting explanations to what the student already knows."

KAFITS address each of the above student model purposes. The record keeping function of the student model is particularly important for the KAFITS framework because of the curriculum flexibility KAFITS supports. For instance, in a knowledge base editing session the domain expert had inadvertently set a system switch to ignore the student model. Due to circular paths in the topic network he was given the same topic four times (each time fulfilling a different part or prerequisite role). If the student model had been activated he would have been given the topic only once (assuming he answered its component questions correctly). KAFITS models whether the student "knows" each topic (item 2 above), and also diagnoses and remedies misconceptions (item 3 above). Finally, KAFITS parameterizes tutoring behavior according to the student model (item 4) by incorporating multiple strategies which can be selected according to the student model.

3.3.2 Representation Issues

At the implementation level (putting aside issues of cognition and pedagogy for the moment) there are two main issues in student modeling. The first is *representation*, i.e. what data structures should be used to model the student's knowledge state, and the second is *diagnosis*, i.e. what procedures should be used to infer the student's knowledge state. Clearly, representation and diagnosis issues are closely linked. First we discuss representation.

All student models have a data structure representing each chunk of knowledge that the student is supposed to learn, or, equivalently, knowledge chunks representing a subset of the knowledge of experts in the domain. The KAFITS student model maintains its own separate code space (similar to a blackboard), containing data structures corresponding to each topic (and presentation, as explained later) given to the student.[23] Many student models also have representations for incorrect, buggy, or sub-optimal (i.e. usable but inefficient) knowledge. The KAFITS student model uses **Mis-KUs to represent incorrect facts, misconceptions, and buggy skills** (there is no mechanism for accounting for sub-optimal knowledge).

There are two classes of student models found in intelligent tutors: overlay student models and runable student models. Runable student models are rule-based systems (like expert systems, but the term "novice system" might be more appropriate) that can be interpreted (run) to produce behavior. Runable models can be run to predict a student's response to a task, and they can be run to compare the student model with a model of an expert's knowledge (as encoded in a domain expert system). Runable models are usually used for tutors that emphasize procedural skills, since it is procedures that can be represented in production rules. In contrast, overlay student models are not runable, they declaratively record the student's level of understanding for each knowledge chunk. Overlay models are usually used in tutors that emphasize non-procedural knowledge. The KAFITS framework assumes a curricular representation of the domain knowledge rather than an expert system

[23] Student model data structures are not created until they are needed. This limits the size of the data base and eliminates the need to initialize the student model at the start of a tutorial session by creating empty data structures for all of the relevant instances in the knowledge base.

representation of domain knowledge. Though it contains a "procedure" knowledge type it is geared more toward teaching non-procedural knowledge such as concepts, principles, facts, and (shallowly represented) complex knowledge (see Section 3.1.6)—therefor **an overlay student model is employed.**[24]

As in many ITSs, KAFITS uses an overlay student model with a bug library. However, instead of assigning a simple symbolic or numeric value for each topic or bug, as in traditional overlay models, the KAFITS student model maintains a separate global code space (similar to a blackboard). Though our student model was designed more from pragmatics than theoretical concerns, is does have several innovative features not found in traditional overlay student models, as described below.

Multi-layered inferencing. Traditional overlay student models record a value (usually true or false), indicating whether the student is assumed to "know" or have learned each topic or skill. Our student model makes inferences at several levels of granularity allowing more precision and expressiveness than found in traditional overlay models (see Figure 3.15). There are five layers of data and/or inferencing: lesson, topic, topic level (corresponding to the performance/mastery levels of the topic), presentation, and transaction (individual student and tutor actions). The values derived at each layer come from a set of symbolic values, and these values are unique to that layer (the layer's range, shown to the right of each layer in the figure). Each value is determined as a function of the values at the next lower layer (the layer's domain). For example, the student model value of a presentation is one of: :shown-only, :correct-with-many-hints, :correct, :wrong-with-answer-given, or :wrong-no-answer-given.[25] The student model value of a topic level is a function of the values of its component Presentations.

Reasoning with uncertainty. The inference rules and language used to express knowledge in the student model explicitly represent and reason with uncertainty [Cohen &

[24] We think that KAFITS could also be useful as a tool for creating, testing, and modifying the production rules of a procedural-skill oriented tutor, but have not attempted this yet.

[25] The set of possible values for each layer were designed to be expressive enough to support the inferencing required, yet not too complicated or more numerous than necessary. We are still determining an optimal vocabulary for the values of each layer.

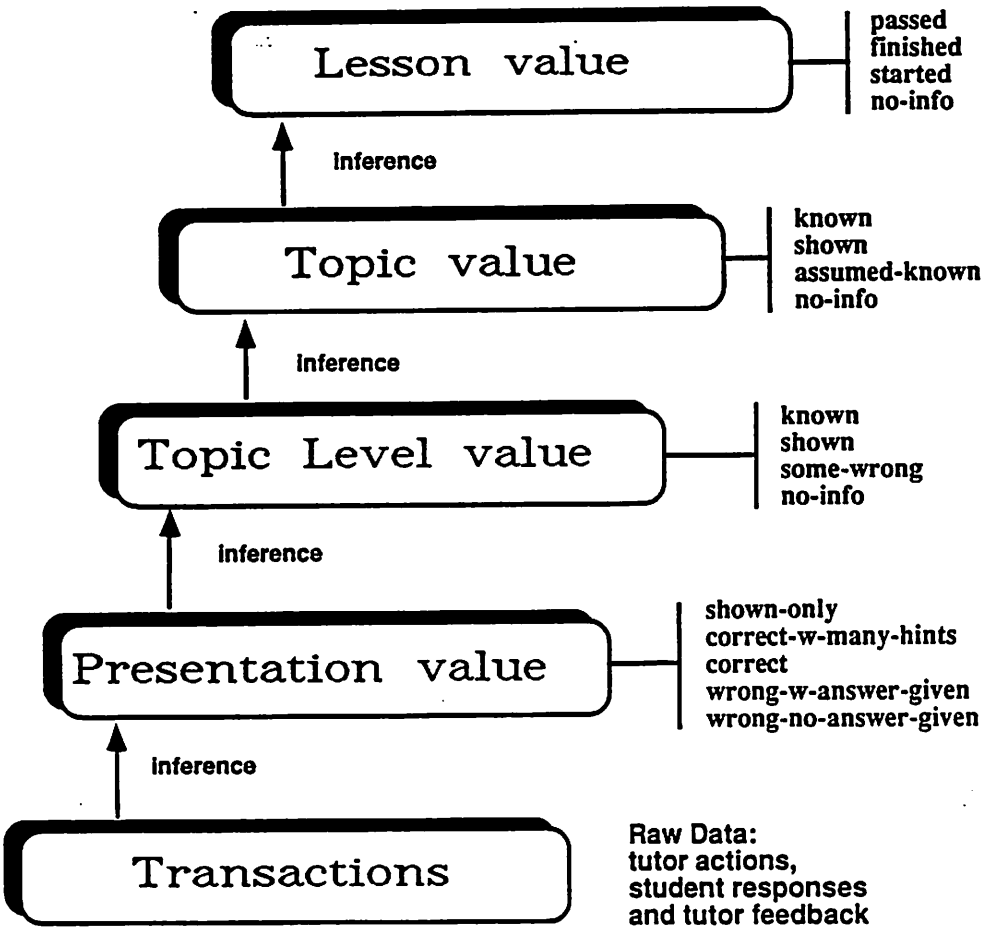


Figure 3.15 Layered Student Model

Gruber 1985]. For example, we incorporate the terms "no-info," "shown," and "assumed-known" into the representational language. "Shown" is used to indicate that a topic or presentation (or a part of one) was presented to the student and that there is not enough evidence has accumulated to determine whether the student knows or does not know it. The topic value "assumed" allows the domain expert to initialize the student model with some topics assumed to be known unless the system accumulates evidence to the contrary. The "suspected" Mis-KU value also represents student model uncertainty, as compared with the "confirmed" Mis-KU value.

Nonmonotonicity allowed. Nonmonotonic inferencing involves making inferences where assumptions are made which may have to be abandoned in the light of new information. Nonmonotonic inferencing is needed in KAFITS for two reasons. First, the teacher can initialize the student model so that the student is assumed to know certain topics, but these initial values change if the system accumulates contradictory evidence of the student's knowledge of a topic. The second reason stems from the flexibility of the curriculum representation. When the tutor tries to teach a topic it infers the level of the student's understanding of the topic based on how well she did on the topic's component presentations. But these presentations may be given again for other reasons, such as for teaching another topic, or as the result of a student initiative. Therefore evidence of the student's understanding of a topic can change even though that topic is not currently being explicitly taught. We have a straightforward method for dealing with nonmonotonicity: only raw data, not inferences, are stored (i.e. inferences are not cached), so that inferences always refer to the most recent student behavior.[26] Raw data is stored in the transaction layer, and includes only information with no uncertainty, such as the student responses to answers, how many hints were given, etc. By allowing inferences to flow from the raw data up the data layers every time a value is used, the value of a topic may change over time even though that topic may not have been visited. Recalculating values every time they are accessed has not affected the response time of the tutor perceptibly.

[26] We do not include a truth maintenance system in KAFITS, therefore while the system deals with nonmonotonicity, it does not do "nonmonotonic reasoning" in the traditional sense; we call this type of capability "cheap nonmonotonic reasoning," or "on-demand inferencing."

3.3.3 Diagnosis and Remediation

"Diagnosis" in KAFITS is data driven (bottom up). Since we are not trying to match combinations of student behaviors to expert behaviors, as is done in some intelligent tutors, combinatorial search is not needed in our diagnostic mechanism. Lists of suspected-mis-kus and confirmed-mis-kus are maintained and checked periodically to activate remediation scripts (in accordance with the current tutoring strategies).[27]

The functions (or rules) that infer values for each student model data layer from raw data are "global" methods for determining student model values. KAFITS also provides "local" means for setting student model values. The Remediation-info presentation slot is used to add evidence that a topic is known or misunderstood based on a single student answer. This slot is used to locally add evidence that a Mis-KU is suspected or confirmed based on specific answers. In the statics tutor there is no global method for updating Mis-KU values, but in general we assume diagnostic functions exist which look at raw data and student behavior to determine whether mis-knowledge exists.

3.3.4 Other Student Model Features

Inspecting the Student Model. The domain expert can inspect the information in the student model in several ways. A pull-down menu has options for "full" or "brief" summaries of the values of all topic and Mis-KU data structures in the student model. The full summary also shows the values for the levels within the topics and Mis-KUs. A Browser operation is provided to view the student model data structure for any topic, Mis-KU, or presentation instance. Also, the topic level monitors (Section 3.2.3) show the values of the levels of the current topic during a tutorial session.

Student Profiles and Saving the Student Model. The student model can be saved and loaded. This allows the student (or teacher testing the system) to quit in the

[27] We do not rule out the possibility of top-down model driven forms of diagnosing topic and Mis-KU values should KAFITS be used for a domain which includes a sophisticated micro-world environment with coach-like tutoring strategies.

middle of a session and continue at another time. It also allows the teacher to create a library of prototypical student profiles that can be loaded to initialize the student model (for example, one for students who have had previous physics experience, one for students with previous science experience, and one for students with little science or math experience).

3.4 Knowledge and Data Management Features

In this section we describe the methods and tools used to automatically record, organize, and manage knowledge and information in the KAFITS system. The information is stored in text files, some of which also contain Lisp source code. First we describe the files used to manage the domain knowledge base and the files used to store data collected while KAFITS is running.

3.4.1 Managing Domain Knowledge

Two files are needed to manage the domain knowledge base: the saved-instances file, which is a record of the contents of the entire knowledge base, and the edit-record, which records changes made to the knowledge base.

Saved-instances file. The domain knowledge base, containing all of the instances of the objects in the system (topics, presentations, lessons, etc.) is written to a text file called the saved-instances file. since the domain knowledge is included in the code "image" of the tutor (see Section 3.6) the saved-instances file is not needed to run the tutor, but it is useful to have available on disk for reasons described below. The saved-instances file has two purposes. First, it is organized to be readable by humans, allowing the domain expert

to reference information when it is not desirable to do this using the browser.[28] Second, the instances saved in the file are in the form of Lisp code (macros), for example:

```
(NEW-TOPIC        :NAME GRAVITY
                  :TOPIC-TYPE CONCEPT
                  :PREREQUISITES (FORCE-DEFINITION)
                  :PARTS (WEIGHT-VS-MASS CENTER-OF-MASS)
                  :SUMMARY ''Gravity is a force which pulls things...''
                  :MOTIVATION ''Gravity is a very important force that..''
                  :USE-EASY (GRAVITY-TWO-PEOPLE GRAVITY-PERSON-BLDG)
.....)
```

The knowledge base is created by loading the saved-instances file after the KAFITS system is loaded. A new or modified knowledge base is saved by writing a new saved-instances file to permanent memory (the Mac's hard disk).

Appendix I shows a portion of a saved-instances file. First each lesson is listed in alphabetical order. Then each topic is listed in alphabetical order. After each topic the presentations it references are listed in alphabetical order. After each presentation the instances it references are listed (crane-booms, pictures, sounds, other presentations, etc.). The resulting listing is organized hierarchically, with miscellaneous instances nested inside presentations, and presentations nested inside topics.[29] Instances that are referenced twice are not listed twice—a comment "see listing above" is written in its place. All instances that were not referenced by topics or presentations (i.e. not accounted for above) are listed alphabetically in a separate section at the end of the file.

In addition to the Lisp-readable representations of all the instances in the knowledge base, the saved-instances file contains several text sections designed to help the domain

[28]For example, the domain expert can take a hard copy of this file home and proofread it, and later use the browser to make corrections. As another example of using a saved-instance hard copy, assume the domain expert wanted to replace all occurrences of the word "friction" in the knowledge base with "frictional force." Using the browser it would be tedious to replace all occurrences of "friction," and even more tedious to first *find* them all. A much simpler alternative is to bring the saved-instances file into a text editor and replace all "friction" with "frictional force" with one editor command. Note that these changes would not be incorporated into the system until the next code image was created.

[29]Topics are not nested inside lessons—all lessons are listed at the beginning of the file.

expert inspect the knowledge base. First is a table of contents which indexes all the topics and Mis-KUs. At the end is a cross reference of all presentations, showing the instances in the knowledge base that refer to each presentation.[30] This feature has been useful in checking for inconsistencies and other errors in the knowledge base. For example, a presentation that is not referenced by anything is likely to have a typo in its name. Following the cross reference is a list of the instances that were referenced but were not created. This provides another way to check the consistency of the knowledge base. Unless the domain expert is in the middle of entering domain knowledge in the knowledge base, all instances that are referenced should exist. At the end of the saved instances file the total numbers of topics, Mis-KUs, presentations, and instances (of all types) are given.

Edit records. As the domain expert uses the Browser to add to or modify the knowledge base, the changes are written to a text file called the edit record. The edit record serves two purposes. First, the changes being made exist only in the computer's temporary memory, and are lost when the KAFITS program is exited, so the edit record is a permanent record of these changes. The next time the system (i.e. the code image) is loaded the knowledge base reflects the state of the saved-instance file when the code image was *created.* Every time the user starts up KAFITS, he must load his edit record to re-instantiate the changes he has recently made.[31] The edit record, like the saved-instance file, contains Lisp code (in a form that can be loaded by Lisp and also read by humans) and various non-code comments. Appendix K has a sample edit record.

The second purpose for the edit record is to allow the knowledge engineer to trace the evolution of the knowledge base and the use of the Browser through time. One can analyze edit records to get information about the use profile of the Browser and the types of modifications made (time, date, and user's name are recorded). Also, the user can insert a typed-in comment into the edit record. For instance, if the domain expert is making a

[30]The hierarchical structure and table of contents makes it easy to locate any topic and to see the presentations used for that topic. However, one cannot easily locate a presentation in the file unless one knows the name of the topic(s) it is associated with. The cross reference facilitates lookup by presentation name.

[31]When the Browser is started up, the user is asked to select his edit record file, which is then loaded automatically.

modification to the knowledge base and has a question for the knowledge engineer, but the knowledge engineer is not there, the domain expert can enter a comment such as "I'm not sure if this was right—does it have to be a positive number?" Later, the knowledge engineer can inspect the edit record to see what changes the domain expert has made and read the comments.

The edit record contains incremental changes to the knowledge base. Periodically, when a new version of the system is compiled, the edit record is loaded on top of the saved-instances file, and a new saved-instances file is created. When the code image is loaded it reflects the recent changes made by the domain expert. A new (blank) edit record file is started for the domain expert.

Edit records also allow multiple users and/or multiple editing purposes. For instance, several domain experts could be working on different parts of the curriculum at the same time, each with his/her own edit record. Also, an edit record can be created to record changes to the knowledge base that are for a specific purpose, but not intended to be permanent changes, such as modifications to the knowledge base for "AAAI-90 demo."

3.4.2 Data Records

The system maintains several files of data records—i.e. files that are updated while KAFITS is running. One, the edit record, was mentioned above. The others are described below.

Session trace file. The session trace file is a record of the tutor's decisions, the tutor's actions, and the student's actions in a tutorial session. The system has a feature which allows the student to interrupt a tutoring session and type in a comment, which is recorded in the session trace file. A sample session trace file is in Appendix L. The session trace file can be analyzed by the knowledge engineer or the domain expert to gather data about student use of the tutor.

Session listing file. There is one session trace file for each tutorial session. The session listing file lists of all of the tutorial sessions that have been run (i.e. all the session trace files created), along with the time, date, and name of the student.

Student model file. The student model can be saved at any time during a tutoring session (though it is usually saved at the end of one). For each saved student model a student model file is created, containing Lisp code which, when loaded, sets the student model to the state it was in when the model was saved. This allows a student to quit in the middle of a lesson and continue later where she left off. The saved student model can also be analyzed for data about the student's final knowledge state at the end of a tutoring session.

3.5 Help and Assistance Features

A software system that has many features and operations, as does KAFITS, has the potential to be confusing or overwhelming to the user. Such systems need features to assist the user in managing these options. Also, the KAFITS knowledge base is large and contains many types of things with diverse relationships between them—there is the potential for the user to get lost in the knowledge base. The user needs assistance in navigating though and gathering information about the knowledge base. The KAFITS system has several on-line help and assistance features, outlined below, to help the user use the system and navigate the knowledge base.

3.5.1 Assistance

On-line help/info system. A hierarchically structured help system has been implemented.[32] Included in the help/info text are explanations of some important KAFITS framework concepts, descriptions of the Browser and KAFITS menu operations, explana-

[32] A "hierarchical" structure is one organized to show various subsumed levels of granularity, for example: a book has parts, chapters, sections, subsections, paragraphs, etc.

tions of the various types of objects and data files, etc. The help/info text is loaded from a text file. This text file has a simple syntax for specifying the hierarchical relationships. It is relatively easy to add new information to the help/info system by editing a text file (no knowledge of Lisp programming is needed).

Browser message window. At the top of the Browser (Figure 3.9) is a small window called the Browser message window which indicates what the user just did and/or should do next while using the Browser. One purpose of these messages is to confirm the successful execution of operations that have no visual repercussion, for instance: "The topic {Newtons-law} has been deleted." Another purpose is to direct the user's attention to the appropriate location on the interface, for instance: "The slot Hints of the instance {Crane-boom-small} has been printed in the output window below."

Disaster avoidance. When the user performs an operation that has serious repercussions, such as deleting an instance, or exiting the tutor, the system asks if s/he is sure they want to do that.

Slot documentation. By selecting Slot-documentation from the slot operations pop-up menu (see Figure 3.9) the user gets the following information: the purpose/description of the slot; restrictions on the value of the slot; and example values of the slot. The domain expert can use this feature when he forgets what a slot is for or what kind of information is stored in a slot.

3.5.2 Knowledge Base Navigation and Information Features.

In a large knowledge base the user often has difficulty knowing "where he is" or what things are related to the thing he is looking at. KAFITS has several features to assist the user in this area.[33]

[33]In Section 3.2.3 we described the Monitoring Tools, which are designed to help the domain expert know were s/he is in the *dynamic* context of running or testing the curriculum and strategies. This section describes methods related to viewing the (static) knowledge base itself.

Cross references. All instances have a slot called Referenced-by, which lists the instances that point to the object. This slot is updated when a saved-instance file is created, when a code image is created, or when the user selects the "calculate cross references" menu operation.

Browser tables. The current type, instance, and slot are highlighted in the three tables on the browser panel (see Figure 3.9). This reminds users of the instance that contains a slot (if s/he is working on a slot), or the object type of an instance (if s/he is working on an instance).

The Browse operation. By selecting the Browse operation the user can see all of the objects connected to (i.e. those pointing to and from) a given object. The connected objects are shown in the instance table (i.e. they are not just listed for viewing—they can be selected and operated upon). This feature allows the user to move easily though the knowledge base via related instances. Another operation, called View-cross-references, shows (in the browser output window) the objects related to an instance, and describes the nature of these relationships.

Topic summaries. As described previously, the topic net gives a visual representation of the relations between topics. Nodes in the net can be clicked on to browse or get concise "summary information" about each topic.

Show-all-slots feature. A tool is provided which allows the user to list the contents of a single slot for all instances of a given type. For instance, if the user wants to know which presentations have more that one hint, he could use this tool to print the Hints slot of all presentations. This is much more efficient than viewing each presentation individually.

3.5.3 Consistency and Error Checking

ITS knowledge acquisition systems which guide the user, step by step, through the creation of a curriculum knowledge base can significantly constrain the content and structure of the knowledge base in their attempt to minimize errors. KAFITS is an open-ended

knowledge acquisition system that does not constrain the order information is entered, which allows for flexibility, but admits more possible errors. To reduce several types of errors we have implemented a small number of features to detect user errors, as described below.

Slot data type warnings. All the slots (of all object types) are defined to be of a specific data type, and the value of each slot is restricted to the required data type. Example data types are: text, a pair of words (symbols), a pointer to a topic, a list of presentations, an integer, etc. When the user creates a new instance or edits an instance slot values are checked to make sure they are of the correct data type. If an error is found, a warning message such as: "slot HINTS of instance {CAR-3} is: 102, but should be a text string; for example: 'hello there'."[34]

Topic net consistency checker. The topic net (Figure 1.3) and the domain knowledge base are not completely integrated. That is, when the user makes a change in the topic net (using the topic net editor) the change is not automatically reflected in the Browser, and vice versa. The user must run the topic-net-consistency-checker (by a menu selection) to make the topic net and the domain knowledge consistent. When this is done, the user is prompted to add or delete items from the net or knowledge base to enforce consistency.

Cross references for consistency. The cross reference and null-reference sections at the end of the saved-instance file (mentioned in Section 3.4.1) allow the user to locate instances that are never used and instances that are referenced but do not exist.

3.5.4 Other Interface Features

Select-screen-configuration. At the beginning of a session the user can specify the hardware screen configuration from a menu of pre-defined configurations, such as: color

[34]The user is not forced to correct these errors, since there may be rare cases when the user is intentionally using a value of the wrong data type. However, if the user types something in the edit window which defies Lisp syntax, such as having a missing or extra quotation mark ("like "this") he is given an appropriate message and is not allowed to exit the editor until this is fixed or the editing is aborted.

monitor, two page monitor, color monitor with two page to the right, etc. The positions of all windows, menus, graphics, etc. are adjusted accordingly.

User types. The KAFITS system incorporates three "user types:" student, teacher, and programmer. The user can change the user type via a menu operation (described in Preferences below). Teacher mode is the usual mode of operation—meant to be used by the domain expert and knowledge base managers. In student mode the user can not invoke the Browser or change the knowledge base. In programmer mode, used only by the knowledge engineer or Lisp programmer, error messages are more detailed and technical.[35] Proposed additions would extend the number of user types to have different modes for new vs. experienced users of the system, and different modes for domain experts vs. knowledge base managers.

Preferences. The user can customize certain aspects of the KAFITS software to his/her personal taste. The Preferences feature allows the user to set several options, including the user type, whether or not to have the session trace window visible, and the name of the default edit record. The setting for these options can be saved (in a "preferences file") and are automatically loaded the next time KAFITS is started.

Short cuts. There are many features that allow the user to take sort-cuts or combine operations, some of which are described below.

At the top of the object type table is an item called Recent-instances (which can not be seen in Figure 3.8 because it is scrolled off the top of the object type table). When this item is selected the instances table is loaded with a list of all the instances recently selected or operated upon (as opposed to the normal situation in which the instance table has a list of the instances of a certain object type). This circumvents selecting the instance from the long alphabetized list of items of a given type (the usual way of selecting instances).

The user can specify (via the Preferences feature) an often used operation to be executed when an item in a table is double-clicked. A different double-click operation can be selected

[35] There are other differences between the modes, which we do not described here.

for each of the three tables (type, instance, and slot). For example, the user can specify that when an instance in the instance table is double-clicked, it is browsed, and that when a slot in the slot table is double-clicked, it is edited.

Some of the most important KAFITS interface operations can be executed by a command keystroke, as well as via menu selection. For example, the command-B key starts the Browser.

3.6 Implementation

The KAFITS system is programmed in Allegro Common Lisp on a Macintosh computer. Standard Common Lisp is used except for interface functions (graphics, windows, menus etc. that are particular to the Mac II computer) and a knowledge representation language called KR. We designed KR to extend the functionality of the object language that comes with Allegro Common Lisp. Below we give an overview of implementation issues related to hardware, software, installation, portability, and extendibility.

The hardware platform. KAFITS runs on a MAC II[36] (or a Macintosh II family computer with higher functionality) with a hard disk and 8 megabytes of RAM memory. Two monitors are needed. A high resolution color monitor is used for the actual tutorial presentations and for the Browser. A two page high resolution black and white monitor is used to display the monitoring tools and the Strategy Editor. A less powerful hardware platform is needed to run only the tutor without the knowledge acquisition interface (which is all that is needed for a student or a teacher who wants to run a tutor designed by another person). The exact specifications of this simpler platform have not yet been determined.

Software needed to run KAFITS. The KAFITS code is released in a Lisp "code image" format. With each new version of the KAFITS code or new version of the domain knowledge base a code image (compiled version of the source code) is created. The code

[36] Apple Computers Inc.

image (which includes the crane boom simulation and the statics knowledge base) takes 1.4 megabytes of hard disk space. To run the tutor and knowledge acquisition interface one needs the code image, and two folders[37]—one containing startup files, and the other containing data files. The startup files are needed to initialize the system at load time. The data files are updated as KAFITS is being run. These files are described in more detail in Section 3.4. Applications software for creating and editing sounds and pictures, and storing these in "resource files," is needed if one wants to create sounds and pictures to include in the curriculum. To run the statics tutor one also needs the resource files for the pictures, sounds, and cursors it uses.

The network editor. A software tool for creating, displaying, and editing graphic networks of nodes and arcs was built. This general tool was used to implement both the topic net editor and the PAN editor.

Portability and the KR language. KR acts as an intermediate programming layer between the KAFITS system and the knowledge representation language provided by the Lisp software platform. The KR language includes features of both object oriented languages and AI frame-based languages. The basic functionality of objects, methods, and frames is incorporated from the Allegro Object-LISP package.[38] In designing KR, the Allegro Object-Lisp inheritance mechanism was modified, and many additional features were added, including object mixins, slot facets, "unknown" values, and type checking. All operations on the knowledge base are programmed using the KR language. Therefore the KAFITS implementation is independent of the knowledge representation language of the underlying Lisp software platform. This makes it easier to port the system to another Common Lisp based software environment (e.g. one based on CLOS). There is no need to rewrite

[37] "Folder" is Macintosh language for "directories" in other operating systems.

[38] The original implementation of KAFITS was on an HP-9000 series computer using the knowledge representation language HPRL. The KR language was written (and the KAFITS system rewritten to be based on KR rather than HPRL) when the KAFITS code was ported from the HP-9000 to the Mac II. Many features of HPRL which do not exist in the Allegro Object-LISP package are included in the KR language.

the KAFITS software to accommodate a new software environment or a new underlying knowledge representation language—only the KR language needs to be rewritten.[39]

Extendibility

When starting to build a tutor using KAFITS, some extensions or modifications to the representational framework are usually necessary. Most of these extensions require programming, so Lisp programming experience is assumed for those implementing the changes. The KAFITS code has been written so as to make most of these code modifications easy, but (until future software versions are written) some modifications are more cumbersome, as noted below. "Essential" aspects of the representation l system (as described in Section 6.3.2), such as the four-level decision model, are not meant to be altered. Below we summarize aspects of the representational framework (not the interfaces) that *can* be modified. Most domains require the addition of new object types or object mixins and, for the objects already provided, require slightly different slots and slots default values. All of these changes are quite easily made, and are automatically reflected in the domain knowledge base Browser.[40] Most domains also require a modification of the topic level scheme (see Section 3.1.5) which requires adding or changing the relevant topic slots.

Low-level programming is needed, to make new topic-types or topic levels apparent in the topic network monitor and the topic level display monitors.

Creating new tutoring strategies is fairly straightforward using the Strategy Editor (but the editor is still in early prototype form). The names of the strategy parameters (antecedents) and primitive tutorial actions (consequents) can be created "on the fly" while designing strategies, but the implementation of parameters and actions must be accomplished by straightforward Lisp programming of the required functions.

---

[39]However, in moving to another programming environment all the graphics-related software, including the net editor package, would have to be rewritten—no small task!

[40]The details of how to make these changes at the code level can be obtained from the author.

Each new domain has its own learning environment (or environments) (unless only pictures are used for learning "situations" and only multiple choice and numerical answers are used for student "tasks"). KAFITS has a flexible mechanism for incorporating new learning situations and task types. First the programmer creates a new KAFITS object type which contains for specifying the parameters for the environment (eg. the cable length and beam angle of the crane boom simulation).[41] Then the programmer must define a finite set of "task-types" (or answer-types) that describe canonical interactions between the student and learning environment (eg. make-point, make-vector, and change-value for the crane boom simulation). For each task-type two methods (Lisp functions) are defined: get-student-response, and process-student-response. Get-student-response invokes the environment (unless is is already active) in a configuration appropriate for the task type. When the student is finished with the task process-student-response determines the correctness (and other properties) of the student's behavior by checking the state of the environment.

New information is easily added to the on-line help/assistance system, as mentioned in Section 3.5.1. It is also easy to add new functions to the student initiative menu (Section 3.2.4).

The most difficult aspect of altering the KAFITS framework is modifying the student model and diagnostic rules, which are represented in Lisp "structures" and procedures. High-level functions are provided for asserting ("tell-dynamic-model") and inquiring ("ask-dynamic-model") student model values from other code modules. However, what the student model does with new information, and how it puts information together to respond to inquiries, must be programmed at the Lisp level. The student model "layers" described in Section 3.2.4 are clearly evident in the source code, but there is no general mechanism for modifying the structure.

[41] The learning environment must be built with programmatic "hooks" (Lisp functions) that allow the tutor to configure and invoke the environment and access any information about the state of the environment that is needed for tutoring or student diagnosis (eg. number-of-vectors-created).

Also, KAFITS does not have mechanisms (such as "demons") which continuously monitor students behavior as they perform tasks (possibly over the course of learning several topics)—these must be implemented in the learning environment.

CHAPTER 4

RESEARCH METHODOLOGY

In this chapter we outline our sixteen month study involving the construction of a tutor for statics by three educators, and describe our methods of collecting data. In Section 2.5.2 we described several research/evaluation paradigms, including formative evaluation, qualitative evaluation, and case study, and explain why all three of these paradigms were chosen for this study. Below we discuss how the choice of subjects, domain, and methodology could effect our ability to generalize the results of this study. Then we discuss our data collection methods and research study time line.[1]

4.1 Description of the Case

As discussed in Section 2.5.2, the case study method trades the advantages of large sample sizes and statistics-based conclusions for depth and diversity of analysis, which contrasts the traditional scientific methodology where experiments are designed to eliminate or factor out differences between samples. In case studies it is crucial that the characteristics of the "case" be described. In this section we describe in detail four components of the case under study: the subjects, the experimenter, the domain and the lack of a stable system, and discuss how each of these components affect our ability to generalize the results.

[1] Also, see Section 5.4.1 for a description of the design of the preliminary KAFITS system that existed at the beginning of the study.

4.1.1 Description of the Subjects

The primary evaluation method is a case study of a domain expert using the KAFITS system over sixteen months to design and test a tutor in statics. Two other subjects, whom we call "knowledge base managers," also participated. Our conclusions about the usability of the system and our identification of important issues in this research area are based on our experience with these three subjects, therefore it is important to describe these subjects and ascertain how closely they correspond to typical potential users of the system.

The main case study subject. Dr. Charles Camp, a physics teacher at the Amherst High School (Amherst, MA) was chosen as the primary subject of the case study. His significant experience in teaching physics, his past involvement in research on science misconceptions, and his proximity to the research laboratory where KAFITS was developed, made him an excellent candidate for this study. He has taught physics at the high school level for over 25 years. He was part of a research team at the University of Massachusetts Scientific Reasoning Research Institute which conducted a five year study identifying misconceptions in Newtonian mechanics and studied classroom-based methods for remediating these misconceptions.

Camp had had some experience with computer programming and some exposure to artificial intelligence concepts before we embarked on this project. He had taught introductory computer programming (APL and PASCAL languages) to high school students, and had no previous experience with Lisp or Macintosh computers. He attended a three week Teacher's Institute on Intelligent Tutoring Systems in the summer of 1988 held at UMass, which introduced basic AI and ITS concepts to classroom teachers organized into small groups to design story boards and screen layouts for hypothetical computer tutors.

First knowledge base manager. Frank Linton worked on the project as a knowledge base manager for approximately one month (totaling 70 hours) in the early phases of creating the domain knowledge base. He is a graduate student in education at UMass studying instructional applications of computers. He had worked as an industry consultant in instructional design for many years prior to becoming a graduate student. Linton had

little experience programming and a fair amount of experience using computers for word processing at the time he joined the project. He also had some familiarity with intelligent tutoring systems concepts.

Second knowledge base manager. Kim Gonzalez worked on the project as a knowledge base manager for 550 hours over a seven month period toward the end of this study. She is a UMass graduate student in education studying science education. Her undergraduate degree is in physics. She had had very little experience using computers before starting on this project. She had no experience with data bases, computer programming, AI, or ITSs.

The knowledge engineer. The author was the knowledge engineer during this study. Though the KAFITS system is designed to be used extensively by the domain expert, the role of the knowledge engineer is crucial in the ITS design process (as it is in the design of any AI expert system). The knowledge engineer must initially train the domain expert how to use the system[2] and must be on call to answer questions the domain expert has while using the system.[3] That the same person conducted the experiment, designed and built the knowledge acquisition system, and acted as the knowledge engineer in this case study limits the ability to generalize of the results. We have no reliable information at this time on the amount of training it would take, or the key issues that would arise, if KAFITS were used by another knowledge engineer.

The author has a bachelors degree in the instructional domain (physics) and has participated (in years past) in cognitive studies of misconceptions in the instructional domain [Murray et al. 1990]. The author attempted not to have any direct influence on the design of the curriculum content, letting the domain expert make all important decisions—however, we did have many conversations about the curriculum, and the author's experience in the domain probably had some effect. But this is not an unusual situation in knowledge engi-

[2]Our experience is that some of the concepts involved in using KAFITS are too difficult to be conveyed with simple written procedural instructions, such as one gets when buying a commercial word processing program.

[3]However, the domain expert did use the system successfully on about 30 days without help, including daily several weeks without the knowledge engineer being available.

neering; it is actually *necessary* that the knowledge engineer have some experience in the domain or learn the basic concepts and structure of the domain if he does not already have this knowledge. Therefore the author's previous experience is beneficial in terms of his ability to do knowledge engineering, but may be detrimental generalizing our experience to ITS knowledge engineering by an arbitrary knowledge engineer.

It is also worth mentioning that Camp and the author were acquainted prior to this project, which made the initial stages of the research more informal than would be the case for an arbitrary knowledge engineer and domain expert.

4.1.2 Discussion of the Prototypicality of the Subjects

Here we argue that the three subjects chosen form a reasonable basis for generalizing to other users of the system, and other non-programmers involved in ITS design. Designing an ITS in an instructional domain is *at least* as complex as writing a text book in that domain. Like designing a textbook, it is a major undertaking, and the product can have great impact and can be used by many people. It is not every teacher who produces a text book, but rather those who are more experienced and motivated than the average—i.e. exceptional teachers. Similarly, we can expect that only above-average or exceptional teachers are capable and motivated enough to participate in designing the intelligent computer tutors of the future. These tutors are designed and built by exceptional teachers, and *used* by other teachers (as are text books or sophisticated curriculum materials).

Also, we do not expect instructors to participate in ITS design without at least an introductory exposure to basic AI and ITS concepts. The undertaking is too complex to expect even exceptional teachers to start participating in ITS design "off the street." Therefore, we claim that Camp, as a highly experienced teacher having had some basic exposure to computers, AI, and ITS, though not a prototypical high school physics teacher, is a prototypical domain expert on an ITS design team (in Section 5.1 we discuss characteristics of a good domain expert, and the expected training time). We had the additional good fortune

that Camp was knowledgeable in cognitive studies of misconceptions in his domain—which is desirable, but not necessary, for the average ITS domain expert.

There are different purposes for using a system like KAFITS and different ways to participate in the design process other than being a domain expert. The *knowledge base manager's* function is to input the knowledge as specified by the domain expert,[4] and test the curriculum for obvious errors (i.e. errors not related to the domain content). Another potential user is the teacher who receives a KAFITS-based tutor and wants to make small modifications to it to suit the needs of his/her class. Still another potential user is the ITS evaluator who runs the ITS with students to ascertain its instructional effectiveness, and perhaps modifies the knowledge base to improve it or to test alternative curricula or strategies.

We argue that the two knowledge base managers who participated in this study are prototypical "low-end" users, i.e., they represent the minimal amount of training and experience for using an ITS knowledge acquisition interface. One of the knowledge base managers (Gonzalez) had very little experience with computers, yet was experienced in the domain (physics), but had no experience teaching the domain. The other knowledge base manager (Linton) had some experience in computers and ITS concepts, but was not a computer programmer, and had no experience in the domain. We further argue that between the three participants the spectrum of types of potential users is represented—at least well enough to make some tentative conclusions in this exploratory study of the ITS knowledge acquisition process.

4.1.3 Choosing the Domain

Below we discuss the choice of statics (or physics) as a domain and note characteristics of this domain that might affect generalizations about building tutors in other domains.

[4] in our study the domain expert originally specified the curriculum on paper worksheets.

Mechanics (sometimes called Newtonian mechanics) is the part of physics that deals with the relationship between the forces on objects and their movement though space. *Statics* is the part of mechanics that deals with systems of objects in a static arrangement—i.e. where no motion is involved. The statics tutor focuses on teaching a *qualitative*[5] understanding of key introductory statics concepts. Simple equations are included in the curriculum but solving equations to obtain numerical answers is not included.[6]

We chose this content area for several reasons. First, we wanted the curriculum to contain interactive simulation so that the student could be actively engaged in exploring concepts and testing hypothesis. We already had available to us a crane boom simulation designed to be used in conjunction with teaching statics. It was designed and built in a collaborative effort by researchers and programmers from UMass and San Francisco State University, as part of a project called Exploring System's Earth [Duckworth et al. 1987]. The second reason for choosing statics was the availability of the domain expert, a master physics teacher who has participated in cognitive studies of learning and teaching mechanics, including misconceptions in statics. Third was the author's previous experience doing research on computer-based method for remediating statics misconceptions [Murray et al. 1990].

Several studies of the United States educational system [U.S. Department of Education 1983, National Science Foundation, 1983] point to the importance of teaching science and mathematics subjects, and to the poor quality of students' understanding of these subjects. This also affected our choice of the statics domain (vs. non-technical domains).

Mechanics (and therefore statics) stands out among most other science and math domains in the amount of research effort aimed at understanding common misconceptions

[5]Dealing with causal and spatial relations rather than quantities and calculations.

[6]Simple force equations are used in the curriculum, but problems involving several equations or trigonometry are not included. Qualitative understanding would, in the larger curriculum of an entire physics course, lead to solving quantitative (equation-based) statics problems. Also, in the early design stages it was decided that the statics topics friction and torque would not be included and that a brief introduction to vectors would be included in the curriculum.

[Clement 1982]. This is only partially a plus for statics, however, since one reason so much research has been done is that mechanics is notoriously difficult to learn.

An important characteristic of instructional domains is the type of knowledge that must be mastered. Physics, and especially the area of physics addressed by the statics tutor, deals primarily with conceptual knowledge and physical principles (and less well-defined types of knowledge such as mental models and physical intuitions). Some other domains are primarily procedural, for instance how to answer the telephone, and how to do long division and teaching them often involves demonstrating skill application and practice with individual procedure steps. Still other domains are primarily factual in nature, such as geography and botany, and teaching them often relies on memorization and classification learning.[7] Of course most domains have a mixture of many types of knowledge, but usually they are taught so that certain types of knowledge are emphasized. The most effective computer tutor designs for each type of knowledge may differ considerably, so generalizations of this study are more relevant for domains involving conceptual or principle knowledge than procedure or factual knowledge.

Anderson [1988] compares intelligent tutors for procedural knowledge with those for "declarative" knowledge. Procedural knowledge, in Anderson's theory, is non-verbal knowledge used to perform skills. Declarative knowledge, which in Anderson's scheme includes concepts and principles, is consciously available to be analyzed and thought about.[8] In procedural tutors the knowledge to be learned is represented in production rules that can be run. Anderson thinks tutors for declarative knowledge are more difficult to design: "the major difficulty posed for [declarative knowledge] tutoring systems is that declarative knowledge cannot be run...so the criterion 'if the student can use it he knows it' does not apply..." In any case, intelligent tutors that focus on procedural knowledge have tradition-

[7] We refer here to the way these domains are usually presented in educational settings. It is conceivable, and perhaps desirable in some settings, to focus the instruction on other aspects, such as botany concepts or physics facts.

[8] The process of learning often involves first learning something at at the declarative level, such as tips for how to ski, and then assimilating it through practice at the procedural level, such as the unconscious (and difficult to articulate) information about skiing that a practiced skier has.

ally been designed quite differently than those focusing on declarative knowledge, so any generalization of this study to procedural knowledge tutors is limited.

To summarize, the domain chosen (statics) has been identified as relatively important by the educational community, and much research has investigated instructional techniques and common misconceptions for this domain. It has also been documented that statics is difficult to learn. The statics tutor focuses on conceptual knowledge, physical principles, and physical intuition, which are probably more difficult to teach than procedural or factual knowledge. Though it is easy to envision a KAFITS-based tutor that teaches facts, or relationships between facts, we are not certain how KAFITS would fare in teaching a domain consisting of primarily procedural knowledge.

4.1.4 Unstable Software Platform.

In accordance with the user participatory design process and formative nature of the evaluation, the system underwent many revisions during the course of the study. The participants in this study were able to suggest changes to the system and saw these suggestions manifest; this was a motivating factor for them. However, the unstable nature of the software made learning and using the system more complicated. On occasion a new release of the software (as often as every two or three weeks during some periods) would contain a programming bug. Usually when something went wrong the user's first reaction was to think s/he had done something wrong, and try to figure out how to mend it—leading to frustration if the problem was due to a programming bug. Also, it would sometimes seem that just as they got used to the way things were, the software would change. The changes were improvements, but it still took effort to learn or accommodate.

In this aspect the study does not represent a typical use of the KAFITS system to build a tutor. Future users of the system (assuming for the moment that the software remains stable) would be using a final version of the system (incorporating many changes that make it easier to use than previous versions) and they would not have to deal with the frustration continually changing software.

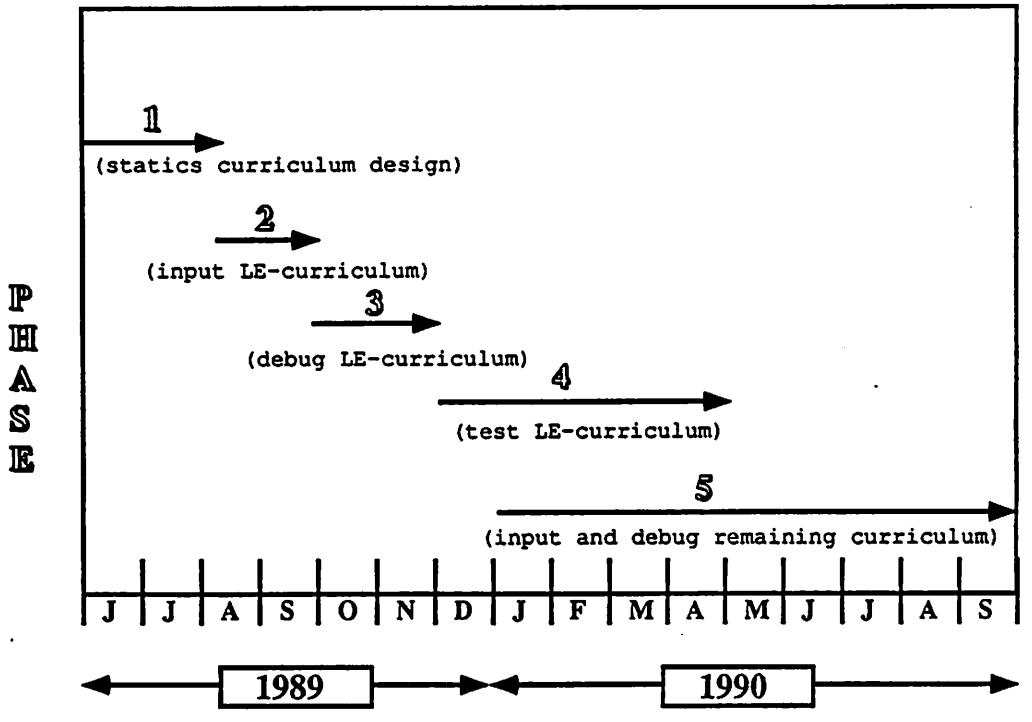


Figure 4.1 Study Time Line

4.2 Case Study Time Line

In this section we outline the progression of events in the case study. The study transpired over a period of 16 months from June 1989 through September 1990 (see Figure 4.1). We have divided the period in to five phases, which cover instructing the domain expert in KAFITS, curriculum design, implementation, and testing with students.[9]

Phase 1. Initial curriculum design—$2\frac{1}{2}$ months. *June to mid August 1989.* The domain expert worked solidly[10] with the knowledge engineer (and at times by himself) for $6\frac{1}{2}$ weeks at the beginning. This was the most intensive work period of the study. The knowledge engineer familiarized the domain expert with ITS concepts and the KAFITS framework (see Section 5.1.2 for a list of what a domain expert needs to know to use

[9] In Section 5.1.1 each phase is further refined and described.

[10] "Solid work" means three to five days a week for two to five hours a day. "Weekly" means once a week for two to five hours on that day. The domain expert also worked some hours at home, as described in Section 5.3.2.

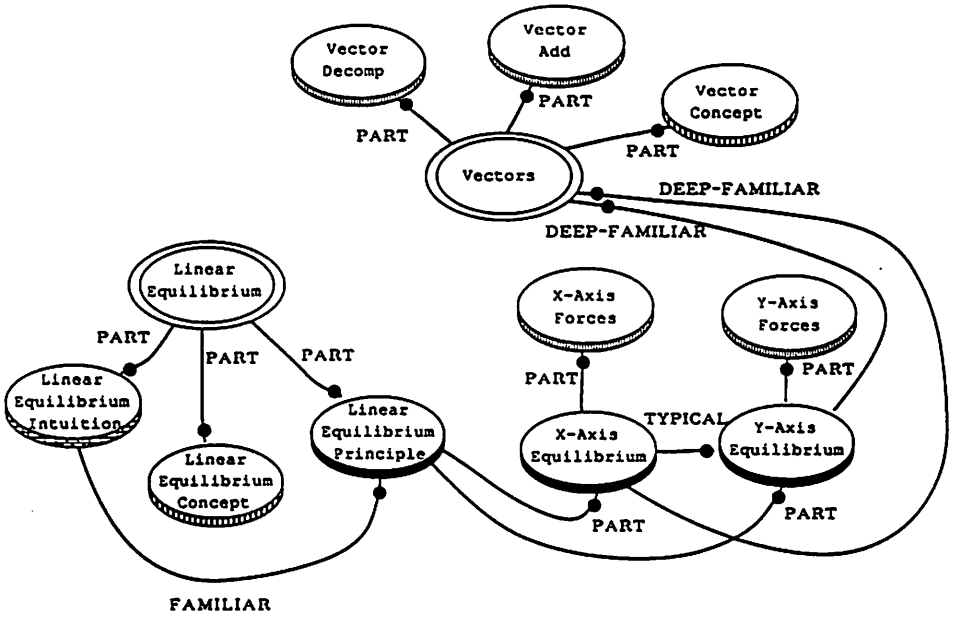


Figure 4.2 Linear Equilibrium Part of Curriculum

KAFITS). The curriculum was designed at the topic grain size (i.e the entire topic net was designed) and details of presentations for several of the topics were begun.

After Phase 1 the domain expert's participation was weekly until the end of the study, except for two weeks of solid work in Phase 5. The expert worked in the research lab on 60 days over the 13 month period spanning Phases 2, 3, 4, and 5.

Phase 2. Entering the Linear Equilibrium portion of the statics tutor—$1\frac{1}{2}$ months. *Mid August to late September 1989.* At the beginning of Phase 2 the first knowledge base manager (Linton) joined the research team. Our initial curriculum focus (indeed our focus for most of this study) was the 8 topics in the linear equilibrium portion of the topic network (called the "LE curriculum," see Figure 4.2). These topics formed a stand-alone curriculum unit that could be designed and tested independently of the rest of the statics curriculum. During Phase 2 Camp cogitated about the curriculum details in this area and filled out worksheets (see Appendix M) for the instances of topics, presentations, and crane booms, which he gave to Linton to enter into the knowledge base. Camp also drew sketches (on paper) of graphics to be shown with some of the presentations.

Linton first built the topic network, entering the information for the approximately 40 topics of the statics curriculum and tested the topics and topics links at a syntactic level. Next he entered the presentations for the LE curriculum as Camp created them. Linton also used a drawing program (Canvas (tm)) to create the graphics for 18 pictures for this part of the curriculum, according to Camp's sketches.[11] Linton's work with the research team ended at the end of Phase 2. During this time he spent 36 hours entering data, 34 hours testing the data, and 43 hours creating graphics.

Phase 3. Debugging the linear equilibrium curriculum—2 months. *October through November 1989.* At the start of Phase 3 the domain expert was trained in the use of the Browser. During Phase 3 he exercised the curriculum that was entered by Linton in Phase 2, and debugged and edited the knowledge base. He worked weekly over this period, a total of nine times. At the end of this period he thought that the LE curriculum was ready to be tested with students.

Phase 4. Testing the linear equilibrium curriculum—5 months *Early December 1989 through late April 1990.* There is a fair amount of temporal overlap of Phases 4 and 5 (see Figure 4.1). They are described separately because they are functionally different and involve independent tasks.

The linear equilibrium portion of the curriculum was tested four times during Phase 4.[12] Between each test modifications were made to the knowledge base and the KAFITS program according to what was observed.

The first test (Test #1) involved eight associates of the lab (students, faculty, and staff) who used the tutor and typed in their comments as they were using it. The testing took two weeks (December 7th to 21st). Test #1 was intended to determine whether the tutor was ready to be tested on high school students—either by bringing them into the lab, or by bringing a computer to the high school. After some minor modifications, it was determined

[11] Some presentations required the crane boom, and others required diagrams or pictures. The crane boom configurations were specified by Camp on paper work sheets.

[12] There were no other tests involving students during this study, therefore the LE curriculum was the only portion tested with subjects.

that the system was indeed ready, and it was decided that we would bring students into the lab.

The second, third, and fourth tests were administered by Gonzalez and involved volunteer high school students coming to the lab to use the tutor for one to two hour tutoring sessions, for which they were remunerated $10. All of the subjects were students in Camp's physics classes, who had covered in class the material which Camp thought was prerequisite for the statics tutor. In addition they had already been introduced to the main concepts of the statics tutor, including linear equilibrium, gravity, static forces, and Newton's Laws.[13] Students worked either alone or in pairs. The tutoring sessions were all at least partly supervised, with Gonzalez sitting next to the students for part or all of the session and taking notes. After each session the participants were asked these questions:[14]

1. What kind of physics experience have you had?
2. Did you find the use of the tutor easy? If not, which aspects were confusing or difficult?
3. Did you enjoy using it? Why or why not?
4. Did you learn anything by using the tutor? If so, what?
5. What additional features would you like to see included in it? Why?
6. (Where applicable:) Did you enjoy using the tutor with another student? How would it have been different if used alone?
7. Any other comments about the tutoring session?

The three tests were given as follows:

- Test #2: February 22nd, three students worked separately.

[13]Charlie intended the tutor to be used to reinforce and deepen existing knowledge in statics, not teach it for the first time.

[14]Some the the interviews were taped and notes were taken during others.

- Test #3: March 22nd, four students participated, two as a pair and two independently.
- Test #4: April 20th, four students participated, two as a pair and two independently. On this day, unlike the others, Camp observed the sessions.

Phase 5. Expanding the Knowledge Base—9 months. *Early January through August, 1990.* During Phase 5 the knowledge base was expanded to include all the topics in the topic network (Figure 1.3). All of the topics had been instantiated in Phase 1, but most were empty or contained only topic summaries or definitions (only topics in the LE curriculum were complete). During the first four months (January through April) Camp worked exclusively on designing the contents of the extended curriculum (including incorporating misconception objects (Mis-KUs) for the first time) and filling out work sheets which Gonzalez entered into the knowledge base. From early May through early August (three months beginning with the last student trail) both Gonzalez and Camp exercised the curriculum and made changes, often working side by side. Gonzalez finished her work with the project in early August and Camp worked alone using the Browser for the final two months of the study.[15]

4.3 Data Collection

The main form of data collection for this study was field notes. Ninety three pages of notes were taken, covering 55 sessions in which the knowledge engineer interacted with the domain expert. Notes were not taken for about thirty of the sessions, since on many days (especially toward the end of the study) Camp came in and continued his work independently. The journal notes include numerous diagrams and sketches, and about 40 short quotes by the domain expert. There were also several large newsprint sheets of brainstorming notes and diagrams generated by the domain expert and knowledge engineer from the early design stages of this study. A separate notebook was kept to record thoughts

[15] At the completion of the study the entire statics curriculum was still not ready for students, and Camp continued to work on it.

and information related to this study that was not associated with actual sessions with the domain expert, to keep data from knowledge engineering sessions separate from other notes. A daily record of all modifications made to the code was also kept. This was used to determine the dates when KAFITS features were added or when code bugs were fixed.

Highlights of the journal notes were transcribed into print in two ways: a chronological listing, and a categorization according to the issues addressed.

In addition two types of data files were analyzed. Edit record files were analyzed to obtain rough quantitative data on the time spent on various types of knowledge base modifications, and session record files (which include typed-in comments from the student) were analyzed to make conclusions about trial runs of the tutor.

Finally, a two-hour post-study interview was done with Camp. Though the interview was audio taped, we did not do a detailed analysis of a trascript of this interview for this study. However we list the questions asked and include many exerpts from the interview in Appendix C.

CHAPTER 5

RESULTS, ANALYSIS AND DISCUSSION

This research project was unique among ITS research in that we studied ITS knowledge engineering with educators and include empirical data about the design steps taken. Our intent was to identify and explore issues and to suggest potential solutions in a new area of study—ITS knowledge acquisition with educators. Therefore we use formative evaluation and case study research methodologies, which allowed us to accomplish two things: 1. develop a benchmark ("existence proof") of a workable ITS shell and knowledge acquisition method that can serve as a *base line* for further work; and 2. identify tradeoffs and problems encountered in using the computer system and knowledge acquisition method that can serve as *constraints* on (or guidelines for) future work. Our research methodology has allowed us to suggest an upper and lower bound for ITS designers, describing what *might* work (and in one case has) and cautioning about what might *not* work.[1]

Littman & Soloway [1988] discuss the importance of doing both "external" and "internal" evaluations of ITSs—we do both in this study.[2] External evaluations involve assessing the behavior of the system in relation to users, while internal evaluation analyzes the relationship between the system's architecture and its behavior. Unlike most external evaluations, ours will focus on the relationship between the system and the domain expert rather than the system and the student. We evaluate the power, usability, and efficiency of the KAFITS system based on computer work logs (edit records), comments from the domain expert, and field notes taken by the experimenter. We also discuss cognitive considerations

[1] This could also be framed as providing sufficiency and constraint conditions on ITS design.

[2] Though we concur with their advice on the general types of evaluation needed, we do not follow many of Littman & Soloway's specific suggestions, since they focus on modeling the student.

in the design of the system. Littman & Soloway describe internal evaluation as answering questions such as: "What does the system know?" (i.e. what could be inferred from its knowledge given infinite processing), and "What can the system do?" (i.e. given what it knows and its ability to infer, what can it tractably infer?). Our internal evaluation includes an assessment of epistemological and representational aspects of the conceptual vocabulary and curriculum representation.

Chapter 3 described the KAFITS system, the use of which is our benchmark for an ITS shell usable by educators; the first section of this chapter describes our benchmark knowledge acquisition process (or method), and the remainder of this chapter addresses issues and tradeoffs encountered. We document our ITS design process, give quantitative results, and discuss knowledge representation issues, interface design issues, and cognitive considerations. Results, analysis, and discussion are combined and interleaved for readability.

5.1 ITS Knowledge Engineering with Classroom Teachers

In this section we describe our design process[3] for building the statics tutor, discussing the steps taken. We also discuss the knowledge and skills domain experts and knowledge engineers need.

5.1.1 Steps in the ITS Design Process

The design process was reminiscent of "ontogeny recapitulating philogeny"—that is, development of the statics tutor over the course of the study roughly paralleled the evolution of computer aided instructional systems over the last few decades. Specification of instructional content passed through classroom-like, CAI-like, and finally ITS-like phases. The

[3]The terms "the design process" and "knowledge acquisition method/process" both refer to the process we followed to build the statics tutor. In general there are also design steps for designing the learning environment and studying student pre-conceptions of the domain, and these steps are part of the ITS design process but not part of the "knowledge acquisition process." We did not do need to do these two steps because they were essentially complete when the project began.

process was one of moving from script-like, procedural, linear representations of the content to increasingly declarative and more flexible representations. The design process addressed this general question: how can a teacher's conceptions of subject matter and teaching methods be transformed from general knowledge based on classroom and one-on-one instruction to a detailed yet flexible conceptualization that is appropriate for computer-based tutoring? Two parallel tasks were necessary. The first was *guiding* the teacher through a series of structured interviews which defined and refined their knowledge according to a specific representational framework (KAFITS in this case). The second task involved *training* the teacher in the skills and concepts needed to design an ITS. The ITS design process described below is the guidance method. The training method, which was interleaved with the design process, is described in Section 5.1.2 (although the reader will find some mention of training in the description of the guidance method).

Figure 5.1 shows an outline of the design process—the knowledge acquisition method used to design the statics tutor. The steps cover the four phases mentioned in Section 4.2: Phase 1 (initial curriculum design) has been refined into steps 1 to 7, Phase 2 (entering the LE curriculum), Phase 3 (debugging the LE curriculum), and Phase 4 (testing the LE curriculum) correspond to steps 8, 9 and 10, respectively. Finally, in Phase 5, steps 5, 7-9 were repeated for most of the remaining statics curriculum. A detailed analysis of the time it took for each step is found in Section 5.3.2. Below I give a detailed description of each of the steps.[4]

Overview meetings. During the first meeting with the domain expert (Dr. Charles Camp) we discussed our goals and the available resources. I gave an overview of these aspects of the KAFITS framework: the domain knowledge base, the topic network, tutoring strategies, and the student model. I also mentioned the availability of the crane boom

[4]For this (and only this) section of the paper the author found it stylistically preferable to refer to himself in the first person singular, because this section discusses the author's interactions with the domain expert.

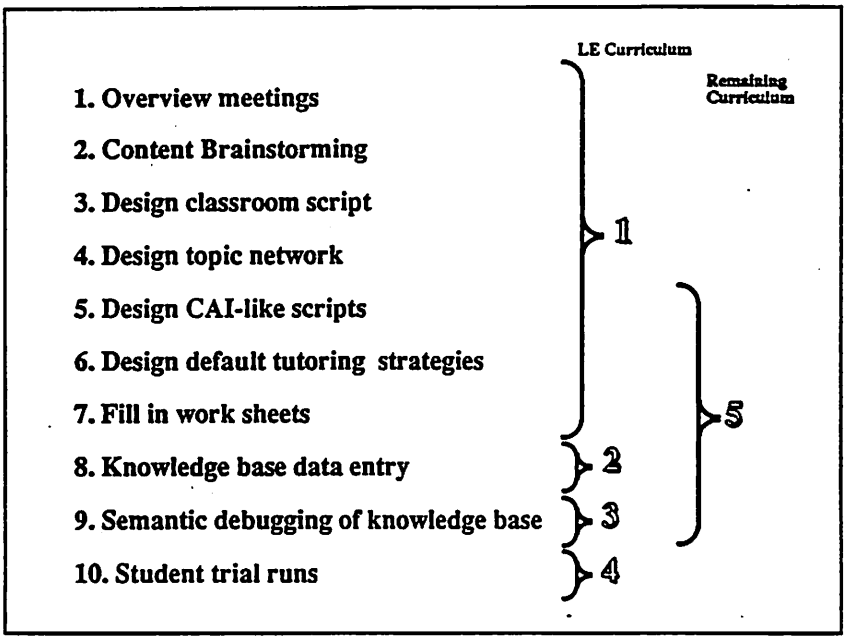


Figure 5.1 ITS Knowledge Acquisition Method
(The Design Process)

simulation and some curriculum materials written by others for topics related to the crane boom.[5]

Content brainstorming. Taking several factors into account, including the availability of the crane boom simulation, we arrived at a general idea of the the high level content, and some rough behavioral objectives for students. Camp wanted the tutor to teach a basic qualitative understanding of Newton's laws for static situations. He wanted to "try to stay away from the big and the messy" and said "even though the simulation calculates the numbers easily [we should] avoid getting mired in [numbers]" [6/15/89]. Camp's behavioral goals were for students to write force equations for simple crane boom configurations and answer some qualitative questions that would indicate a good intuitive grasp of the material. We discussed the knowledge that the average student was assumed to have before starting the tutor. Camp described common misconceptions in statics and the difference between misconceptions and "stuck points" (or "stall out places"), where students often get hung

[5]The first three meetings were held in Camp's office at the high school were he taught so that he would be more comfortable. Among the materials I brought were large sheets of newsprint for brainstorming activities. After the third meeting all meetings were held at the UMass lab.

up or need extra help. At the end of the brainstorming sessions we had sketched a high level topic network, containing ten nodes and five prerequisite relationships.[6]

Classroom script. To anchor discussion of the curriculum in a concrete and familiar context, I next asked Camp to design an outline for classroom-style lessons of the subject matter, including key examples, explanations, and questions. He framed his goal this way: "we need a sequence of qualitative problems that will move the student along to a good understanding" [6/15/89]. The script was at the level of a detailed overview; it did not include remedial or advanced material and did not branch for misconceptions. The classroom script afforded a more concrete example of what Camp wanted to teach.

Topic network. We analyzed the classroom script to produce a hierarchical list of topics and sub-topics, clarifying the content to a greater level of detail than in the high level network produced during content brainstorming. Next I introduced Camp to some basic aspects of the KAFITS framework, that the topics are arranged in a semantic network, each containing presentations specifying student-tutor interactions. We organized the list of topics into a network, at first without attending to the semantics of node types or link types. The goal was for completeness and circumscription of the curriculum. Next I instructed Camp in additional aspects of the KAFITS framework including: knowledge types, topic levels, and types of relationships between nodes. I also introduced the "spiral teaching" effect noted in Section 5.4.3. These new concepts allowed important distinctions and relationships in the network to be articulated, and the network was refined accordingly. For instance, the distinction between principles and procedures helped Camp sort some subject matter into more refined topics.

CAI-like scripts. At one point, after several hours discussing fine points of the topics' contents and relationships at an abstract level, uncertainty and ambiguity about the scope and overlap of the topics became an issue (i.e. What did each topic represent?). This may seem like a trivial problem, solvable by writing a clear definition or behavioral objectives for

[6]Only 6 of these high level topics eventually made it to the final 41 node topic net; the rest were pruned out as we refined the scope of the curriculum.

each topic, but fuzzy areas kept arising and our discussions were straying too far from the presentations Camp had designed for the classroom script. To move toward the concrete and specific again, I asked Camp to compose CAI-like tutorial scripts, or "story boards." For each script he was to assume that a student started at one of the topic nodes and took a typical path which would pass though several nodes. This was done, starting at several high level topics, so that much of the curriculum was considered. These detailed scripts specified curriculum features such as diagrams, crane boom configurations, motivations, summaries, explanations, examples, tasks, hints, and branches according to student behavior. They were written in terms of presentations and topics, and contained much of the information needed for topic and presentation objects. These scripts were CAI-like because all curriculum paths and branches were explicitly specified.

Designing default tutoring strategies. Next I introduced the concept of strategies that would determine: 1. How to traverse the topic network, 2. Which of the available presentations would be given, and 3. What type of response was to be given to the student (these correspond to the topic, presentation, and response decision levels in Figure 3.2). Before strategies were discussed ideas and constraints for tutoring were discussed in terms of "rules." For example: "to teach a composite topic teach all of its parts," "teach sub-concepts before other sub-parts," "check that a topic is not already known before teaching it," and "don't congratulate the student after a correct answer if they got it wrong previously." Some of these rules were suggested by Camp and some by me. The very structure of topic and presentation objects automatically suggests certain possibilities and orderings, which could also be seen as rules. For example, topics have a slot called Motivation, which should come near the beginning of teaching a topic; and presentations have a slot called Hints, which should be given (if at all) soon after an incorrect student answer.[7] Taking all of the "rules" into consideration I drafted preliminary strategies for the topic and response levels. Early drafts of the strategies were like scripts, showing the order of all of the actions as if all would be included (as in the most verbose possible strategy). Later versions were in

[7]Though most of the structure of KAFITS objects was determined before the case study began, several object slots were added or changed over the course of the statics curriculum design as a result of suggestions from Camp or needs that I perceived while working with him.

terms of flexible action networks (PANs) and switches (see Section 3.1.7 for a discussion of PANs, and Appendix E for diagrams of the final strategies). Three strategies for the presentation level were discussed. The default presentation strategy was trivial: to give the presentations for a topic level in the order they appeared in the topic's slot. Camp also designed a presentation strategy specifically for the {FBD-identify-forces} topic, which is shown in Appendix E. We also discussed using a bridging analogies strategy [Murray et al. 1990] for some topics, although this was not implemented.

KAFITS worksheets: re-designing for flexibility and modularity. Camp was informed that the instructional context of topics and presentations is not predetermined when the curriculum is conceptualized in a flexible way allowing for multiple strategies. For example, he could not assume that {Linear-equilibrium-concept} would be, in all situations, preceeded by {Linear-equilibrium-intution}, or that a topic's definition would be preceeded by its motivation, or that an answer-reason will be preceeded by the answer give-away because different strategies could order them differently. I also introduced the student initiative feature, which allows students to jump from one part of the curriculum to another. Consideration of these additional degrees of flexibility created a need for the components of the curriculum to be modified toward modularity and multiple uses (with dependencies on other parts made explicit if possible).[8] Camp was also advised that information relevant to many different strategies should be included in *all* objects (or as many as are feasible, in order to maximize the flexibility made possible with multiple tutoring strategies); i.e. most instance slots should be filled in (eg. most topics should have a motivation, summary, examples, etc.). With these suggestions in mind, Camp re-designed the curriculum for the final time, specifying it at a level detailed enough for entry into the computer.

To facilitate data entry I designed paper worksheet forms with templates showing the information required for topics, presentations, Mis-KUs, and crane-boom objects (see Appendix I for samples). The templates served as reminders of the attributes of objects as well as worksheets for curriculum design. Camp needed to be introduced to some basic syntax

[8]Designing for *complete* flexibility was not feasible—tradeoffs are discussed in Section 5.4.3.

at this point, since the contents of slots were restricted by data type (for example: a list of numbers, text, an integer, etc.).

Knowledge base data entry. To enter Camp's curriculum into the computer, Linton (the first knowledge base manager) first created blank topics for all nodes in the topic net. Linton ran preliminary local and global tests of the knowledge base and tested the the tutor's flow through the topic network using a strategy called Skim, which traverses the network according to prerequisite and other topic relationships, giving only a brief description of each topic (which at this point was blank), without trying to "teach" the topic. Then Linton entered and tested the information from the topic and presentation work sheets. The tests were mostly syntactic, checking that the material was entered as specified by the domain expert. Linton needed one hour of instruction about the basics of the Browser to get started using it (though more time was spent learning additional features). Being the first extensive user of Browser, he had many suggestions that resulted in code modifications.

Semantic debugging of the knowledge base. Next Camp performed a semantic debugging of the computer-based curriculum. He ran the tutor, tried numerous combinations of student responses and strategies, and made many changes to the knowledge base using the Browser. (Camp was trained in basic Macintosh concepts and basic Browser features just prior to this). The first runs were performed with verbose strategies, so that all elements of the curriculum would appear. Test runs were performed choosing all correct answers, then all wrong answers, then a mixture. Less verbose strategies were then used, to test the flow of the tutorial dialog when some components of the curriculum were skipped.

Student trials. Nineteen subjects test ran the LE curriculum. Camp and Gonzalez analyzed the test results to improve the knowledge base. A description of the student trials is given in Section 5.2.

In summary, the above description provides a schematic view of our design process. The steps also serve as a methodological framework for others building ITSs and engaged in ITS knowledge acquisition. The design process assumes that the instructor (or knowledge base

managers) will actually build a tutor, doing most of the knowledge entry and testing, and it assumes a knowledge engineer is available to supervise the process.[9]

5.1.2 Training the Domain Expert

Here I discuss what the domain expert needed to know and how he was trained, and make general suggestions for training ITS domain experts.

Encouraging a Vision of the Final Product

Very early in the design process the teacher should be encouraged to form a fairly concrete vision of the look and feel of the proposed tutor. Fortunately, we were able to show Camp both a prototype of the crane boom simulation and an example of a small prototype KAFITS-based tutor in operation, including examples of tutoring sessions and the Browser in use. These experiences significantly clarified his conception of the look and feel (but not the content) of the proposed tutor (a more detail discussion of the importance of a concrete context in training is given in Section 5.5.2). Though I had explained the functionality of the simulation to him earlier, its potential as a learning tool became much more apparent when he played with the system. He envisioned scenarios for asking students to draw force vectors on free-body diagrams that he did not consider when given only pictures and a description of the simulation ("It was really good that we looked at the [simulation] yesterday" [6/30/89]).[10]

A Galaxy of Knowledge

Teachers have much to learn on the road to becoming ITS domain experts. As mentioned above, the domain expert was trained in parallel with the early design steps. An overview of the galaxy of knowledge needed by the domain expert for this project is shown in Figure

[9]However, an almost identical process could be followed in cases were the knowledge engineer interviews the domain expert to extract knowledge and then does the data entry and testing him/herself.

[10]Camp also had numerous suggestions for improving to the simulation, most of which were implemented.

5.2 (at the start the teacher already knew some of the items under the domain pedagogy category, and bits and pieces of other categories from an ITS Summer Teacher's Institute). I claim that most of the knowledge listed will be need by a domain expert involved in building *any* intelligent tutor, if they participate in design, implementation, and testing.[11]

Training in Instructional Theory

Originally I had planned to introduce the domain expert to instructional design theories (by Reigeluth, Merrill, and Gagne, as mentioned in Section 2.2), and encourage him to incorporate instructional design principles into the curriculum. I soon realized that this would lead to an information overload and detract from the goals of the study. It was enough for Camp to re-conceptualize his own ideas about teaching statics in terms of the KAFITS framework and to participate in a formative evaluation of the KAFITS interface, without assimilating new theories of instruction. Learning and applying instructional design principles so that they are second nature could have taken additional months. Also, these principles usually consist of constraints on the content and form of the material presented to the student, whereas Camp was already sufficiently constrained by the limitations inherent in representing human teaching knowledge in a computer.[12] Therefore, the only elements of instructional theory used were those that were already incorporated into the KAFITS framework (see Section 2.2.7).

As a benchmark for the degree of information complexity a knowledge engineer can introduce to a domain expert, consider our use of knowledge types. Appendix B shows a handout which explains knowledge types. After reading this document, Camp said "I thought it was really quite clear...a reasonable thing to ask [a domain expert] to do...a lot of the examples in the handout are from domains I am familiar with" [7/7/89]. At this point

[11] More powerful knowledge acquisition interfaces will minimize the need to know the details of syntax and representation framework.

[12] Camp demonstrated his understanding of this constraint during a session where we were talking about the difficulties of designing sophisticated tutoring strategies that might approximate some human tutoring capabilities: "We could put a probe on their forehead for telling if they are confused, and we need one attached to their butts to see how much they are wiggling around 'cause they're bored" [6/5/90].

- **Basic ITS and AI concepts.**
  - Distinction between declarative and procedural information; objects, slots, classes, instances, semantic networks.
  - Lisp basics (code as data, source code vs. compiled code).
  - Familiarity with expert systems (rules).
  - Familiarity with seminal ITS research and ITS systems.
- **The KAFITS framework.**
  - Conceptual vocabulary.
  - Overall structure (four-level decision model, etc.).
  - The student model.
  - Tutoring strategies (switch sets and PANs).
- **The KAFITS interface.**
  - Macintosh basics (using the mouse, Mac file folders, windows, menus).
  - Basic syntax ((lists), :KEYWORDS, "text").
  - Using the Browser (operations, help features, etc.).
- **The learning environment.**
  - Using the crane boom simulation (dozens of features and parameters).
  - The student interactions and the accessible information about student behavior.
  - Canonical specifications of crane boom configurations.
  - The student interface and student initiative feature.
- **Design process and knowledge management**
  - Computer information storage concepts .
  - Familiarity with knowledge modularity issues (see Section 5.4.3).
  - Filling out template forms and managing edit records and trace files.
- **Domain pedagogy.**
  - Abilities and knowledge of average (and non-average) students.
  - Common misconceptions and buggy knowledge.
  - Key examples, questions, and explanations, prerequisites, etc.
  - Teaching strategies (that have worked in the classroom or one-on-one tutoring).
  - Relevant cognitive and educational research.
  - Other sources of curriculum knowledge (such as workbooks, textbooks, films, etc.).

Figure 5.2 Knowledge Needed by the Domain Expert

he had assimilated the information at a *recognition* level, meaning he understood quite well when I talked in terms of knowledge types, and could engage in discussions about subtle issues. However, his understanding was not integrated at the *recall* level, since he could not on his own give definitions of knowledge types, and several weeks later he became fuzzy about the meanings of knowledge types, saying "a lot of water's gone under the bridge since [I read that handout]!" [8/2/89].[13]

There is a tradeoff between letting the teacher use his practical knowledge and domain expertise vs. constraining him to use principles from instructional theories. Constraining him too much may "cramp his style" and inhibit creativity and intuition, yet some instructional principles are very powerful and have a high likelihood of improving instructional quality. One solution to this tradeoff is having the domain expert learn instructional principles and then modify them according to his practical knowledge and decide the most effective and practical contexts in which to apply them. However, the concepts and guidelines of instructional theory are not trivially applied, and it may take an unreasonable amount of time to practice using them in various contexts to gain sufficient mastery. This was evidenced in several situations in which I recommended an instructional design principle to Camp (such as having a conceptual prerequisite to a principle) and he agreed that it was a good idea, but later was not able to piece together the reasoning for the decision he had made.

Training for Other Users

The guidance and training methods described above are best suited for domain experts who are novices at using KAFITS. The method is designed to incrementally introduce the concepts and representational framework as they are needed, in parallel with the design of

[13]By the second month of the design process we had labeled all topics according to knowledge type. As mentioned above, knowledge types helped Camp clarify his organization of the curriculum. I had originally thought we would eventually design different strategies for each knowledge type, but this was not done because of the problems of introducing too much new instructional theory, and because the scope of the study was to study knowledge acquisition of domain knowledge, not strategy knowledge.

the curriculum.[14] Domain experts who do not build a tutor from scratch, but modify an existing tutor, will need to know less. Knowledge base managers need to know as much about the KAFITS interface as the domain expert, less about the KAFITS framework, and little about ITS concepts and domain pedagogy.

5.1.3 Working with the Domain Expert

In this section I describe several experiences or aspects of working with the domain expert which I believe is of general concern to ITS knowledge engineers.

Opportunism in design guidance. As in many design processes, the pre-implementation stages of the ITS design process involved alternating between creative, expansive, brainstorming phases, where "completeness" was a goal, and contracting, refining, pruning phases, where "concreteness" was a goal. As knowledge engineer I needed to be sensitive to when it was most propitious to step back for a global overview or analysis and when it was appropriate to focus in and refine, get concrete, and/or implement. As mentioned, the design process described is only a schematic; in any particular knowledge engineering session bits from several phases may have been present, and I had to make decisions about the appropriateness of leaving the main objective to review material or preview future issues.

Opportunism in training. The earlier description of the design process showed several occasions where I introduced the domain expert to new material needed for the next design step. In addition to having a general plan for how and when this information should have been be communicated, I had to be prepared to make opportunistic training decisions. For instance, I had planned on introducing knowledge types *after* the topic network was complete, at which time I had planned to refine the curriculum by determining the knowledge types of the topics. But I found it necessary to introduce knowledge types earlier than that because it helped Camp clarify some of his thoughts about how to divide the curricu-

[14] A domain expert who is already familiar with the KAFITS framework can skip many steps, and domain experts who are interviewed by a knowledge engineer but do not use the KAFITS system (as in more traditional AI knowledge acquisition) clearly will not need to learn as much or go through the same process.

lum into discrete topics [7/5/89]. On another occasion [7/20/89], when I asked Camp to design CAI-style scripts, he said he needed to know what kinds of student parameters he should assume he had access to, since the branches in the script should depend on student behavior. At this point I introduced more detail about the KAFITS student model, long before I had planned to.[15]

Scaffolding and context in training. In general, one does not want to overwhelm the domain expert with information, and it is best to introduce new information in a context in which it makes sense and can be used.[16] This is consistent with constructivist learning theories [von Glasersfeld (in press)] and apprenticeship teaching theories [Collins et. al 1986]. As an example of the power of learning in concrete and realistic instructional contexts, consider the following. Approximately eight months before this study began, Camp attended a three week ITS Summer Teacher's Institute[17] in which he was introduced to ITS and AI concepts, and worked in small groups to design simulation environments and "story boards" for hypothetical intelligent tutors. I was therefore surprised when, after only one day of working with Camp on this study he said "[Now I have a] hell of a lot better idea of the name of the game," not as a criticism of the Summer Institute, but as an expression of a picture that had finally become clear to him. The institute's designers (I was one) included group work to anchor the participant's knowledge in experience and give them a chance to apply what they had learned, but the participants were not as constrained or guided as they would have been if they had been designing actual ITS systems within a clear implementation framework.

As mentioned in the discussion on training, it is important for the domain expert to have an early vision of the "look and feel" of the proposed tutor—and the more concrete and realistic this vision the better. Teachers not exposed to intelligent computer tutors will have difficulty envisioning the range of possible actions and decisions a computer tutor could

[15]During this discussion Camp suggested several student model modifications which were later implemented.

[16]My notes indicate that I was often concerned about overwhelming and/or discouraging Camp with too much information. Fortunately he had a high tolerance for complexity and ambiguity.

[17]At the UMass Dept. of Computer and Information Science.

take.[18] Their specification of computer-based curriculum will be more limited by their concept of curriculum as preparation for classroom lectures and activities. Therefore exposing new domain experts to working ITSs in domains similar to theirs is highly recommended.

The usefulness of lateral thinking. As knowledge engineer I often had to strike a balance between focusing the discussion on content relevant to the ITS and allowing the discussions to stray as the domain expert circled around for a landing, or spun off on a tangent. Apparently, it is extremely rare that classroom teachers have the opportunity to talk with an interested party about what they teach, the way they teach, and the problems that come up in their teaching. Working with an ITS knowledge engineer is one such opportunity. The motivational aspects of this opportunity mitigated the frustration and tedium that were part and parcel of ITS construction. Camp's tangential thoughts and anecdotal classroom stories served several purposes. They were divergent ("lateral") thinking patterns that often led to creative solutions or reminders of important information. In addition, even as the content of discussions seemed to drift away from necessary information, they anchored the discussion and Camp's thoughts in the concrete reality of a classroom or one-on-one tutoring situation. It was apparent that forcing the discussion to remain focused on creating knowledge base instances would have been frustrating, dry, and overly abstract for Camp.

The domain expert's initial knowledge. I have mentioned elsewhere that some of the ideas a domain expert brings to an ITS project can be detrimental to the ITS design process, such as notions of classroom teaching which limit his concept of the range of possibilities the intelligent tutor affords. But our discussion would not be complete without noting that a good domain expert comes to the work with a wealth of essential knowledge. Numerous times during the curriculum design Camp demonstrated his vast and detailed knowledge of teaching high school physics, as evidenced by the quotes in Figure 5.3.[19]

[18]The situation is probably even worse for teachers not exposed to *any* type of computer aided instructional system, though exposure to traditional CAI *might limit* the teacher's sense of the range possibilities for ITSs.

[19]These quotes also illustrate that Camp was quite sensitive to whether students were actually *learning and believing* what was presented, as opposed to simply wanting them to get correct answers to questions.

- *"An awful lot of the mistakes are made in the preliminary parts [of the free body diagram solution]" [6/26/89].*
- *"Not many that get the free body diagram right get the sum of forces [equaling zero] wrong" [6/26/89].*
- *"Let's stay away from cans of worms like friction" [7/5/89].*
- *"That area [around {Types-of-forces}] is a nasty little area in there," and jokingly: "Unfortunately I know too much about it from [involvement in classroom physics teaching research projects]" [8/29/89].*
- *"Fortunately we are not trying to teach the* dynamic *third law; I'm convinced* nobody *knows how to teach that...and fortunately nobody believes it [anyway]!" [12/12/89].*
- *"[In my participation on research in classroom methods for teaching physics] the thing I've learned mostly is how* hard *it is to [teach this material]. There are really some problems in teaching this stuff." [1/9/90].*

Figure 5.3 Quotes by the Domain Expert on Teaching Physics

ITS designers should search for domain experts with this level of understanding about the subject matter.

The complexity and magnitude of designing an ITS. Though Camp worked very productively and (to us at times surprisingly) enthusiastically on this project for its entire duration, the magnitude and complexity of the task before him was sometimes a source of frustration and momentary exasperation, as exemplified by the quotes in Figure 5.4. Tolerances for uncertainty, ambiguity, and complexity are desired characteristics of ITS domain experts. Camp was a good candidate for this project in this respect; he started working on the statics tutor in the dawn of his growing understanding, well before having a detailed picture of the overall KAFITS system.

Making efficient use of the domain expert's time. A substantial commitment is needed by the domain expert to design an ITS from scratch. Camp had many other commitments (school teachers' commitments during school session are usually numerous and overwhelming), so his time was precious. Having knowledge base managers on the

- *"[There are] sure a hell of a lot of options!" [6/15/89].*
- *"Boy, there's plenty of room for creative thinking here!" (said somewhat sarcastically, but with excitement, putting his head down on the desk, shaking it, grinning) [6/15/89].*
- *"Boy, there's a lot of levels to think about this thing on!" [7/30/89].*
- *"When you pass out your dissertation you better give them a bottle of Tylenol!" [7/30/89].*
- *"When I think about the range of ideas we have here on this paper [a printout of the knowledge base] I sit back and say 'woah!' We have enough physics ideas here to choke a horse!" [9/18/89].*

Figure 5.4 Quotes About ITS Design Process Complexity.

design team helped us make the best use of Camp's time. (See Section 5.3 for an estimate of the time commitment needed to design the statics tutor.)

Most of Camp's participation was once a week. This is, in part, disadvantageous compared with more frequent visits.[20] Progress on this project was more efficient during the periods when Camp came in frequently (several days a week) and was "on a roll." Camp had to build complex mental models of the KAFITS software and his curriculum and have these readily accessible in his mind for efficient work.[21] Camp's life as a teacher was so full during the school session that it took a significant amount of time (about a half hour or more within each session) to get re-established after a week's break. On several occasions Camp expressed his frustration with the (necessary) infrequency of his work with me, for example: "Getting my head around this *whole* [topic network] is mind boggling. Getting my head around *parts* of it [at a time] is no problem. If I had my druthers I'd take a team of about four physics teachers and work on it for six months" [2/14/90].

However, in our case, having the domain expert come weekly did have some advantages. Modifications to the KAFITS system to add new features or re-conceptualize aspects of

[20]During the school year frequent visits may not be possible.

[21]Even "simple" tasks like driving a car or using a hand held calculator require complex mental models [Young 1983].

the framework often took weeks, and fixing code bugs sometimes took days. In many ITS design efforts the simulation environment will be built or modified while the tutor is being built (the crane boom simulation underwent substantial modifications according to Camp's recommendations), and spreading the teacher's involvement over a longer time allows a more complete system to be developed.

The amount and complexity of information the domain expert must learn, and the "preciousness" of his time, highlight the importance of usability and effectiveness for the knowledge acquisition interface. Structures and concepts that are visually reified act as reminders of the underlying framework and limit the user's cognitive processing load.

Computer vs. classroom instruction. On several occasions Camp commented on hypothesized benefits of computer tutoring as compared to classroom instruction. When the simulation was first described to him he was impressed with the possibility of animating the objects in a statics problem and the possibility of showing diagrams being constructed piece by piece, for example having the vectors of a free body diagram added one at a time. He described how he had often been frustrated that textbook or blackboard diagrams do not easily convey important concepts because of their static nature and limited number of pictures. He also thought that the computer tutor would give more instructional leverage to less advanced students, who benefit more from repetition and diagrams. He also liked the idea of being able to allow more than one right answer to questions, so that correct answers with different shades of meaning or emphases were possible. He noted that putting more than one correct answer in textbook or handout questions sometimes leads to confusion due to the delayed feedback, saying "[students] are left in suspense too long" [8/7/89]. In contrast, the immediate feedback potential of computer tutors allows students to be told that theirs was not the only correct answer, or be given a reason why another correct answer was preferred.

Summary of the Design Process

In this section we described our "design process" (or knowledge acquisition method) for building the statics curriculum.[22] It has ten steps spanning design, implementation, and testing activities, and should serve as a first pass road map for other ITS knowledge engineering efforts. We also described the content of and methodology for instructing/training the domain expert in the concepts and skills required.

The design process is a schematic which does not show the overlap of steps. Since there actually *was* overlap, the need for the knowledge engineer to make opportunistic guidance and training decisions was discussed (training was interleaved with design, and new concepts were introduced as needed). The wide range of knowledge needed by the domain expert was listed, most of which, we anticipate, would be needed by a domain expert involved in designing and implementing any intelligent computer tutor. Important characteristics of domain experts were noted, including a high tolerance for complexity and ambiguity, significant knowledge of the pedagogy of his domain, and the ability to make a significant time commitment. Important knowledge engineering skills were noted, including flexibility in moving between expansive brainstorming phases and refining concretizing phases, balancing listening with guidance, and sensitivity to when new concepts need to be introduced. See Section 2.4.2 for a discussion of the knowledge acquisition methods we used.

We pointed out the value of giving the domain expert a clear vision of the look and feel of the final product, including the possibilities for student-tutor interaction, early in the design process. We also described tradeoffs involved in teaching the domain expert new instructional theories.

We gave anecdotal evidence for potential frustration of domain experts involved in this type of project, and noted how precious their time is. Having knowledge base managers on the design team and designing the KAFITS interface for usability mitigated the size and

[22]For the remainder of the document the author will use the first person plural.

complexity of the demands placed on the domain expert and allowed us to make efficient use of his time.

5.2 Results of the Statics Tutor Test Runs

As mentioned in Chapter 4, 19 subjects test ran the LE curriculum, including 8 lab associates and 11 high school students, in four test groups, each separated by about a month. It took an average of about 1.5 hours to complete the LE curriculum. Two of the tutoring sessions involved pairs of students running the tutor; the rest were one-on-one tutoring. The domain expert was interested in getting feedback about the quality of the student interactions and the pedagogical quality of the lesson. However, the experimenter was mainly interested in how the domain expert and knowledge base manager gathered evaluative information and used the Browser to modify the knowledge base. Below we will describe and give results of the statics tutor test runs and discuss these results.

Suggested Changes from Student Trials

Students' comments (both verbal and typed) and observations by the test administrator (Gonzalez) lead to several dozen minor changes to KAFITS code and the statics knowledge base. There were between five and twenty suggestions per session (with about 30% overlap from duplicate comments), which we regard as a low number, considering the amount of curriculum presented and the complexity of what the students saw. That relatively few suggestions were made and that most of them were superficial leads us to these hypotheses: 1. the look and feel of the KAFITS student interface was minimally acceptable, 2. the LE curriculum was minimally acceptable; and 3. Camp and Gonzalez's test runs (semantic debugging) of the domain knowledge base were fairly thorough.[23]

[23]Note, however, that had we been measuring whether students learned physics, more changes would probably have been made.

Although changes were made to both KAFITS code and the knowledge base in between each of the four test groups, the quantity of suggested changes per session did not decrease noticeably from the first to the last test group. This indicates that the curriculum quality (which started at an "acceptable" level) had not begun to "level off" yet. We cannot estimate how many more student trials it would have taken to have student comments level off to a "very low" number. Due to fundamental limitations in the "intelligence" of the computer tutor, the needs of all students could never be met at a level comparable to one-on-one human tutoring, and perhaps the quantity of student comments would always be at the level observed in our tests.

Suggested changes to the domain knowledge base were of several types, including: wording (eg. spelling and grammar), pedagogy (eg. answers or explanations that didn't make sense), and format (eg. a question being too big so that part of it scrolled off the window before it could be read; a picture that was shown before its text description rather than after it). Gonzalez's analysis of session trace files to produce summaries of student comments went smoothly. Camp's and Gonzalez's editing of the domain knowledge base to account for student comments also went smoothly. Camp used his own judgment to decide which of the suggestions warranted changes in the knowledge base.

Student Comments and Critiques

Most of the subjects reported overall enjoyment of their tutoring sessions, though there were moments of frustration. On the average they did not seem to learn a great deal of new material, but they found several features of the tutor helpful. To give the reader a feel for the student's experiences, we include the following student comments, first positive ones, and then critical ones.[24] There were substantially more critical comments than positive ones, but the vast majority of critical ones concerned problems or questions with specific pieces of the curriculum content, and only general comments are listed below.

Positive comments:

[24] All quotes in this section are paraphrases.

- The overall [dialog] flow seems to be smooth and continuous....Once I became familiar with how the system worked [menus, etc.] things seemed to run quite smoothly [session 175].
- The hints were very helpful, providing ways of thinking about the problem I hadn't thought of [session 234A].
- It is helpful that the tutor gives more than one way of answering the questions [session 256].
- Being able to see the vectors change and meter values change without having to calculate the numbers is nice [session 234A].

Critical comments:

- There is too much reading [text] in some areas [session 234A].
- The tutor gives too much information after you get the correct answer [session 249].
- The flow of the tutor does not make sense to me here. It does not go back to the crane boom simulation problems [session 252].

KAFITS code and strategy changes. A small number of Lisp code and tutoring strategy changes (about 15) were made as a result of student trials.[25] This number is small partially because the system was exercised significantly by the domain expert and knowledge base managers prior to student trials, and suggested changes were made at that time. Code changes from the student trials included the interruptable menu feature, the trace file format, and changes to the simulation (eg. changing the vector colors for more visibility). Also, about three code bugs were discovered by students, suggesting that no matter how much the design team exercises the software, students will think of new ways to use it—therefore, student trials were essential for working out code bugs.

[25]Strategies were represented in computer code in the version of KAFITS used for the student trials.

Most of the code changes were strategy related. A single strategy was used for the duration of each tutoring session; the "default" response strategy, which is rather verbose, was used for the first three test groups, and a less verbose strategy (called New-r-brief) was used for the fourth test group. Students were not shown how to use the student initiative menu (Section 3.2.4) to alter the current strategy. An often repeated comment during the first three test groups was that the tutor was too wordy (see critical comments above). Yet in the fourth test group (with the less verbose strategy) two students mentioned that they would liked to have seen an explanation or elaboration after they answered a question correctly, just as they did when then they answered a question incorrectly. Their wish would have been granted if the default response strategy were in use rather than New-r-brief. This weakly demonstrates the potential utility of multiple tutoring strategies and the need for student control of tutoring style. As a result, of these comments a feature was added to the student initiative menu that allowed students to get an answer explanation and elaboration of the previous presentation.

Several other strategy changes were suggested and later implemented, including: a student initiative feature allowing the student to backtrack and review or re-answer the last presentation given; and a modification to the response strategy so that the student was not congratulated (for example: "that's good") when she answers correctly after her second or third try at the question.

There were several useful suggested code and strategy changes that we did not implement (but plan to), including: the capability for a student to see her progress in the lesson (for example, to know whether she was almost done); a "there is only one possible answer left" message given when the student has tried all but one answer choice unsuccessfully; a "you have already tried that choice" message given when the student answers the same way twice; and the capability for advanced students to skip the easy level presentations in all topics.

Other Student Trial Observations

Camp intended the statics curriculum to be used after the learner had some initial exposure to important concepts from a classroom or textbook. He also assumed (see Section 4.1.3) that the tutor would be most useful to students of average or below average ability. All of the high school participants had had one semester of physics and were currently enrolled in an advanced physics class. About half were of greater than average ability and the other half were of average ability in their physics classes, according to Camp. Therefore the students that volunteered for this study were not optimal from the perspective of the anticipated instructional leverage of the statics tutor, which was expected to have greatest impact on average or below average students. Camp's intentions and assumptions held true, as evidenced by the following sample of student comments:

- I didn't learn any new concepts, but it was helpful as a review or supplement [session 234].
- The tutor helped clarify some ideas about statics [session 232].
- It was good as a review for material already learned, but not to learn it initially [session 245].

Different learning styles were noted. One student [session 232] spent a lot of time playing with the crane boom simulation; he enjoyed exploring various crane boom configurations and tried to discover rules governing the force vectors. This student took the longest to finish the LE curriculum, $2\frac{1}{2}$ hours.

The pair of students in group #4 worked well together and had many motivated exchanges. Fewer student comments were typed during this session than during the other one-on-one sessions, perhaps because students tend to interact with the computer less when they can communicate with each other.

The pair of students in group #3 was mismatched—one student was familiar with Macintosh computers; the other student was not totally fluent in English and had no familiarity

with Macs. In this session the first student tended to take control and not include the second student (who seemed timid) in decision making.

Domain expert observation. Test #4 was unique in that Camp sat nearby and observed the sessions. This was the first time Camp had ever witnessed anyone being given the curriculum he had designed, and, significantly, he said there were "no big surprises" from his observation of the two one-on-one sessions and one paired session. Camp thought that his presence may have affected the confidence of two of the students. In addition he thought that using the tutor with pairs of students was "valuable."

Student comment facility. Students were quite willing to take the time to type in their comments. This may have been because they knew they were part of an experiment or software evaluation. Also their willingness to enter comments may have been due to the fact that students rarely have the opportunity to critique the teacher (though they would undoubtedly have much to say) and the KAFITS student comment facility provided such an opportunity.

Summary of the Test Runs

In summary, the KAFITS student interface was fairly robust and complete after being exercised by the knowledge engineers and the domain expert, and the statics knowledge base was fairly bug free after being debugged by the domain expert. However, student trials did result in the detection of interface and knowledge base bugs, leading to important modifications. The LE curriculum trial runs, involving 19 subjects and taking an average of $1\frac{1}{2}$ hours, went fairly smoothly. The overall style and content of the curriculum was acceptable according to students and according to the domain expert who observed four students running the tutor. However, several of the students, who were on the average more advanced in physics than students for whom the statics curriculum was thought to be optimal, complained of it being too wordy, and said it gave a good review of the material but that they did not learn anything new.

These were supervised lab experiments, and we were not particularly concerned with student learning. We assume that many changes would need to be made to both the student interface and the statics curriculum for it to be robust enough to be used in a classroom, but the student trials suggest that the KAFITS system (both the framework and the interface) is a viable tool for designing, implementing, and testing ITS curriculum.

5.3 Quantitative Analysis of the Curriculum and Design Process

The driving questions for quantitative analysis of the data are:

- How can the size and structure of the knowledge base be measured to allow for comparative analysis of knowledge bases?
- How much effort is required by each participant in each step of designing the statics tutor?
- Approximately how much time did it take (per hour of instruction, and per curriculum topic) to design the statics tutor?

Much of the analysis is approximate, entailing estimations and extrapolations (we explicitly note points of estimation and approximation in our discussion below). We consider this acceptable since we are only looking for order of magnitude answers to the above questions.

5.3.1 Curriculum Size and Complexity

Here we give metrics for analyzing and comparing the domain knowledge base and describe the size and complexity of the topic net, the entire domain knowledge base, and the LE portion of the curriculum.

The Topic Network

Following are important characteristics of the topic net:

- Total nodes: 41.
- Breakdown by completeness of node: 24 full topics, 8 empty composite topics, 7 overview topics, and 2 (full) Mis-KUs.[26]
- Equivalent full nodes: 28.[27]
- Node fullness ratio: (28/41) 0.68.
- Total topic net links:[28] 51.
- Links per node: (51/41) 1.2.
- Topic time: (assuming an average of 6 hours to run the entire curriculum)) 8.8 minutes of instruction per topic.

The total nodes, node fullness ratio, and links per node can be used to compare the statics topic net with other topic nets, or to compare parts of the statics topic net with each other. The total number of nodes may not yield a fair comparison, since some portion of them will be empty or only partially completed; the equivalent full nodes is a better method for comparison. The links per node is a metric of the complexity of the network. Since we have no data on KAFITS topic nets for other domains, these figures are provided

[26] A "full" topic contains presentations for teaching it; an "overview" topic contains no presentations, but may contain a motivation, summary, wrap-up, etc.; and "empty" nodes contain only pointers to other nodes. Topics may be overview or empty intentionally, or because they have not been completed. All of the composite topics in the statics curriculum are empty (i.e. their whole is equal to the sum of their parts).

[27] The equivalent full nodes gives an indication of the size of the curriculum that is independent of how many nodes are empty or overview, and is calculated as follows: empty nodes count as one twentieth of a full node, and overview nodes count as one fifth of a full node. This roughly reflects both the time it takes to implement the node and the on-line tutoring time for full vs. empty vs. overview topics.

[28] The number of topic net links does not give a complete picture of the topic relations. Implicit links (such as ordering determined by text order in a slot), and local links (such as reaction and remediation links associated with particular question answers) are not included.

for comparison with future topic nets, and to compare the LE curriculum with the entire statics curriculum, which we do below.

The Domain Knowledge Base

Following are more figures (not related to the topic network) for the domain knowledge base:

- Total knowledge base instances: 252.
- Breakdown according to object type: 38 topics, 2 Mis-KUs, 81 presentations, 48 pictures, 40 crane-booms, 43 others (including pictures, sounds, storage instances, lessons, examples, and questions).
- Object type ratios: (81/41) 2.0 presentations per node;[29] (81/26) 3.1 presentations per full node.
- Total transactions: 511;[30] about 12 transactions per node.
- Domain knowledge base (Saved-instances file) size: approx. 200 kbytes of text/source code on disk; 90 hard copy pages.

The presentations per node gives a measure of the amount of subject matter in (or the "depth" of) the average topic. Technically, presentations are used to represent expository and inquisitory interactions with the user. However, when an expository interaction is a block of text, it is entered as a text string (i.e. there is no need to create an entire presentation object just to store a block of text).[31] Therefore, to get another perspective

[29] "Nodes" are used rather than topics so that Mis-KUs will be included.

[30] Transactions are the smallest units of discourse. Each block of text stored in a slot, and each picture, simulation, etc., accounts for one transaction (randomly chosen text is which is not included in this figure). All KAFITS text is canned. A more complicated method for calculating the number of potential student transactions would be needed for generated text. The number of transactions may be a more useful metric for comparing the size of a KAFITS knowledge base with other computer-based curriculum than the number of topics or presentations.

[31] Object slots that contain information for interacting with the student can point to a presentation *or* contain text.

on the amount of information in the knowledge base and to enable comparison with other computer tutoring systems, we total the potential student "transactions," the text strings in the knowledge base. Each slot (including Motivation, Definition, Question, Hint, Reaction, etc.) counts for one transaction if the slot contains text to be given to the student.[32]

Linear Equilibrium Part of the Curriculum

The following data are for the LE curriculum (see Figure 4.2):

- Total topics: 12.
- Breakdown according to object type: 12 topics, 16 presentations.
- Object type ratios: (16/12) 1.3 presentations per node; (16/7) 2.3 presentations per full node.
- Equivalent full nodes: 8 (7 full topics; 2 empty topics, {Linear-equilibrium} and {Vectors}; and 3 summary topics, the parts of {Vectors}).
- Node fullness ratio: (8/12) 0.67.
- Total links: 14 (only links *from* LE topics).
- Links per node: (14/12) 1.2.

Much of the most detailed data for this study was taken during the development of the LE curriculum. Values for node fullness ratio, links per node, and object type ratios for the entire curriculum are quite similar to those for the LE curriculum. This allows us to extrapolate some data from the LE curriculum to the entire curriculum.

[32]Six of the primitive actions (the default hint, tell-wrong, tell-ok, encourage wrong, encourage-ok, and congratulate) use randomly chosen text from pre-defined sets (see Section 3.1.2) if the corresponding presentation slot is empty—these are not included in the total transactions.

		Tasks	
		Development	Training
Steps	Design		
	Implementation		

Figure 5.5 Project Task and Step Terminology Relationships

5.3.2 Design Steps and Person-hours

Here we look at the person-hour effort involved in building the statics tutor. There are many perspectives for analyzing available data and many relevant variables to compare, including: curriculum part (the LE curriculum vs. the entire curriculum); participant roles (domain expert vs. knowledge engineer vs. knowledge base manager); project steps and tasks (development vs. training vs. design vs. implementation). Below we present and analyze the data, describing and comparing variables.

Figure 5.5 shows the relationships between the terms we use to describe the design process (this diagram is also a schematic of the layout of Figure 5.6). There are two kinds of "steps:" design and implementation. Within each step there are two kinds of "tasks:" development (which includes production and guidance)[33] and training. "Training" refers to the initial instruction for how to carry out a task, plus all other instruction not related to any specific task, such as teaching about the KAFITS framework, ITS concepts, etc. The table in Figure 5.6 contains a detailed analysis of person-hours involved in the project. It is organized with project steps in the rows and project tasks in the columns. A description of each of the steps was given in Section 5.1.1.

Person-hour Analysis Table description and assumptions. Below we explain the table in Figure 5.6. The notes below the table explain exceptions to the assumptions and calculations given here.

1. We make a working assumption that there is no overlap between steps.

[33] "Guidance" refers to ongoing guidance related to the task at hand.

Who & Note	Step	Total days	Lab hours	Home hours	Total hours	Train -ing	Dev- elop.	LE-dev -elop.	Stat. dev.
1. C-1	Overview	.5s	3.5	0	3.5	3.5	0	0	0
2. C	Brainstorm	2.5s	12.5	5	17.5	6.2	11.2	2.8	11.2
3. C	Class. Script	2s	10	4	14	5	9	2.2	9
4. C	Topic Net	2s	10	4	14	5	9	2.2	9
5. C-2	CAI Script	2s	10	10	20	5	15	15	45
6. C	Strategies	1.5s	7.5	3	10.5	3.8	6.7	1.7	6.7
7. C	Work Sheets	4.5s	22.5	9	31.5	5	26.5	26.5	79.5
8. L-3	Data Entry	NA	52	0	52	3	49	49	147
9. C-4	Debugging	9w	27	4.5	31.5	4	27.5	27.5	82.5
10.G-5	Testing	NA	30	0	30	3	27	27	81
Design (steps 1-4, 6)			43.5	16	59.5	23.5	35.9	NA	35.9
Implement. (steps 5, 7-10)			141.5	23.5	165	20	145	NA	435
Totals	all	NA	185	39.5	224.5	43.5	181	154	471

Table Notes:

1. The overview step differs from others in that the teacher had no home hours and the entire step constitutes training time.
2. An estimated 6 hours extra home hours over a vacation period was added.
3. Linton's hours were recorded by him, not estimated by the number of days he worked: 36 hrs. data entry plus 17 hours to organize and test the knowledge base (I subtracted from his 34 hours an estimated 17 hours wasted on unnecessary organization because there was no knowledge base save feature yet) equals 52 hours. Linton's time does not include time spent on picture drawing: 43 hours for 18 pictures (2 of which were not for the LE curriculum).
4. Estimation for semantic debugging: 3 days off line, 4 days on-line to "first pass," plus estimated 2 more days on line before testing, equals 9 days.
5. Gonzalez's hours spent setting up and administering tests are estimated as follows: preparation 5 hours, administration 15 hours, data collection 10 hours, totaling 30 hours.

Figure 5.6 Person-hour Analysis for Building the Statics Tutor

2. Total days. The estimated number of days working: "s" means solid time (approximately daily work), "w" means weekly time. All other numbers in the table are in hours.

3. Lab hours vs. home hours: We estimate that solid days are 5 lab hours and 2 home hours, and weekly days are 3 lab hours and 1/2 home hours.

4. Total hours is lab hours plus home hours.

5. Training time is estimated to be one half of the lab time until after step 6, then it is estimated on a step by step basis.

6. Development time is total time minus training time.

7. LE curriculum Development time for those steps that were relevant to the entire curriculum (steps 2, 3, 4, and 6) is estimated to be 25% of total development time. For other steps it is equivalent to the development time.[34]

8. The development time for the statics curriculum was estimated as follows: For steps 1-4, and 6 it is equivalent to the total development time; for the rest of the steps it is four times the LE development time, times 0.75, for an assumed (and quite conservative) 25% efficiency increase.

9. Key for "Who" column: C– Camp, L–Linton; G–Gonzalez. For notes see the numbered notes below the table.

10. Subtotals—design vs. implementation. Steps 1, 2, 3, 4, and 6 apply to the design of the entire curriculum. The implementation subtotal includes only steps working on the LE curriculum, and does not include design work. The LE development total includes design and implementation, and does not include training or time spent on other curriculum parts.[35]

---

[34]There are 41 nodes in the topic net. The LE curriculum has 12 nodes, or 25% of the total (also, the LE curriculum has (8/28) 29% of the equivalent full nodes of the entire curriculum). Therefore we estimate that the LE curriculum development time is 25% of the development time for the statics curriculum.

[35]I.E. the difference between the LE development column and the implementation subtotal. As a data cross check, note that the total for the LE development column (151.1 hours for LE work, which includes

	Domain Expert		KB Managers		Knowledge Engineer		All		Total
	Train.	Devel.	Train.	Devel.	Train.	Guidance	Train.	Devel.	
Design	22.7	36.8	0	0	22.7	3.7	46	40.5	86.5
Implem.	14	203	6	234	20	32	40	469	509
Totals	36.7	240	6	234	42.7	35.7	86	510	596
	277		240		79		596		

Figure 5.7 Time vs. Participant Role

11. Totals are for design of the entire curriculum, i.e. all ten steps (or equivalently, design steps plus implementation steps).

Person-hours vs. participant role. Figure 5.7 shows a breakdown of the time commitment vs. participant role for the entire statics curriculum. The domain expert and knowledge base manager hours are from the training and statics development columns of Figure 5.6. Steps 8 and 10 from Figure 5.6 were summed for the knowledge base manager time, and the remaining steps in Figure 5.6 go toward domain expert time. The knowledge engineer's time (which was not included in Figure 5.6) consists of training time and guidance time. Knowledge engineer training time is equivalent to the domain expert training time plus the knowledge base manager training time. Knowledge engineer guidance time is estimated at 10% of the domain expert development time plus 5% of the knowledge base manger development time.

Analysis and Comparisons

Summary calculations. The total time spent to build the statics tutor is estimated to be 596 hours. This includes training and development for all participants, and does not include development of the KAFITS system, development of the crane boom simulation, or time spent creating picture graphics. It took (596/41) *14.5 hours per topic net node* to build the statics curriculum. Assuming the entire curriculum provides an average of six hours of teaching it took (596/6) *98.8 hours per hour of on-line instruction* to build the

all non-training time) is equal to the implementation subtotal of the development column (142 hours of LE work for only implementation steps) plus 25% of the design subtotal for the development column (because 25% of 36.8 hours is the part of the design time used for the LE curriculum).

statics tutor. These numbers are estimates intended to yield order of magnitude figures. We believe that they are applicable to similar ITS projects using KAFITS, to within 50 percent.

Factoring out training. Training time is considered to be one-shot, i.e. a participant is instructed in something only once, regardless of the size of the curriculum. Reminders or reviews of this information are considered to be "guidance," which is included in the development time for the domain expert and knowledge base managers. The domain expert's total training time was about 40 hours, spread over about 24 working days, interleaved with production work.[36] The total training time was (37.5+6+43.5) 87 hours, or (87/596) 14.7% of the total time. Factoring out training time it took (.853x596/6) 84.3 hours per hour of on-line instruction (i.e assuming prior training). This is the time to create each additional hour of instruction, assuming training is complete.

Following are comparison figures expressed in percentage of the total time:

As a function of role: The knowledge engineer's, domain expert's, and knowledge base managers' efforts were (79.2/596) 13.3%, (280/596) 47.2%, and (234/596) 39.4%, respectively, of the total time. The domain expert's time was comparable to the knowledge base managers' time, which was about three times as much as the knowledge engineer put in.

As a function of step: Design took (86.5/596) 14.5%, and implementation took (507/596) 85.5% of the total time. I.E. implementation took about six times as much effort as design.

As a function of task: Training was ((37.5+43.5)/596) 14.7%, and development was ((243+228+35.7)/596) 85.3% of the total time. Note that knowledge base managers'

[36] We do not include time from the ITS Summer Teacher's Institute in the domain expert's training time. The domain expert learned many general ITS concepts in this institute, but this covers only a small percentage of the total training material (see Figure 5.2).

training time was only (6/37.5) about one sixth as much as the domain expert's training time, even though they had comparable development times.[37]

5.3.3 Analysis of Other Data

All of the above figures were based on the nature of the tasks and amount of days spent on each task. Below we analyze data from edit records and work sheets (which have the dates of design and data entry on them) and note how these data compare with the above figures.

Edit record data exist for 69 separate working days.[38] About 25% of these days consisted of creating instances from work sheet data, and the remaining 75% involved testing and editing the curriculum.[39]

A cursory analysis of edit record data gave the following figures:

- The average time it takes to enter one worksheet form into the knowledge base using the Browser, i.e. create a new instance, is about 25 minutes (data ranges from 19 to 30 minutes).
- Data from editing records logged during test runs of the statics tutor indicate that the average time it takes to find a bug in the curriculum and edit the knowledge base to fix the bug is about 6 minutes (data ranges from 2.4 to 11.6 minutes with outliers removed).
- From Camp's and Gonzalez's edit records, we can hypothesize that there was an initial start-up time for using the Browser, during which the average time it took to make a

[37]It is only a coincidence that total design time is almost equal to total training time, and that total implementation time is almost equal to total development time.

[38]On some days there was more than one editing session. We cannot give the total number of editing sessions or the total amount of time spent editing, because some edit records were combined, and some data were lost. The breakdown of person vs. how many days for which editing information exists is: Linton 7, Camp 15, Gonzalez (or Camp and Gonzalez working together) 47—totaling 69.

[39]There were a total of 18 worksheet entry days, consisting of 25% of Gonzalez's edit record days, 70% of Linton's edit record days, and none of Camp's edit record days.

change was about 11 minutes; i.e., we saw a two-fold improvement in efficiency after a start-up period lasting one to three editing sessions.

- The data do not confirm or reject the hypothesis of a longitudinal trend toward increased efficiency over the months as improvements were made to the Browser.

Comparison of actual time vs. extrapolated time. Some of the numbers in Figure 5.6 for the entire curriculum are extrapolations based on data from the LE curriculum (justified because the LE Curriculum and the entire curriculum were shown to have similar characteristics). It would be desirable to compare the *extrapolated* total time to build the statics tutor (596 hours) with the actual time Camp, Gonzalez, and Linton worked over the sixteen months of the study. Unfortunately the development work was not closely monitored in Phase 5, after the LE curriculum was tested, when the most of the remainder of the curriculum was designed, entered, and debugged.

Camp's total time was well recorded—he put in a total of 75 days (27 solid days and 48 weekly days) over the course of the entire study (29 of these days were totally unsupervised). The previously given method for estimating total time (7 hours per solid day and 3.5 hours per weekly day) gives a total of 357 hours. This figure corroborates (is only about 25% more than) the estimated 280 hours in Figure 5.6. Linton worked only during Phase 2, and his time has been given as 52 hours (see note 2 in Figure 5.6). Gonzalez put in a total of 550 hours (according to payroll data) in Phases 4 and 5. A task/time analysis of Gonzalez's participation is difficult to determine. It was estimated that she spent 30 hours supervising the student trials. She also spent some time developing a small KAFITS tutor explaining how to use the crane boom simulation (which was not completed). A fair amount of time was non-productive, due to time spent reorganizing edit records before we had a clear system for managing them, and hardware problems (Lisp system crashes and bad floppy disks) leading to loss of data which had to be re-entered. Also, during many of the editing sessions in Phase 5, Gonzalez and Camp tested and modified the tutor together, which confounds calculations of total hours spent editing.

The estimated total knowledge base manager time for building the statics tutor is 240 hours (Figure 5.7). Subtracting Linton's 52 hours leaves 188 hours for Gonzalez's effort. That 188 of her 550 hours were spent on work directly contributing to the implementation of the statics tutor is not unreasonable, given the above discussion. In summary, though an analysis of the actual time spent by the participants is too uncertain to show close corroboration with the figures extrapolated from the LE curriculum, the actual time spent is comparable to the extrapolated time.

5.3.4 Summary of the Quantitative Analysis

Our estimations indicate that the 41 node (39 topics and 2 Mis-KUs) statics curriculum would provide an average of six hours of on-line instruction. Its design and implementation took about 14 hours per topic, each topic representing an average of about 9 minutes of on-line instruction. It took approximately 100 hours to produce each hour of on-line instruction (85 hours if training is not included), which is comparable to the estimated design time for conventional CAI given in the literature [Gery 1987].[40] This figure is to be interpreted with caution,[41] and suggests that computer tutors with the "intelligence" and flexibility inherent in a KAFITS-based tutor can be built with effort comparable to that of traditional CAI.

The following comparisons were calculated for building the statics curriculum: the total training time was about 40 hours for the domain expert and 6 hours for the knowledge base managers; the domain expert and knowledge base managers put in a comparable amount of time, which was about three times as much as the knowledge engineer put in; and implementation took six times as much effort as design.

---

[40] Our estimate includes training and development time for the domain expert, knowledge base managers, and training and guidance time for the knowledge engineer. It does not include time to create picture graphics or program the simulation environment.

[41] The hours of development per hour of instruction is not a rigorous comparative metric, in part because the longer students take to learn, the more efficient the design process seems, using this measurement.

The figures above for the entire curriculum were extrapolated based on figures for the LE curriculum. An analysis of data from edit records and work sheets for the entire curriculum shows slight corroboration with some aspects of, and otherwise does not contradict, the extrapolated figures.

5.4 Knowledge Representation Issues

Human knowledge, whether about driving a car, designing a space craft, or teaching, does not exist in neatly defined, clearly named packages—it is inherently complex, densely connected, fuzzy, and ambiguous. Yet to use knowledge in computer tutors we try to represent it in individual units with clear structure. The problems of classifying and organizing human knowledge to create external representations have been dealt with extensively from many different perspectives in philosophy, psychology, education, library science, computer science, etc. (eg. [Mervis & Rosh 1981, Winograd & Flores 1986]). Brachman & Levesque [1985, pg. xiii] note that "there are tremendous subtleties in the notions of 'representation,' 'knowledge,' and their combinations" in AI knowledge representation. That so much effort has been spent on the problem, and that this effort has not resulted in any widely accepted general solutions, attests to its salience as a fundamental issue in ITS knowledge acquisition. In this study we do not contribute to the general theory of knowledge representation. We discuss key issues related to knowledge representation and describe how these issues were discovered and handled. In addition we discuss the design of the KAFITS framework, paying particular attention to the conceptual vocabulary, and discuss problems and potential solutions in designing highly flexible and modular domain knowledge bases. First, however, we discuss experiences that influenced knowledge representation in the original prototype KAFITS system.

5.4.1 Development of the Original KAFITS Framework

It took several years to develop the KAFITS framework and software that served as the prototype for this study. Many changes were made during the course of this study, but the basic structure and vocabulary remained fairly constant. In the first month of the study we noted (in the research journal) that the KAFITS framework was anticipating many of Camp's needs, indicating that he "was on the right track" [6/30/89]. Many times in our early conversations Camp would ask: "but how do we deal with [so-and-so]," and the author would answer that the framework could account for it in [such-and-such] a way. Examples of KAFITS features that anticipated his questions are: multiple correct answers, reactions (allowing short responses particular to each possible student answer), diagnosis of misconceptions, synthesizer nodes, and elaborations to correct answers–plus many other such features.

How was the prototype framework developed? What were the reasons for design decisions? Unfortunately, the years previous to the start of the case study were not as closely documented as the case study itself, but we can describe some foundation for the conceptual vocabulary. There were two main sources of information: personal experience working with educators, and instructional design theory literature, most notably the work of Merrill [1983], Reigeluth [1983] and Gagne [1985].[42] We will not attempt to analyze why we used certain parts of instructional design theories and not others; the author assimilated much from articles and books over the years and included a small part of this information into KAFITS—only what the author thought to be immediately useful and understandable to users.

Personal experiences that influenced this study but took place before this study include working with many educators trying to formalize their curriculum and instructional knowledge. We worked with:

- Three college professors who were associated with our research group;

[42]Familiarity with other ITS systems must also have had some influence our design decisions, but not in specific ways that we can document.

- Four high school and college educators associated with the Exploring Systems Earth project, centered at San Francisco State University; and
- A dozen high school teachers who participated in the UMass ITS Summer Teachers' Institute.

We also conducted an informal study with eight associates of our lab (most of whom were also teachers) on tutoring strategies and primitive tutorial actions, which involved group analysis of taped tutorial sessions.

We succeeded in extracting from these experiences a set of ubiquitous "primitive" tutoring actions, some general patterns or scripts for tutoring actions, and parameters and reasons for the actions. The list of commonly occurring actions is not surprising, and corresponds to many of the topic and presentations slots: motivations, examples (of different kinds), definitions, prerequisites, summarizing, hinting, elaborating, etc. Common strategies or patterns include (this is a small subset): reframing—reflecting the student's ideas back to her with irrelevant information filtered out and important aspects emphasized; alternating between focusing in on details and stepping back to overview, summarize, or give a context or reason; beginning instruction with declarative statements—definitions or procedure descriptions—and then continuing with an example or operationalization; and offering meta-comments such as "this is a hard one." Since this study did not focus on representing multiple alternative tutoring strategies, most of these were not incorporated into the case study. Similarly, strategy parameters were not studied much in the case study.

In summary, the prototype KAFITS conceptual vocabulary evolved over years in response to working with educators to represent pedagogical knowledge. It was modified according to the findings of this study to produce the final KAFITS conceptual vocabulary (Section 3.1.6). Despite all of this foundational work, the vocabulary is relatively simple—of the many dozens of descriptors identified, covering a broad band of behavior and many shades of meaning, only a few were used, mainly due to our goal of simplicity. We included those terms which seemed necessary and clearly explainable, even though a larger vocabulary would have been more precise.

5.4.2 Conceptual Vocabulary and Semantic Uncertainty

The KAFITS conceptual vocabulary (see Section 3.1.6 for a description) is a language for representing knowledge about curriculum and teaching. The KAFITS vocabulary worked well in this study, and constitutes a benchmark for further efforts. There were many issues and tradeoffs involved in designing it, and in this section we discuss design considerations related to the understandability and uncertainty of vocabulary terms and discuss how procedural knowledge affects the declarative meaning of terms.

Vocabulary Understandability

Our goal was to minimize abstract and technical terminology, and to minimize complexity—but still maintain a workable degree of flexibility, power, and clarity. We cannot give formulas for how these decisions were made, but will describe the tradeoffs involved, and give examples of specific decisions made.[43]

In the original framework the term "KU," or knowledge unit, was used for what eventually became "topic" objects. Terms such as "topic" or "subject-unit" were initially avoided because they lacked precision and had differing meanings in different mundane contexts. The term "knowledge unit"[44] seemed a more technically precise term for a piece of domain knowledge of arbitrary knowledge type and arbitrary grain size. However, in trying to explain the system to educators we often found that this term was confusing; its meaning didn't stick in their minds, but drifted and got fuzzy from one training session to the next. Some users confused knowledge unit with "slot" or "object," both of which, in different senses, are also units of knowledge. So we changed it to the term "topic" which, though less precise, has not caused confusion. In contrast to the case of "knowledge unit," is the case of the term "Mis-KU," where we decided to stay with a more technical term. In the early system design we wavered about calling Mis-KUs "misconceptions," because the lat-

[43] And see Section 2.2 for a discussion of how the Performance-Content Matrix was designed with usability in mind.

[44] Originally coined by Deborah Servi [1986].

ter term has the advantage of being commonly understood. But it was not precise enough, since we wanted a term that would stand for any type of buggy knowledge, whether misconception, misinformation, skill-bug, etc. The term "Mis-KU" did not cause confusion, probably because it did not bring competing interpretations to the user's mind.[45]

Semantic Uncertainty

In analyzing the documented instances of terminological uncertainty from the case study we found two useful distinctions, one between ambiguity and fuzziness, and the other between intentional and inherent uncertainty.[46] Terms can be uncertain because they overlap with other terms ("ambiguity") or because the boundaries of the term are unclear ("fuzziness"). The **ambiguity** problem manifests itself in questions such as "Is this thing an X or a Y?"—where "this thing" is something you are trying to classify. It occurs when there is more than one term available for similar things. The **fuzziness** problem manifests itself with questions like "What is an X?" and occurs when the uncertainty does not concern similar competing terms. Example questions revealing fuzziness are: "What is supposed to go in the elaboration slot?" and "What is a composite knowledge type?"

Uncertainty in the conceptual vocabulary was either "intentional" or "inherent". **Intentional terminological uncertainly** occurs when a term is created with intentional fuzziness or ambiguity.[47] As an example of intentional uncertainty consider the term "motivation." Prior to and during this study, the following ways to motivate or introduce a topic were identified:

---

[45] In hindsight, there are some KAFITS terms we would like to have changed, but the effort to re-tool the code and teach the new term to the expert in the midst of the study was prohibitive: "K-bug" (knowledge bug) would be more self-explanatory than "Mis-KU;" "situation" is more precise than "example;" "task" is more precise than "question." We think these new terms would work as well or better than the existing terms, but will not know unless they are tried.

[46] There are more rigorous ways of categorizing semantic uncertainty, but we will keep it simple here.

[47] It may appear unusual or sound self-serving to say that any uncertainty that is not intentional is inherent. Actually, any particular instance of ambiguity or fuzziness could probably be engineered out, perhaps at a cost of increased complexity or by being more esoteric. But there will always be some uncertainty in terminology, and getting rid of it in one place moves it elsewhere (for example, replacing a term with several more precise terms takes care of uncertainty at one level, but the new terms will have their own uncertainty)—*some* amount of uncertainty is inherent.

- Playing: allow the student to play or explore in order to get an intuitive feel of the domain from a concrete context;
- Reminding: remind the student of previous knowledge (and relate it to the current topic);
- Enticing: pose a question or describe a phenomenon that instills curiosity;
- Persuading: motivate the need to learn by posing a problem that is not solvable without new knowledge;
- Advanced organizing: give the big picture or road map; and
- Epitomizing: give a concrete prototypical example or anecdote which epitomizes the topic (this term borrowed from Reigeluth [1983b]).

But only one slot, Motivation, was implemented to cover all of these possibilities. We mentioned all of the possible senses of Motivation to Camp, and left it up to him to decide what to put in Motivation slots. Similarly the Wrap-up topic slot could be used to conclude a topic by elaborating on it (giving incidental information, e.g. "by the way..."), comparing or contrasting it to other topics, summarizing the content or purpose of a topic, or cleaning up "white lies" (dealing with previous simplifying assumptions)—but only the Wrap-up slot was implemented and covers all of these possibilities.

The above examples involve intentional uncertainty, where uncertain in terms' precise meaning allowed for flexibility of its contents and uses. Intentional uncertainty is useful when the domain expert does not need to explicitly distinguish between the meanings of a term in a tutoring strategy. As an example of a case were a term *was* refined into several more precise terms, consider "feedback," which was refined to allow for strategies that differentiated types of feedback. Early versions of KAFITS had one term, "feedback," for what was eventually refined into "reaction," "give-away," "reason," and "elaboration."

There were also cases of **inherent terminological uncertainty**. For example, the distinction between topics and Mis-KUs was not always clear. A "common misconception"

to one teacher may be an "important topic" to another. Camp decided to call {Force-on-vs-by-confusion} (confusing forces *on* objects with forces *by* objects) a topic, yet called {Tension-on-vs-by-confusion} a Mis-KU. His reasoning was that the former was a topic he assumed every student needed to be taught, while the latter was something to be remediated when a student was diagnosed as having the misconception. Similarly, there were times when Camp was not sure whether the feedback he wanted to give the student was an answer-reason or an elaboration. Similar uncertainty was documented with the terms motivation vs. summary, summary vs. definition, and concept vs. principle. Though each individual instance of terminological uncertainty could probably have been engineered out, some degree of uncertainly is unavoidable, and the exact meanings of many terms will come only through common understanding from a dialog between domain expert and knowledge engineer.

Level of abstraction. Uncertainty in terminology is often closely relate to design decisions about the level of abstraction or generality of terms in a system. For instance, we believed that having fewer first class object types in the KAFITS framework facilitated understandability and extendibility, and also made the KAFITS interface simpler. For example, we decided that the general term "motivation" was more appropriate than its refinements, and that "feedback" was *too* general and needed refinement. A related issue is deciding what entities in the system will be "first class objects." Topics are first class objects in the KAFITS system. We chose not to have "fact," "procedure," etc. (the knowledge types for topics) be first class objects. Rather, the knowledge type is indicated by a topic slot.[48] (See the discussion of mixins in Section 3.1.4.) In brief, the designer must balance terminological precision with usability when deciding the appropriate level of abstraction of terms.

Next we discuss another significant source of terminological uncertainty, the conflation of declarative and procedural knowledge.

---

[48]We also limited the proliferation of first class objects in the following cases: Mis-KU objects were used rather than types of mis-knowledge ("misinformation," "procedural bug," etc.); and "example" objects were used rather than types of examples (extreme cases, analogies, counter examples, etc.).

Interaction of Declarative vs. Procedural Knowledge

In a sense, the "meanings" of terms depend fundamentally on how they are used. In AI systems this concerns the distinction between declarative and procedural knowledge. Ideally, ITS designers (and the designers of all expert systems) try to represent declarative and procedural knowledge separately. This is done with the working assumption that declarative and procedural knowledge are independent, that a piece of knowledge can be represented without regard to how it is used, and that a piece of knowledge can be used in multiple ways. This assumption is exactly what gives KAFITS its flexibility and power—for example, a topic is represented in a way that allows it to be used to teach, summarize, give examples, etc. However, we found much evidence attesting to the fact that declarative and procedural knowledge are highly interdependent. Specifically, the meaning of a term (or slot) can not be separated from how it is used—in fact one can make an argument that quite the opposite is true: that a term's meaning is *completely* determined by how it is used.

For example, Summary is a topic slot that contains text. We could have called the slot "topic-slot-7," or "foobar," but "summary" is a more useful term because it reminds the user of its *intended* meaning.[49] What does "summary" really mean? It depends on how strategies *use* it.[50] Strategies could conceivably use it in several ways: to summarize after teaching the topic, to preview the topic, to provide a stand-alone overview of the topic, or to review a topic after it was taught.[51] Each of these potential uses provides a different sense of the term "summary," and that meaning affects what the teacher specifies as the slot's content. An "official" meaning of "summary" might have been put in a user's guide or on an on-line help system, but in our case it was a shared meaning stemming from conversations between the domain expert and the knowledge engineer.

---

[49] Lenat et. al (1981, pg. 71) calls inappropriately reading meaning into symbol names a "knowledge illusion."

[50] Actually a slot name's meaning depends on how strategies use it *and* on how it is related (implicitly or explicitly) to other bits of information in the knowledge base.

[51] That there are so many potential meanings of the term might indicate that we should have used a more precise term (or several) in its place.

There were several other instances of procedural vs. declarative interdependence documented in the case study. On different occasions the meanings of the Summary and Elaboration slots were unclear to Camp without reference to how they were used in the default strategies. There were also instances of co-dependence between two or more meanings: the meanings of Motivation and Prerequisite depend on which comes first (i.e. whether prerequisite means prior to motivating a topic or prior to teaching it); the meanings of the Summary and Definition slots were seen by Camp as co-dependent because both summary and definition could be given while teaching a topic, so they should not say the same thing.

Summary of Tradeoffs in Designing the Conceptual Vocabulary

One of the main design issues of the KAFITS conceptual vocabulary was the uncertainty (i.e. fuzziness and ambiguity) of its terms. Some degree of terminological uncertainty is unavoidable, but in specific cases it can be repaired by replacing a term with a clearer one or replacing a term by several more precise ones. However, the introduction of esoteric terms or unchecked expansion of terminology has negative effects on usability, so new terms should not be introduced unless a clear need is perceived. Also, some terminological uncertainty is intentional and allows flexibility because multiple senses of a term allow multiple uses.

One source of terminological uncertainty is dependence of (declarative) meaning of terms with how they are used (procedurally). Although independence of declarative and procedural knowledge is a useful working assumption, determining the meanings of attributes in a computer system often defies this assumption. And though the separation of declarative and procedural knowledge is a crucial aspect of the KAFITS system, we found ample evidence in the case study that the meaning of slots has some dependence on the strategies that access them. Fortunately, even though there were instances where procedural knowledge was intertwined with declarative knowledge, the working assumption of their independence was still used and useful.

Usability implies sufficient simplicity and understandability. Problems with conceptual terms often stem from confusion with other terms in the conceptual vocabulary, or with

the mundane use of words. Ultimately, empirical data or experience is needed for any determination of usability.

5.4.3 Flexibility and Modularity in the Domain Knowledge Base

The previous section on conceptual vocabulary and semantic uncertainty dealt with the *general meaning* of KAFITS slots (and other terms). In this section we discuss issues concerning the *contents* of the slots—specifically, how the contents can be designed to allow for the potential flexibility inherent in the KAFITS framework.

To take advantage of the power of multiple tutoring strategies the domain knowledge base must be designed with a high degree of flexibility. There are two types of flexibility involved: "object sequencing" and "action selection." **Object sequencing flexibility** refers to the diversity of potential paths through a space of similar objects. For example, a variety of strategies can be designed for traversing the topic net (a space of topics and Mis-KUs), and (though we did not take advantage of this capability in the statics tutor) a variety of strategies for traversing a space of presentation objects (for example, a space of positive and negative examples). **Action selection flexibility** refers to the capability to choose and order primitive actions in numerous ways according to strategies. Another way to think about action selection flexibility is to say that the objects in the knowledge base are represented so that they can have many uses (eg. Mis-KUs can be remedied and diagnosed).

If a topic or presentation is written in a way that assumes a certain instructional context (what comes immediately before it in a tutoring session) then it is less flexible; and when it is given to the student in a context that was not intended it will not make sense. Therefore, topics and (to a lesser extent) presentations should be relatively modular and context-free—independent of sister objects, when possible. But it is difficult to design the knowledge base contents so that the flexibility allowed by multiple tutoring strategies can be taken advantage of. As mentioned previously, human knowledge is not easily repre-

sented in modular, independent chunks. Several issues have been identified: granularity, interdependence, structure, and discourse flow; each is discussed below.

Knowledge Granularity

Our answer to the question "How big is a topic?" is based on pragmatics rather than epistemology: a topic should be whatever size the expert needs it to be, i.e. if the expert wants to refer to a chunk of knowledge (for example as a prerequisite of another topic) the system should have an abstract representation for it. Conversely, even if a topic could technically be broken into many parts, this should not be done if the whole topic is the only thing that is ever referred to.[52] The KAFITS system has three mechanisms for chunking domain knowledge: the lesson object, topic levels, and the Parts attribute, each described below.

The **lesson object** (see Section 3.1.2) is the first chunking mechanism. Topics can be grouped arbitrarily in a lesson, and, by specifying a teaching strategy in the lesson object, they can be taught with a particular pedagogical style or perspective. [53]

Topic levels, distinguishing levels of performance and mastery within a topic, are the second mechanism for dealing with grain size. Slots and strategies can refer to these levels individually, such as in teaching only the use-easy level of a topic. Without levels within topics, entire topics would have to be created to be able to refer to each level of performance/mastery.

A third mechanism for chunking is the **Parts topic attribute**. Using this attribute, topics at any level of curriculum generality can be represented and the topic network can have hierarchical relationships of arbitrary depth (eg. we can create topics as general as "introductory statics," and as specific as "equilibrium of static forces in the x direction for

[52] Often it would be possible to re-interpret the presentations or levels within a topic as entire topics.

[53] We did not make much use of the chunking capability of lesson objects in the statics tutor, but as an example use: four different lessons could be created containing the same group of topics but taught in introductory, review, qualitative, and quantitative perspectives, by specifying different tutoring strategies in each lesson.

two simple bodies"). If the knowledge that a parent topic refers to is more than the sum of its parts, then the parent topic contains instructional material, otherwise the parent topic is empty (it only points to its part topics); teaching it is equivalent to teaching its parts, and knowing it means knowing its parts.

Normally the parts of a topic should be of the same type as the parent topic. For example, a concept has conceptual parts, and a procedure has procedural parts. But some groups of topics that the domain expert wanted to refer to could not be categorized according to knowledge type because they were the union of several different types of knowledge. To address this we created a node type called *composite*. The parts of a composite topic can be of any topic type (eg. {Linear-Equilibrium} has parts {Linear-equilibrium-concept}, {Linear-equilibrium-intuition}, and {Linear-equilibrium-principle}).

Knowledge Modularity vs. Interdependence

To allow flexibility, topics should be designed modularly, with all dependences on other topics made explicit (as in prerequisite relationships)—Lesgold (1988) calls this "internal coherence."[54] Though our goal was to design topics modularly we were often faced with situations in which topics were inherently interdependent or overlapping. For example, the student has to know something about {Types-of-forces} to fully understand {Linear-equilibrium}, yet some understanding of {Linear equilibrium} is prerequisite to learning about {Types-of-forces}. These two topics are prerequisites of each other, an untenable situation for most ITSs. Also, some knowledge is about the relationship between two topics and should not be contained in either one of the topics exclusively. The KAFITS framework has two mechanisms for dealing with topic interdependence: synthesizers and spiral teaching, each described below.

[54]The same is true for presentations, which are more flexible if written independently of other presentations. However, one can safely assume a fair amount about presentation order within a topic level. In addition, in some cases the relationships in a section of the topic net so tightly constrain the ordering of topics that one can make assumptions about topic order.

Some of the curriculum concerned *relationships* between two topics and could not properly be associated with any single topic. Lesgold [1988] emphasizes the need to provide curricular "glue" to relate topics. The Motivation and Wrap-up topic slots (as well as the Introduction and Conclusion lesson slots) are appropriate places to mention how topics are related, but this relationship is only salient for the topic whose slot has the information. Reigeluth [1983b] uses the term synthesizer for instructional components used to interrelate and integrate individual content units. KAFITS includes a **synthesizer node type** which points to two or more topics and compares or contrasts them. Like the composite node type, it is not a knowledge type and was created to address a representational need. In the end, synthesizers were not used for the statics curriculum (Motivation and Wrap-up slots were used to compare topics), but we give two hypothetical examples of how strategies could use synthesizer objects. One possible strategy relates two topics the student has learned: after a topic is taught check whether a synthesizer connects it with another known topic, and if so, teach the synthesis material. Another possible strategy connects new information with existing information before the new information is presented: before a topic is taught, check whether a synthesizer connects it with an already known topic and, if so, present the synthesis material.

Spiral Teaching

In classroom teaching and one-on-one tutoring topic interdependence is often dealt with using some form of "spiral teaching" [Van Heuvelen 1987]. In the prototypical example of spiral teaching several topics are presented at an introductory level, and then again at a more advanced level, and so forth, as many times as necessary. We have had some success in implementing spiral teaching in KAFITS, by utilizing the levels of performance/mastery and the ability to encode different levels of prerequisites. As Lesgold [1988] points out: "the concept of prerequisite [relations between topics in tutoring systems] has been inadequate in the past." KAFITS incorporates a refinement to the typical prerequisite relationship by allowing *levels* of prerequisites (corresponding to the different types of prerequisite links in Figure 1.3, see Section 3.1.5).

Using this refined encoding of prerequisite relationships, along with the ability to encode topic part-whole relationships, KAFITS produces spiral teaching.[55] For example, {Linear-equilibrium} has {Linear-equilibrium-intuition} and {Linear-equilibrium-principle} as parts. {Linear-equilibrium-intuition} has {Linear-equilibrium-principle} as a familiar prerequisite. When {Linear-equilibrium} is taught it first starts to teach {Linear-equilibrium-intuition}, which in turn needs {*Linear-equilibrium-principle*} to be taught at the *familiarity* level. Later (after {Linear-equilibrium-intuition} and {Linear-equilibrium-concept} have been taught), {*Linear-equilibrium-principle*} is revisited, this time taught at the *easy* level.

Alternative Curriculum Structures

Many structures for curriculum have been proposed in the literature: networks, tables, trees, scripts, etc. When humans teach they are free to organize topics in any way and use any strategy to traverse this organization, and they can choose different structures for different needs. However, in the KAFITS system the method for structuring curriculum must be consistent to reduce complexity in the representational framework and the resulting confusion for users.[56] The basic curriculum structure of the KAFITS system is flexible: topics (and Mis-KUs) are arranged in a network; the topics are of different types; the relationships that link the nodes of the network are of different types; and each topic has levels of understanding. The specific topic types, link types, and level names used in the statics tutor were developed to be general, but are prototypes, likely to change when a sufficiently different domain is represented. Below we discuss alternative methods of structuring curriculum and how these fit into the KAFITS system.

Some curriculum structures are hierarchical or tree-like. For example Ausubel's [1963] "meaningful learning" theory suggests that each discipline consists of sets of hierarchically organized concepts. Burton's [1982] Buggy system uses a *lattice* to represent subsumption

[55]The resulting tutoring is similar to a breadth-first traversal of the topic network.

[56]However, with a given structure, strategies can be designed to traverse that structure in many ways.

relationships between skills and sub-skills. Tree and lattice structures are kinds of networks, so tree and lattice structures are easily implemented in KAFITS by using the "part" link. Some curriculum structures use a prerequisite network structure, such as Gagne's [1985] "essential prerequisites" and the BIP system's "curriculum information net" [Barr et al. 1975]. Goldstein's [1982] genetic graph is a network structure which represents analogy, refinement, generality, and buggy relationships between procedural rules. These are all network structures, so they could also be implemented using KAFITS by implementing the appropriate link types.

The methods above do not sufficiently account for the overlapping or multi-perspective nature of knowledge. Lesgold's [1988] "curriculum goal lattice" accounts for the fact that there are many views on the same curriculum material. He shows how curriculum for an electricity tutor can be taught from the perspective of circuit types, electricity laws, electricity concepts, or problem types. Though teaching from each perspective gives an approximately identical set of student explanations and tasks, each perspective results in a different ordering of the explanations and tasks. KAFITS has the capability to model this behavior, since it has lesson objects "above" topics which can specify high level goals and strategies, and presentation objects "below" topics, which can be referenced by more than one topic. However, though the goal lattice can be implemented using the KAFITS framework, we relied on the spiral teaching method mentioned above to account for topic overlap in the statics curriculum.

Statics Domain Dimensions

During the statics tutor case study many curriculum structures were discussed (the discussions being initiated by both the domain expert and the knowledge engineer) that constituted different views (or perspectives or dimensions, as in Rissland et al. [1984]) on the curriculum material. Most of these were not implemented in the course of the case study but are worth mentioning because they illustrate the multidimensional nature of curricula and how KAFITS can represent these curricula structures. For instance

PEDAGOGICAL DIMENSION	LEVELS OR VALUES
Intuition/Application	force existence, force direction (opposite), force magnitude (equality), write equations, solve equations (quantitative)
Difficulty/Complexity (or mastery)	easy, typical, difficult problems
Problem Types/Contexts	hanging objects, falling objects, rolling objects, inclined planes, colliding objects, exploding objects, etc.
Force Type	tension forces, gravity forces, rigid body contact forces, compressible body contact forces, etc.
Spatial Orientation	horizontal forces, vertical forces, all orientations, rotational (torque)

Figure 5.8 Perspectives for Teaching Newton's Third Law

there are many approaches to teaching that bodies exert equal and opposite forces on each other, i.e. {Newtons-third-law}. Figure 5.8 shows several perspectives (dimensions) and levels of presentation for each perspective.[57] Each of these perspectives has substantial overlap with others, and theoretically each perspective could be a topic, for example: {Qualitative-newtons-third-law}, {Quantitative-newtons-third-law}, {Understanding-inclined-planes}, {Understanding-collision-situations}, {Horizontal-forces}, etc.[58] It would be exceedingly difficult to represent the curriculum so that {Newtons-third-law} could be taught from *all* of these perspectives, but it is possible to incorporate them in a limited way.

One way to implement these dimensions in tutoring involves conceptualizing the curriculum in terms of a table, with one of the dimensions in the rows and another dimension in the columns. For example, for a table containing force types vs. problem type, teaching could involve moving across the rows and columns according to some strategy, such as teaching all force types for each problem type (see Figure 5.8). (This scheme can be extended from a two dimensional table to an array of arbitrary dimensions.) Though conceptually simple, tabular representations of curriculum are restrictive. For example, a straightforward method for simulating spiral teaching, one we did not adopt, involves a tabular representa-

[57] The dimensions listed are not arbitrary; they all have pedagogical relevance which we will not explain in detail here. In classroom instruction many divergent examples of a concept are usually necessary to promote a deep understanding. Multiple views and contexts are useful for teaching a single topic because students' learning is context dependent.

[58] White & Frederiksen's [1986] model evolution instructional strategy is similar, teaching the same material from increasingly more sophisticated perspectives.

tion of curriculum with one dimension (rows) for topics and the other (columns) for levels of difficulty. In this method spiral teaching involves weaving back and forth over all topics from simple to more difficult levels. Though workable, this method is too restrictive, and much of the flexibility of network representations is lost when a tabular representation is used, and topics would be confined to a rigid ordering.[59] As in our implementation of spiral teaching, the topic levels of KAFITS could be used (if they were renamed for some topics) to incorporate multiple perspectives in a limited way.

Discourse Flexibility vs. Context Dependence

As stated above, the tutor's flexibility is directly related to the modularity and independence of the knowledge base contents. We have mentioned that *complete* modularity and independence is not possible, so the goal of the knowledge engineer is to achieve, or to help the domain expert achieve, the highest degree of context independence feasible, given the difficulty of the task. We found that, in practice, it is often difficult for the domain expert to design curriculum that assumes no global discourse context because to do so he must consider many alternative paths through the topic net and alternative permutations of the primitive actions (see Section 5.5 for a discussion of cognitive factors). For instance {X-axis-forces} is a part of {X-axis-equilibrium} and is also a part of {FDB-WRITE-EQUATIONS} (as shown in Figure 1.3), so {X-axis-forces} could be reached from two different paths, each having a different discourse history. Also, prerequisite relations can ensure that one topic has been taught before another but not that the prerequisite topic was presented *immediately* before (it could have been taught at any time in the past). As a third example of the difficulty of context independence, the domain expert was advised that the answer-description could not assume that hints were given, since some strategies do not give hints.

---

[59] Actually there is some flexibility inherent in the tabular method—two teaching strategies are possible. Moving straight down a column results in teaching a single topic at all levels of difficulty (i.e. depth first traversal); moving straight across a row results in teaching all the topics at the same level of difficulty (i.e. breadth first traversal).

Designing *too much* modularity into the knowledge base can result in abrupt or choppy tutorial discourse. Making every chunk independent can (at least with canned text systems) lead to repetition and can prevent smooth transitions between ideas (topics). We have taken a pragmatic approach to this tradeoff, resulting in a compromise between curricular flexibility and smooth discourse: designing with modularity is encouraged, but not required of the domain expert. The domain expert is not expected to test every conceivable network path and combination of primitive actions but, rather, tests many paths with several tutorial strategies and adjusts the knowledge base for smooth discourse. This approach worked well in this study, and student trials indicated that the tutorial discourse for the statics tutor was reasonably smooth.[60]

A related problem is that when a *change* is made to the domain knowledge base, the domain expert must keep in mind the many ways that the modified information could be used, including how it might affect nearby topics or presentations, and various potential trajectories through the topic net. Potentially even more difficult is changing or adding tutoring strategies. Though the domain expert tries to consider different possibilities when designing the curriculum, his vision of the range of possible situations will be somewhat limited by what he knows of the existing strategies. For example, after the third set of student trials Camp noted that it would be feasible for advanced students to skip all the easy level presentations. This would require that the easy and typical level presentations would have to be written so that the easy level could be skipped and teaching could start at the typical level. We never implemented this strategy change, but it is possible that many of the presentations would have had to be edited had we done so.

5.4.4 Summary of Knowledge Representation Issues

In this section we first described how the original KAFITS framework (especially the conceptual vocabulary) grew out of experience doing knowledge engineering with educators,

[60]However, we did not experiment with very diverse strategies in this study, therefore the degree of difficulty in designing smooth discourse with more elaborate strategies is an open question.

Knowledge Representation Issue or Problem	DESIGN TRADEOFFS			Solution or KAFITS Mechanism
Vocabulary Understandability	Minimize overly abstract or technical terms	–vs.–	Maximize flexibility, clarity, expressiveness	Analysis of empirical evidence of confusion
Vocabulary Uncertainty	Minimize ambiguity and fuzziness	–vs.–	Some uncertainty is unavoidable and some is useful and intentional	Allow ambiguity that provides flexibility without confusing the user
Vocabulary Simplicity	Minimize size and complexity of vocabulary	–vs.–	Maximize clarity and expressiveness	Introduce new terms only when need is demonstrated
Level of Abstraction	Minimize proliferation of first class object types	–vs.–	More first class object types increases expressiveness	Use slots to specify some sub-categories
Knowledge granularity	Knowledge is chunked at many levels of abstraction			Lessons, topic levels, Part slot, composite topic type
Modularity & Interdependence	Maximize knowledge modularity	–vs.–	But some interdependence is unavoidable	Synthesizers, spiral teaching, "glue slots" (Motivation, Wrap-up)
Discourse Flexibility & Context Dependence	Maximize smooth flow of discourse transactions and ideas	–vs.–	Maximize flexibility and context independence	Interface allows easy movement between testing and modifying

Figure 5.9 Summary of Knowledge Representation Design Tradeoffs

analysis of taped tutorial sessions, and the instructional design literature. We then described knowledge representation design tradeoffs, documenting examples from the case study when possible, and gave suggestions or mechanisms which address knowledge representation issues. The issues were organized into two categories: conceptual vocabulary/semantic uncertainty, dealing with the general meaning of KAFITS terms, and curriculum flexibility/modularity, dealing with the contents of the KAFITS knowledge base. The table in Figure 5.9 summarizes the knowledge representation issues identified, the design tradeoffs involved, and KAFITS mechanisms that constitute partial solutions. We also gave evidence that the declarative meaning of terms can depend on how they are used procedurally.

We mentioned several alternative curriculum structures, including knowledge hierarchies, sub-skill lattices, prerequisite networks, genetic graphs, curriculum goal lattices, and curriculum views (pedagogical dimensions), and indicated how each could be implemented within the KAFITS framework. We also showed how the statics curriculum could be taught from a number of pedagogical perspectives.

5.5 Cognitive Considerations, Interface Design, and User Participation

Some of the issues and tradeoffs identified in this study are *epistemological* in nature, related to fundamental characteristics of knowledge such as terminological fuzziness and ambiguity, and the massively interconnected nature of knowledge; these epistemological issues were discussed in Section 5.4. Other issues identified are *psychological* in nature, having to do with human cognitive capabilities and limitations. In this section we document issues related to the domain expert's cognition and discuss how cognitive considerations affected the design of the KAFITS interface and the knowledge acquisition methodology. We also discuss the importance of user participation in designing KAFITS.

5.5.1 Procedural Nature of Curriculum Mental Models

Norman [1983, pg. 7] makes a useful distinction between conceptual models and mental models. A conceptual model is a model that a designer or teacher invents in order to provide an appropriate (accurate, consistent, and complete) representation of a system to a user or student. A mental model is a naturally occurring evolving property or structure in a human mind. Mental models are by their nature incomplete, unstable, and runable; and are not necessarily accurate. Below we discuss one case where a KAFITS conceptual model was at odds with the domain expert's mental model.

Normally teachers are exposed to predominantly *linear* representations of instructional material, such as in textbooks and lesson plans, and the experience of teaching itself is, of necessity, composed of a linear ordering of events. Thus it is not surprising that we have found that teachers' mental models of the instructional process tend to be linear and procedural.[61] [62] This is in sharp contrast with the declarative nature of domain knowledge represented in intelligent tutors. For instance, we have found that perceiving the KAFITS topic net as a *declarative* representation of relationships between topics is difficult for those who, like most teachers, are not trained in computer science.[63] There is a persistent tendency for teachers to interpret the network as a *procedural* specification of topic ordering.[64] Camp himself noted this tendency: "It's hard to get away from linear types of notions [about teaching, when one is] teaching [in the] classroom all the time" [7/5/89].

---

[61] Observations concerning teachers are from experiences working with over a dozen high school and college teachers over several years.

[62] Garg-Janardan & Salvendy [1988] report similar findings (in the expert systems field) of users confounding process (procedural) and content (declarative) knowledge. They hypothesize that the expert's internal schema, compiled over a history of solving problems, contains process and content knowledge, and suggest that it is not feasible to attempt to structure knowledge elicitation to elicit only one or the other.

[63] The network is (in part) a set of *prerequisite constraints* and hierarchical relationships between topics, not a specification of how topics will be ordered when tutoring.

[64] They seem to have little trouble accepting an explanation of the declarative nature of the network when it is presented to them, but fall back into procedural interpretations of it when designing tutors.

We observed four phenomena related to the difficulty of assimilating the declarative nature of the topic network, described below. All of these occurred in the first months of the study, after which Camp had a deeper understanding of the declarative nature of the domain knowledge.

Confusing parts with prerequisites. We noted several instances where Camp interpreted the relationship between two topics as "topic X needs to be taught before topic Y" and used a *prerequisite* link when the topics actually had a part-whole relationship and the *part* link was more appropriate.[65]

Confusing evidence with action. On one occasion, while we were discussing Mis-KU values in the student model, Camp said "so if it's confirmed, then we go remediate it, right?" He forgot for a moment that Mis-KU evidence in the student model does not imply branching immediately to remediate, but that this information may (or may not) be used at some later point to cause the Mis-KU to be remediated.

Inversion of a subordinate topic relationship. Figures 5.10 and 5.11 show non-optimal topic relationships as drawn by Camp (on the left), along with the more appropriate representation that was eventually included in the curriculum (on the right). Topic {Gravity} is subordinate to (i.e. a part of) topic {Forces-at-a-distance}, as shown to the right of Figure 5.10. Normally topic parts are taught before topics, so {Gravity} should be taught before {Forces-at-a-distance}. Camp's original sketch of the relationship between these topics is shown on the left in the figure. He was thinking of the topic net procedurally, and put {Gravity} before {Forces-at-a-distance}.[66]

Linearization of a subordinate topic relationship. {X-axis-forces} and {Y-axis-forces} are subordinate to {FBD-write-equations}, as shown to the right of Figure 5.11. Normally the first topic listed in the Part slot to be taught first, and Camp wanted {X-

[65]The difference is significant because strategies can specify that parts and prerequisites be ordered differently.

[66]Similarly, beginning computer science students often have difficulty conceptualizing that in depth first network traversal sibling nodes are evaluated before their parents, even though the search starts with the parents.

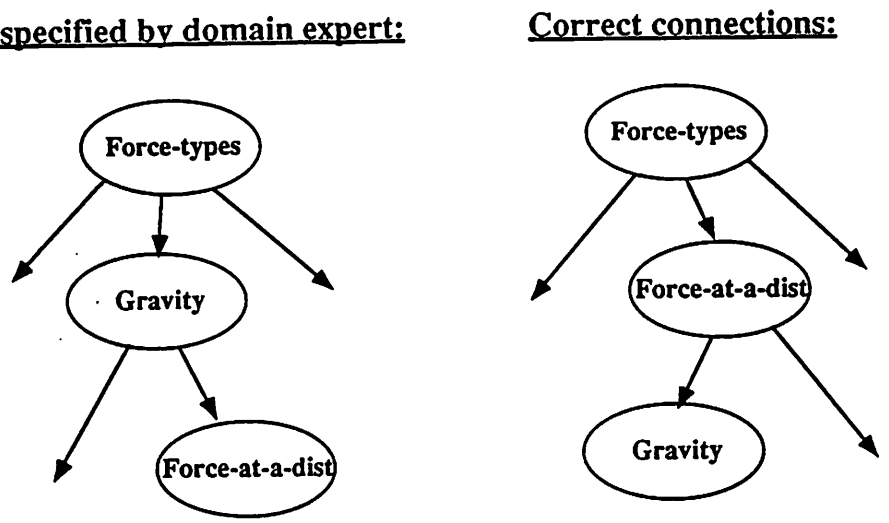


Figure 5.10 Inversion of Subordinate Topic Relationship

axis-forces} is taught before {Y-axis-forces}. His initial sketch of this relationship is shown on the left in the figure. Again he was thinking of the net procedurally, showing the topics in the order they would be presented to the student.

The above discussion underscores the importance of considering the compatibility between the cognitive model offered by an interface or representational framework, and the mental models that users bring with them to the task. If enough information is available this compatibility should be considered during system design. In any case, software evaluators should be aware of incompatibilities between cognitive and mental models. In some cases, such as when the user has a misconception, the best approach to the incompatibility may be to instruct the user in a new model, rather than alter the interface to reflect the user's model. This was the case for the domain expert's tendency to see a declarative network as a procedural network.

5.5.2 Cognitive Factors in Curriculum Size and Complexity

In Section 5.4.3 we discussed how tradeoffs between domain knowledge modularity, knowledge overlap, and discourse smoothness made the domain expert's job more difficult. We found that the domain expert was more often frustrated by the sheer *size* of the

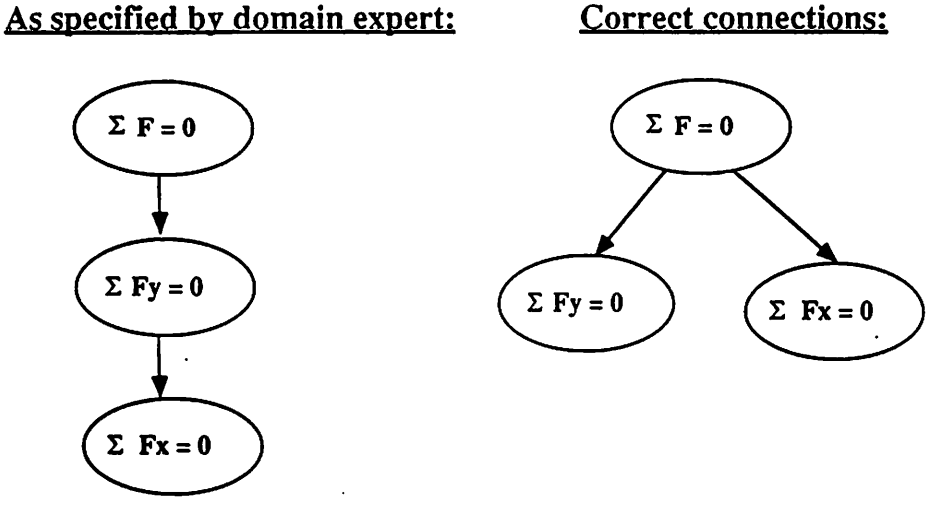


Figure 5.11 Linearization of Subordinate Topic Relationship

knowledge base than these epistemological issues. In addition, it was observed that issues of curriculum size and complexity were more pronounced than conceptual vocabulary issues.[67] Here we document how cognitive factors relate to the size and complexity of the knowledge base.

For most of the first two and a half months of the project Camp worked several days a week and arrived at the final version of the topic network only after repeated revisions. He had constructed a complex mental model of the curriculum, much of which involved subtle reasoning and distinctions about the meanings of topics, their boundaries, and their relationships (such as the meanings of and difference between {X-axis-forces} and {X-axis-equilibrium}). For most of the later months of the project Camp worked weekly, often focusing on a small part of the network for a long period of time; some the details of the curriculum faded from his long term memory or were not immediately accessible without reconstructing his reasoning process. He often found it difficult to "get his head around" [his words] the entire curriculum, i.e. maintain an accurate mental model of it. Figure 5.12 has several quotes that illustrate Camp's frustration with the magnitude of the curriculum he was designing (see also the quotes in Section 5.3.1).

[67] "The biggest problem for Camp is not with the syntax or semantics of the framework, but with the semantics of the [curriculum]" [lab journal 3/22/90].

- "There are so many fuzzy little pieces" [6/26/89, referring to breaking {Linear-equilibrium} into parts].
- "I get really discouraged trying to think of which is the *best* way to [organize the topic net]" [7/5/89].
- "Part of what's taken a lot of time is all this checking back [to previous work sheets and how] to make [this work sheet] different or consistent with [previous ones]" [8/3/89].
- "One of the big issues is: How does the person dealing with the content figure out where they are?" [10/25/89].
- "It's so easy to fall off the wagon with some of this stuff ...I start putting stuff in [{Rigid-bodies}] that belongs in [{Newtons-3rd-law}]" [3/6/90].

Figure 5.12 Quotes about Curriculum Size

- "Oh God! we can't do everything for everybody here! Argh!!" [7/30/89].
- "I keep thinking there is some awful path in there that is just insane, where if someone stepped in there they would get garbage" [8/1/89].
- "No matter what way they go though here, fastways, backward, forward—you have to make sure it makes sense!" [8/8/89].
- About test running the tutor to debug the curriculum: "After two or three wrong answers I get my brain sort of unhinged as to where I am" [11/14/89].

Figure 5.13 Quotes about Modularity

A related problem, mentioned in Section 5.4.3, is the difficulty of designing the knowledge base modularly, trying to account for the many possible paths through the curriculum and various uses for each object. Keeping track of all of these possibilities puts strong demands on working memory. The quotes in Figure 5.13 illustrate Camp's occasional frustration with these demands.

In summary, the curriculum is large and complex, and in order to design or modify it the domain expert needs to maintain an accurate mental model. This complex mental model can fade (become less accessible or less accurate in long term memory) if is it not used (accessed) regularly. In addition, designing and modifying the curriculum requires having

large parts of this mental model in working memory to reason about multiple interactions between pieces of it, and this is sometimes challenging. In the next section we describe aspects of the KAFITS system and our knowledge acquisition design process designed to mitigate these and other problems.

5.5.3 Design Principles

Our goals were to design the KAFITS system and the knowledge acquisition process for power, efficiency, and ease of use. We have discussed some of the many design tradeoffs and cognitive considerations involved in meeting these *design goals*. Here we abstract a set of *design principles* which describe how we addressed the design goals and which also serve as recommendations for future investigations of ITS knowledge acquisition interfaces and methods. We list these design principles below, and in succeeding sections expand upon them, giving examples of how they were followed.

1. **Cognitive fidelity of the interface.** The interface should provide the user with a clear and accurate cognitive model of the underlying framework.

2. **Accessibility and management of the domain knowledge.** The user should be able to view the (static) knowledge base in multiple ways and easily navigate among related items.

3. **Monitoring and testing the knowledge base.** It should be easy for the user to test and modify the knowledge base and monitor dynamic processes and data structures.

4. **Assistance and efficiency of the interface.** The interface should have features which facilitate efficient work and provide assistance when needed.

5. **Managing the curriculum development process.** The knowledge acquisition process should facilitate both the on-line and off-line aspects of building a tutor throughout all phases of the project.

Cognitive Fidelity of the Interface

In designing the KAFITS framework we tried to follow this "conceptual simplicity design guideline": no conceptual aspect of the framework should be so complex that it can't be clearly represented visually in the interface.[68] Though this guideline was not followed completely, it helped us keep the representational structure simple. Visually reifying the structure of the framework through the topic network display and the Browser reduces the user's working memory load by "off-loading" information to the interface that would normally take up space in the user's working memory, and reduces demands on long term memory because the interface serves as a reminder of the underlying representational structure. The following concepts and structures were given visual counterparts in the interface:[69]

- The structure of the curriculum and relationships between topics is reified by the topic net graphical display;
- The topic level displays illustrate that topics are composed of several levels of understanding;
- The hierarchical organization of information (i.e. the types, instances, and slots) is reified by laying out adjacent type, instance, and slot tables in the Browser, with the instance table visually changing when the type table is clicked on, and the slot table visually changing when the instance table is clicked on;
- Strategies are reified by the graphical PAN displays and the switch set displays;
- The correspondence between Browser operations and the things they operate on is reified by locating pop-up menus of type, instance, and slot operations directly below the corresponding tables.

---

[68] This applies to the conceptual level of the framework. At the implementation level this restriction does not hold.

[69] These interface features relate to the static information in the knowledge bases; later we will list features related to dynamic information and processes.

Some structural aspects of the framework do not have corresponding interface realizations, most notably the four-level decision model and the structure of the student model. An interface showing the decision levels would be more important if we dealt more with multiple strategies at each of the four levels, and could be added in the future. An interface for the student model could also be added in the future, assuming a more general framework for diagnosis and student modeling is developed (as will be discussed in Section 6.4).

Domain Knowledge Accessibility and Management

The user should be able to view the knowledge base from different perspectives and different levels of abstraction and should be able to easily navigate among related items. The Referenced-by slot, and the "recent-instances," "browse-slot," "browse-instance," and "describe-connections" browser operations allow the user to take note of and/or access instances that are connected to the instance currently being worked on in the Browser. The following features gave the user a variety of levels of granularity from which to view the curriculum: the topic net shows the entire curriculum; topic level displays show the topic levels of all current topics; the "describe topic details" operation on topic nodes gives an overview of the presentations called by each topic level; and the Browser shows the lowest level details of each topic and presentation.

Monitoring and Testing the Knowledge Base

It should be easy for the user to exercise the knowledge base, monitor how knowledge base items are being evoked during tutoring, monitor changes in dynamic data structures, and modify the contents of the knowledge base. Dynamic processes in the tutor are reified by: highlighted topic net nodes that follow the path through the curriculum, highlighted PAN nodes that follow the flow of control through strategies, and topic level displays that show movement from one topic level to the next and show dynamically changing values of the student model. In addition, the KAFITS interface facilitates easy movement between testing and modification tasks, as mentioned previously.

Assistance and Efficiency Features

The non-technical background of the assumed user and the preciousness of the domain expert's time underscore the importance of assistance and efficiency features. Several help, assistance, and efficiency features were provided, as described in Section 3.5.1, including: an on-line help/info system, disaster avoidance, data type checking, and keystroke short-cuts. In addition, the features listed in previous sections related to cognitive fidelity, knowledge base accessibility, and monitoring are all assumed to contribute to easy and efficient use.[70]

Managing the Curriculum Development Process

The knowledge acquisition method should facilitate both on-line and off-line aspects of the design, implementation, and testing phases. Two issues we encountered are worthy of note: off-line support and file management, each described below.

Work sheets. When the statics tutor project began, we had envisioned much of the design work being done on-line. However, we found that off-line tasks were sometimes more appropriate, and that off-line support artifacts were needed.[71] For instance, we envisioned the domain expert working with the Browser to design the curriculum in much the same way that a writer uses a word processor to design a document. But it soon became clear that a paper worksheets would better fit our needs for opportunistic design and working on the curriculum away from the lab (such as at home). Appendix M has samples of worksheets used. Their format went through several revisions, according to input from the domain expert and knowledge base managers (for example, the diagram sketching area to the right on the topic worksheet was suggested by Camp as a way to get a quick visual overview of the contents of the topic[72]).

---

[70] We make a tentative assumption that increased ease of use increases efficiency, but this has not been tested and may not be true for some features or for advanced users.

[71] It is conceivable that new software tools could be added that would alleviate the need for the off-line tasks and artifacts we found necessary.

[72] Camp mentioned that a graphical topic overview such as this would be helpful to have on line, but this was not implemented.

Knowledge base hard copy. At first the Saved-instances file (see Section 3.4.1) was used only to store code representations of the knowledge base objects so that they could be loaded back into the system. We thought the KAFITS interface would be a sufficient tool for all tasks involving inspection of the knowledge base. But we found that a hard copy printout of the knowledge base, formatted for readability (see Appendix I), had some advantages. It was easier for the user to see at a glance (or on adjacent pages) all of the information related to a specific topic or presentation. Also, the domain expert could take the printout home and proofread it, marking changes for later revision.

File management. Although mundane, supervision in managing knowledge base versions, paper forms, and edit records was important. Before the Saved-instances file feature was completely working, Linton spent a significant amount of "non-productive" (in retrospect) time combining and organizing edit records. Gonzalez initially started a new edit record every working day and also tried to organize editing records according to purpose, such as "new instances" and "modified instances." It was eventually suggested that she use a single edit record for a week or more, until the next version of the code-image was created, but until this happened she spent a significant amount of non-productive time managing, combining, and re-combining edit record contents. Accordingly, one of the jobs of the knowledge engineer is to supervise and provide guidance on data file management.

5.5.4 User Participatory Design

Our user participatory design process has given rich and detailed information on usability. We used this information to improve the KAFITS system and the design methodology and tried to follow two guiding implementation principles: 1. design in what the user needs; 2. don't add anything else (Occam's Razor). Below we argue for the importance of user participatory design by describing features that directly resulted from user participation as well as features that were designed *without* user participation that were not usable.[73]

[73]The prototype KAFITS system existing at the beginning of this study was in part based in several years experience working with educators (see Section 5.4.1). These experiences had a significant effect on the

Features inspired by user participation. We believe the following features would not have been conceived without user participation:

- topic level displays (Section 3.2.3),
- pop-up operations menus (Section 3.5.1),
- readable Save-instances file (Section 3.4.1),
- topic detail feature (Section 3.5.1), and
- numerous menu and Browser operations.

Features modified due to user participation. We delayed the implementation of some features until sufficient experience working with users was gained to inform the design specifics. This "wait and see" design attitude was quite useful since our conception of features changed, or we discovered that some features were not needed, *before* we expended the effort of implementing them.[74] Below we list features which were conceived without user participation, but were subsequently modified due to user participation:

- In the original framework all presentation objects had only slots for pointing to an example instance (which specified the picture, crane boom set-up, etc.) and a question instance (which specified the question, hints, elaborations, etc.). KAFITS was designed in this way for flexibility purposes, as explained in Section 3.1.2. However, this extra level of indirection proved to be very frustrating for the users, so example

---

design of the representational framework, but very little of this prior experience involved on-line work, so feedback about the KAFITS *interface* came mostly during this study.

[74] As a programmer the author has experienced the common desire to implement new program tools, add new features, and re-conceptualize representational structures based solely on his own experience using a program. Code changes, usually for purposes of generality, clarity, and/or efficiency, can be made without discretion if they affect only the implementation level, but caution must be taken if they affect the conceptual level of the system or the interface to the user. The software developer is not a typical user, and his needs and conceptualization of the software might differ radically from those of a typical user. Acting on the temptation to add new capabilities to a system (sometimes called "creeping featurism") can also drain valuable time from a software development project. Thus an "implement only as needed" attitude is recommended.

and question slots were added (via mixins, described in Section 3.1.4) to presentations. This was cited by one knowledge base manager (Gonzalez) as a substantial improvement.

- The student model was not implemented until one or two months after the project began, since during that time the domain expert had come sufficiently up to speed to provide important suggestions for its design.
- On several occasions the syntax of object slots was changed to improve usability, for example, the correct answer slot, which contains a list of indices into the answer-description list, originally contained the correct answers verbatim. It was originally thought that the additional effort of representing the information twice would be offset by not having the indirection of using an index, but this proved to be a false assumption.
- Other user motivated changes included: changes to worksheet forms, changes to the crane boom simulation, and changes to the conceptual vocabulary. Also see Section 5.2 for changes made to KAFITS due to data from student trials.

Answers without questions: features implemented but not used. Another way to argue for the importance of user participation is to list features that were implemented without user input that were deemed inadequate or went unused.[75] Such features include:

- the "recent instances" feature,
- the "topic details" feature,
- most of the command-key short-cuts,
- the help/info system,
- and several menu and browser operations.

[75]This is not a strong argument because the features may have been underutilized due to inadequate training, poor interface design, or idiosyncratic characteristics of the participants.

With the help system in particular, we discovered that the design was not completely responsive to the user's needs. Most of the assistance the domain expert needed was along the lines of "something funny is going on here" or "I thought I was here [in this mode or function] but I ended up here." A (standard, non-intelligent) help system is of little use in such situations. After the first couple of months the domain expert had extremely few conceptual questions regarding the framework or the interface. However, the knowledge engineer did occasionally give unsolicited advice of a conceptual nature while looking over the domain expert's shoulder. If the domain expert had a misconception about the system, or was in a situation where he could have used a more efficient method to accomplish a task, *he did not know it*, and a traditional help system would not be useful.

However, failure to learn or utilize software features may be due to fundamental psychological phenomena. In "Paradox of the Active User," Carroll & Rosson [1987] point to two tendencies of users using sophisticated software such as word processing programs: "people have considerable trouble learning computers," and "their skills tend to asymptote at relative mediocrity." They claim that these are empirical phenomena and hypothesize that they are due to fundamental conflicts in the user's cognitive and motivational strategies, rather than poor software design. They describe two "paradoxes." The first paradox, called "production bias," is that people's desire to get the job done often reduces their motivation to learn new ideas, even when the new ideas might get the job done faster. The second, called "assimilation bias," is that irrelevant similarities between new information and old information (eg. other features of the same program, other programs, or other technical artifacts) can mislead or blind the user so that they make assumptions about the software that inhibit learning.

User participation and software testing. We have argued for the importance of user participatory design by documenting aspects of the system that benefited from user participation and aspects of the system that were poorly designed due to lack of user participation. User participation is useful for software *testing* as well as software design. For most of the study we had at least two users using the system. Each change we made

to the software was tested, if not rigorously then at least realistically by two users from the target audience. (But this can be frustrating for them, as mentioned in Section 4.1.4.)

5.5.5 Summary of Cognitive Considerations and Interface Design

Designing systems for usability requires considering cognitive aspects of software use. Since no *a priori* theory exists which will predict the cognition of an average user (or any particular user), and since new technology, by its very nature, introduces unknown variables into situations, user participation is crucial to ensure usability in software design. In this Section we discussed several cognitive factors for designing ITSs:

1. Designers must evaluate whether the cognitive model they want to convey corresponds with the mental model the user actually builds;
2. The user's previous knowledge must be considered, especially when misconceptions or confusion with mundane uses of terms could interfere with understanding the system;
3. Teachers' mental models of curriculum tend to be procedural in nature (as documented by four observed phenomena), and this must be considered when training them to use a declaratively represented curriculum knowledge base.
4. The domain expert's mental models of curriculum and software are complex and put strong demands on working memory and long term memory (as documented by numerous quotes);
5. Designing cognitive fidelity into the interface helps to convey the correct cognitive model, reduces demands on working memory by off-loading information onto the medium, and reduces demands on long term memory by reminding the user of the underlying representational framework;
6. Failure to learn or utilize software features is sometimes due to a fundamental conflict between the user's goals of producing vs. learning, rather than poor software design.

We presented several ITS knowledge acquisition interface design principles for realizing (and dealing with tradeoffs between) our design goals of power, efficiency, and ease of use. These principles relate to: cognitive fidelity; facilitating accessibility, monitoring, and management of both static and dynamic information; assistance and efficiency; and providing support for off-line activities. We also gave two implementation guidelines related to user participation: "include what the user needs," and "don't include anything else." To argue for these guidelines and for user participation in general, we documented features affected by user participation, and features that were poorly developed because of lack of user participation.

CHAPTER 6

SUMMARY AND RECOMMENDATIONS

In this Chapter we summarize the results and contributions of this study and discuss limitations to the generality of these results and contributions. Then we offer several extensions to our framework as recommendations to the ITS research community, including: (1) a generalization of our software design which provides a context within which to compare KAFITS with other generic ITSs and serves as a design specification with which to build future generic tutor shells; (2) a general theory for incorporating knowledge classification schemes ("K-types") into intelligent tutors; and (3) a discussion of "collaborative, inspectible, persuadable computer tutors." Then we give a number of recommendations for future research; and finally we offer recommendations to the educational community, including a vision for how generic ITS systems could be incorporated into public education and industry training.

6.1 Contributions

We have reported on the formative evaluation of a knowledge acquisition tool for intelligent computer tutors and on a case study of the tool being used by three educators to design a tutor for statics. Using the case study and formative evaluation methodologies has allowed us to *describe a benchmark ITS knowledge acquisition tool and to identify key design issues* and design tradeoffs for building similar tools. We have also described a benchmark ITS *knowledge acquisition process* and identified key design issues and tradeoffs for the ITS design process. Below we summarize the main contributions of this study.

Involving Educators in ITS Design

We have demonstrated that it is feasible to involve educators in ITS construction at a highly collaborative hands-on level, i.e. fully participating in design, implementation, and testing.[1] Building an ITS is analogous to writing a textbook in terms of the magnitude of expertise and effort required, so we expect only select teachers to participate in ITS construction. We listed several characteristics of instructors that make them good ITS domain experts (see Section 5.1.2). We described the method and content of our instructor training in Section 5.1.2 and summarize the overall time commitment below. Building an ITS is at times frustrating for the domain expert (as described in Section 5.1.3); however it is on the whole rewarding and enjoyable (see Appendix C). Finally, the domain expert's initial knowledge, mental models, and cognitive limitations must be considered when designing knowledge engineering sessions and tools (see Section 5.5.2). For instance, we have identified a tendency for educators to construct *procedural* internal representations of ITS curriculum even when this information is declarative (non-procedural) (Section 5.5.1).

A Generic ITS Representational Framework

We have designed, implemented, and successfully used a generic framework for representing "what to teach" and "how to teach it" in intelligent tutors. The framework includes: a conceptual vocabulary that is designed for expressiveness and usability; a curriculum model consisting of a topic network, misconceptions, knowledge types, and levels of performance/mastery within topics; a method for representing tutorial strategies (called Parameterized Action Networks), and a method for overlay student modeling. Our framework incorporates some considerations from instructional design theory and is unique among generic ITS frameworks in that multiple tutoring strategies are represented and dynamically selected during tutoring.

---

[1] Though others have built ITS shells intended to be used by non-programmers we know of no other inquiry that studies educators' use of such a system.

We discussed issues and design tradeoffs affecting the representation of domain knowledge in intelligent tutors and gave mechanisms and solutions for them (see Figure 5.9). The main tradeoffs identified were: (1) the expressiveness/power/precision vs. the understandability/simplicity/usability of the conceptual vocabulary, and (2) the modularity/flexibility vs. smooth pedagogical/discourse flow of the curriculum. Although it is desirable to separate declarative and procedural knowledge in ITS knowledge bases, we have noted that some degree of interdependence between declarative and procedural information is unavoidable (Section 5.4.3).

An ITS Knowledge Acquisition Interface

We have implemented and successfully used a set of tools for acquiring pedagogical and domain knowledge from instructional experts. The tools support an accurate and fairly complete cognitive model of the underlying representational framework of the system by visually reifying key system concepts and structures. Tools were built to browse (inspect and navigate within) the knowledge base, modify the knowledge base, test the knowledge base, and monitor the state of the system while it is tutoring. The tools also constitute an experimental workbench usable by instructional researchers for rapid prototyping and testing of curriculum and tutoring strategies. In Section 5.5.4 we discussed how our user-participatory design process led to a highly usable system, and we discussed issues and tradeoffs for designing powerful yet usable tools.

An ITS Knowledge Acquisition Process

We have outlined a process for involving educators in building an ITS using knowledge acquisition tools. The ten-step process used to build the statics tutor (see Figure 5.1) spans design, implementation, and testing, uses several knowledge acquisition techniques, and enables the instructor's conception of the curriculum to evolve through classroom-like, CAI-like, and ITS-like phases. We have also *identified important issues and tradeoffs* for ITS knowledge acquisition, covering issues of domain expert training, important characteristics

of domain experts and knowledge engineers, plus techniques and guidelines for conducting knowledge acquisition sessions.

Quantitative Analysis of the Design Process

The statics curriculum covers 41 topics and misconceptions and represents about six hours of on-line instruction.[2] Our analysis of size, complexity, and other properties of the topic network and knowledge base is given in Section 5.3 (these properties allow comparison of the statics knowledge base with knowledge bases of other domains). The main result of our person-hour analysis of the design process (in Section 5.3.2) is that *it took about 100 hours to build the statics tutor per hour of on-line instruction*,[3] or about 85 hours of development time for each hour of on-line instruction if training is subtracted. These figures are (surprisingly) comparable to similar estimates for building traditional computer aided instructional systems [Gery 1987], but should be interpreted cautiously because they incorporate many assumptions and because hours of effort per hour of instruction is not a very satisfactory metric for the effectiveness of ITS tools (see Section 5.3.2). The major conclusion we take from these figures is that intelligent tutors with modular knowledge representation and flexible tutorial response can be built with about the same effort as CAI systems (which do not have this degree of flexibility). Additional figures are: implementation took about six times as long as design; total training time was 40 hours for the domain expert and 6 hours for the knowledge base managers; and the knowledge engineer's time was about one third the domain expert's time. (See Figures 5.6 and 5.7 for additional results of quantitative analysis.)

---

[2]Much of our analysis is extrapolated from data collected during the design and testing of the Linear-equilibrium portion of the statics curriculum, which constitutes about one quarter of the entire statics curriculum.

[3]This figure includes both production and training time spent by the domain expert, knowledge engineer, and knowledge base managers to design, implement, test, and refine the statics tutor using the KAFITS system; but does not include the development of the KAFITS system itself, building the simulation environment (the crane boom), or creating graphics pictures.

Other Contributions to ITS Research

The contributions mentioned above all relate to ITS knowledge engineering, the focus of our study. There are several tangential contributions to the ITS field that are not related to this focus:

- Student Modeling. Our overlay student model is more sophisticated than most student models reported in the literature and has several unique features, including nonmonotonic reasoning, reasoning with uncertainty, and multiple levels of inferencing.
- Student Initiative. The KAFITS student initiative feature represents the beginnings of a facility allowing students to have significant control over the style and content of their learning (see also Section 6.3.4 below).
- Strategy Representation. Parameterized Action Networks (PANs), described in Section 3.1.7, are an efficient and powerful way to represent multiple tutoring strategies, and facilitate easy prototyping and experimentation of tutoring strategies.
- ITS Design and Evaluation. The evaluation of ITSs is a research issue in itself, and we have discussed and given examples of many evaluation methods and tradeoffs (Section 2.5). Our study is an example of using a combination of about a dozen evaluation methods (mainly qualitative methods, with some semi-quantitative methods) to investigate an exploratory research area.

Relation to Previous Work

Our survey and analysis of the AI and ITS literature (Chapter 2) in the areas of generic ITS shells, empirical research and iterative design, knowledge acquisition methods, and evaluation methods offers many general suggestions to ITS designers in each of these areas. In Chapter 2 we also relate our study to previous work in these areas (at the end of each of the sections). In addition, below (in Section 6.3.2) we propose a classification of ITS systems which places our work in perspective with previous and future generic ITSs.

6.2 Limitations

Methodological Limitations

In Section 1.5 we noted general methodological factors which limit our ability to generalize our results, including: that we studied the building of a single tutor in one domain, that there was only one teacher (and a total of three users of the software) involved in the study, that the researcher's objectivity was potentially affected because he was also the knowledge engineer and software designer, and that the software was not constant over the course of the study since it was periodically modified according to user needs. Limitations due to the low number of subjects and domains is partially mitigated by the fact that our original framework was designed from years of experience prior to this study working with educators in computer tutoring (see Section 5.4.1).

Limits to the KAFITS System and ITS Design Process

Here we note specific limits to the contributions mentioned above (our list of limitations parallels the list of contributions).

- Involving Educators in ITS Design. Though we have demonstrated that it is feasible and useful to involve educators in ITS design, we have not demonstrated that incorporating ITSs into education is beneficial or cost-effective. We discuss (below, in Section 6.5) issues in using KAFITS-built tutors in classrooms.

- A Generic ITS Representational Framework. Intelligent tutoring systems fall roughly into one of two categories: curriculum-oriented tutors for teaching declarative knowledge (including concepts, facts, and principles), and expert system tutors which teach procedural knowledge and contain rule-based representations of the domain and student knowledge (usually having sophisticated diagnostic capabilities). KAFITS is most suited to building curriculum-oriented tutors for teaching declarative knowledge. Though the framework does not prohibit building other types of tutors, we

have no data yet on how KAFITS needs to be modified to build procedural tutors. In Section 6.4 we discuss future research in these areas.

- An ITS Knowledge Acquisition Interface. We are confident that our domain knowledge base Browser and our monitoring tools are robust and usable (though many improvements are possible).[4] The strategy interface is still in prototype form; it has not been used extensively by typical users. Similarly, the student interface an early prototype. We discuss future research on the KAFITS interfaces below (in Section 6.4).

- An ITS Knowledge Acquisition Process. Since we have gone through our ITS design/knowledge acquisition process in its entirety only in one domain, it is only a base-line from which to build future ITSs.[5] Our process does not include steps for task analysis of domain expertise (because we were not representing procedural domain knowledge) or cognitive studies of students' misconceptions (since the domain expert had participated in previous such studies) which are necessary for some domains.

- Numerical Analysis. The validity of our numerical results is particularly susceptible to uncertainty due to our low number of subjects (and single domain). (Though we argue for the prototypicality of the subjects in this study in Section 4.1.2.) The quantitative results are meant to be order-of-magnitude and require further verification.

---

[4]Note however that KAFITS is still a prototype system. For any type of software the effort required to move the software from the prototype stage to a bug-proof "product" with documentation is considerable.

[5]This process worked for a domain expert who had some previous exposure to ITS concepts and otherwise was not familiar with the design process. The design process (especially training) would have to be altered for experts with more or less starting knowledge.

6.3 Recommendations and Proposals to the ITS Research Community

6.3.1 Recommendations

In general, we have recommended more teacher participation in building ITSs and better evaluation in ITS research. Specific recommendations for those designing ITSs or doing ITS-related research are distributed throughout this document: in Section 2.1.3 we listed six desirable features for ITS shells (or generic ITSs); in Section 2.2 we recommended that ITS designers become more familiar with instructional design theory and that ITSs incorporate knowledge type classification schemes; in Section 2.2.3 we listed seven "basic areas of instruction" that we recommend be included in all ITSs; in Section 2.3.1 we listed six steps to follow in doing ITS research; in Section 2.3.1 we recommended that an iterative design process be combined with a user-participatory design process for building ITS tools; in Section 2.4 we described a number of ITS knowledge acquisition methods (summarized in Figure 2.3) and discussed tradeoffs in using them; in Section 2.5.3 we described a number of ITS evaluation methods (summarized in Figure 2.4) and recommend using primarily qualitative and formative evaluation methods for "exploratory" studies and for reporting within the research community.

6.3.2 NEO-KAFITS and Class Z Tutors

Below we generalize the KAFITS representational framework to provide a pruned-down design specification on which to base future work and to provide a framework for comparing KAFITS with other ITS shells. We also discuss tradeoffs in deciding how much "intelligence" or inferencing power to include in an ITS.

NEO-KAFITS

The KAFITS representational framework is a prototype and we do not expect future research to duplicate it exactly as specified in this document—some of its features are more essential than others. Therefore we offer a high level design specification of a generalization of KAFITS, called NEO-KAFITS (after Clancey's [1986b] NEOMYCIN, a generalization of the MYCIN system).[6] A high-level specification of the NEO-KAFITS generic ITS framework is as follows:

1. Three knowledge bases are used. Curriculum objects and their attributes are stored in a **domain knowledge base**. Tutoring rules or strategies, which access the information in the domain knowledge base to produce tutorial behavior, are represented using a clearly defined set of actions and predicates, and are stored in a **strategic knowledge base**. The system also has a **dynamic knowledge base** which includes dynamically updated models of the student's mental state and the tutorial discourse.[7]

2. Curriculum is represented in a **curriculum or domain knowledge network** which includes topic nodes and other curriculum entities, such as Mis-KUs and Synthesizers.

3. The framework is object-oriented, including but not limited to these **object types: topics** (used to specify the "macro-level" of instruction), **and presentations** (used to specify the "micro-level" of instruction).

4. Topic objects are classified according to **knowledge type** (see the discussion in Section 3.1.5 on knowledge types).

5. The nodes of the topic network are related by a variety of **topic links**, which may include, but are not limited to, prerequisite and subsumption links.

---

[6] NEO-KAFITS is a generalization of the KAFITS representational framework, but not the KAFITS knowledge acquisition interface.

[7] Refinements on these three knowledge bases are possible, such as representing the student model and discourse models in separate knowledge bases, and distinguishing diagnostic rules from tutoring rules. A knowledge base for domain expertise is also needed for systems that include a performance model of domain problem solving.

6. Topics can have multiple **levels of understanding** (or performance or mastery) (see Section 3.1.5 for a discussion of topic levels).

7. **Presentation objects** define the expository and inquisitory interactions with the user, including specifications of the learning situation and the task given to the learner.

8. A **layered overlay student model** is used which calculates its values as follows: topics values are based on topic levels, topic level values are based on presentations, and presentations values are based on individual student transactions (see Section 3.3).

The NEO-KAFITS representational framework is a powerful (general, extendible, flexible, expressive) and usable (understandable and non-complex) core on which to build KAFITS-like knowledge acquisition tools. The following specifics of the KAFITS system have been extracted: the four-level decision model; the use of PANs to represent strategies; the specific types of topic network nodes and links; the specific topic levels; all but two of the object types; the names of the object slots, primitive actions, and strategy parameters; and most features of the KAFITS student model.

Like KAFITS, NEO-KAFITS is best suited for non-procedural domains which do not require rule-based representations of expert and novice knowledge. It is also best suited for directive (or curriculum driven) tutoring as opposed to reactive (or coaching or diagnosis driven) tutoring.[8] [9] In addition, since it is curriculum-driven, domains with highly interactive overlapping topics, in which it is difficult to distinguish individual topics or in which many topics are taught simultaneously, are not well suited for NEO-KAFITS.[10]

---

[8]Note that considerable student control is still possible in NEO-KAFITS systems.

[9]These opinions are subject to change as a result of future studies using KAFITS in procedural domains or with reactive tutoring strategies.

[10]Rich interactive learning environments are best suited for such domains, but representing domain knowledge in these domains is difficult, so guiding and measuring student learning will be difficult for *any* style of computer tutor.

Class Z Computer Tutors: A Proposal

It is difficult to make direct comparisons of this work with other ITS research because research goals differ among research efforts. Unlike our work, most ITS research does not have generality as a main goal, and instead focuses on: embodying a specific theory of cognition, learning, or instruction (such as Anderson's ACT* theory and Merrill's component display theory); applying a specific AI technology to computer tutors (such as machine learning or case-based reasoning); and/or optimizing learning in a specific domain (such as algebra or PASCAL programming). Efforts that *do* have generality as a main goal attempt to design powerful and general frameworks but do not focus on usability, teacher involvement, or the knowledge acquisition process (as was discussed in Section 2.1). However, in order to generalize the results of this study and make recommendations for future work, we must put this work in context with previous studies. Toward this end we define characteristics of a class of generic ITSs that address our criterion for usability. The following characteristics serve as design guidelines for realizing the goals of practicality and teacher participation.

- **Separate domain and strategic knowledge bases.** Domain knowledge and strategic knowledge should be represented in separate knowledge bases.. The tutoring strategies or rules in the strategic knowledge base determine how and when to use the domain knowledge. Domain knowledge should be represented in a flexible manner, allowing multiple tutoring strategies to use it in diverse ways.

- **Clearly defined representational framework and conceptual vocabulary.** The basic representational entities, their attributes, and the allowed relationships between them should be pre-defined or defined early in the design process (though minor modifications may occur during the ITS design).[11] The syntax and semantics of the data structures of the domain and strategic knowledge bases should be explicit and general.

[11]This contrasts with ITS projects which start with design goals for the performance of the final computer tutor and then determine what representational and control formalisms will best suit their needs.

- **Knowledge acquisition tools.** Tools must be built which allow easy inspection, monitoring, and modification of knowledge bases.
- **Usability and understandability.** The conceptual vocabulary and tools should be designed to be used by educators who are not computer scientists.

We call intelligent tutors or ITS shells with these characteristics "Class Z computer tutors." Class Z tutors bridge the gap between ITS research and practical application of this research, allowing educators to participate hands-on in ITS design and testing. For reasons given in the next section, Class Z tutors will tend not to incorporate "state of the art" AI technology that has not been "hammered out" and broadly tested (though they may employ novel strategies or structures *using* proven AI technology). Because Class Z tutors have clearly defined representational frameworks, their knowledge acquisition and monitoring tools, student modeling methods, and student interfaces can be standardized and used in many tutors.

To put Class Z tutors and the KAFITS system in perspective with each other and with other ITS systems, we show a subsumption classification scheme of types of ITSs in Figure 6.1. This hierarchy is sparse because it is a *prescriptive* classification designed to emphasize and recommend important characteristics of generic ITS shells, not a *descriptive* classification designed to compare existing ITSs.

There are five levels to the hierarchy: 1. **ITSs** are defined as in Section 2.1.1; 2. **Generic ITS shells** are defined and described in Section 2.1; 3. **Class Z tutors** are described above (we know of no system other than KAFITS that fits the definition of Class Z tutors); 4. **NEO-KAFITS** as described above; and 5. the KAFITS system as described in Chapter 3. KAFITS, NEO-KAFITS, and Class Z tutors employ AI modeling and knowledge representation paradigms but do not include heuristic search of problem spaces as is characteristic of many AI systems.[12] We call them "low inference" systems, as described next.

[12]However, heuristic search is not *excluded* by these frameworks.

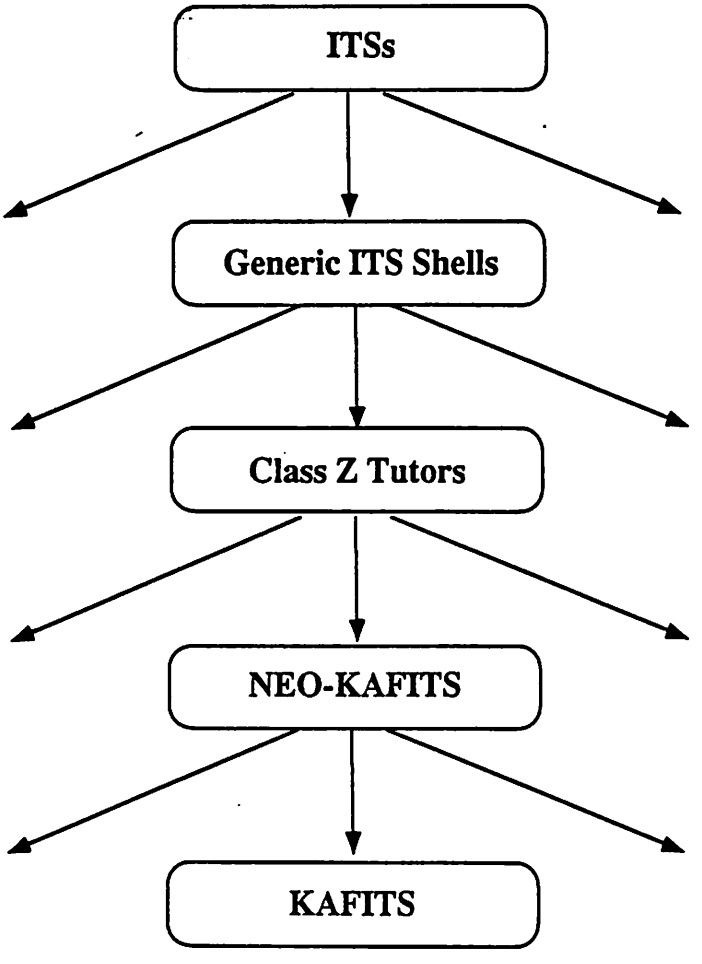


Figure 6.1 Class Z Hierarchy

"Low Inference" Intelligent Tutors

Our experience building the statics tutor suggests that practical and effective intelligent computer tutors can be built without incorporating sophisticated heuristic search techniques or runable cognitive models of expert and student knowledge. The exclusion of these elements may seem detrimental, but we argue that it is sometimes *desirable* to build tutors with these limitations.

Inferencing in tutoring systems can happen in three areas: domain reasoning, tutorial reasoning, and diagnostic reasoning. The inferences performed by KAFITS in these areas are few and not very sophisticated: (1) Since our representation of the domain knowledge is not rule-based, we do not "reason" about the domain. (2) Our representation of strategies uses a procedural network, which, unlike some rule-based methods of representing strategies, does not require search among alternative tutorial actions,[13] and does not plan ahead or backtrack. (3) Our student model performs inferencing through its several levels as described in Section 3.3 but does not employ search techniques.

There are many areas where deeper forms of reasoning (more sophisticated search and inferencing) could be added to KAFITS to calculate what is canned or pre-determined, for example:

- generating natural language rather than using canned text;
- accepting natural language student input in the place of multiple choice menus and mouse gestures;
- calculating which topics are prerequisites of other topics based on pedagogical properties;
- inferring correct answers, hints, and answer reasons based on an underlying representation of domain knowledge, and;

[13]Except for cases where a conflict resolution scheme is used to pick a node that "passes" because more than one arc emanating from a node "passes."

- making the tutoring strategies self-improving based on successful student achievement;

One reason we have not incorporated these more sophisticated computational mechanisms is that our study focuses on representational adequacy rather than inferencing power. That is, we have attempted to identify primitive actions and properties that are sufficient or necessary for encoding knowledge about what to teach and how to teach it, and have represented most of this knowledge in shallow forms. But there is a more fundamental reason for not incorporating some of this more sophisticated inferencing: the technology required is, with the current state of the art, not robust enough and/or (in cases requiring detailed cognitive task analysis) is too labor-intensive to meet our goals for practical intelligent tutors that can be built with educators participation.

If all the knowledge in KAFITS (except the student model) is stored (canned), can it be said to be "intelligent?"[14] Wenger [1987, pg. 5] describes the intelligence in ITSs by comparing ITSs "generative" capability with traditional CAI, noting that traditional CAI encodes an expert's *decisions* while ITS encodes the *knowledge* and/or reasoning that underlies decisions. KAFITS is generative in that it generates *tutorial dialog* and infers a student model "on the fly" (at run time). The knowledge underlying this dialog is represented explicitly in action networks. However, the distinction between encoding decisions and knowledge is not clear cut, since we can treat any tutoring rule as a decision for which there is a deeper reason or cause. For instance, the knowledge underlying KAFITS's action networks is not represented explicitly. In fact, the enterprise of AI can be seen as an attempt to encode increasingly deeper (or more generative, abstract, and explicit) levels of meaning toward, in the extreme, encoding first principles from which all other information can be inferred.

As an illustration of this progression from encoding decisions to encoding general rules and deeper principles, consider the following ("English-ized") hypothetical ITS tutoring rules and principles, where each item is intended to be an abstraction or reason encompassing the previous one:

[14] We will ignore philosophical issues about the definition of intelligence.

1. If question-12 is answered wrong, give explanation-5.

2. If the student gets a question wrong twice, then give a canned explanation.

3. If the student is very confused, then give an additional level of feedback.

4. Give students several opportunities to think about each situation so that they may learn from their mistakes, then scaffold feedback of increasing levels of specificity.

5. Learning happens through an active process of concept formation while trying to account for new information within in the context of previous knowledge.

This progression of hypothetical ITS tutoring "rules" goes from the trivial to the impossible. The first item illustrates the low-level coupling of diagnosis and action found in (non-intelligent) CAI. The second item illustrates a type of tutorial reasoning that is typical of today's intelligent tutors. A tutor using this rule must keep a record of the student's behavior, but the reason why the rule is applicable is not explicit. The third item is well within the state of the art for ITS. A tutor using this rule must have abstract models of the student's mental state and the tutoring process.[15] The fourth item states a pedagogical belief or strategy, and represents the principle behind the previous rule. It could be operationalized in a limited way but is not precise enough to be part of a robust ITS (with today's technology). The final item is based on a theory—a psychological, or philosophical assumption. It represents the reason for the previous principle and the purpose for the rule above it. Representing and using knowledge at this level of abstraction is clearly out of the reach of current technology.

The more abstract items listed above, as well as the potential additional KAFITS inferencing mechanisms listed above are conceivable, but may not be practical or tractable in realistic settings. In the extreme one could ask: "Why go through all the trouble of defining a curriculum and tutoring strategies at all? Why not use a deep causal representation of

[15] A diagnostic strategy must infer the level of "confusion" from student behavior (such as number of times asking for help), and the appropriate interpretation of "feedback" must be inferred based on the current state of the tutorial session.

domain knowledge and its pedagogical properties, and let AI rules infer the relationships and ordering of the subject matter?" The answer is perhaps obvious: the problem is intractable except in limited cases. Educational researchers have not successfully developed a general, well defined theory for constructing curricula from first principles, so we will not be able to program a computer to do so. The tradeoffs between "intelligence" and practicality/usability implicit in the above discussion are illustrated in the continuum shown in Figure 6.2. On a continuum of knowledge and inference sophistication, with CAI-like systems simulating book- or lecture-style learning at one extreme, and future generation ITS systems simulating human one-on-one tutoring at the other extreme, systems which are geared toward teacher participation and realistic applications must focus on the less sophisticated end of the spectrum.

Difficulty realizing AI's potential is not only due to limitations in human knowledge but also limitations in our ability to *represent* human knowledge in machines. Much artificial intelligence research is based on a reductionist assumption that all knowledge can be represented in simple modular units and that complex behavior results from interpreting a large number of these simple knowledge units with a simple mechanism [Winograd & Flores 1986, Simon 1981], and this reductionist assumption has yet to be substantially supported empirically. Currently we have a limited repertoire of formalisms and mechanisms for representing knowledge (including rules, frames, and propositions). The vast majority of types of human knowledge and skills (see the complex knowledge types in Figure 2.2) have not been simulated non-trivially in machines. We have been able to model facts and simple procedural skills, but language, mental models, complex problem solving, metacognition, creativity, etc. have been simulated only in very simplified forms.[16] One school of thought hypothesizes that all that is missing is a critical mass of enough knowledge (Lenat et. al [1986] write of "knowledge acquisition from strength"). But the successes of AI research in toy domains has been notoriously difficult to scale up to realistic situations. Buchanan

[16]We have given evidence in this study of some seemingly intractable problems related to the fuzziness and ambiguity of knowledge (Section 5.4.2).

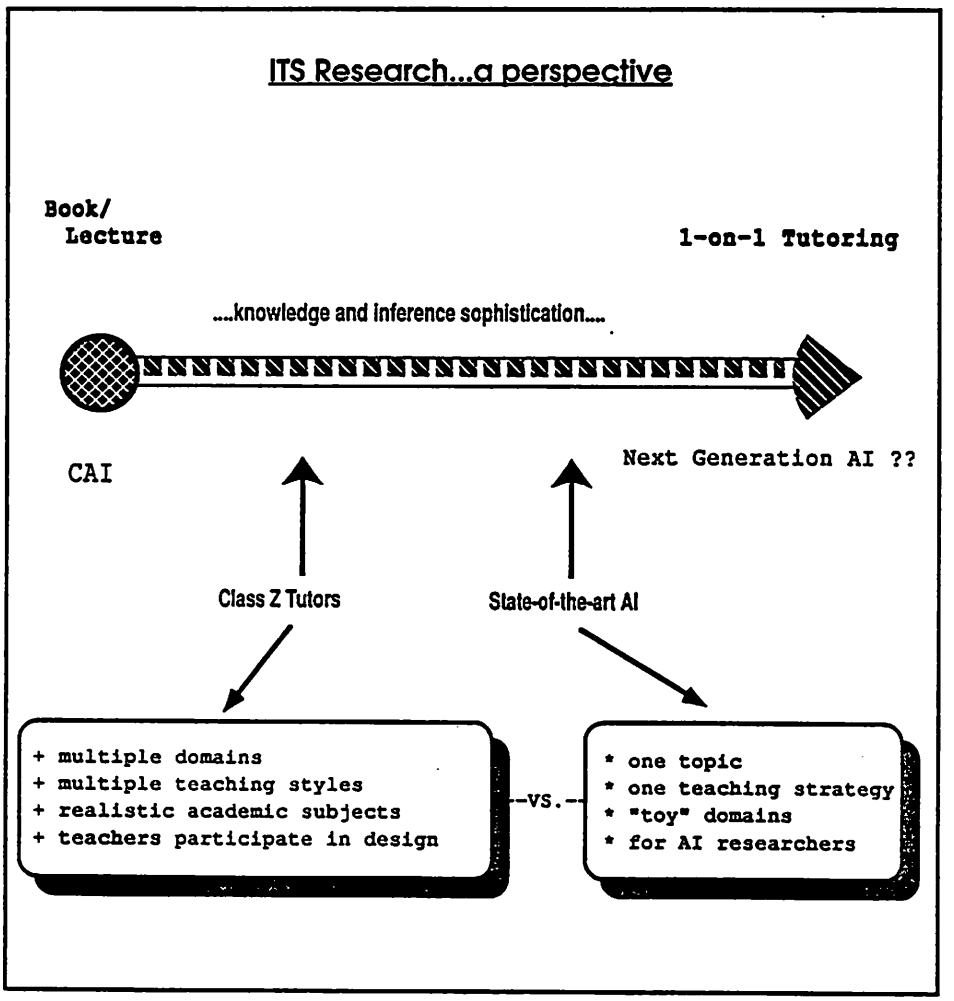


Figure 6.2 ITS and AI State of the Art—A Perspective

[1987], of the "paradox of increased knowledge," argues that "we rarely can predict whether the effect of more knowledge will be positive or negative."

Therefore, in designing general frameworks for computer tutors with *practicality and usability* as key goals, we must be aware of limitations in the state of the art of intelligent systems, regardless of whether these limits are the result of insufficient knowledge base depth and breadth, fundamental limitations of mechanisms used to represent knowledge, or unavailability of articulated human knowledge.

We suggest that, given the current state of the art, ITS rules should be implemented on the level of item three above, and that all such rules be annotated with the principles, assumptions, and/or theories that justify them. This allows systems to be evaluated and modified on the basis of their underlying assumptions and allows tutoring systems to explain (in a limited way) why they are performing tutoring or discourse actions.

To guide ITS developers in deciding the depth or level of sophistication of reasoning mechanisms they use, we suggest that three factors be considered: technical practicality, human factors limitations, and educational needs, as described below.

- **Technical practicality.** The representational framework must be tractable for instructional domains of realistic size and complexity. That an AI mechanism has been used successfully in a limited context is not sufficient to assume it will scale up or transfer to new domains.

- **Human factors.** For some knowledge, even if it *can* be represented tractably, the state of the art may dictate that its representation is too complicated and esoteric for use by those who are not AI researchers or programmers. If educators are to build and experiment with intelligent tutors then they must understand, access, and modify the encoded knowledge.

- **Educational needs.** Learning complex and deep forms of knowledge is essential in most domains. The content of intelligent tutors should not be limited to the simple types of knowledge that AI technology can currently represent. ITSs must attempt to

teach complex knowledge even if this knowledge must be represented in shallow forms such as canned text.

Summary of NEO-KAFITS and Class Z Tutors

In this Section we gave eight high-level design specifications which generalize the KAFITS framework (not the interface) by excluding all but essential features of the representational framework. The specification for proposed new system, called NEO-KAFITS, is given as a core upon which to build future systems, and also serves to highlight those features of KAFITS that are most central.

Then, to place our work in perspective with other ITS projects, we proposed a class of ITSs aimed at practicality and usability by educators. We gave four guidelines for designing the representational frameworks and interfaces for such systems, which we call Class Z tutors. Class Z tutors facilitate educators' full integration into the design, implementation, and evaluation of computer tutors.

Finally, we discussed the appropriate amount of "intelligence" or inferencing sophistication in intelligent tutors. We discussed tradeoffs in usability vs. inferencing power and argued that Class Z tutors should be "low inference tutors," given the current state of the art of AI.

6.3.3 Toward A Theory of Knowledge Types

There is considerable lack of agreement in the ITS research community over optimal, or even adequate, tutoring strategies (or rules) for computer tutors.[17] This lack of agreement, though somewhat attributable to divergent underlying psychological, cognitive, or epistemological assumptions, can be largely attributed to the existence of divergent instructional goals that are not clearly articulated. For example, disagreement over the optimal level of feedback or learner control in intelligent tutors often boils down to differing priorities about

[17]There is also little agreement in the educational research community over optimal teaching strategies.

the type of knowledge to be learned. Consider these diverse pedagogical goals: (1) learn specific domain facts and solve standard problems; (2) obtain a conceptual and intuitive understanding of the content; (3) learn general problem solving and metacognitive skills. Discussions of alternative tutoring strategies rarely articulate fundamental differences in pedagogical priorities. The inadequate articulation of instructional goals or priorities is, in part, due to the lack of a sufficient technical vocabulary for describing the subject matter. Here we propose the beginnings of a model for "knowledge types" which provides a descriptive vocabulary for specifying instructional content and strategies in intelligent tutors.

We follow "Gagne's hypothesis" that there are different types of knowledge (or "learned capabilities") and that there are different methods appropriate for promoting the learning of each type (see Section 2.2.7). A knowledge classification scheme can be based on characteristics of the behavior that the knowledge allows, or can be based on an underlying theory of cognition (or on both, as in Merrill [1983]).

The Procedural/Declarative Distinction

Gagne's [1985] instructional theory, Merrill's [1983] PC-matrix, and the KAFITS Modified PC-matrix (Figure 2.2 and Appendix B), are examples of knowledge type classifications. Knowledge type distinctions are made throughout the ITS literature, but the vast majority of them are much less elaborate that the three schemes mentioned above. Many discussions of knowledge types in the ITS literature are founded upon the distinction between declarative and procedural knowledge,[18] yet instructional scientists are clear that we need to distinguish many types of knowledge if we are to make headway in articulating pedagogical principles. The procedural vs. declarative distinction not only lacks sufficient expressiveness, it may actually be detrimental. VanLehn [1987, pg.60], speaking from an AI perspective, says that the procedural/declarative distinction is "notorious...as a fuzzy, seldom useful differentiation." We recommend that the procedural/declarative distinction

[18]Other types of knowledge are mentioned, such as mental models, and common sense knowledge, but these are not clearly defined; and some systems add minor refinements such as describing two classes of procedural knowledge.

be abandoned (except in contexts where it has a precise meaning, as in the ACT* theory of cognition) and that more descriptive and precise ones be used in its place.

Mapping From Observables to K-types to Tutoring Actions

Our model of K-types (knowledge types) will not prescribe a specific classification scheme; it proposes properties for K-type schemes in general. We assume that knowledge classification will take a form similar to Merrill's, in that operationally determinable characteristics (such as behavioral objectives) of each piece of subject matter can be mapped to a knowledge type and knowledge types map to instructional methods. A simplified hypothetical example of this two step mapping is: (1) If a student is required to read and memorize a piece of information with the goal of being able to recall it verbatim in the future, that piece of knowledge is classified as a fact. (2) Facts are taught by repeated presentation of the information in different contexts until the student remembers the fact in a novel context. This mapping can be done manually, or can be automated or assisted by a rule-based system (as in [Merrill 1983]), but we believe that the classification of knowledge and the selection of corresponding strategies should not be forced upon the instructional designer (however, the system could suggest alternatives and argue against unusual designer choices, giving reasons for its suggestions). K-type schemes represent both a descriptive and a prescriptive model for instruction; they provide a language for describing the concepts, conditions, methods, and outcomes of instruction, and they provide a prescription of how to sensitize instruction to knowledge types.

K-type Internal Structure

A major feature of our preliminary theory of K-types is that all knowledge type representations have a common structure. This common structure allows general representational, inferencing, and control mechanisms to be built, and it also facilitates the pedagogical anal-

ysis of knowledge. We propose that all of the following attributes be represented for all knowledge types:[19]

- **Name**—the indicator used to identify a piece of knowledge, such as the name of a concept or procedure. Some K-types, such as facts, may not have names.
- **Definition**—a description, statement, or definition for the knowledge piece. It may be a statement for facts, a definition (including necessary and sufficient properties) for concepts, a list of steps for procedures, etc.
- **Examples,** including positive examples, negative examples, analogies, etc. Examples may take different forms for different K-types (e.g. pictures for concrete concepts and "walk throughs" for procedures).
- **Components.** Each K-type will have a characteristic method for representing sub-parts (e.g. concepts are made of attributes and sub-concepts, procedures are made of concepts and sub-steps, principles are made of concepts and relations).
- **K-bugs.** Each K-type will have a characteristic type of "mis-knowledge" (e.g. misconceptions for concepts, buggy rules for procedures, and misinformation for facts).
- **Performance levels.**[20] Levels can be similar to the PC-Matrix levels, or can be any of the other possibilities mentioned in Section 5.4.3. Each K-type could have its own characteristic levels, such as levels of explanatory depth for principles, and levels of detail for procedures.
- **Strategies.** Each K-type will have recommended methods for (1) conveying the knowledge, (2) giving feedback, and (3) remediating K-bugs.
- **Alternative representations.** Each K-type will have characteristic alternative methods for representing the knowledge, for example with pictures, text, graphs, animated "walk-throughs," etc.

[19]These attributes can be either stored or inferred.

[20]This is an extension to Merrill's performance levels (Section 2.2.4).

Note that the overall structure of all K-types is the same, but that specifics of the structure usually depend on the K-type (or sub-K-type, see below). In a tutoring system, all topics (knowledge pieces) may share additional attributes, as with the Motivation, Prerequisites, and Summary slots of the KAFITS system.

Sub-K-types

We will assume for expository reasons that the K-type scheme is implemented in an object-oriented ITS representational framework, although this is not required. The scheme will define a number of K-type classes, and each topic (or instructional unit) will be an instance of a K-type class, inheriting its default attributes and procedures.

For any knowledge categorization scheme developed there will be cases were refinements to K-types (i.e. sub-types of the K-type classes) are needed. For instance, principles can be descriptive or prescriptive [Reigeluth 1983b], concepts can be classical or fuzzy [Mervis & Rosh 1981], and procedures can be linear or hierarchical. Sub-K-types will inherit default values and procedures from their parents, and can override these defaults. The K-type scheme designer must pay attention to complexity and usability issues; more baroque schemes are less usable, and designers should avoid a proliferation of K-type distinctions that do not map into concrete differences in tutorial behavior or knowledge acquisition clarity (as was found to be the case in Bloom's [1956] early knowledge taxonomy). Also, tabular schemes, such as Merrill's PC-Matrix, may be more understandable than hierarchical schemes.

Domain Types

Domains tend to have characteristic overall structures, and K-type schemes can facilitate articulating this structure. For example, Reigeluth [1983b] prescribes that each domain be assigned an "organizing content type"—conceptual, theoretical (principle-like), or procedural—that best fits the characteristics of the domain and the instructional goals.

His "elaboration theory of instruction" specifies methods for selecting and sequencing content according to the organizing content type. Others have categorized domains according to whether their structure is predominantly procedural, historical, structural, causal, teleological, inferential, etc. (see also Wenger's [1987, Chapter 15] description of types of domain articulation). Domain types have characteristic links between topics, for example analogy, physical-part, a-kind-of, etc.[21] K-type schemes should incorporate, or at least provide guidance for, schemes for categorizing entire domains, and they should be useful in determining optimal overall instructional strategies for domains (as well as for individual topics).

Benefits of K-types

We believe that K-types have the following benefits to ITS performance and the ITS knowledge acquisition process:

1. K-types provide a precise vocabulary for **articulating** the domain expert's **instructional objectives**;
2. K-types provide a technical vocabulary allowing ITS designers to articulate characteristics of instructional content, facilitating **comparison among ITSs**;
3. K-types facilitate ITS content representation by providing clear **inter-topic structure** and **extra-topic structure**;
4. K-types facilitate the **decomposition and organization** of high level curriculum goals, content, and tasks;
5. K-types facilitate the **acquisition and representation** of domain content and tutoring strategies;

[21] Wenger [1987, pg. 331] describes several "justification types," such as structure, functionality, and constraints, that could form the basis for topic links.

6. For K-type schemes based on instructional theories, instructors designing an ITS will **automatically be using instructional or cognitive theories** and would indirectly be learning some of this theory;

7. Tutoring rules and behavior will be less ad-hoc, and **more diverse and effective.**

The Cost of Increased Complexity

Incorporating the above described aspects of K-type schemes into the representational framework of an ITS involves serious tradeoffs. The main factors mitigating against the benefits listed above are increased complexity and the subsequent decreased usability and learnability of the more powerful framework. Adding the extra power may move systems out of alignment with the design principles for Class Z tutors and may affect their accessibility by the educational community, as demonstrated by our description of problems with introducing the domain expert in our study to K-types (Section 5.1.2). However, most of the aspects of K-types are inspired by instructional design theory, so, while such systems *may* be out of reach for classroom teachers, they should be usable by instructional designers. In addition, if the rules for K-type categorization and strategy selection were automated in an expert system (as in Merrill's [1983] theory), and if the expert system explains its reasoning to the teacher, this may offset the increased complexity. However, such systems must allow the user to override the imposed rules so that their knowledge acquisition sessions can be structured opportunistically (bottom up or top down). We do not want to tie the instructional designer's hands behind his back with obligatory rules since there are sure to be important exceptions to most rules.

Summary of the K-types Model

We have proposed the beginnings of a model of knowledge types (K-types) for ITS, which we claim will provide ITS researchers with a technical vocabulary for articulating important characteristics of domain contents, help domain experts articulate and organize instructional content, and improve the quality of computer tutoring by sensitizing tutoring

rules to K-types. We have not suggested a specific K-type classification scheme but suggest structures and properties that K-type schemes should have. Our discussion included a suggested canonical internal structure for K-types, advice about refining K-types into sub-K-types, and a discussion of how K-types can generalize into "domain types." We also discuss usability/power tradeoffs in implementing K-type schemes.

6.3.4 Collaborative, Inspectible, Persuadable Computer Tutors

Ideally, learning involves a significant amount of collaboration between the student and the teacher. Thus far a high degree of collaboration has not been realized in computer tutors. Most ITSs give the student little control over the goals, content, or style of the tutorial session. ITSs that offer unobtrusive advice or "coach" a student are a step in the right direction but typical do not let the student feel as if she is in control of her learning. ITSs that allow free exploration in simulated environments without any guidance give the student control of her *activities*, but not of her *learning*, since she is usually not sure what she is learning or how well she is learning it. We believe that part of what has been missing is an acknowledgment of where different areas of expertise lie. There are three areas of expertise to consider, corresponding to the three prototypical functions of an ITS: the student model, the domain model, and the teaching model. Clearly the computer tutor is the expert at teaching, i.e. an expert in specific pedagogical knowledge about the domain and general knowledge about communication skills. The tutor has knowledge such as what topics are prerequisites of others, which topics are more difficult, and the best strategies for conveying different types of knowledge. Less recognized is the fact that it is the *student* herself who is the expert on the student model. She knows much more than the tutor about what she knows and doesn't know, what her plans are, her personal history, and how she likes to learn—the computer tutor's student model represents only a guess at what is in the student's head. Some researchers are starting to realize this and are designing ITS architectures that let students convey more directly what they think, asking students explicitly about their knowledge and beliefs, or allowing their actions to give evidence of their knowledge or plans [Self 1988]. Even *less* recognized is the fact that the student and

the computer tutor *share* expertise about the domain. There are two reasons for this. First, the student, unlike the "intelligent" tutor, is an intelligent being, having common-sense knowledge about the world and the ability to put information together in flexible new ways. In this respect the student actually "knows" much more about the subject matter domain than the computer tutor even before the first lesson begins. Second, the student's knowledge state evolves from novice to expert as she learns, and if she learns well she will, by virtue of her flexible intelligence, surpass the computer tutor and be able to solve problems that the tutor could not solve.[22]

Our view that the student and tutor have complementary areas of expertise argues for the need for more collaborative tutors. The KAFITS student initiative feature (Section 3.2.4) embodies the primitive beginnings of student/tutor collaboration, and our design goal of having a clearly defined representational framework and conceptual vocabulary facilitates building collaborative tutors. Having a consistent unified format for curriculum material allows the tutoring rules and student interface to be re-used in many tutors and domains. For instance, a framework with a standard mechanism for representing hints allows a student interface which lets the student ask for hints in any situation, for tutors in any domain.

Ridgeway [1988] calls for "transparent" ITSs that give the student access to the following information that most systems possess implicitly or explicitly: a specification of the knowledge to be acquired, the teaching techniques, and beliefs about the current state of the user. Along these lines, we propose that ITSs be designed to be "collaborative, inspectible, persuadable tutors." Their domain knowledge, student model, and teaching knowledge should be open to inspection, and both the content and the inferences of these components should be explainable. Since students share expertise of the student model and domain knowledge with the tutor, ITSs should allow a student to take control of her learning, set the instructional agenda, and even manipulate the student model and tutoring style. And since it is the tutor who has the expertise about teaching the domain knowledge,

[22]Humans are good at what in AI is called "explanation based learning," in which a small number of understood examples, in combination with general common-sense knowledge, yield deep and broad knowledge that goes far beyond what could be induced based solely on the characteristics of the examples.

ITSs should provide firm guidance when needed and be able to *prevent* the student from doing things that there is sufficient reason to believe are counterproductive or damaging (such as choosing a teaching strategy that does not make sense in the current context, or irresponsibly altering the data in the student model)—thus the term "persuadable" is used rather than "controllable." Again, the KAFITS student initiative feature illustrates a primitive start toward this goal, which needs much further study. Following is a preliminary compendium of features we recommend for future ITSs that are collaborative, inspectible, and persuadable, along with hypothetical natural language statements by the tutor (T) or student that illustrate the features.[23]

1. Transparent student model
    - What misconceptions do you think I have?
    - Do you think I have mastered vectors?
2. Transparent tutor
    - What strategy are you using?
    - Why did you give me that example?
3. Inspectible, navigable knowledge base
    - What does "vectored interrupt" mean?
    - Give me another example of a convex polyedron.
    - Had electricity been invented when Kant was born?
    - How did you conclude it was a carburetor malfunction?
4. Persuadable tutoring strategies
    - Slow down, explain each step, I'm getting confused.
    - Don't give so many hints, I want to think it through myself.

[23] Some of these may be beyond the current state of the art, or at least outside the scope of Class Z tutors, but we present them to point the direction for future work.

- I think I learn better with quick feedback for wrong answers.

5. Persuadable agenda

   - First teach me about vectors.
   - Skip this topic.
   - Lets stop here, I'll be back tomorrow.
   - I want to experiment on my own now.

6. Collaborative advice

   - What do I need to know to be able to learn about entropy?
   - Is this topic difficult to learn?
   - T: My analysis suggests that you may have one of these misconceptions.....do you want to choose one to try to remediate?[24]
   - T: It seems like you are confused, should I start giving more detailed explanations?

7. Manipulatable learning environment

   - What if the universal law of gravity were an inverse cube law?
   - I'm approaching Jupiter's third moon, fire the left thruster for three seconds.

Making all of this power available to the learner does not mean it will be utilized. Students are not accustomed to having this level of information about, or control over, a learning situation. They are not accustomed to monitoring their learning or problem solving progress [Confrey 1985] and/or altering their learning environment according to reflective metacognitive analysis. When we evaluate the success of such systems we are sure to find that students initially under-utilize the potential. Students must be assisted (by a human or automated teacher) in assimilating the new possibilities available to them. Collaborative, inspectible, persuadable intelligent tutors not only serve to teach subject matter, but provide exciting new arenas for learning and practicing metacognitive skills.

[24]Brown's [1985] physics ITS is a good example of this sort of collaborative tutoring.

Summary of Collaborative, Inspectible, Persuadable Tutors

We began our discussion of student/tutor interfaces by noting that, although we can assume that the computer tutor has significantly more expertise in pedagogy than the student, it is the *student* who has more expertise in issues of the "student model," and that the tutor and student *share* (or have complementary types of) expertise of subject matter. These considerations lead us to propose that future ITS student interfaces be collaborative, inspectible, and persuadable. To elaborate on these three terms we gave a compendium of student interface features with example student interactions for each. Finally we noted that utilizing the flexibility provided by such student interfaces would not come naturally to students and that such interfaces provide unique environments for improving students' metacognitive skills.

6.4 Recommendations for Future Research

This study has unearthed more questions than it has answered and has identified more new issues than addressed previously identified ones. In this section we suggest several fruitful areas for continued research which directly extend this work, describing relevant issues and offering questions that could guide research design.[25]

Representing Additional Domains

The conclusions reached in this study are tentative, partly because the case study involved only one domain and one domain expert. Working with other domain experts and/or building tutors in other domains would provide important new data. Working on domains "near" to the one used in this study (such as adding torque or quantitative problem solving

[25] We also maintain a long list of potential KAFITS modifications and new features, such as a context sensitive help system and a domain knowledge base consistency checker, but we will not present these here. Several suggestions for inferring (rather than storing) slot contents, and possibilities for improvements using AI technology were mentioned in Section 6.3.2.

to the statics domain, or working on kinematics or dynamics) has the benefit of extending our current work and corroborating our findings but has the drawback of not addressing important issues of generality. Representing "farther" domains could answer some of these questions: How much does the KAFITS framework need to be modified in order to represent domains that are primarily procedural or factual? Can KAFITS be used to provide "coaching" style tutorial guidance for micro-world learning environments? Can KAFITS be used to implement a "model tracing" [Anderson et. al 1985a] style tutor in a domain where domain knowledge is represented in an expert system?

Tools for Acquiring Strategic Knowledge

In this study we focused on the teacher's construction of the domain knowledge base, using the associated tools. Unlike the domain knowledge base, which was designed and built entirely by users, the strategic knowledge base (containing the tutoring strategies) was designed by the knowledge engineer with some input from the domain expert.[26] The strategies incorporate recommendations from several instructional design sources but are fairly ad-hoc, few in number, and designed to be adequate, not optimal. An important next step for this work is studying knowledge acquisition processes and tools for strategic knowledge. Motivating questions include: What kind of interface features are most useful for inspecting, modifying, and monitoring strategies? What aspects of designing and managing strategic knowledge are most difficult for educators (especially those who are not programmers)? Are there important properties of teacher-designed strategies and teachers' mental models of their teaching strategies?

Experiments with Instructional Strategies

Another future direction for this work is the incorporation of several strategies from cognitive science and/or instructional science. Here the questions would be: What are the difficulties in formalizing these strategies or principles so they can be represented in a

[26] We have designed a preliminary Strategy Editor (Section 3.2.2), but not through user-participation.

computer system? How do these strategies compare in test runs with students? Can the strategies be represented so that they are understandable and "tweakable" by teachers? Does KAFITS facilitate the evaluation and improvement of these theory-based tutoring strategies?

In Section 6.3.2 we discussed knowledge type classification schemes. KAFITS has the capability to represent K-type schemes and tutoring strategies that are sensitive to K-types, but this capability has not been significantly used. Therefore, another research direction involves implementing a more elaborate K-type classification scheme and the associated tutoring strategies.

Student Modeling and Diagnosis

The KAFITS overlay student model and diagnostic mechanism are its least general and least developed components. Although it has more expressiveness and functionality than most ITS overlay student models, its implementation is ad-hoc and not easily extendible. This is in part because student modeling in non-procedural domains (for which KAFITS is best suited) is quite difficult [Anderson 1983, VanLehn 1983], and little has been done in the field to formalize general characteristics of overlay models (although several working examples of overlay models can be found, for example Goldstein [1982]). The representational scheme and tools for student modeling in KAFITS should be made as general and flexible as its domain and strategic knowledge base representations and tools. Motivating questions include: How should diagnostic rules be represented? Can a single student modeling/diagnostic framework be used for both procedural and non-procedural domains? Can the structure of the student model be made to automatically reflect the structure of the domain and strategic knowledge bases?

In addition, there are a number of features we would like to see added to KAFITS student modeling, including: student-selected confidence or "makes sense" measurements that augment student answers; having students specify "reasons" for their answers; more

recent student answers being given more weight; and storing information about student learning styles and preferences.

Interfaces and Tools

Ideally an intelligent tutor should have powerful and usable interfaces or tools for inspecting, modifying, monitoring, and evaluating its knowledge bases. KAFITS has such tools for the domain knowledge base and the beginnings of such tools for the strategic knowledge base. We also need such tools for the student model (containing declarative information) and the diagnostic mechanism (containing rules). Solving student model representational issues mentioned above is prerequisite to developing these tools.

In Section 6.3.4 we discussed the desirability for collaborative, inspectible, persuadable computer tutors. The KAFITS framework allows for a significant range of student control, but the KAFITS student interface was, like the student model, not extensively developed or studied. This study focused the interaction between the domain expert, the computer system, and the knowledge engineer. More work is needed in designing and testing the student interface. This work is limited because the student interface, more that any other ITS component, is intimately linked with the instructional domain, especially if simulations or micro-worlds are employed. Still, many general student interface features, such as those in the KAFITS student initiative menu (Section 3.2.4), can be studied. Motivating questions include: To what extent can students comprehend and use such a wide variety of options? How can we encourage students to take control? What kinds of questions do students want answered while they are learning? What is the student's cognitive model of her own knowledge and learning process while using a computer tutor? What types of tools will give students a conceptual picture of what the tutor is doing so that they can make informed decisions to control tutoring sessions.

Moving Into the Classroom

Finally, further development and evaluation is needed to push application of the KAFITS tool from the lab to the classroom or training context. Doing this would involve extending KAFITS to address some of the issues mentioned above related to the student model and student interface. It has been said that a piece of software that runs in the lab is only ten percent of the way to being a product. Similarly, there are many unforeseen yet important issues related to routine use of KAFITS by teachers and students in natural settings. Questions motivating research include: What is the best role for the teacher—how much and what type of student guidance is optimal? What is the best mix of classroom teaching and computer tutoring? Are students more motivated when they work in pairs? Do students use the student control capabilities more when they work in pairs? How can KAFITS integrate into existing classroom structures, such as grading, labs, and classes of limited length? Does KAFITS facilitate multiple experts (teachers) designing and modifying the knowledge base?

6.5 Recommendations for the Instructional and Educational Communities

Although we have argued for the importance of including educators in building ITSs, we have not offered much evidence of benefits of incorporating ITSs in education (except that the domain expert indicated that he learned things from the experience of building a tutor that could be used in his classroom teaching—see Appendix C). In Section 1.1 we discussed the great "potential" of ITSs, and Shute [1990] offers some recent evidence that using ITSs can result in significant enhancement to learning in both classroom and workplace training situations. We will therefore take it for granted that ITSs do enhance learning when designed appropriately, used in domains where they have shown to be effective, and when properly integrated into educational infrastructures where students and administrators fully support ITS learning.

But this still leaves many questions. Most relevant to this study is the question of whether an ITS *shell* is useful and effective for widespread use in realistic educational settings. First we will present an idealist (best-case) vision of how a KAFITS-like (or Class Z) system could be integrated into the school setting. Then we will discuss issues hindering the realization of this vision. Finally we predict that ITSs will, in the near future, have more impact in industry training than public schools.

A Best-Case Future Scenario

Class Z tutors are now standard tools for designing intelligent tutoring, training, and assistance programs. The price of the necessary hardware and software has fallen to a reasonable level, and ample funding has been provided to integrate ITSs into schools. Some teachers integrating ITSs into their courses can be found at all grade levels for most academic subjects (though ITS use is by no means universal). Peer groups of teachers exist in most schools so that teachers can support and learn from each other in their efforts to make the best use of this new technology. Most of these teachers have participated in one-week teacher training seminars on how to incorporate ITSs and their associated curriculum materials into the classroom. These seminars include discussions of grading, progress monitoring, classroom management[27] and an overview of basic ITS concepts.[28] Some teachers have gone on to further workshops that instruct them in how to use ITS shells such as KAFITS to alter an ITS, so that they can, for example, change the text of an explanation, add new examples, change the prerequisites of a topic, or change the teaching strategy to give more hints. A small number of teachers have become "domain experts" who participate in ITS development teams.

The number of domain experts building intelligent tutors is of the same order of magnitude as the number of educators writing textbooks and designing professional quality

[27]Including discussions of classroom "topologies" (eg. a single computer used for demonstrations in front of the class; one computer per pair; per group; per student).

[28]Including a discussion of what the "intelligent" in ITS means and does not mean, so that teachers do not project too much "smarts" into computer tutors.

curriculum materials for widespread use. ITSs are accompanied by curriculum materials such as workbooks, descriptions of lab experiments, teacher guidelines, etc. The rapid-prototyping nature of ITS shells (tools) allows designers to: easily test and modify the tutors, build new tutors on top of existing tutors, and evaluate the effectiveness of alternative instructional theories and strategies. As with textbooks and current-day CAI, teachers have several published tutors to choose from for any given subject. The number of teachers trained to modify the content or strategies of a tutor is on the order of the number of "master" teachers in a school system. Typical teachers may make a few minor modifications if they are ambitious and very comfortable with computers in general, but most do not. Also, some ITSs are published with a small number of categories of changes pre-defined, so that teachers can tailor the software for their needs easily, if their needs are captured by the pre-defined list of options. The incorporation of practicing teachers as co-researchers in academic studies of ITSs is becoming widespread, as is including teachers on design teams that produce ITSs for distribution.

Most teachers using these systems, especially those who study the topic network and strategy networks of a tutor (tools are provided to view these) report that they have a more sophisticated and organized understanding of the relationships between topics in their subject, have come to appreciate the importance of anticipating misconceptions, and are thinking of their own teaching more in terms of "strategies." Those teachers who go the extra step to modify the tutor to fit the needs of their classroom, curriculum, or teaching style, experience an increased sense of "ownership" and understanding of the intelligent tutor. They are excited by the possibility of trying new strategies and curriculum structures. They report that modifying ITSs causes substantial reflection on their teaching and considerable reconceptualization of their understanding of the domain and the instructional process.

Problems with Realizing the Vision

Clearly, the vision described above is not a probable extrapolation of current trends in education. We suggest several current trends or factors which act as barriers to the

realization of our vision of incorporating ITSs into the educational system (assuming that eventually adequate technology will be readily available). These trends and factors are given to enumerate obstacles to using ITSs in schools, and point to areas of potential problems *should* ITSs be used routinely in schools.

- Funding and other rescources. Substantial funding would be required to realize our vision in public schools. Some of the cost would involve hardware and software, but even assuming these were donated or priced reasonably, resources are needed to restructure how classes are taught to take best advantage of this new technology. Additional classroom and/or lab space would be needed in most schools. Also, in introducing computers into the classroom there has been a fairly consistent trend to spend more money (proportionally) on hardware, less on software, and very little on teacher training or support (as expanded on below). This disproportionate allocation of funds [Johnson 1988] leaves schools burdened with computers no one is trained or willing to use, or worse yet, with outdated hardware that has no useful software written for it.

- Teacher training and creative planning time. Using ITSs in the classroom could eventually lighten the teacher's load, but there is a startup time required for learning how to incorporate ITSs into classrooms. Unfortunately, teachers do not get sufficient paid time for teacher training or job enhancement, and when they are given the opportunity to attend a workshop, there is usually no support system at their job site that encourages them to integrate what they have learned into their classroom, and there is no follow-up evaluation and feedback on their attempt to integrate the new technology. In addition, our vision assumes that teachers have the time to think creatively about their curriculum and try out new content or strategies—but "extra" time for creative planning of classes is not available to many teachers.

- Teacher acceptance. "Intelligent" tutors may be threatening to some teachers for many reasons. First, as has been found with incorporating existing computer software into classrooms, students tend to quickly learn more about the software than the

teacher and are more familiar with computers in general than the teacher. This requires that teachers take on new roles in the classroom and challenges some teachers' sense of competence, autonomy, control, and authority. "Intelligent" tutors pose the additional threat of a machine that knows more about the subject matter than the teacher. Though this may rarely be true in reality, the *perceived* threat will be very real. In an ethnographic study of the attitudes and beliefs of students, teachers, and administrators in a school where the GEOMETRY tutor (an ITS developed by Anderson & Boyle [1985]) was being used in five geometry classes, Schoefild & Verban [1988, pg. 20] note "evidence of many teachers' indifference to or even resistance to the idea of using computers in their teaching." Allowing teachers to inspect and modify computer tutors will tend to demystify the systems and give teachers a sense of control, but the teachers most likely to feel threatened are the ones *least* likely to "open up" and play with this new technology.

- Organizational support. The support (on many levels) of educational administrators is crucial for new technology to have an impact, but the educational institution/infrastructure is often slow to change, or changes only in reaction to crisis or public opinion.

- Computer naivete. Unfamiliarity with the nature of computers can have two manifestations. At one extreme is fear and recalcitrance to change, at the other is passive acceptance of ITSs, over reliance on them, and over-estimation of their capabilities (or "intelligence").

- Embedding ITS concepts into the educational culture. When a new technology is introduced into a sub-culture, there is a (usually awkward) startup time in which the creators and users of the technology assimilate some of each other's models and concepts. It will take a while for educators to understand the key concepts, capabilities, and limitations of ITSs, and it will take a while for the builders of ITSs to establish design criteria that meet the needs of users (both teachers and students). Consider the first word processing programs. Though they are commonplace now, when word

processors were introduced, there was no shared corpus of models or vocabulary about integrating them into the home or work environment. The key concepts of malleable text, separating text from formatting, permanent vs. temporary computer memory, etc., took a while to sink in. It also took some time for industry to stabilize good standards for manuals, help features, and tutorials (if they yet have). ITSs are more complex than word processors, perhaps more comparable to CAD/CAM programs. The learning curve for embedding ITSs into the educational culture will probably be even longer than that for word processing systems.

Individually, each of the above factors could conceivably be mitigated by a concerted effort involving parents, educators, and administrators. But when these factors are considered as a whole, the author must admit to a fair amount of pessimism regarding the probability of such dramatic change in the educational system.[29] Therefore, we must look to the work place and home for a more optimistic prediction of near-term ITS use.

ITSs in Industry

Although all of the above trends and warnings are applicable to industry as well as public education, they are less severe in industry settings. Businesses are freer to allocate resources, prioritize human efforts, and instigate follow-up and support systems, and can implement these changes at any level of an organizational structure (for example they can try it out on a small scale). Also, adults in training programs tend to be more motivated learners than children and young adults in public schools. For these reasons, we predict that in the near future ITSs and ITS shells will have more impact in industry than public education.[30] Once ITSs are accepted and understood in the context of industry training (and perhaps in home education) they will integrate more easily into schools.

---

[29]It is conceivable that the hardware and software would be available, but the other problems are more difficult to tackle. Schofield & Verban [1988, pg. 1] note that the "rapid proliferation of microcomputers in schools [over the last ten years has been] truly startling [but that] the effect of this change...is not [as obvious]."

[30]See Johnson [1988] for pragmatic considerations in implementation of ITSs in industry and military training.

APPENDIX A

Terms and Definitions

Below we give word senses, meanings, and synonyms for some of the important concepts and terms used in this document.

- AI. Artificial Intelligence. Discussed in Sections ?? and 6.3.2.
- Browser. The interface for viewing and editing information in the domain knowledge base.
- CAI. Computer aided (or assisted) instruction. In ITS research CAI usually refers to "non-intelligent" or traditional educational software.
- Design process. In most of this document "the design process" refers to the process we used to build the statics tutor; it is synonymous with "the knowledge acquisition process" since our study focused on knowledge acquisition.
- Domain. We use the word "domain" to mean the content area or subject matter area. Domain expertise is expertise in solving problems in the domain.
- Domain expert. Traditionally in AI, the domain expert is the person serving as the source of expertise in building an expert system. The terms "teacher," "tutor," "domain expert," "subject matter expert," and "instructional designer" are used interchangeably to refer to the hypothetical user of the KAFITS tool, or, in some contexts, the teacher who participated in this study.
- Educator. Term used to cover teachers, tutors, instructional designers, and educational researchers.
- Instance. (See Object.)
- ISD. Instructional systems design. Also called instructional theory, instructional design theory.
- ITS. Intelligent tutoring system. Intelligent tutoring systems, intelligent learning environments, knowledge based tutoring systems, and intelligent computer aided instruction (ICAI) systems are different terms with similar meaning, i.e computer assisted learning programs which incorporate artificial intelligence technology and paradigms. These terms can imply a focus on different issues for different authors, but for the purposes of this document we treat them as equivalent.

- Knowledge acquisition (KA). Knowledge acquisition is the process of acquiring an expert's knowledge for representation in an AI system.
- KAFITS. Knowledge Acquisition Framework for ITSs. KAFITS is a representational *framework* and an *interface* allowing an instructor to represent his/her knowledge in terms of that framework. Usually "KAFITS" refers to both the framework and the interface, unless the distinction is relevant and otherwise noted.
- Knowledge base (KB). The information stored in an expert system. KAFITS has two main knowledge bases: the domain knowledge base and the strategic knowledge base.
- Knowledge base manager. The knowledge base manager is a member of the ITS design team whose task is to input the knowledge as specified by the domain expert (usually on paper worksheets) into the knowledge base and test the curriculum for obvious errors (i.e. errors not related to the domain content).
- Knowledge engineer, knowledge engineering (KE). The knowledge engineer is a scientist or engineer who works with a domain expert to represent domain expertise in an AI system. "Knowledge engineering" is the process of eliciting the domain expert's knowledge and encoding it in an AI system.
- LE. Linear equilibrium. A topic in the statics curriculum dealing with the balancing of forces acting on a stationary abject.
- LE-curriculum. The portion of the statics curriculum surrounding the linear equilibrium topic. This subset of topics was the focus of the first several phases of this study.
- Lesson, topic, presentation. These terms refer to KAFITS object types unless otherwise stated.
- Micro/macro levels. The macro level of instruction involves "what to teach" and the micro level involves "how to teach it."
- Object, instance, slot. An object (or frame) is the fundamental unit of representing things in many AI systems. AI objects (and frames) refer to an entity and specify the important attributes of that entity, and usually also specify values or default values for these attributes. Objects are different from "frames" in that objects have procedures called methods associated with them. There are typically two types of objects in object-oriented systemsD: classes and instances. Classes refer to categories of things (such as Dogs) and instances refer to specific entities (such as Fido, an instance of the class Dog). The attributes (or parameters, or properties) of objects are called *slots* (eg. Color, Size, and Owner for Dogs).
- PC Matrix. Performance-Content Matrix. A knowledge type classification scheme originally designed by Merrill [1983]. KAFITS uses a modification of Merrill's PC Matrix.
- Pedagogical knowledge (or pedagogical expertise, sometimes called propeadutics). Information about how to teach, including tutoring strategies and specifics about the

topics such as prerequisite, examples, etc. used specifically for teaching (as opposed to being used in problem solving or performance in the domain).

- Slot. (See object.)
- Statics tutor. The KAFITS system combined with the knowledge base for the statics domain.
- Strategy, strategic knowledge. Tutoring strategies are explicit representations of tutoring rules or principles.
- Strategy Editor. The interface for viewing and editing the information in the strategic knowledge base.
- Teacher, tutor, instructor. Used interchangeably unless use indicates a more specific meaning. (See domain expert.) In this document when we refer to a *computer "tutor"* we usually mean a tutor built using KAFITS, i.e. the KAFITS system combined with the knowledge base of a particular domain. Similarly "the tutor" will usually refer to the statics tutor built during this study.
- Tutoring strategy, tutoring rule. These are used interchangeably unless use clearly implies a more specific meaning.
- User. The term "user" refers to persons using the KAFITS system for knowledge engineering (primarily domain experts and knowledge base managers), *not* students using the tutor.

The following abbreviations are used in the Bibliography:

- **AAAI.** American Association of Artificial Intelligence.
- **ACM.** Association of Computing Machinery.
- **IJCAI.** International Joint Conference on Artificial Intelligence.
- **ITS.** Intelligent Tutoring Systems conference.
- **LRDC.** Learning Research and Development Center, University of Pittsburgh, Pittsburgh, PA.

APPENDIX B

Knowledge Type Descriptions

The following document was given to KAFITS users to familiarize them with the knowledge types used in KAFITS. (This classification scheme is more elaborate that the one used in KAFITS for this study, shown in Figure 2.2.)

A Classification Scheme for Types of Knowledge and Instructional Objectives

In this document we explain a system for classifying types of knowledge, instructional objectives, learned behavior (all three of there terms have essentially equal meaning for our present purpose). (The system is an extension of a system designed by Dr. David Merrill called a Performance-Content Matrix.)

CLASSIFY ACCORDING TO COMPLEX VS. BASIC KNOWLEDGE

We define two broad categories of knowledge (information, instructional objectives, skills, or learned abilities), Basic and Complex.

Basic knowledge

Definition: An instructional objective is Basic Knowledge if it can be categorized as a fact, concept, skill, or principle. These four terms are defined later.

Complex knowledge

Definition: Complex knowledge includes any instructional objective that is not Basic knowledge. Complex knowledge is a catch-all category for the types of knowledge that are too complicated to be defined clearly and concretely (by educational and psychological researchers).

Examples: Complex knowledge includes the following types of abilities and skills:

- General problem solving skills (such as breaking the problem into parts, checking the answer, etc.)
- Metacognitive skills (self-diagnosis or self-analysis of one's thinking or problem solving process).
- Scientific inquiry, discovery, and hypothesizing skills (includes data collection and analysis skills).
- Formal logical skills, such as deduction.
- Mental models. Complex "gestalts" of densely connected information allowing a system to be modeled and mentally "run."
- Creativity. (According to any number of definitions.)

Complex skills are ubiquitous, needed for the mastery of many subjects. From the list of examples given above, you can see that some of these overlap and most of them are

hard to define precisely. Thus, we are mainly interested in determining whether an instructional item is Complex or not, and not concerned with categorizing sub-types of Complex knowledge.

THE PERFORMANCE-CONTENT MATRIX

The diagram [Figure 2.2 in this dissertation] shows the Performance-Content Matrix, with the classification scheme for Basic Knowledge There are four content types: Facts, Concepts, Principles, and Procedures. We classify knowledge in this way to help organize instructional objectives and to aid in determining teaching methods (since different knowledge types usually require different methods).

There are six levels of performance possible with each content type (not including the parts of the matrix blocked out). These levels allow further useful distinctions in instructional objectives, for example, memorizing of the steps of a procedure (Remember); vs. actually being able to use the procedure (Apply); vs. being able to invent a new and better procedure which accomplishes the same goal (Create); vs. knowing in what situations the procedure is useful (Meta-knowledge).

The terms we are using may have other meanings in other contexts. We have tried to use clear terminology, but some confusion is inevitable, therefore we include examples of the knowledge types and discuss possible confusions in this document.

CONTENT TYPES

Fact

Description. Facts are arbitrary associated pieces of information.

Examples. A proper name, a date, an event, the name of a particular object.

Possible confusions. Fact vs. Remember. The Remember performance level is sometimes confused with the Fact content type. The *definition* of a concept, the *statement* of a principle, and the *list* of the steps in a procedure can all be memorized and recalled. These are all instructional objectives at the Remember level. We use the category Fact for memorized bits of associated information that *are not* part of the definition of Concepts, Procedures, or Principles.

Fact vs. Meta-knowledge. Facts are also sometimes confused with the Meta-knowledge performance level. The Meta-knowledge level is for information *about* knowledge, such as why it is good to know about it, when it is used, where you learned it, etc.

Classification hints. Due to the possible confusions with the Fact content type vs. the Remember and Meta-knowledge performance levels, make sure the thing you are classifying is not a Remember or Meta-knowledge level item (for a Concept, Procedure, or Principle) *before* you decide to classify it as a Fact.

Elaboration. The learner's knowledge of a fact is evidenced by recalling something from memory. The student is given some stimulus, such as a name, symbol, picture, etc., and is asked to recall some associated information.

Concept

Description. Concepts are groups of objects, events, situations, attributes, or symbols that share some common characteristics and are identified by the same name or phrase. Using concepts involves recognizing or analyzing the characteristics (or properties) of things.

Examples. Most of the words in a spoken language are concepts. Some concepts are concrete, such as "toaster oven", and some are more abstract, such as "equity". Some concepts have very exact criterion (definitions), such as "mammal", "president", and "Monday", and some have more fuzzy criterion, such as "chair" and "symmetry".

Possible confusions. The word "concept" is used for many things, such as in "having a conceptual understanding of." Our meaning here is limited to the *classification* sense of the word "concept". That is: identifying instances of a Concept, being able to invent new instances of a concept, etc. If you want to know whether a knowledge type for some piece of content is a Concept, ask yourself whether you are talking about classification of things, or distinguishing between different types of things, as in "is this a.....", or "what kind of thing is this?" or "is this a situation where you need...?"

Concept vs. Principle. Concepts are sometimes confused with Principles (which are relationships between concepts). For example, the equation "F = m a" (Newton's second law) is a Principle. Each of the components, force, mass, and acceleration, are Concepts. We can ask the student to determine whether a situation involves the Concept of force (as opposed to, say, momentum). This involves a Concept because it is a classification of situations (into those which do and do not involve force). However, to "understand Newton's second law" is to understand a Principle.

Concept vs. Meta-knowledge. Concepts are sometimes confused with Meta-knowledge (a Performance Level) and Complex Knowledge (such as problem solving skills). The word "conceptual understanding", used to mean deep understanding of some topic, is not the meaning of Concept we use here (though having a deep understanding usually *includes* having an understanding of the Concept).

Concept vs. Procedures. Procedure sometimes followed to label or classify things. If the instructional goal is to memorize or use this specific procedure, the content type is Procedure. For example, "Igneous rock" is a Concept, and understanding the concept involves begin able to identify things that are and are not instances of igneous rock, but a specific step-by-step process for identifying igneous rock is a Procedure.

Elaboration. The purpose of this document is to assist in learning the meaning (or *our meaning*) of Concepts such as Fact, Performance-level, Concept, Basic-Knowledge-type, etc. and being able to classify instructional topics using these Concepts.

Procedure

Description. An ordered sequence of steps and decisions necessary to accomplish some goal.

Examples. Divide 455 by 15 (the long division algorithm). The procedure given for filing tax returns. Solve the following linear equations.

Possible confusions. (See the Concept vs. Procedure possible confusion above.)

Elaboration. Being able to perform a step in a procedure or make a decision within in a procedure requires sufficient understanding of the Concepts used in the step or decision.

Principle

Description. Principles are explanations or predictions of why things happen in the world. They are cause-and-effect, correlational, or constraint relationships.

Examples. Physical laws in the form of equations, such as "PV=NRT" (the Ideal Gas Law), are principles. Other principles, Hot air rises; He who laughs last laughs best.

Possible confusions. Principles are often stated in terms of equations. The *general* ability to solve word problems and equations is a Complex Knowledge Type, not a Principle.

Mental Models often consist of a closely related set of Principles, such as how the weather system or how car engines work. Each individual relationship is a Principle (such as "heating air causes it to rise" or "the starter causes the engine to turn over"), an understanding of the entire set (such as "what causes rain fall?" or "how does a car engine work?") is (very roughly) a Mental Model.

Elaboration. Principles consist of a relationship between Concepts. The Concepts must be sufficiently understood to be able to use the Principle.

PERFORMANCE LEVELS

Remember

Performance at the Remember level requires the learner to search memory in order to reproduce or recognize information.

Sub-levels of the Remember level. There are four sub-levels of the Remember performance level, i.e. there are four types of performance behavior possible. They depend on two things: whether the information is recalled verbatim or paraphrase, and whether the information recalled is a generality or instance. Thus, the four levels are: Remember-generality-verbatim, Remember-generality-paraphrase, Remember-instance-verbatim, and Remember-instance-paraphrase.

Verbatim vs. Paraphrase. Verbatim is used in its usual sense, meaning that a fact, definition, steps, diagram, etc. must be recalled exactly as originally learned. A paraphrase is an

alternate representation which has the same meaning. The recall of the alternate representation could be via an alternate wording, or via an alternate mode of representation such as a diagram, by pointing, graphs, etc.

Instance vs. Generality. Generalities are abstract definitions, rules, procedures, etc., which are true or applicable for many situations. An instance is a specific example of a concept or a specific application of a procedure of principle. All facts are Instances. Instances of Principles are *explanations* of how or why the Principle applied to a specific situation. Instances of Procedures show the steps taken to use the procedure in a specific situation. Instance of Concepts are exemplars of the concept.

Possible confusions. Recalling alternate representations can be a Complex Knowledge Type if it involves a lot of inferencing, such as converting between written descriptions of a situation, formulas centered representations, graphical representation, and diagrammatic representations. We do not include such conceptually difficult or complex tasks in the Remember level.

Use (or Apply-Use)

Description. Successful performance at the Use level requires that the student be able to apply the knowledge within a context where it is clear that the piece of knowledge is applicable.

Problem-Solve (or Apply-Problem-Solve)

Description. Successful performance at the Problem-Solve level requires that the student recognize that the piece of knowledge is applicable, and then apply it. Given a physics problem which requires using "F=MA," a Use performance task would be: "Use Newton's second law to solve the following problem...". A Problem-Solve performance task for the same problem situation would be "Solve this problem...". In the later case, the student needs to *recognize the need* for Newton's second law and *then* apply it.

Create-Instance

Description. The Create-instance performance level requires that the student find or create a new instance of a generality.

Create-Generality

Description. The Create-generality performance level requires that the student derive, invent, or find a new abstraction, given two or more instances of it.

Meta-Knowledge

Description. Meta-knowledge is knowledge about knowledge, such as: why it is important to know, where it is used, where you learned it, whether it is hard to apply, etc. It is *explicit* knowledge, i.e. the student demonstrates the knowledge verbally (or in written form).

APPENDIX C

Post-Study Interview with the Domain Expert

What follows are excerpts from an interveiw with the domain expert done on March 26, 1991, several months following the end of this study. The domain expert had continued to work improving the statics tutor in the period between the end of this study and the interview. We prepared a series of interview questions, most of which were answered, but the ordering of topics in the interview was partly determined by the flow of the conversation. We do not include a complete transcription of this interview, nor an analysis of the interview, as part of this study (though these could be done at some future date). In the excerpt below "KE" refers to the knowledge engineer (the interviewer), and "DE" refers to the domain expert (Camp). All knowledge engineer questions are paraphrases rather than quotes.

[First the knowledge engineer (KE, interviewer) overveiwed the project steps (see Figure 5.1) with domain expert (DE).]

KE: What was good about the entire process?

DE: What's exciting about designing and working on something like this is that there's just an incredible amount of room for creativity...the environment is rich enough so that...if a teacher is interested in getting ideas across to people...[he/she has the flexibility and power to do so]. It's a really great challenge; it's a really fun environment for somebody like me to work in.

KE: What wasn't good, what was hard, or what could have been improved about the entire process?

DE: The thing that jumps to mind first is [that there were times] we were having sort of difficulty between hardware and software or something [and] doing the same thing over again about three weeks in a row, at least it seemed like that...Kim and I were spinning our wheels quite a bit, where we kept loosing stuff or blowing our stuff up or something...ya know—[the software was] in development there for a while and things seemed to get in trouble. ...For me that was one of the more frustrating parts.

KE: What was good about the interface, the tools and software?

DE: What really jumps to mind is the ability to go plowing into it and saying...I'm going to act like a student and test this thing out and...going along and saying 'that's not right...I want to fix that,' and not having to write some long complicated note and come back two hours later and get into the right module and the right section, but be able to stop the damn thing to go in, fix it [chuckles], get back out, pick up where you left off and keep going. To my mind this is really amazing, and...makes it so much more possible to [improve the tutor] during development and testing. That feature makes a tremendous difference in how far you get when you work on a big project like this.

It was really nice the way the system continued to evolve and got so much easier to use. [For instance, with the topic level displays] showing where you are [in the curriculum]...I can just see things cranking along over there on the right hand screen...[it is] nice to visually keep in touch with where I am at [as the information] evolves. [Before that tool was developed] I can remember [Kim and I] suffering from...'I want to fix something, but where the heck [in the knowledge base] is it?'—so the [displays are] such a powerful feature. There is so much information available to help figure out where you are and what's wrong and where to fix it in reasonably short order.

KE: What was the most difficult or hardest to use about the tools and software?

DE: I certainly remember for while in the beginning thinking, 'gee, how long is it going to take me to get comfortable with this?' As I look back I can't think of anything in particular that was] silly or needlessly complicated for what we are trying to do here...it seems like a really fine structure.

...Being clear about [the difference between] topics and presentations took a while.

[Interviewer asked again about problems with the interface, but Charlie did not have any more specific comments, except to say there were some features he didn't use.]

KE: What was good about the statics tutor itself?

DE: In spite of my sort of initial reservations I would have to say [the flexibility of the curriculum]. At first I thought 'That's crazy, he wants [the student] to be able to start anywhere in here!' [i.e. in the curriculum topic network]. However, I do feel as I look at this now that there are really about four major logical starting places where a person could dip in here and start to learn a good deal, with a little background.

To me its amazing the different kinds of [curriculum] relationships that this has made possible...[for instance] the idea that the misconception strategy is really different than other strategies, [and] the free-body diagram solution strategy, [and] that you can say 'let's let them play with [the crane boom] for a while' in the linear equilibrium intuition stuff—the ability to allow for that much variety is really amazing.

KE: What could have been better about the statics tutor?

DE: The thing that troubles me is that I wish I might have been able to use a bit more variety in terms of the style in which I designed questions and [answers]. I don't know if that was so much a limitation of the system or a limitation of my imagination. I have thought 'aw, another multiple choice question here with a sort of not wholly exciting picture to go with it,' it would be nice to think of a wilder way to go at it. At times I wish there were not such a percentage of multiple choice. Good multiple choice really requires a lot of attention to build. But good multiple choice has some advantages [over other response methods]. It would be nice [for the student] to be able to type in equations, or type in an explanation to an answer.

KE: If you were going to do it all over again what would you do differently?

DE: In the best of worlds I would have more people power resources...three teachers, a couple more helpers...

KE: What about the help features and assistance features?

DE: I didn't tend to look at the system a lot in terms of the system helping me, which may be partly due to the way I learned about computers and programming on some pretty unfriendly systems. [My attitude was] 'if you've got a problem, you've got to figure it out yourself, and if [one attempt] doesn't work, try something else.' That may say something about my mind set. I do notice my [high school] programming students using help utilities..with great ease..but it is not my immediate reaction to say 'Oh, let me look it up.' I was to a large degree focused on content.

KE: What was good and bad about working with the knowledge base managers?

DE: When I was trying to get a lot of content out [on worksheets] it was really good not to be distracted with hardware and trying to push it all into the machine. Kim [Gonzalez] was a physics person [has a degree in physics] so I interacted more with her than Frank [Linton]...she asked some valuable questions. I felt comfortable around both of these folks.

[There was] probably more frustration on their side that mine..since I was only here once a week.

KE: Do you have any suggestion for training future domain experts, or any suggestions for future domain experts?

DE: I thought all of [the information I was introduced to was] really important stuff. It was tricky enough to get started from scratch...it does take a bit of a feel to get on board...and we reviewed [some material and terms] quite a few times. I would say that they would definitely want to be patient with themselves. You really don't want to do this on an island, but work with someone else. I have a hard time seeing someone just sitting down to tackle this alone [without a knowledge engineer.] They will be happier if they interact and share this with somebody.

KE: How important was the ITS Summer Teacher's institute?

DE: ...[some of the material was not directly relevant but] having those concepts rattling around my mind for a year had some effect.

KE: What's your gut feeling about how this kind of tutor can be used in a classroom (given that the students we tested were above average in ability)?

DE: That is a very interesting issue. I'd love to be able to fool with it more [and] see how we could do with more average kind of students. I'd like to see how much the advanced students could do with [only] a little background [in the prerequisite concepts]...but I don't have any solid answers on that.

KE: Are there any big pieces missing before moving this into the classroom?

DE: [Not really, but] the student model [i.e. incorporating confidence measurements and more elaborate abstractions of student mental states] is an interesting challenge down the line...sounds pretty tough to me...a whole other can of worms.

Following are questions asked the domain expert that are not summarized here:

- How do you think things would have been different if you came in every day instead of once a week?
- About how many changes were made to the knowledge base as a result of the student trails? On the average, what kind of changes were they?
- What do you think about the way the curriculum was designed in a linear classroom style, then later on worksheets? Did the worksheets work out OK; can you picture doing it without them?
- What do you think about the quantitative results [Camp was shown total time spent on training and implementation for the four participants, see Figure 5.7] and the assumptions made in the calculations (as listed in Section 5.3.1)? [He found no obvious problems with the figures or assumptions.]

APPENDIX D

KAFITS Menu Operations

Below is a hierarchical list the KAFITS menu operations, showing the features available to the user (instructional designer or domain expert) and the student (in the "student menu").

- **Student Menu**
  - Run the Statics LESSON
  - Make a student COMMENT
  - TEACH a topic
  - DESCRIBE a topic
  - TEST my knowledge of a topic
  - Give me a HINT
  - Tell me the ANSWER
  - REPEAT the last presentation
  - Change the TEACHING STYLE
  - Display session STATUS
  - Play with the SIMULATION
  - QUIT the tutor
- **Knowledge Engineering Menu**
  - **Student Model Utilities**
    - IGNORE the student model
    - RESET the student model
    - SAVE the student model
    - LOAD a student model
    - Full SM SUMMARY
    - BRIEF SM summary
    - LIST all SM objects
  - **KB Utilities**

    * CLEAR all windows
    * Change the screen CONFIGURATION
    * Test a PICTURE object
    * Test a CRANE BOOM object
    * Alter crane boom COLORS
    * List SLOTS in knowledge base
    * Load changes for DEMO
    * Toggle show TOPIC NET
    * Install NET-EDITOR menu
    * Install NET-PROGRAMMER menu

- **Strategy Utilities**
    * Show CURRENT strategies
    * NEW current RESPONSE strategy
    * ALTER response strategy
    * CREATE response strategy
    * ERASE response strategy
    * NEW current TOPIC strategy
    * ALTER topic strategy
    * CREATE topic strategy
    * ERASE topic strategy

- **PAN Utilities**
    * CREATE a new PAN
    * HIDE current PAN
    * Hide ALL PANs
    * SHOW all PANs
    * Show SELECTED PANs
    * SAVE PANs
    * Toggle TRACE PANS
    * Toggle use DEFAULT NAMES
    * Toggle DUMMY nodes& arcs
    * SELECT a PAN

- **Preferences**
    * Trace window on?
    * User type
    * Trace dialog?
    * Use student model?
    * Trace file name
    * Demo lesson name
    * SAVE preferences

        * SHOW preferences
        * DESCRIBE preferences
    - REMOVE tutor menus
    - Tutor HELP/INFO
    - Start BROWSER
    - Install LISP menus
    - QUIT tutor and Lisp

- **Browser Menu**
    - **Record and File Operations**
        * Make a COMMENT
        * SAVE all instances
        * View an EDIT RECORD
        * View SESSION TRACE
        * View KNOWLEDGE BASE
        * Choose any FILE to open
    - **Misc. Operations**
        * Reset RESENT INSTANCES
        * CROSS-REFERENCE the K-base
        * Raise Browser WINDOWS
    - **Preferences**
        * Operator name
        * Record-file name
        * Set Type double-click action
        * Set Instance double-click action
        * Set Slot double-click action
        * SAVE preferences
        * SHOW preferences
        * DESCRIBE preferences
    - QUIT Browser

- **Browser Pop-up Menus**
    - **Type Operations**
        * New INSTANCE
        * New instance with MIXINS
        * DESCRIBE types
    - **Instance Operations**
        * VIEW instance

    * EDIT instance
    * COPY instance
    * DELETE instance
    * BROWSE instance
    * TEST instance
    * Add instance NOTE
    * DOCUMENTATION on slots
    * View CROSS-REFERENCES
    * STUDENT MODEL status
  - **Slot Operations**
    * VIEW slot
    * EDIT slot
    * DOCUMENTATION slot
    * BROWSE slot
    * LISP inspect

- **Browser Editor Menu**
  - DONE editing
  - ABORT editing
  - REVERT buffer
  - Editor HELP
  - CLEAR buffer
  - Slot DOCUMENTATION
  - PASTE previous edit

APPENDIX E

Strategies Used in the Statics Tutor

In this Appendix we show the PANs for the default strategies used in this study. KAFITS allows strategies at four levels: Lesson, Topic, Presentation, and Response.

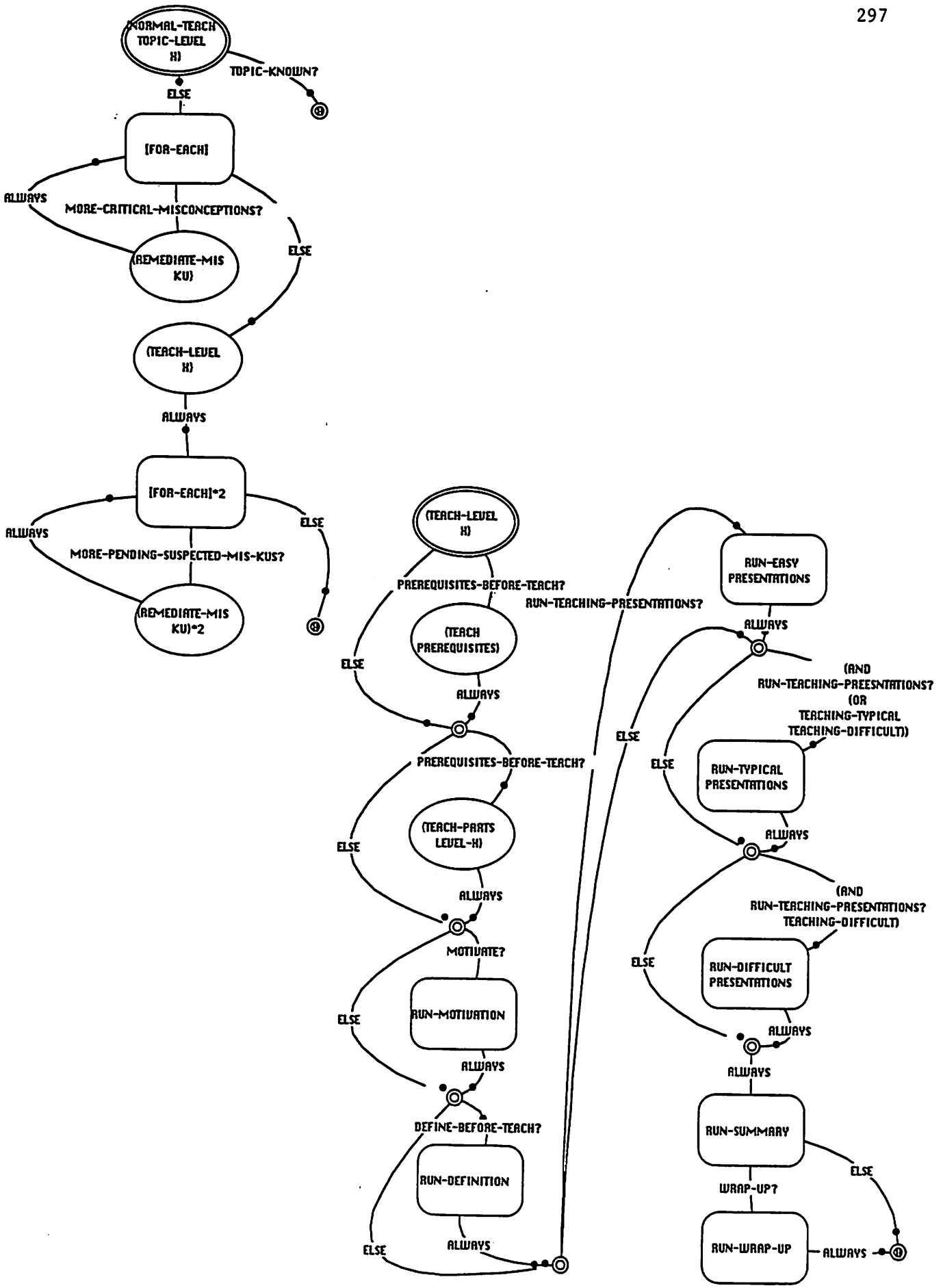
The default lesson level strategy is trivial (and not shown below): all the topics listed as goal topics in the lesson sequence are taught (at the typical performance level) in the order listed.

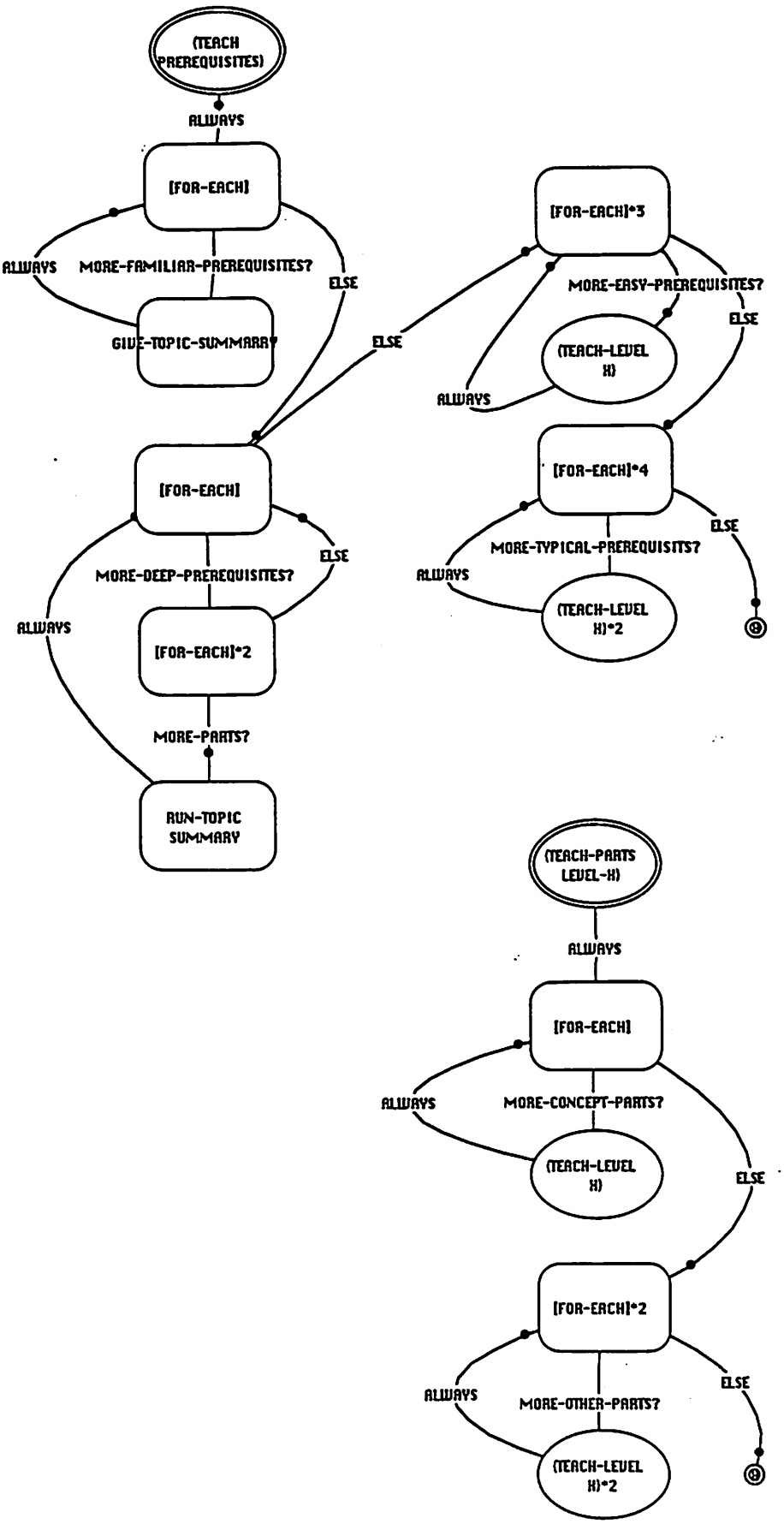
The default presentation level strategy is also trivial (and not shown below): all of the presentation for a given topic level are run in the order listed. An alternate presentation level strategy was used for the FBD-identify-forces topic. This strategy involves giving a series of tasks related to a sequence of crane boom configurations, and is shown at the end of this Appendix.

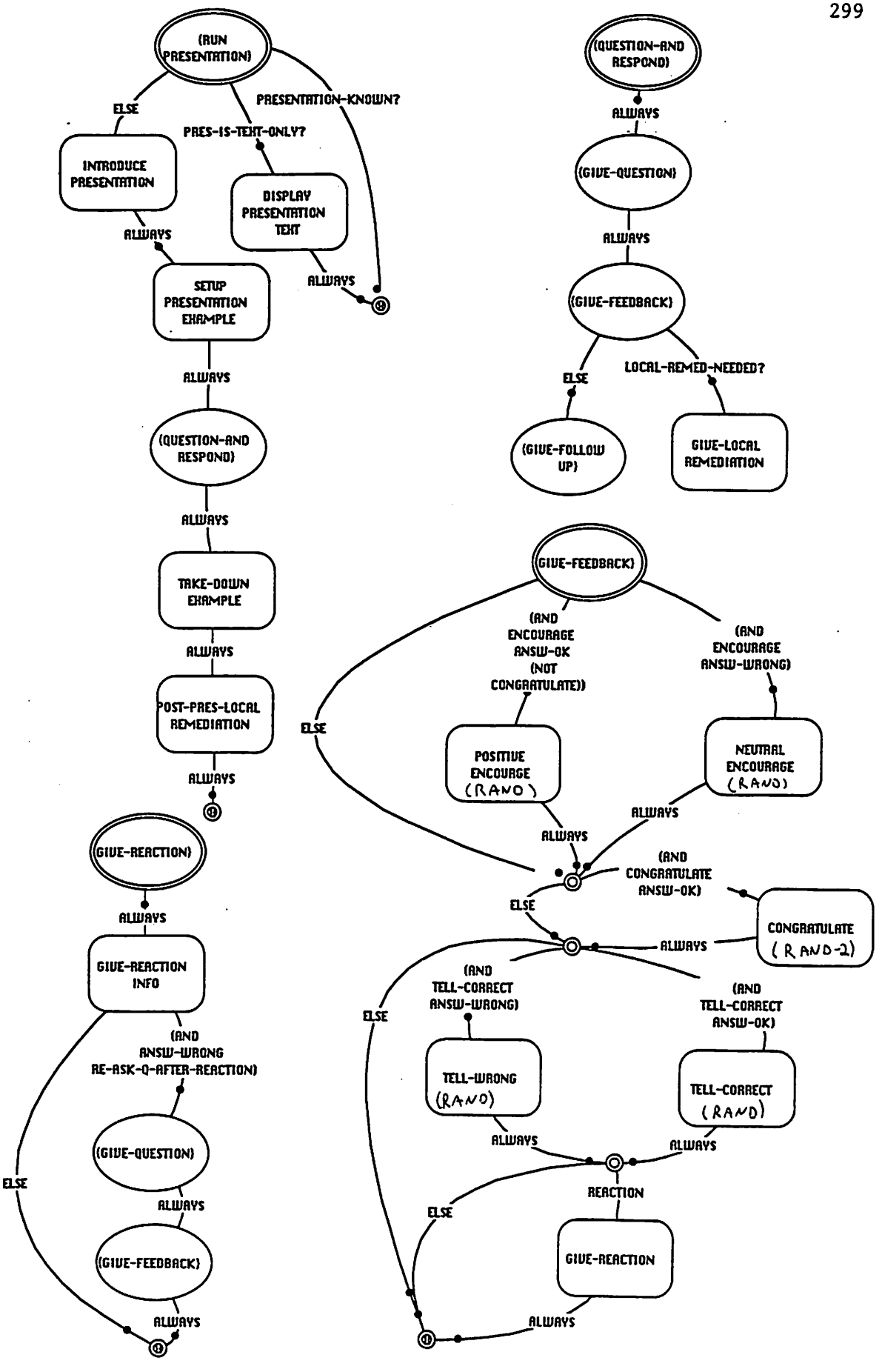
The default PANs for the topic and response levels are more elaborate, and their PANs are shown below. Several switch sets (not shown) were defined for these PANs.

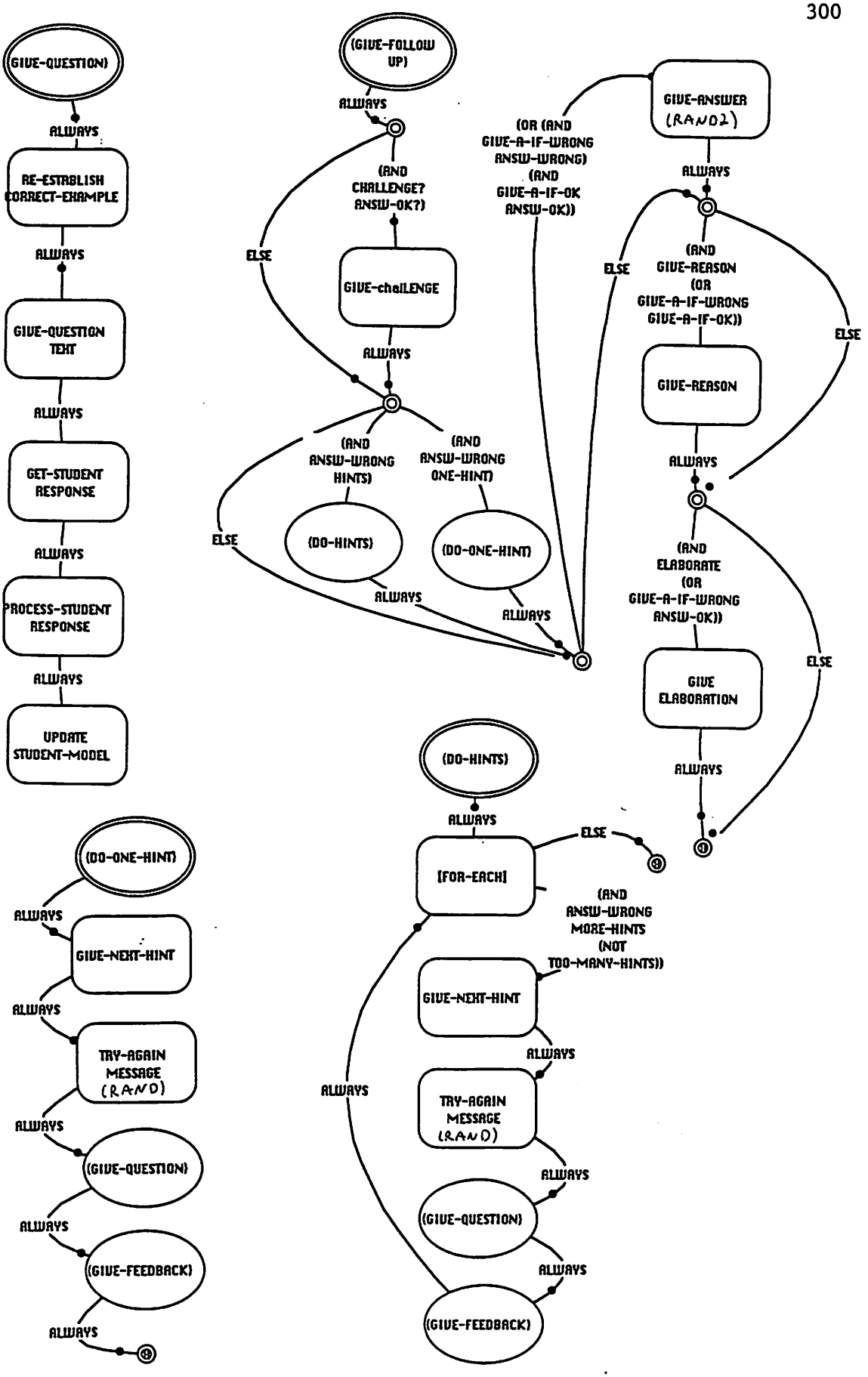
The first two pages show the five PANs for the default topic strategy. The next two pages show the eight PANs for the default response strategy. The last page shows the presentation level PAN for the FBD-identify-forces topic.

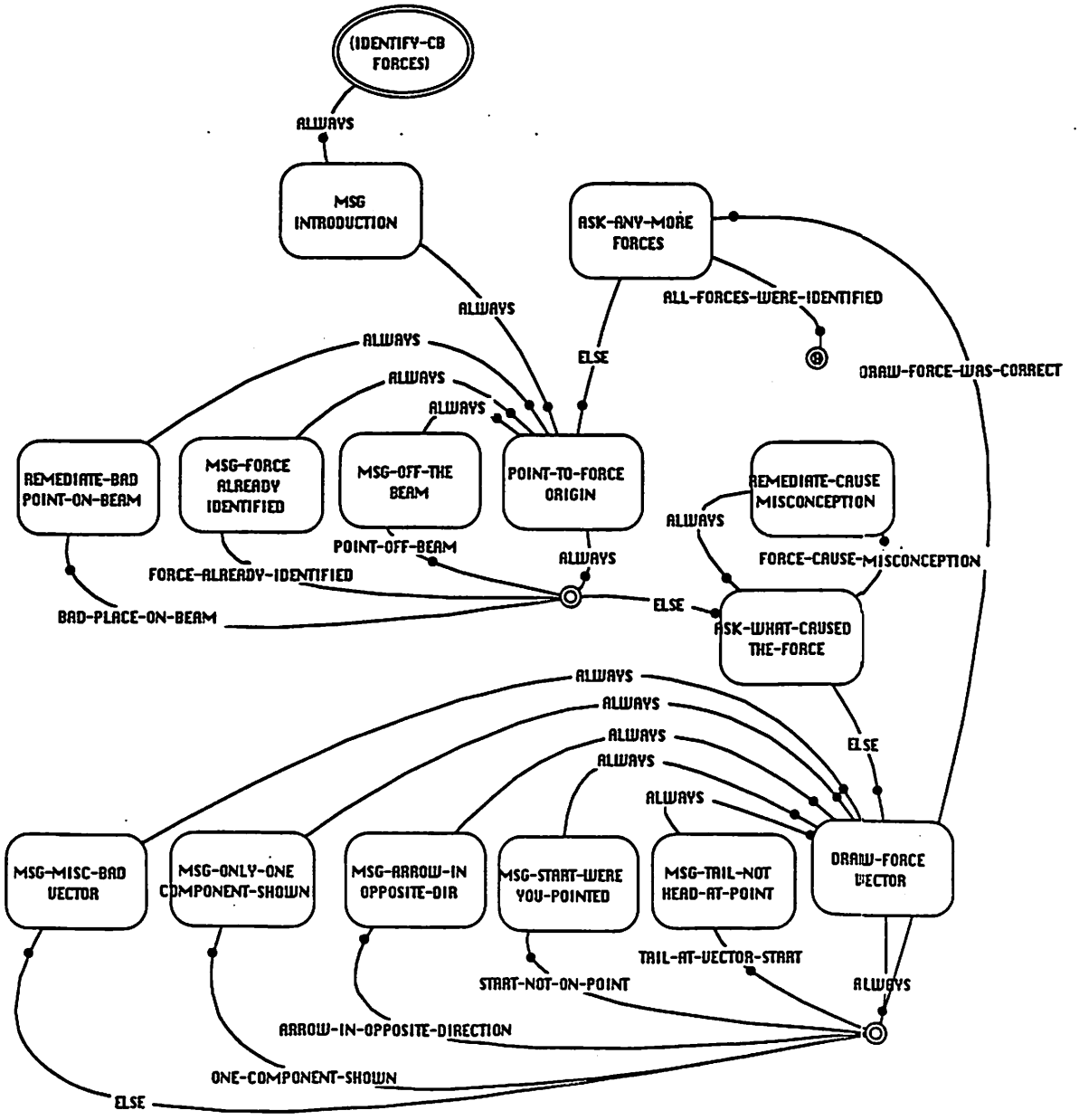
"(Rand)" under the name of a node indicates that the node action produces a random text item from a list of similar items. "(Rand2)" indicates that random text is given if there is no text specified in the knowledge base.


(NORMAL-TEACH TOPIC-LEVEL X)
TOPIC-KNOWN?
ELSE
[FOR-EACH]
ALWAYS
MORE-CRITICAL-MISCONCEPTIONS?
(REMEDIATE-MIS KU)
ELSE
(TEACH-LEVEL X)
ALWAYS
[FOR-EACH]*2
ALWAYS
MORE-PENDING-SUSPECTED-MIS-KUS?
(REMEDIATE-MIS KU)*2
ELSE
(TEACH-LEVEL X)
PREREQUISITES-BEFORE-TEACH?
(TEACH PREREQUISITES)
ELSE
ALWAYS
PREREQUISITES-BEFORE-TEACH?
(TEACH-PARTS LEVEL-X)
ELSE
ALWAYS
MOTIVATE?
RUN-MOTIVATION
ELSE
ALWAYS
DEFINE-BEFORE-TEACH?
RUN-DEFINITION
ELSE
ALWAYS
RUN-TEACHING-PRESENTATIONS?
RUN-EASY PRESENTATIONS
ALWAYS
(AND RUN-TEACHING-PREESNTATIONS? (OR TEACHING-TYPICAL TEACHING-DIFFICULT))
ELSE
RUN-TYPICAL PRESENTATIONS
ALWAYS
(AND RUN-TEACHING-PRESENTATIONS? TEACHING-DIFFICULT)
ELSE
RUN-DIFFICULT PRESENTATIONS
ALWAYS
ALWAYS
RUN-SUMMARY
ELSE
WRAP-UP?
RUN-WRAP-UP
ALWAYS


(TEACH PREREQUISITES)
ALWAYS
[FOR-EACH]
ALWAYS
MORE-FAMILIAR-PREREQUISITES?
ELSE
GIVE-TOPIC-SUMMARY?
ELSE
[FOR-EACH]
MORE-DEEP-PREREQUISITES?
ELSE
ALWAYS
[FOR-EACH]*2
MORE-PARTS?
RUN-TOPIC SUMMARY
[FOR-EACH]*3
MORE-EASY-PREREQUISITES?
ELSE
(TEACH-LEVEL H)
ALWAYS
[FOR-EACH]*4
ELSE
MORE-TYPICAL-PREREQUISITS?
ALWAYS
(TEACH-LEVEL H)*2
(TEACH-PARTS LEVEL-H)
ALWAYS
[FOR-EACH]
ALWAYS
MORE-CONCEPT-PARTS?
(TEACH-LEVEL H)
ELSE
[FOR-EACH]*2
ELSE
ALWAYS
MORE-OTHER-PARTS?
(TEACH-LEVEL H)*2


(RUN PRESENTATION)
ELSE
PRESENTATION-KNOWN?
PRES-IS-TEXT-ONLY?
INTRODUCE PRESENTATION
DISPLAY PRESENTATION TEXT
ALWAYS
ALWAYS
SETUP PRESENTATION EXAMPLE
ALWAYS
(QUESTION-AND RESPOND)
ALWAYS
TAKE-DOWN EXAMPLE
ALWAYS
POST-PRES-LOCAL REMEDIATION
ALWAYS
GIVE-REACTION)
ALWAYS
GIVE-REACTION INFO
(AND ANSW-WRONG RE-ASK-Q-AFTER-REACTION)
ELSE
(GIVE-QUESTION)
ALWAYS
(GIVE-FEEDBACK)
ALWAYS
(QUESTION-AND RESPOND)
ALWAYS
(GIVE-QUESTION)
ALWAYS
(GIVE-FEEDBACK)
ELSE
LOCAL-REMED-NEEDED?
(GIVE-FOLLOW UP)
GIVE-LOCAL REMEDIATION
GIVE-FEEDBACK)
(AND ENCOURAGE ANSW-OK (NOT CONGRATULATE))
(AND ENCOURAGE ANSW-WRONG)
ELSE
POSITIVE ENCOURAGE (RAND)
NEUTRAL ENCOURAGE (RAND)
ALWAYS
ALWAYS
(AND CONGRATULATE ANSW-OK)
ELSE
CONGRATULATE (RAND-2)
ALWAYS
(AND TELL-CORRECT ANSW-WRONG)
(AND TELL-CORRECT ANSW-OK)
ELSE
TELL-WRONG (RAND)
TELL-CORRECT (RAND)
ALWAYS
ALWAYS
REACTION
ELSE
GIVE-REACTION
ALWAYS


(GIVE-QUESTION)
ALWAYS
RE-ESTABLISH CORRECT-EXAMPLE
ALWAYS
GIVE-QUESTION TEXT
ALWAYS
GET-STUDENT RESPONSE
ALWAYS
PROCESS-STUDENT RESPONSE
ALWAYS
UPDATE STUDENT-MODEL
(GIVE-FOLLOW UP)
ALWAYS
(AND CHALLENGE? ANSW-OK?)
ELSE
GIVE-CHALLENGE
ALWAYS
(AND ANSW-WRONG HINTS)
(AND ANSW-WRONG ONE-HINT)
ELSE
(DO-HINTS)
(DO-ONE-HINT)
ALWAYS
ALWAYS
(OR (AND GIVE-A-IF-WRONG ANSW-WRONG) (AND GIVE-A-IF-OK ANSW-OK))
GIVE-ANSWER (RAND2)
ALWAYS
ELSE
(AND GIVE-REASON (OR GIVE-A-IF-WRONG GIVE-A-IF-OK))
ELSE
GIVE-REASON
ALWAYS
(AND ELABORATE (OR GIVE-A-IF-WRONG ANSW-OK))
ELSE
GIVE ELABORATION
ALWAYS
(DO-ONE-HINT)
ALWAYS
GIVE-NEXT-HINT
ALWAYS
TRY-AGAIN MESSAGE (RAND)
ALWAYS
(GIVE-QUESTION)
ALWAYS
(GIVE-FEEDBACK)
ALWAYS
(DO-HINTS)
ALWAYS
[FOR-EACH]
ELSE
(AND ANSW-WRONG MORE-HINTS (NOT TOO-MANY-HINTS))
GIVE-NEXT-HINT
ALWAYS
TRY-AGAIN MESSAGE (RAND)
ALWAYS
(GIVE-QUESTION)
ALWAYS
(GIVE-FEEDBACK)
ALWAYS

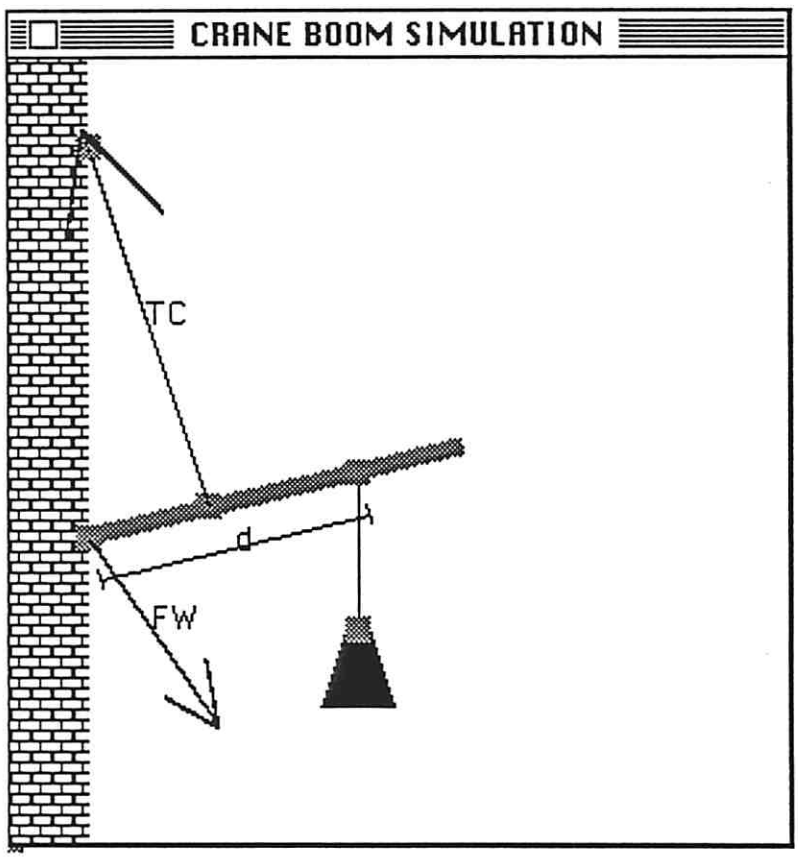

(IDENTIFY-CB FORCES)
ALWAYS
MSG INTRODUCTION
ALWAYS
ASK-ANY-MORE FORCES
ALL-FORCES-WERE-IDENTIFIED
DRAW-FORCE-WAS-CORRECT
ELSE
ALWAYS
ALWAYS
ALWAYS
REMEDIATE-BAD POINT-ON-BEAM
MSG-FORCE ALREADY IDENTIFIED
MSG-OFF-THE BEAM
POINT-TO-FORCE ORIGIN
REMEDIATE-CAUSE MISCONCEPTION
ALWAYS
POINT-OFF-BEAM
ALWAYS
FORCE-CAUSE-MISCONCEPTION
FORCE-ALREADY-IDENTIFIED
ELSE
BAD-PLACE-ON-BEAM
ASK-WHAT-CAUSED THE-FORCE
ALWAYS
ALWAYS
ALWAYS
ALWAYS
ALWAYS
ELSE
MSG-MISC-BAD VECTOR
MSG-ONLY-ONE COMPONENT-SHOWN
MSG-ARROW-IN OPPOSITE-DIR
MSG-START-WERE YOU-POINTED
MSG-TAIL-NOT HEAD-AT-POINT
DRAW-FORCE VECTOR
TAIL-AT-VECTOR-START
START-NOT-ON-POINT
ALWAYS
ARROW-IN-OPPOSITE-DIRECTION
ONE-COMPONENT-SHOWN
ELSE

APPENDIX F

Crane Boom Simulation Details

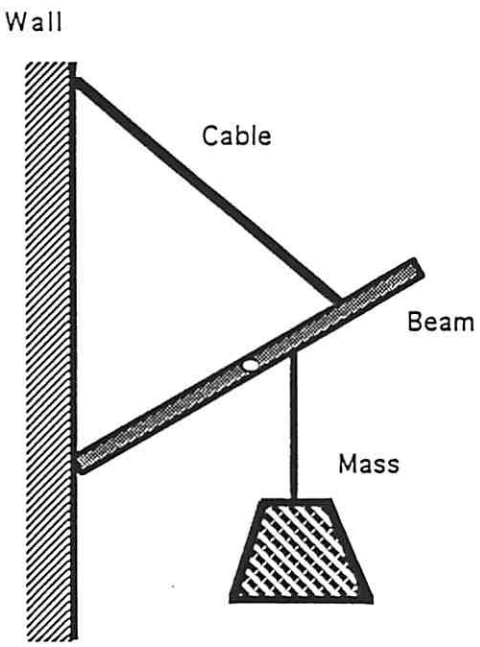
We will not describe the functionality of the crane boom simulation in detail, but below are diagrams that illustrate some of its features and parameters.

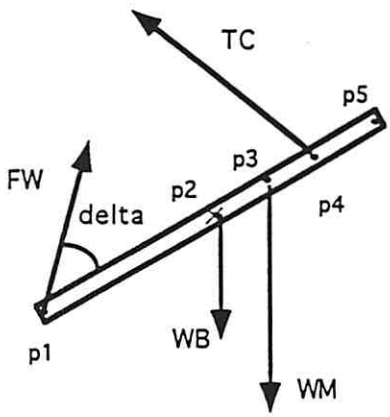
Crane Boom Simulation

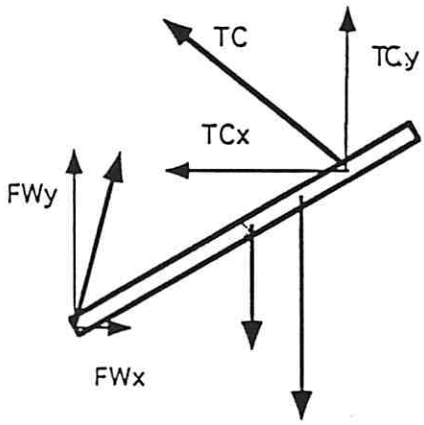


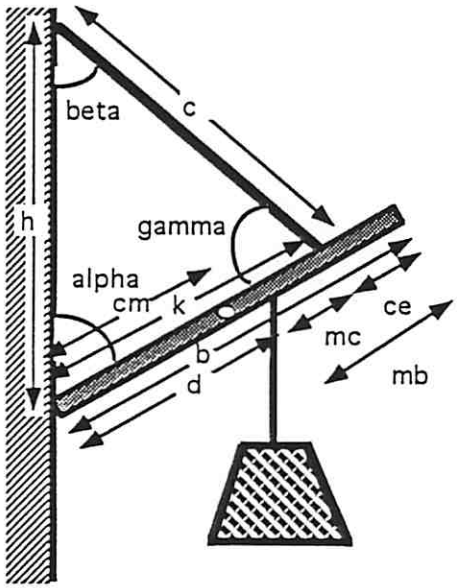
Meters
TC= 51.44 N
FW= 28.84 N
d= 7.13 m

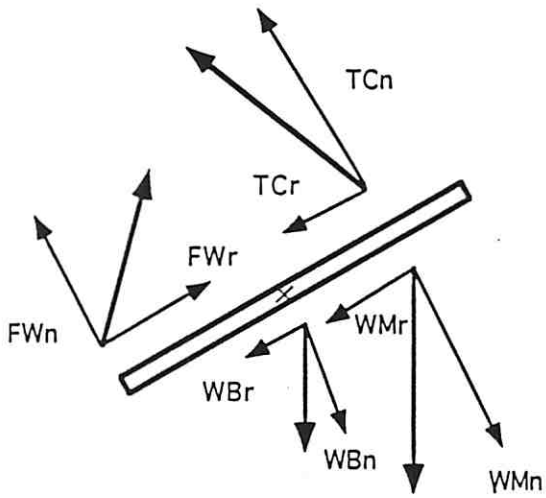
Crane Boom and Forces











APPENDIX G

Sample Tutorial Dialog

Below is an annotated sample of a typical tutorial dialog to give the reader a sense of what it is like to run the statics tutor. Note that the inclusion of the various elements of the knowledge base (elaborations, hints, etc.) depends on the current strategy.

SAMPLE TUTORIAL DIALOG

[TOPIC: {LINEAR-EQUILIBRIUM-INTUITION}]

[MOTIVATE-TOPIC] You are going to be given some crane boom situations and then asked what will happen when one variable in the simulation is modified.

[DEFINE-TOPIC] Given an object in equilibrium, changing one of the forces must result in a change in one or more of the other forces, if the total system is to remain in equilibrium (i.e. not move).

[START-QUESTION: {LE-INTUITION-EASY1}]

[A CRANE BOOM DIAGRAM IS GIVEN]

How would you expect the tension force in the cable to change if the weight were moved to the left?

:::Choose:
Tension force increases.
Tension force decreases.
Tension force remains the same.
:::Chosen:Tension force decreases.
[Response-ok? CORRECT]

[CHALLENGE] Please use the simulation now to move the weight left and right to see that it confirms your prediction.

[SIMULATION COMES UP. STUDENT PLAYS WITH IT, AND EXITS IT.]

[GIVE-AWAY] When you moved the weight to the left you may have noticed that the force of the wall holding up the boom is greater and the cable tension is reduced.

[REASON] If you think of the weight moved far to the left, it should seem that the wall is providing most of the support. When the weight is on the right side, the cable must provide most of the support.

[ELABORATION] Think about the extreme case when the weight is moved almost all the way to the wall. At this location the weight should put very little load on the cable.

[.....]

[START-QUESTION: {LE-INTUITION-TYPICAL1}]

[A CRANE BOOM DIAGRAM IS GIVEN]

If the cable were shortened, raising the end of the
boom, how would the tension force in the cable change?

:::Choose:
Tension force increases.
Tension force decreases.
Tension force stays the same.
:::Chosen:Tension force decreases.
Response-ok? CORRECT

[CONGRATULATION] Good job!

[REASON] Notice, as the end of the boom swings up, how the wall
must support more and more of the weight.

[ELABORATION] Think about the extreme case when the boom is almost
vertical. Then you can probably see that the wall is
holding up the whole boom.

[.....]

[SUMMARIZE-TOPIC] Since one may always think of the crane boom as being in
equilibrium, one should try to see that an increased
force in one location should always be compensated by
increased balancing force(s) elsewhere. The opposite
for decreasing forces should be true.

[WRAP-UP-TOPIC] When working on this type of problem you are urged to
draw a picture showing all the forces on the beam.
Then you will hopefully be able to reason how a change in
one force will effect the other force(s) acting on the beam.

[.....]

[START-QUESTION: {LE-PRINCIP-EASY1}]

[A PICTURE IS GIVEN]

A person is pushing a refrigerator straight across a
warehouse floor at a steady speed of two miles/hour.

How could one possibly think of the refrigerator being in equilibrium?

:::Answer Choices:
:::A. The refrigerator is not in equilibrium because the worker is much stronger than any other force in the problem.
:::B. The refrigerator is in equilibrium because the refrigerator is pushing back on the worker exactly as hard at the worker is pushing on the refrigerator.
:::C. The refrigerator is in equilibrium because the friction force from the floor is opposite and equal to the force of the worker.
:::D. The refrigerator is in equilibrium because the force of the worker is balanced by both the force of inertia and the force of friction combined.
:::E. The force of inertia is balanced by the force of gravity that pulls the object down.
:::Chosen: B.
Response-ok? NO

[ENCOURAGEMENT] I think I understand your answer.

[REACTION] Be careful. The two forces named are equal but only one of these forces acts on the refrigerator.

[GIVE-HINT] You need to find an explanation as to how all the forces acting on the refrigerator could be balanced.

Please try again...

[.....]

[...Response-ok? YES]

[CONGRATULATION] That's good!

[REASON] Since friction is caused by tiny bumps on the floor it is hard to imagine that the friction force could balance the force of a strong worker. It may help to think of many of these little bumps working together.

[.....]

APPENDIX H

Sample Object Instances

Example instance of one of each type of object in the system are shown below to document the slots associated with the object types.[1]

[1]Not all objects of a given type have the same slots, because the mixin feature allows other slots to be added. The instances shown below are for the default mixins for each object type, as used in ninety five percent of all objects in the knowledge base.

SAMPLE INSTANCES

```
;; Sample LESSON

(NEW-LESSON :NAME {LINEAR-EQUILIBRIUM-LESSON}
 :NOTES  ("This lesson was used in the June student tests.")
 :REFERENCED-BY  ()
 :INTRODUCTION  "This lesson will focus on a qualitative understanding
    of Linear Equilibrium for  static objects.  "
 :LESSON-SEQUENCE  ( (SET-TOPIC-STRATEGY 'GENERAL-TEACH)
                          (SET-RESPONSE-STRATEGY 'VERBOSE)
                          {LINEAR-EQUILIBRIUM})
 :CONCLUSION  "This concludes the LINEAR-EQUILIBRIUM lesson--Have a nice day!"
 )

;; SAMPLE TOPIC

(NEW-TOPIC :NAME {LINEAR-EQUILIBRIUM-INTUITION}
 :NOTES  ("should have familiarity with newtons laws?-tm" "i need
    to re-check the motivation later.-cc")
 :REFERENCED-BY  ({FBD-SOLUTION-ANALYSIS} {LINEAR-EQUILIBRIUM})
 :TOPIC-TYPE  COMPLEX
 :DEFINITION  "Given an object in equilibrium, changing one
   of the forces must result in a change in  one or more of
   the other forces, if the  total system is to  remain in equilibrium
   (i.e. not move)."
 :PREREQUISITES  (:FAMILIAR ({LINEAR-EQUILIBRIUM-PRINCIPLE})
           :DEEP-FAMILIAR ({?}) :EASY ({?}) :TYPICAL ({?}) :DIFFICULT ({?}))
 :PARTS  (:CONCEPTS ({?}) :OTHER ({?}))
 :MOTIVATION  "You are going to be given some crane boom
   situations and then asked what will happen
   when one variable in the simulation is modified."
 :SUMMARY  "Since one may always think of the crane boom as being in
   equilibrium, one should try to see that an increased force in one
   location should always be compensated by increased balancing force(s)
   elsewhere.  The opposite for decreasing forces should be true."
 :WRAP-UP  "When working on this type of problem you are urged to draw
    a picture showing all the forces on the beam.  Then you will
    hopefully be able to reason how a change in one force will effect
    the other force(s) acting on the beam."
 :META-KNOWLEDGE  ()
 :REMEMBER  ()
 :USE-EASY  ({LE-INTUITION-EASY1} {LE-INTUITION-EASY2})
 :USE-TYPICAL  ({LE-INTUITION-TYPICAL1} {LE-INTUITION-TYPICAL2})
 :USE-DIFFICULT  ()
```

```
 :DIAGNOSTIC-QUESTIONS  ()
 :CRITICAL-MISCONCEPTIONS  ()
 :LOCAL-TOPIC-STRATEGY ()
 )

;; Sample PRESENTATION

(NEW-PRESENTATION :NAME {CENTER-OF-MASS-EASY1}
 :MIXINS  ({EXAMPLE-MIXIN} {QUESTION-MIXIN})
 :NOTES  ()
 :REFERENCED-BY  ({CENTER-OF-MASS})
 :SET-UP  (SHOW-PICTURE {TWO-BLOCKS-ON-BEAM})
 :TAKE-DOWN  ()
 :ANSWER-TYPE  :MULTIPLE-CHOICE
 :INTRO-TEXT  "The 2 kg block and the 5 kg block are on a massless beam."
 :QUESTION-TEXT  "Where would the center of mass of the system be located?"
 :ANSWER-DESCRIPTIONS  ("Nearest to point A."
                        "Nearest to point B."
                        "Nearest to point C."
                        "Nearest to point D."
                        "Nearest to point E.")
 :REACTIONS  (5 "You have the right idea but consider the fact that
   the boom is massless.")
 :REMEDIATION-INFO  ()
 :CORRECT-ANSWERS  (3)
 :CONGRATULATE  ()
 :CHALLENGE  ()
 :HINTS  ("Think of where you could put a support under the beam and
   have it balance.")
 :GIVE-AWAY  ("The center of mass must be nearer the 5 kg than the
    2 kg mass.")
 :REASON  ("If you placed a 7 kg mass at the center of mass it would
   balance the same as the two  separate masses.")
 :ELABORATE  ("Actually the distance from the 2 kg mass to the center
   of mass is 2.5 times as far as the distance from the 5 kg to the
    center of mass.  This is the inverse of the mass ratio.")
 :LOCAL-RESPONSE-CONTROL  ()
 :RESPONSE-STRATEGY  NORMAL-RUN-PRESENTATION
 :CB-INTERVENTION-FUNCTION  NUL-INTERVENTION-FUNCTION
 )
```

[Note: The presentation above includes mixins for example and question object slots. Example objects and question objects (rarely used in the statics domain) have a subset of the slots shown for the presentation above.]

```
;;  Sample PICTURE

(NEW-PICTURE :NAME {TWO-BLOCKS-ON-BEAM}
 :MIXINS  ()
 :NOTES  ()
 :REFERENCED-BY  ({CENTER-OF-MASS-EASY1})
 :RESOURCE-ID  7
 :DESCRIPTION  "A massless beam with a 2 kg block on one
                end and a 5 kg block on the other."
 :RESOURCE-FILE  "statics-7.pic"
 :DRAWING-FILE  "statics-pictures"
 )

;; Sample CRANE BOOM

(NEW-CRANE-BOOM :NAME {SUM-Y-FORCES-2}
 :NOTES  ("add in the wall force and display in free body
          mode in right screen with answer reason?")
 :REFERENCED-BY  ({PROCED-SUM-Y-FORCES-TYPICAL2-EX})
 :TYPE  LEFT-DIAGRAM
 :MODE  PHYSICAL
 :CABLE-HEIGHT-H  INFINITE
 :ALPHA  90
 :BEAM-LENGTH-B  10.0
 :BEAM-CTR-MASS-C  5.0
 :WEIGHT-POS-D  10.0
 :CABLE-POS-K  6
 :MASS-WEIGHT-WM  10.0
 :BEAM-WEIGHT-WB  4
 :ANGLES-SHOWN  ()
 :VECTORS-SHOWN  (TC WB W)
 :LENGTHS-SHOWN  ()
 :METERS-SHOWN  (TC WB WM)
 :VECTOR-SCALE  :AUTO
 :PICTURE-SCALE  :AUTO
 :SHOW-WHEN-CREATED?  T
 :LEAVE-SHOWN-WHEN-DONE?  ()
 )

;; Sample MIS-KU

(NEW-MIS-KU :NAME {COMPRESSION-ON-VS-BY-CONFUSION}
 :MIXINS  ()
 :NOTES  ()
 :REFERENCED-BY  ({COMPRESSIBLE-BODY-FORCES} {FORCE-ON-VS-BY-CONFUSION})
 :DIAGNOSTIC-SCRIPT  ({COMPRESSION-ON-VS-BY-DIAG1} {COMPRESSION-ON-VS-BY-DIAG2})
```

```
 :REMEDIATION-SCRIPT  ({COMPRESSION-MIS-KU-REMED1} {COMPRESSION-MIS-KU-REMED2})
 )

;; Sample SOUND

(NEW-SOUND :NAME {CATS-MEOW}
 :MIXINS  ()
 :NOTES  ()
 :REFERENCED-BY  ()
 :RESOURCE-ID  25903
 :DESCRIPTION  "cat meowing"
 :RESOURCE-FILE  "tev-sounds"
 :ORIGINAL-SOUND-FILE  ""
 )

;; Sample STORAGE object

(NEW-STORAGE :NAME {RANDOM-TEXT}
 :MIXINS  ()
 :NOTES  ("This instance is not meant to be run.
           It only stores values and text to be used elsewhere.")
 :REFERENCED-BY  ()
 :TRY-AGAIN  ("Please try again..." "Try it one more time..."
              "Try it again...")
 :GIVE-AWAY-INTRO  ("The correct answer is: " "The answer is: "
                    "This is the right answer: ")
 :TELL-WRONG  ("That's not quite right." "Your answer is incorrect."
               "Sorry, but that is wrong.")
 :TELL-OK  ("That's right." "That's exactly right!"
            "Your answer is correct.")
 :ENCOURAGE-WRONG  ("I think I understand your answer." "Well...")
 :ENCOURAGE-OK  ("OK," "fine,")
 :CONGRATULATE  ("Very good!" "That's good!" "Good job!")
 :HELLO  ("hi" "hello" "hi there" "welcome")
 )
```

APPENDIX I

Sample Saved-instances File

Portions of a saved-instances-file are shown below, including a table of contents, instances, and cross references.

SAVED INSTANCE FILE

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Tutor Instance SAVE file for tutor edit session.
;;;  Time: 13:49:31.  Date: 12/12/1989.
;;;  File: "Tutor-data:Tupits-saved-instances:tutor-inst-7.lisp"
;;;  Operator: tom
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;========================================================================
;                          TABLE OF CONTENTS
;========================================================================
#|

TOPICS:
------



{ALL-FORCES-ARE-ELASTIC}................ ______
{CENTER-OF-MASS}........................ ______
{COMPRESSIBLE-BODY}..................... ______
{CONTACT-FORCES}........................ ______
{DUM-TOPIC}............................. ______
{FBD-CONCEPT}........................... ______
{FBD-IDENTIFY-FORCES}................... ______
{FBD-IDENTIFY-MAIN-BODY}................ ______
{FBD-SOLUTION-ANALYSIS}................. ______
{FBD-SOLVE-EQUATIONS}................... ______

[......]

{Y-AXIS-FORCES}......................... ______


MISC. PRESENTATIONS:.................... ______

MISC. OTHER INSTANCES:.................. ______

CROSS REFERENCES:....................... ______

;========================================================================
;                    END     TABLE OF CONTENTS
```

```
;================================================================================
|#

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  LESSON: {LINEAR-EQUILIBRIUM-LESSON}
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(NEW-LESSON :NAME {LINEAR-EQUILIBRIUM-LESSON}
 :NOTES  ()
 :REFERENCED-BY  ()
 :INTRODUCTION  "This lesson will focus on a qualitative understanding
    of Linear Equilibrium for  static objects.  "
 :LESSON-SEQUENCE  ((SET-TOPIC-STRATEGY 'GENERAL-TEACH)
                    (SET-RESPONSE-STRATEGY 'VERBOSE)
                    {LINEAR-EQUILIBRIUM})
 :CONCLUSION  "This concludes the LINEAR-EQUILIBRIUM lesson--Have a nice day!"
 )


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  TOPIC: {FORCES-AT-A-DISTANCE}
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(NEW-TOPIC :NAME {FORCES-AT-A-DISTANCE}
 :MIXINS  ()
 :NOTES  ("This topic tries to make clear the difference between
   forces that act at a distance as compared to contact forces."
   "For Use Easy and Use Typical, see the notes on the original sheet.
   Synthesizer link  to 3rd-law-existance.")
 :REFERENCED-BY  ({TYPES-OF-FORCES})
 :TOPIC-TYPE  CONCEPT
 :PREREQUISITES  (:FAMILIAR ({?}) :DEEP-FAMILIAR ({?}) :EASY ({?})
                  :TYPICAL ({?}) :DIFFICULT ({?}))
 :PARTS  (:CONCEPTS ({GRAVITY} {OTHER-FORCES}) :OTHER ({?}))
 :SUMMARY  "In order for you to be sure that a particular force can act
    at a distance, you should be able to remember an example where you
    have seen this kind of force reach out and attract or repel an object
    through empty space."
 :MOTIVATION  "A most amazing type of force is the kind that can reach out
    through empty space and influence (push or pull) another body."
 :DEFINITION  "When a body can reach out through space with an apparently
```

```
    invisible influence and push or pull another object, we say this type
    of force is a force that acts through a distance and does not require
    contact of the two bodies."
 :USE-EASY  ({FORCE-AT-A-DISTANCE-EASY1})
 :USE-TYPICAL  ({FORCE-AT-A-DISTANCE-TYPICAL1})
 :USE-DIFFICULT  ()
 :WRAP-UP  "In our every day lives gravity would seem to be the strongest
    force that can act at a distance."
 :TEACH-STRATEGY  NORMAL-TEACH-TOPIC
 :LOCAL-TEACH-CONTROL  ()
 :CRITICAL-MISCONCEPTIONS  ()
 )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(NEW-PRESENTATION :NAME {FORCE-AT-A-DISTANCE-EASY1}
 :MIXINS  ({EXAMPLE-MIXIN} {QUESTION-MIXIN})
 :NOTES  ("This is a counter example.  Tries to deal with the impetus misconception.")
 :REFERENCED-BY  ({FORCES-AT-A-DISTANCE})
 :INTRO-TEXT  ""
 :LOCAL-RESPONSE-CONTROL  ()
 :RESPONSE-STRATEGY  NORMAL-RUN-PRESENTATION
 :SET-UP  (SHOW-PICTURE {BAT-HITS-BALL})
 :TAKE-DOWN  ()
 :ANSWER-TYPE  :ABC-CHOICE
 :QUESTION-TEXT  "When the ball hits the bat and flies away, what
type(s) of forces are involved?"
 :ANSWER-DESCRIPTIONS  ("Only contact force (when the ball is touching
   the bat)."
   "Both contact forces (when the ball touches the bat) and the force at
    a distance (that keeps it going when it has left the bat)."
   "Only forces that act at a distance really matter in this problem.")
 :REACTIONS  ()
 :REMEDIATION-INFO  ()
 :CORRECT-ANSWERS  (1)
 :CONGRATULATE  ()
 :CHALLENGE  ()
 :HINTS  ()
 :GIVE-AWAY  "It is tempting to think of the bat as still pushing on
   the ball after it has left contact with the bat.
   However, you may find it helpful to think of the ball as
   coasting after it stops touching the bat."
 :REASON  "The contact force between the bat and the ball only
   influences the ball while they are touching.  There are
   no forces that act at a distance between the bat and the ball."
```

```
 :ELABORATE  ()
 :CB-INTERVENTION-FUNCTION  NUL-INTERVENTION-FUNCTION
 )

(NEW-PICTURE :NAME {BAT-HITS-BALL}
 :MIXINS  ()
 :NOTES  ()
 :REFERENCED-BY  ({FORCE-AT-A-DISTANCE-EASY1})
 :RESOURCE-ID  18406
 :DESCRIPTION  "[no picture description available]"
 :RESOURCE-FILE  "statics-2.pic"
 :DRAWING-FILE  "CRANE-BOOM-DWGS.1"
 )

[......]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Misc. Presentations:
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

[......]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Misc. other instances::
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

[......]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Cross references for PRESENTATIONS:
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


{FORCE-AT-A-DISTANCE-EASY1} cross references:
  Appears in slot USE-EASY of instance {FORCES-AT-A-DISTANCE}.

{FORCE-AT-A-DISTANCE-TYPICAL1} cross references:
  Appears in slot USE-TYPICAL of instance {FORCES-AT-A-DISTANCE}.

{GRAVITY-FORCE-EASY1} cross references:
```

```
  Appears in slot USE-EASY of instance {GRAVITY}.

[......]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Non-existing instances found in Cross References:
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 Instance {DUM-MIS-KU} refers to nonexisting: {DUM-KU}.
 Instance {GRAVITY} refers to nonexisting: {EARTH-ROTATION}.
 Instance {GRAVITY} refers to nonexisting: {AIR-PRESSURE}.

|#
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  End of Cross References
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;**************************************************************************
;;41 Topics.

;;3 Mis-KUs

;;68 Presentations.

;;250 Total instances.

;;**************************************************************************
```

APPENDIX J

Sample Lesson Listing

The lesson listing file has one entry for each tutorial session started. A sample of this file is given below.

File: Session-listings.txt

```
Demo lesson tutoring session. Lesson: {LINEAR-EQUILIBRIUM-LESSON}.
   Trace file: Breakfast:tutor:tutor-data:tup-traces:trace-235.out.
   Student: Steve Brown. Date: 2/23/1990 Time: 16:35:8


Demo lesson tutoring session. Lesson: {LINEAR-EQUILIBRIUM-LESSON}.
   Trace file: Breakfast:tutor:tutor-data:tup-traces:trace-237.out.
   Student: Gint Flarb. Date: 2/27/1990 Time: 15:5:26


Demo lesson tutoring session. Lesson: {LINEAR-EQUILIBRIUM-LESSON}.
   Trace file: Breakfast:tutor:tutor-data:tup-traces:trace-244.out.
   Student: Trent Todlis. Date: 3/15/1990 Time: 14:14:22


Demo lesson tutoring session. Lesson: {LINEAR-EQUILIBRIUM-LESSON}.
   Trace file: Breakfast:tutor:tutor-data:tup-traces:trace-245.out.
   Student: Barb. Date: 3/15/1990 Time: 16:5:59


[......]
```

APPENDIX K

Sample Edit Record

A portion of an edit record containting changes made by the domain expert is shown. Note the comments entered by the domain expert for the knowledge engineer.

SAMPLE EDIT RECORD

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; New Record file for tutor edit session.
;;;  Time: 12:7:33.  Date: 8/7/1990.
;;;  File: :Tutor-data:Tupits-edit-records:DB7/24/90
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Starting tutor edit session.
;;;  Time: 12:9:36.  Date: 8/7/1990.
;;;  Operator: cc
;;;  Session record file: :tutor-data:tup-traces:trace-290.out
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Recording:

(RL:ASSERT-VAL {LE-CONCEPT-EASY1}
               'QUESTION-TEXT
               '"The person shown weighs 450 Newtons.  If the earth is
pulling down on this  person with  a 450 N force, what
other force is acting to keep this body in equilibrium?")

#|COMMENT:
Tom -- Notice the question scrolls off the screen when all 4 ans
choices are displayed. cc
|#



(RL:ASSERT-VAL {LE-CONCEPT-TYPICAL2}
               'ANSWER-DESCRIPTIONS
               '("2 Newtons Southwest" "1 Newton South and 1 Newton West"
                                       "1 Newton Southwest"
                                       "1.41 Newtons Southwest"))



;;Recording:

(RL:ASSERT-VAL {X-AXIS-FORCES}
               'MOTIVATION
               '"Most crane boom problems involve both X and Y forces.
While this section involves practicing with only
horizontal forces, you are reminded that you can ignore
any Y-forces present.")
```

```
#|COMMENT:
Tom,  Why isn't this starting with  Y-Forces ??  CC
|#

;;Recording:

(NEW-CRANE-BOOM :NAME {CB-TWO-FORCES}
                :MIXINS NIL
                :NOTES ("Massless boom with mass hanging from center
                        and vertical cable in center.")
                :REFERENCED-BY NIL
                :TYPE LEFT-DIAGRAM
                :MODE PHYSICAL
                :CABLE-HEIGHT-H 9.0
                :ALPHA 45.0
                :BEAM-LENGHT-B 10.0
                :BEAM-CTR-MASS-C 5.0
                :WEIGHT-POS-D 5.0
                :CABLE-POS-K 5.0
                :BEAM-HEIGHT 7.5
                :MASS-WEIGHT-WM 10.0
                :BEAM-WEIGHT-WB 0
                :ANGLES-SHOWN NIL
                :VECTORS-SHOWN NIL
                :LENGTHS-SHOWN NIL
                :METERS-SHOWN NIL
                :VECTOR-SCALE :AUTO
                :PICTURE-SCALE :AUTO
                :SHOW-METERS? T
                :SHOW-WHEN-CREATED? T
                :LEAVE-SHOWN-WHEN-DONE? NIL)

[......]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; ENDING tutor edit session.
;;; Time: 13:8:22.  Date: 8/7/1990.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

APPENDIX L

Sample Trace File

Trace files, like the one shown below, record the transactions of tutorial sessions. Note the comment entered by the student near the end.

SAMPLE TRACE FILE

```
New trace file name: :tutor-data:tup-traces:trace-159.out

Starting Crane Boom tutoring session.

Student Model Reset.

********************

******Time stamp: 13:13:10.

START-LESSON: {LINEAR-EQUILIBRIUM-LESSON}
  :::
LESSON-INTRO:
  :::This lesson will focus on a qualitative understanding of
Linear Equilibrium   for  static objects.


NEW-TOPIC-STRATEGY: GENERAL-TEACH
NEW-RESPONSE-STRATEGY: VERBOSE
START-TOPIC: {LINEAR-EQUILIBRIUM}
START-PREREQUISITES-OF: {LINEAR-EQUILIBRIUM}
FINISH-PREREQ-OF: {LINEAR-EQUILIBRIUM}
TEACHING-PARTS-OF: {LINEAR-EQUILIBRIUM}
CONCEPT-PART-OF: {LINEAR-EQUILIBRIUM}

******Time stamp: 14:33:13.

START-TOPIC: {LINEAR-EQUILIBRIUM-CONCEPT}

[......]

MOTIVATE-TOPIC: {LINEAR-EQUILIBRIUM-CONCEPT}
  :::
  :::Many problem situations in physics are much simpler if
the forces acting on an  object are balanced.  It is
important to learn to recognize these balanced force situations.

DEFINE-TOPIC: {LINEAR-EQUILIBRIUM-CONCEPT}
  :::
  :::An object is in equilibrium if the forces acting on the
object balance each other.   We sometimes say there is
no unbalanced force or the sum of the force vectors is zero.
```

```
  :::Choose:
     Click here to CONTINUE
     ****************
     INTERRUPT the session
  :::Chosen:Click here to CONTINUE
CHOOSING-NEXT-EASY-PRESENTATION: {LINEAR-EQUILIBRIUM-CONCEPT}
START-PRESENTATION: {LE-CONCEPT-EASY1}
START-EXAMPLE: {LE-CONCEPT-EASY1}
START-QUESTION: {LE-CONCEPT-EASY1}
  :::The person shown weighs 450 Newtons.  If the earth is
pulling down on this  person with  a 450 N force, what
other force is acting to keep this body in equilibrium?
  :::Choose:
     No other force is needed because the person is standing still.
     Balanced forces are not needed for objects in contact with the earth.
     The person's legs push up to keep him from falling down.
     The ground pushes up with a force of 450N.
     ****************
     INTERRUPT the session
  :::Chosen:The ground pushes up with a force of 450N.
Response-ok? CORRECT
FINISH-QUESTION: {LE-CONCEPT-EASY1}
  :::Good job!
  :::Although it is often hard to believe, the solid ground
does push up with a force of  450N.  This force just
balances the gravity force which acts downward on the person.
  :::Choose:
     Click here to CONTINUE
     ****************
     INTERRUPT the session

SESSION-SUSPENDED:

[Student comment entered here.]
****************  STUDENT COMMENT: ********************
could put a picture with the three forces here
**********************************************************



[......]
```

APPENDIX M

Sample Paper Worksheets

Below are examples of the worksheets used by the domain expert to specify the curriculum for data entry by the knowledge base manager. Paper forms were designed for topics, presentations, Mis-KUs, and crane-booms.

TOPIC

Topic name TENSION

Topic type Concept

Notes

1. Force same on both ends of a spring
2. Force same on both ends of a rope
3. Tension any where in a rope = the force on one end.

Motivation We need to be clear about the relationship between the forces pulling on the two ends of a rope and the idea of tension in the middle of a rope.

Definition
Tension equals: the force acting on one end of an element within the rope. If you think of a rope as having many short pieces, you may realize that the tension is the same throughout the rope.

Use-easy

Tension - Use - easy 1

Tension - Use - easy 2

Use-typical

Tension - Use - Typical 1

Tension - Use - Typical 2

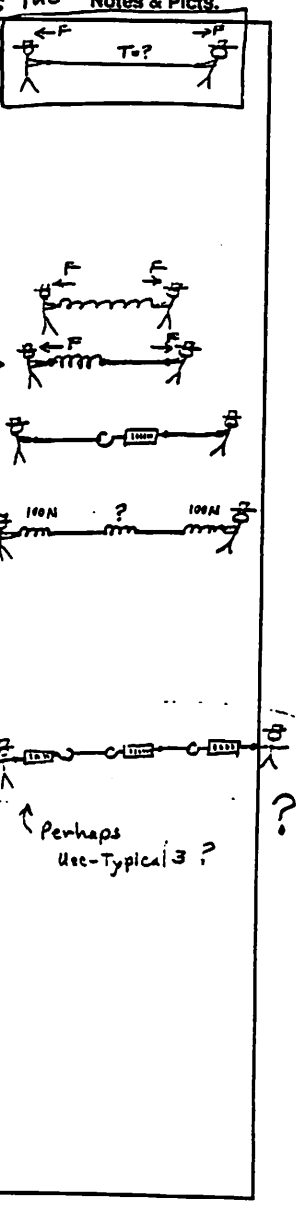
Summary A piece of rope or spring must feel equal and opposite forces on its two ends. Since a rope can be thought of as composed of many connected pieces, we can think of the force as being equal on any piece within the rope.

Wrap-up
The force on one end of a a rope equals the tension force at any spot within the rope.

Notes:
↳ (see top of the page)

perhaps include this picture?

Notes & Picts.

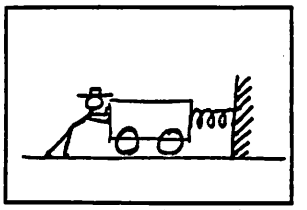


Entered: FL — **Who:**

8/8/89 — **When**

Presentation

Name Compression - MIS-KU-Remed1



Title: Man pushes cart on spri.

Diagram-descripton (set-up)
Picture of a person pushing a cart against a spring bumper attached to a wall (No motion).

✓ **Intro-text** The man is pushing on a cart which partly compresses a strong spring attached to the wall.

✓ **Question-text**
How many horizontal forces are acting on the cart?
A.) 1 B.) 2 C.) 3 D.) 4

Correct-answers B.

✓ **Hint 1** Are you including some force or forces of the cart pushing on other objects?

Hint 2

✓ **Answer give-away** There are two forces. One caused by the man pushing on the cart and the other caused by the spring pushing on the cart.

✓ **Answer reason**
The only things that can exert horizontal forces on the cart are objects that are touching it.

Elaboration:

✓

List of possible answers
(for each you can specify:
Reaction, Misconception/Branch)
Reaction to Answer A.) The cart is not moving so there must be two forces acting on the cart that cancel each other.

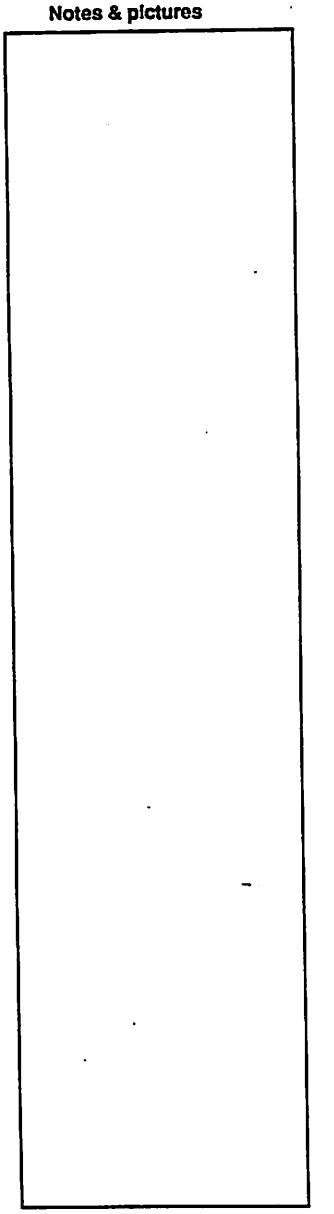
Notes

Entered: KDG (Who:) 03-20-90 (When)

MIS-KU

Name _____

Notes & pictures



Diagnositc-script

Remidiation-script

For Topics...

Notes

Entered: _____ _____

Who: When

Crane Boom

Name Sum Y-forces 2

Copy of crane boom Default-Horiz-CB1

[Interactive Simulation]
or [left Diagram] or [right Diagram]

cable height (h) Infinity

beam length (b)

weight postition on beam (d) 10

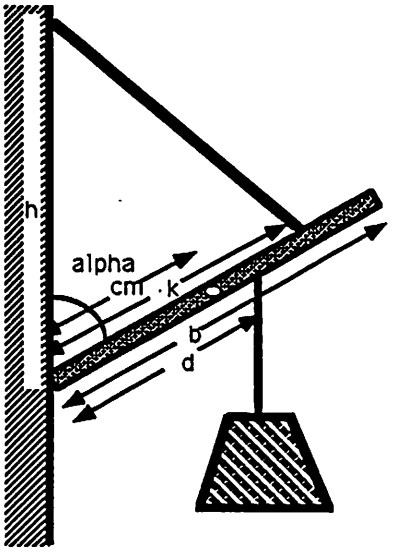
cable postion on beam (k) 4

beam-cntr-mass (cm)

beam angle (alpha)

beam weight (WB) 5

mass's weight (WM) 25



Lengths shown [d b k cm h]

Angles shown [alpha beta]

Vectors shown [(CT) FW (WB) (WM)]

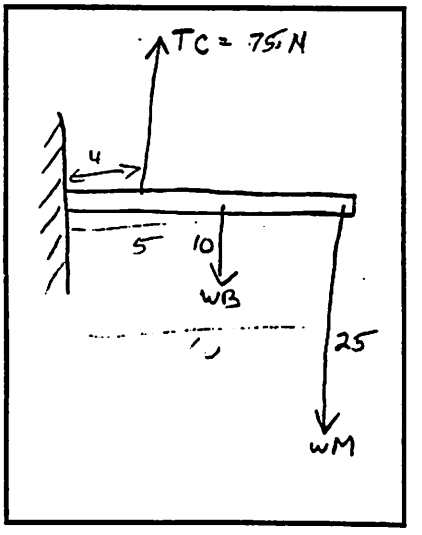
Vectors components shown
[Actual X Y Normal Radial]

Meters shown (for simul. only)
[(CT) FW (WB) (WM)]
[d b k cm h alpha beta]

Mode [abstract] [physical] [free body]

Show when created? [yes] [no]

Leave shown when done? [yes] [no]



Notes Ref. by: Procedure - Sum - Y - forces - Typical 2

? Should we add in the wall force and display in free body mode on the right screen with answer reason

Entered: [illegible] 11/7

Who: When

Bibliography

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

[2] Anderson, J. R. (1988). "The Expert Module." In Polson & Richardson (Eds.), *Foundations of Intelligent Tutoring Systems*, pg. 23-55. Hillsdale, NJ: Lawrence Erlbaum.

[3] Anderson, J. R. (1990). "Analysis of Student Performance with the LISP Tutor." *Diagnostic Monitoring of Skill and Knowledge Acquisition.* Hillsdale, NJ: Lawrence Erlbaum.

[4] Anderson, J. R. & Reiser, B. (1985). "The Lisp Tutor." *BYTE*, April 1985, pg. 159-175.

[5] Anderson, J. R. & Skwarecki, E. (1986). "The Automated Tutoring of Introductory Computer Programming." *Communications of the ACM*, Vol. 29 No. 9, pg. 842-849.

[6] Anderson, J. R., Boyle, C., and Reiser, B. (1985a). "Intelligent Tutoring Systems." *Science*, Vol. 228, pg. 456-462.

[7] Anderson, J. R., Boyle, F. & Yost, G. (1985b). "The Geometry Tutor." *Proceedings of IJCAI-85.* Los Angeles, CA.

[8] Anderson, J.R., Farrell, R. & Sauers, R. (1984). "Learning to Program in Lisp." *Cognitive Science*, Vol. 8, pg. 87-129.

[9] Aronson, D. & Briggs, L. (1983). "Contributions of Gagne and Briggs to a Prescriptive Model of Instruction." In Reigeluth (Ed.), *Instructional Design Theories and Models.* Hillsdale, NJ: Lawrence Erlbaum.

[10] Ausubel, D.P. (1963). *The Psychology of Meaningful Verbal Learning.* NY: Grune & Stratton.

[11] Baker, E. (1991). "Technology Assesment: Policy and Methodological Issues." In Burns, H., Parlett, J.W., & Redfield, C.L (Eds.), *Intelligent Tutoring Systems.* Hillsdale, NJ: Lawrence Erlbaum.

[12] Barker, V.E. & O'Connor, D.E. (1989). "Expert Systems for Configuration at Digital: XCON and Beyond." *Communications of the ACM*, Vol. 32 No. 3, pg. 297-318.

[13] Barr, A., Beard, M., & Atkinson, R. C. (1975). "Information Networks for CAI Curriculums." In O. Lecareme & R. Lewis (Eds.), Computers in Education. Amsterdam, The Netherlands: North-Holland.

[14] Begg, I.M. & Hogg, I. (1987). "Authoring Systems for ICAI." In Kearsley, G. (Ed.), *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

[15] Blomberg, J. L. & Henderson, A. (1990). "Reflections on Participatory Design: Lessons from the Trillium Experience." *CHI '90 Proceedings*, April, 1990, pg. 353-359.

[16] Bloom, B. S. (1984). "The 2-Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring." *Educational Researcher*, Vol 13, pg. 4-16.

[17] Bloom, B. S. (Ed.) (1956). *Taxonomy of Educational Objectives, Handbook I: Cognitive Domain.* New York: David McKay.

[18] Bloom, B., Madaus, G. & Hastings, J. (1981). *Evaluation To Improve Learning.* New York: McGraw Hill, Inc.

[19] Bogdan, R. & Biklen, S. K. (1982). *Qualitative Research for Education: An Introduction to Theory and Methods.* Boston: Allyn and Bacon, Inc.

[20] Bonar, J. & Soloway, E. (1985). "Pre-Programming Knowledge: A Major Source of Misconceptions in Novice Programmers." *Human-Computer Interaction*, Fall 1985, pg. 1-45.

[21] Bonar, J., Cunningham, R., Beatty, P., & Weil, W. (1988). "Bridge: An Intelligent Tutoring System with Intermediate Representations." LRDC Technical Report. Univ. of Pittsburgh, Pittsburgh, PA.

[22] Bonar, J., Cunningham, R., & Schultz, J. (1986). "An Object-Oriented Architecture for Intelligent Tutoring Systems." *Proceedings of OOPSLA-86.*

[23] Bonar, J.G. (1991). "Interface Architectures for Intelligent Tutoring Systems." In Burns. H., Parlett, J.W., & Redfield, C.L. (Eds.), *Intelligent Tutoring Systems: Evolutions in Design.* Hillsdale, NJ: Lawrence Erlbaum.

[24] Bonarini, A., Filippi, V., & Muti, S. (1988). "DOCET: An Environment to Build Intelligent Tutors." Milano Poyltechnic, Dept. of Electronics, Technical Report 88-033.

[25] Boose, J. H. (1988). "A Survey of Knowledge Acquisition Techniques and Tools." 3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop, November 1988, pg. 3.1-3.23. Banff, Canada.

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

[27] Brachman, R. & Schmoltze, J. (1985). "An Overview of the KL-ONE Knowledge Representation System." *Cognitive Science*, Vol. 9.

[28] Breuker, J., Winkels, R., & Sandberg, J. (1987). "A Shell for Intelligent Help Systems." *Proceedings of IJCAI-87*, pg. 167-173.

[29] Brooks, F. P. Jr. (1975). *The Mythical Man-Month.* Reading, MA: Addison-Wesley.

[30] Brown, D. (1989). "A Knowledge Acquisition Tool for Decision Support Systems." *SIGART Newsletter*, No. 108, pg. 93-97.

[31] Brown, S. F. (1985). "The Use of Learning Theory in the Application of Artificial Intelligence to Computer Assisted Instruction in Physics." Ph.D. Dissertation, North Texas State University.

[32] Bruner, J. (1966). *Toward a Theory of Instruction.* Cambridge, MA: Harvard University Press.

[33] Buchanan, B. (1987). "Artificial Intelligence as an Experimental Science." Stanford Univ. Knowledge Systems Lab Technical Report KSL 87-03.

[34] Bumbaca, F. (1988). "Intelligent Computer-Assisted Instruction: A Theoretical Framework." *International Journal of Man-Machine Studies*, Vol. 29, pg. 227-255.

[35] Burke, R. & Funaro, G. M. (1990). "Case-based Environments for Learning." *Working Notes of the AAAI Spring Symposium on Knowledge-Based Environments for Learning and Teaching*, pg. 63-67: March, 1990, Stanford, CA.

[36] Burns, H., & Parlett, J. (1989). *Proceedings of the Second Intelligent Tutoring Systems Research Forum.* San Antonio, TX: Air Force Systems Command.

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

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

[39] Carroll, J. M. & Rosson, M. B. (1987). "Paradox of the Active User." In Carroll (Ed.), Interfacing Thought: Cognitive Aspects of Human-Computer Interaction. Cambridge, MA: The MIT Press.

[40] Cerri, S. A. (1988). "The Requirements of Conceptual Modelling Systems." In Self, J. (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, pg. 88-108. New York, NY: Chapman & Hall Computing.

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

[42] Clancey, W. J. (1985). "Representing Control Knowledge as Abstract Tasks and Metarules." In Coombs & Bolc (Ed.), *Computer Expert Systems.* New York: Springer Verlag.

[43] Clancey, W. J. (1986a). "Qualitative Student Models." In Annual Reviews, Inc. (Eds.), *Annual Review of Computer Science*, pg. 381-450: Palo Alto, CA.

[44] Clancey, W. J. (1986b). "From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ONR Final Report 1979-1985." *The AI Magazine*, August, 1986, pg. 40-187.

[45] Clancey, W., & Joerger, K. (1988). "A Practical Authoring Shell for Apprenticeship Learning." *Proceedings of ITS-88*, pg. 67-74. June 1988, Montreal.

[46] Clark, C. & Peterson, P. (1986). "Teacher's Thought Processes." In M. C. Wittrock (Ed.), *Handbook of Research on Teaching, Third Edition.* New York: Macmillan Publ.

[47] Clement, J. (1982). "Student's Preconceptions in Introductory Mechanics." *American Journal of Physics*, Vol. 50 No. 1, pg. 66-77.

[48] Cohen, P. (1991). "Research Methodologies: Finding a Common Ground." *AI Magazine*, Spring 1991, pg. 15-41.

[49] Cohen, P. & Howe, A. (1988a). "How Evaluation Guides AI Research." *AI Magazine*, Winter 1988.

[50] Cohen, P. & Howe, A. (1988b). "Toward AI Research Methodology: Three Case Studies in Evaluation." Univ. of Massachusetts Dept. of Computer and Information Science Technical Report 88-31.

[51] Cohen, P., & Gruber, T. (1985). "Reasoning About Uncertainty: A Knowledge Representation Perspective." Univ. of Massachusetts Dept. of Computer and Information Science Tech. Report 85-24.

[52] Collins, A. (1977). "Processes in Acquiring Knowledge." In Anderson, R., Spiro, R., & Montagne, W. (Eds.), *Schooling and the Acquisition of Knowledge.* Hillsdale, NJ: Lawrence Erlbaum.

[53] Collins, A. & Stevens, A. (1983). "A Cognitive Theory of Inquiry Teaching." In Reigeluth, C. (Ed.), *Instructional-Design Theories and Models*, pg. 247-278. Hillsdale, NJ: Lawrence Erlbaum.

[54] Collins, A., Brown, J. S., & Newman, S. E. (1986). "Cognitive Apprenticeship: Teaching the Craft of Reading, Writing, and Mathematics." BBN Laboratories Incorporated, Technical Report No. 6459.

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

[56] Corbett, A. T. & Anderson, J. R. (1990). "The Effect of Feedback Control on Learning to Program with the LISP Tutor." *Proceedings of the Twelfth Annual Conference of the Cognitve Science Society*, pg. 796-806. July 1990, Cambridge, MA.

[57] Crandall, B. W. (1989). "A Comparative Study of Think-Aloud and Critical Decision Knowledge Elicitation Methods." *SIGART Newsletter*, No. 108, pg. 144-146.

[58] Davis, R., Buchanan, B., & Shortlife, E. (1977). "Production Rules as a Representation for a Knowledge-Based Consultant Program." *Artificial Intelligence*, Vol. 8 No. 1.

[59] Derry, S. J., Hawkes, L. W., Ziegler, U. (1988). "A Plan-Based Opportunistic Architecture for Intelligent Tutoring." *Proceedings of ITS-88*, pg. 116-123. June 1988, Montreal, Canada.

[60] Dillenbourg, P. (1988). "A Model of Knowledge Acquisition by Intelligent Tutoring Systems." *Proceedings of ITS-88*, pg. 145-153. June 1988, Montreal, Canada.

[61] Dreyfus, H. (1985). "From Micro-Worlds to Knowledge Representation: AI at an Impasse." In Brachman & Levesque (Eds.), *Readings in Knowledge Representation.* Los Altos, CA: Morgan Kaufmann.

[62] Duckworth, E., Kelley, J., & Wilson, S. (1987). "AI Goes to School." *Academic Computing*, November, 1987, pg. 6-63.

[63] Ericcson, K. A. & Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data.* Cambridge, MA: The MIT Press.

[64] Flavell, J. H. (1981). "Metacognition and Cognitive Monitoring: A New Area of Cognitive Development Inquiry." *American Psychologist*, Vol. 34, pg. 906-911.

[65] Ford, L. (1988). "The Appraisal of an ICAI System." In Self (Ed.), *Artificial Intelligence and Human Learning.* New York: Chapman and Hall.

[66] Frye, D., Littman, D. C., Soloway, E. (1988). "The Next Wave of Problems in ITS: Confronting the 'User Issues' of Interface Design and System Evaluation." In Psotka, J., Massey, L. D., Muttter, S. A. (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, pg. 451-478. Hillsdale, NJ: Lawrence Erlbaum.

[67] Gagne, R. M. (1974). "Task Analysis—Its Relation To Content Analysis." *Educational Psychologist*, Vol. 11, No. 1, pg. 11-18.

[68] Gagne, R. M. (1985). *The Conditions of Learning.* New York: Holt, Rinehart, and Winston.

[69] Gaines, B. R. & Boose, J. H. (1988). "Knowledge Aquisition for Knowledge-Based Systems." In Gaines & Boose (Ed.), *Knowledge Acquisition for Knowledge-Based Systems: Knowledge-Based Systems Volume 1*, pg. xiii-xix. New York: Academic Press.

[70] Gane, R. & Briggs, L. (1979). *Principles of Instructional Design.* Holt, Rinehart and Winston.

[71] Garg-Janardan, C. & Salvendy, G. (1988). "A Conceptual Framework for Knowledge Elicitation." In Boose & Gaines (Ed.), *Knowledge Acquision Tools for Expert Systems: Knowledge-Based Systems Volume 2*, pg. 119 - 129. New York: Academic Press.

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

[73] Gery, G. (1987). *Making CBT Happen.* Boston, MA: Weingarten Publ.

[74] Gilbert, G. N (1987). "Question and Answer Types." In D. S. Moralee (Ed.), *Research adn Development in Expert Systems IV*, pg. 162-172. Cambridge, MA: Cambridge Univ. Press.

[75] Gilmore, D. & Self, J. (1988). "The Application of Machine Learning to Intelligent Tutoring Systems." In Self, J. (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, pg. 179-196. New York: Chapman & Hall Computing.

[76] Glaser, B. G. & Strauss, A. I. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Hawthorne, NY: Aldine de Gruyter.

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

[78] Gould, J. D. (1988). "How to Design Usable Systems." In Helander, M. (Ed.), *Handbook of Human-Computer Interaction*, pg. 272-298. Amsterdam: Elsevier Science North Holland.

[79] Gruber, T. (1987). "A Method for Acquiring Strategic Knowledge from Experts." *Univ. of Massachusetts Dept. of Computer and Information Science Tech. Report.*

[80] Gruber, T. (1988). "The Acquisition of Strategic Knowledge." Univ. of Massachusetts Dept. of Computer and Information Science Tech. Report 88-101.

[81] Halasz, F. G., Moran, T. P., & Trigg, R. H. (1986). *Notecards in a Nutshell.* Palo Alto, CA: Intelligent Systems Lab.

[82] Halff, H. (1988). "Curriculum and Instruction in Automated Tutors." In Polson & Richardson (Eds.), *Foundations of Intelligent Tutoring Systems.* Hillsdale, NJ: Lawrence Erlbaum.

[83] Hart, A. (1986). *Knowledge Acquisition for Expert Systems.* New York: McGraw-Hill.

[84] Hoffman, R. (1987). "The Problem of Extracting the Knowledge of Experts From the Perspective of Experimental Psychology." *AI Magazine*, Summer 1987.

[85] Hoffman, R. (1989). "A Brief Survey of Methods for Extracting the Knowledge of Experts." *SIGART Newsletter*, No. 108, pg. 19-27.

[86] Hunt, B (1962). *Concept Learning: An Information Processing Problem.* New York: John Wiley and Sons.

[87] Issac, S. & Michael, W. (1977). *Handbook in Research and Evaluation.* San Diego, CA: EDITS Publishers.

[88] Johnson, W. B. (1988). "Pragmatic Considerations in Research, Development, and Implementation of Intelligent Tutoring Systems." In Polson & Richardson (Eds.), *Foundations of Intelligent Tutoring Systems*, pg. 191-208. Hillsdale, NJ: Lawrence Erlbaum.

[89] Johnson, W. L. (1988). "Modelling Programmers' Intentions." In Self, J. (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, pg. 374-390. New York, NY: Chapman & Hall Computing.

[90] Johnson, W.L. & Soloway, E. (1987). "PROUST: An Automatic Debugger for Pascal Programs." In Kearsley, G. (Ed.), *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

[91] Jones, M. & Wipond, K. (1989). "Intelligent Environments for Curriculum and Course Development." In Goodyear (Ed.), *Teaching Knowledge and Intelligent Tutoring.* Norwood, NJ: Ablex.

[92] Joyce, B. & Weil, M. (1972). *Models of Teaching.* Englewood Cliffs, NJ: Prentice-Hall.

[93] Kearsley, G. (Ed.) (1987). *Artificial Intelligence and Instruction, Applications and Methods.* Reading, MA: Addison-Wesley.

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

[95] Koedinger, K. R. & Anderson, J. R. (1990). "Theoretical and Empirical Motivations for the Design of ANGLE: A New Geometry Learning Environment." Working Notes of the AAAI Spring Symposium on Knowledge-based Environments for Learning and Teaching, pg. 6-10: March, 1990, Stanford, CA.

[96] Kopec, D. & Latour, L. (1989). "Towards an Expert/Novice Learning System With Application to Infectious Disease." *SIGART Newletter*, No. 108, pg. 140-143.

[97] LaFrance, M. (1989). "The Quality of Expertise: Understanding the Differences Between Experts and Novices." *SIGART Newsletter*, No. 108, pg. 6-14.

[98] Lajoie, S. (1986). "Cognitive Task Analysis: Implications for Intelligent Tutor Development." Presented at the AERA Annual Meeting, San Francisco, April 1986.

[99] Lawler, R. & Yazdani, M. (Eds.) (1987). *Artificial Intelligence and Education, Volume One, Learning Environments and Tutoring Systems.* Norwood, NJ: Ablex Publishing.

[100] LeClair, S. R. (1989). "Interactive Learning: A Multiexpert Paradigm for Acquiring New Knowledge." *SIGART Newsletter*, No. 108, pg. 34-44.

[101] Leinhardt, G. & Greeno, J. G. (1986). "The Cognitive Skill of Teaching." *Journal of Educational Psychology*, Vol. 78, No. 2, pg. 75-95.

[102] Lenat, D, Prakash, M., & Shepherd, M. (1986). "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks." *AI Magazine*, Winter 1986, pg. 65-85.

[103] Lepper, M.R. & Chabay, R.W. (1988). "Socializing the Intelligent Tutor: Bringing Empathy to Computer Tutors." In Mandl, H. & Lesgold, A. (Eds.), *Learning Issues for Intelligent Tutoring Systems.* NY: Springer-Verlag.

[104] Lesgold, A. (1988). "Toward a Theory of Curriculum Development for Use in Designing Insrtuctional Systems." In Mandl & Lesgold (Eds.), *Learning Issues for Intelligent Tutoring Systems.* New York: Springer-Verlag.

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

[106] Lewis, M. W., McArthur, D., Stasz, C., Zmuidzinas, M. (1990). "Discovery-Based Tutoring in Mathematics." Working Notes of the AAAI Spring Symposium on Knowledge-Based Environments for Learning and Teaching, pg. 21-28: March, 1990, Stanford, CA.

[107] Lewis, M.W., Milson, R., & Anderson, J.R. (1987). "The Teacher's Apprentice: Designing an Intelligent Authoring System for High School Mathematics." In Kearsley, G. (Ed.), *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

[108] Littman, D. & Soloway, E. (1988). "Evaluating ITSs: The Cognitive Science Perspective." In Polson & Richardson (Eds.), *Foundations of Intelligent Tutoring Systems*, pg. 209-242. Hillsdale, NJ: Lawrence Erlbaum.

[109] Lockhead, J. & Clement, J (Eds.) (1979). *Cognitive Process Instruction.* New York: The Franklin Institute Press.

[110] Macmillan, S., Emme, D., & Berkowitz, M. (1988). "Instructional Planners, Lessons Learned." In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned.* Hillsdale, NJ: Lawrence Erlbaum.

[111] Mandl, H. & Lesgold, A. (Eds.) (1988). *Learning Issues for Intelligent Tutoring Systems.* New York: Springer-Verlag.

[112] McCalla, G. & Greer, J. (1988). "Intelligent Advising in Problem Solving Domains: The SCENT-3 Architecture." *Proceedings of ITS-88*, pg. 124-131. June, 1988, Montreal, Canada.

[113] McCaslin, H. H. Jr. & Boord, P. M. (1990). "Project Development: Taking a Closer Look at the Subject Matter Expert." *Instruction Delivery Systems*, July/August 1990, pg. 21-24.

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

[115] McGraw, K. L. (1989). "Knowledge Acquisition for Intelligent Instructional Systems." *Journal of Artificial Intelligence in Education*, Vol. 1 No. 1, pg. 11-26.

[116] Means, B. & Gott, S. P. (1988). "Cognitive Task Analysis as a Basis for Tutor Development: Articulating Abstract Knowledge Representations." In Psotka, J., Massey, L. D., Mutter, S. A. (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, pg. 35-58. Hillsdale, NJ: Lawrence Erlbaum.

[117] Merriam, S. (1989). *Case Study Research in Education, A Qualitative Approach.* San Francisco: Jossey-Bass Publ.

[118] Merrill, D. & Tennyson, R. (1987). *Teaching Concepts: An Instructional Design Guide.* Englewood Cliffs, NJ: Educational Technology Publications.

[119] Merrill, M. D. (1983). "Component Display Theory." In Reigeluth (Ed.), *Instructional Design Theories and Models* Hillsdale, NJ: Lawrence Erlbaum.

[120] Merrill, M. D. (1987). "An Expert System for Instructional Design." *IEEE Expert*, Summer 1987, pg. 25-37.

[121] Merrill, M. D. & Li, Z. (1989). "An Instructional Design Expert System." *Journal of Computer-Based Instruction*, Vol. 16, No. 3, pg. 95-101.

[122] Merrill, M. D., Li, Z., & Jones, M. (1990). "Second Generation Instructional Design." *Educational Technology*, February 1990, pg. 7-14.

[123] Mervis, B. & Rosh, E. (1981). "Categories of Natural Objects." *Annual Review of Psychology*, Vol. 32.

[124] Miller, J. R. (1988). "The Role of Human-Computer Interaction in Intelligent Tutoring Systems." In Polson & Richardson (Eds.), *Foundations of Intelligent Tutoring Systems*, pg. 143-190. Hillsdale, NJ: Lawrence Erlbaum.

[125] Murray, K. S. (1988). "KI: An Experiment in Automating Knowledge Integration." Univ. of Texas at Austin AI Lab, Technical Report AI88-90.

[126] Murray, T. (1987). "Toward a General Architecture for Intelligent Tutoring Systems." *Univ. of Massachusetts Dept. of Computer and Information Science Tech. Report 87-89.* Amherst, MA.

[127] Murray, T. (1988). "Teaching Concepts with Examples: A Literature Exploration and Synthesis with Implimentation for ITS Design." , (Working paper).

[128] Murray, T. & Woolf, B. P. (1990). "A Knowledge Acquisition Framework Facilitating Multiple Tutoring Strategies." Working Notes for AAAI Symposium on Knowledge-Based Environments for Learning and Teaching. May 1990, Stanford, CA.

[129] Murray, T., Schultz, K., Brown, D., Clement, J. (1990). "An Analogy-Based Computer Tutor for Remediating Physics Misconceptions." *Interactive Learning Environments*, Vol. 1, No. 2, pg. 79-101.

[130] National Science Foundation (1983). *Educating America for the 21st Century.* Washington, DC.

[131] Nickerson, R. S. & Pew, R. W. (1990). "Toward More Compatible Human-Computer Interfaces." *IEEE Spectrum*, July 1990, pg. 40-43.

[132] Nicolson, R. (1988). "SCALD–Towards an Intelligent Authoring System." In J. Self (Ed.), *Artificial Intelligence and Human Learning.* New York: Chapman and Hall.

[133] Nielsen, J. (1990). "The Art of Navigating Through Hypertext." *Communications of the ACM*, Vol. 33, pg. 296-310.

[134] Norman, D. A. (1983). "Some Observations on Mental Models." In Gentner & Stevens (Eds.), *Mental Models.* Hillsdale, NJ: Lawrence Erlbaum.

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

[136] Ohlsson, S. (1986). "Some Principles of Intelligent Tutoring." *Instructional Science*, Vol. 14.

[137] Park, O. P., Perez, R. S., & Seidel, R. J. (1987). "Intelligent CAI: Old Wine in New Bottles, or a New Vintage?." In G. Kearsley (Ed.), *Artificial Intelligence and Instruction: Applications and Methods*, pg. 11-46. Reading, MA: Addison-Wesley.

[138] Patterson, A. C. & Bloch, B. (1987). "Formative Evalution: A Process Required in Computer-Assisted Instruction." *Educational Technology*, November, 1987, pg. 26-30.

[139] Patton, M. Q. (1980). *Qualitative Evaluation Methods.* Beverly Hills, CA: Sage Publications.

[140] Payne, S. J. (1988). "Methods and Mental Models in Theories of Cognitive Skill." In J. Self (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, pg. 69-87. New York: Chapman & Hall Computing.

[141] Perau, D. (1987). "Knowledge Acquisition in the Development of a Large Expert System." *AI Magazine*, Summer 1987, pg. 43-51.

[142] Polson, M, & Richardson, J. (Eds.) (1988). *Foundations of Intelligent Tutoring Systems.* Hillsdale, NJ: Lawrence Erlbaum Associates.

[143] Psotka, J. Massey, L. & Mutter, S. (Eds.) (1988). *Intelligent Tutoring Systems, Lessons Learned.* Hillsdale, NJ: Lawrence Erlbaum.

[144] Reigeluth, C. (1983a). "Instructional Design: What is it and Why is it?." In Reigeluth (Ed.), *Instructional Design Theories and Models.* Hillsdale, NJ: Lawrence Erlbaum.

[145] Reigeluth, C. (1983b). "The Elaboration Theory of Instruction." In Reigeluth (Ed.), *Instructional Design Theories and Models* Hillsdale, NJ: Lawrence Erlbaum.

[146] Reigeluth, C. (Ed.) (1983c). *Instructional-Design Theories and Models.* Hillsdale, NJ: Lawrence Erlbaum.

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

[148] Ridgeway, J. (1988). "Of Course ICAI is Impossible...Worse though, It Might Be Seditious." In J.A. Self (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* New York: Chapman & Hall.

[149] Rissland, E, Valcarce, E. & Ashley, K. (1984). "Explaining and Arguing With Examples." *Proceedings of AAAI-84*, August 1984. Austin, TX.

[150] Rissland, E. (1985). "The Structure of Knowledge in Complex Domains." In Chipman, Segal, & Glaser (Eds.), *Thinking and Learning Skills, Vol. 2: Research and Open Questions.* Hillsdale, NJ: Lawrence Erlbaum.

[151] Rosenberg, R. (1987). "A Critical Analysis of Research on Intelligent Tutoring Systems." *Educational Technology*, November, 1987, pg. 7-13.

[152] Russel, D. M., Moran, T. P., & Jordan, D. S. (1988). "The Instructional Design Environment." In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems: Lessons Learned.* Hillsdale, NJ: Lawrence Erlbaum.

[153] Russell, D. (1988). "IDE: The Interpreter." In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned* Hillsdale, NJ: Lawrence Erlbaum.

[154] Sacerdoti, E. (1974). "Planning in a Hierarchy of Abstraction Spaces." *Artificial Intelligence*, Vol. 5 No. 2.

[155] Sandberg, J., Breuker, J., Winkels, R. (1988). "Research on HELP-Systems: Empirical Study and Model Construction." Submitted for ECAI-88, Munchen.

[156] Schoenfeld. A. H. (1985). "Metacognitive and Epistemological Issues in Mathematical Understanding." In Silver (Ed.), *Teaching and Learning Mathematical Problem Solving*, pg. 361-380. Hillsdale, NJ: Lawrence Erlbaum.

[157] Schofield, J. & Verban, D. (1988). "Barriers and Incentives To Computer Usage in Teaching." LRDC Technical Report No. 1.

[158] Schooler, L. J. & Anderson, J. R. (1990). "The Disruptive Potential of Immediate Feedback." *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, pg. 702-708. July, 1990, Cambridge, MA.

[159] Schultz, K., Murray, T., Clement, J., & Brown, D. (1987). "Overcoming Misconceptions with a Computer-based Tutor." Presented at the Second International Seminar on Misconceptions and Educational Strategies for Science and Mathematics, Vol. 3. July, 1987, Ithica, NY.

[160] Self, J. (1985). "Intelligent Computer Assisted Instruction." Unpublished paper presented at the ICAI Spring Seminar. Cambridge.

[161] Self, J. A. (1988b). "Bypassing the Intractable Problem of Student Modelling." *Proceedings of ITS-88*, pg. 18-24. June 1988, Montreal, Canada.

[162] Self, J. A. (Ed.) (1988a). *Artificial Intelligence and Human Learning.* New York: Chapman and Hall.

[163] Servi, D. & Woolf, B. (1986). "E.T.: An Architecture for Error-Triggered Remediation in Intelligent Tutoring Systems." Presented to the 1986 AI in Education Conference. Pittsburgh, NJ.

[164] Shadbolt, N. & Burton, M. (1989). "The Empirical Study of Knowledge Elicitation Techniques." *SIGART Newsletter*, No. 108, pg. 15-18.

[165] Shaw, M. L. G. (1989). "A Grid-Based Tool for Knowledge Acquisition: Validation with Multiple Experts." *SIGART Newsletter*, No. 108, pg. 168-169.

[166] Shaw, M. L. G. (Ed.) (1981). *Recent Advances in Personal Construct Technology.* New York: Academic Press.

[167] Shaw, M. L. G. & Gaines, B. R. (1986). "Advances in Interactive Knowledge Engineering." Submitted to Expert Systems '86. University of Calgary, Alberta, CANADA: Dept. of Computer Science.

[168] Shute, V. J. (1990). "Rose Garden Promises of Intelligent Tutoring Systems: Blossom or Thorn?." Presented at Space Operations, Applications and Research Symposium, June 1990, Albuquerque, NM.

[169] Simon, H. A. (1981). *The Science of the Artificial.* Cambridge, MA: MIT Press.

[170] Skinner, B.F. (1957). *Verbal Behavior.* New York: Appleton-Century-Crofts.

[171] Slagle, J. & Wick, M. (1988). "A Method for Evaluating Candidate Expert System Applications." *AI Magazine*, Winter 1988, pg. 45-63.

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

[173] Sleeman, D. (1987a). "PIXIE: A Shell for Developing Intelligent Tutoring Systems." In Lawler & Yazdani (Eds.), *Artificial Intelligence and Education: Volume One.* Norwood, NJ: Ablex.

[174] Sleeman, D. (1987b). "Micro-Search: A Shell for Building Systems to Help Students Solve Nondeterministic Tasks." In G. Kearsley (Ed.), *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

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

[176] Stevens, A. & Collins, A. (1977). "The Goal Structure of a Socratic Tutor." Bolt Beranek and Newman Inc. Technical Report 3518. Cambridge, MA.

[177] Stevens, A. & Steinberg, C. (1981). "A Typology of Explanations and its Application to Intelligent Computer Aided Instruction." Technical Report No. 4626. Cambridge, MA: Bolt Beranek and Newman, Inc.

[178] Tennyson, R. (1986). "MAIS: An Educational Alternative of ICAI." *Educational Technology*, May 1987, pg. 22-28.

[179] Tennyson, R. D. & Rasch, M. (1988). "Linking Cognitive Learning Theory to Instructional Prescriptions." *Instructional Science*, Vol. 17, pg. 369-385.

[180] Tennyson, R. & Christensen, D. (1988). "MAIS: An Intelligent Learning System." In Jonassen (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, NJ: Lawrence Erlbaum.

[181] Tennyson, R. & Park, O. (1980). "The Teaching of Concepts: A Review of Instructional Design Research Literature." *Review of Educational Research*, Vol. 50 No. 1, pg. 55-70.

[182] U.S. Department of Education (1983). *Proceedings of the Office of Education Research and Improvement.* Washington, DC.

[183] U.S. Department of Education (1983). *Computers in Education: Realizing the Potential.* Washington, DC.

[184] Van Heuvelen, A. (1987). "A Hierarchical Spiral Method of Teaching Physics." Presented at the Summer Meeting of the American Association of Physics Teachers, Bozeman, MT.

[185] VanLehn, K. (1988). "Toward a Theory of Impasse-Driven Learning." In Mandl, H. & Lesgold, A. (Eds.), *Learning Issues for Intelligent Tutoring Systems.* NY: Springer-Verlag.

[186] VanLehn, K. (1988b). "Student Modeling." In Polson & Richardson (Eds.), *Foundations of Intelligent Tutoring Systems.* Hillsdale, NJ: Lawrence Erlbaum.

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

[188] Von Glasersfeld, E. (1989). "Knowing without Metaphysics: Aspects of the Radical Constructivist Position." In F. Steier (Ed.), Research and Reflexivity: Toward a Cybernetic/Social Constructionist Way of Knowing. London: Sage Publications.

[189] Wenger, E. (1987). *Aritificial Intelligence and Tutoring Systems.* Los Altos, CA: Morgan Kauffmann.

[190] White, B. & Frederiksen, J. (1986). "Progressions of Qualitative Models as a Foundation for Intelligent Learning Environments." Technical Report No. 6277. Cambridge, MA: Boldt Beranek and Newman.

[191] Whiteside, J., Bennett, J., & Holtzblatt, K. (1988). "Usability Engineering: Our Experience and Evolution." In M. Helander (Ed.), *Handbook of Human-Computer Interaction* Amsterdam: Elsevier Science North-Holland Press.

[192] Winkels, R., Breuker, J., & Sandberg, J. (1988). "Didactic Discourse in Intelligent Help Systems." *Proceedings of ITS-88*, pg. 279. Montreal, Canada.

[193] Winkels, R., Sandberg, J. & Breuker, J. (1986). "Coaching Strategies and Tactics of Intelligent Help Systems." Memo 78 of the VF-Project. University of Amsterdam.

[194] Winne, P. (1989). "Theories of Instruction and of Intelligence for Designing Artificially Intelligent Tutoring Systems." *Educational Psychologist*, Vol. 24 No. 3, pg. 229-259.

[195] Winne, P. & Kramer, L. (1988). "Representing and Inferencing with Knowledge about Teaching: DOCENT." *Proceedings of ITS-88*. June 1988, Montreal, Canada.

[196] Winograd, T. & Flores, F. (1986). *Understanding Computers and Cognition: A New Foundation for Design*. Norwood, NJ: Ablex.

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

[198] Woolf B. P. & Cunningham, P. (1987). "Multiple Knowledge Sources in Intelligent Tutoring Systems." IEEE Expert, Summer 1987, pg. 41-54.

[199] Woolf, B. P. (1990). "Knowledge-Based Tutors: An Artificial Intelligence Approach to Education." Ed.D. Dissertation, Univ. of Massachusetts School of Education. Amherst, MA.

[200] Woolf, B. P. (in press). "AI in Education." In Shapiro (Ed.), *Encyclopedia of AI, Second Edition* John Wiley & Sons.

[201] Woolf, B. & Blegen, D., Jansen, D., Verloop, A. (1986). "Teaching a Complex Industrial Process." *Proceedings of AAAI-86*, pg. 722-727.

[202] Woolf, B. & McDonald, D. (1984). "Building a Computer Tutor: Design Issues." *IEEE Computer*, September 1984, pg. 61-73.

[203] Woolf, B. & Murray, T. (1987). "A Framework for Representing Tutorial Discourse." *Proceedings of IJCAI-87*, pg. 189-192.

[204] Woolf, B. & Murray, T. (1987). "Discourse Transition Networks for Intelligent Tutoring Systems." Unpublished paper presented at the Third International Conference on Artificial Intelligence and Education. May 1987, Pittsburgh, PA.

[205] Woolf, B., Murray, T., Suthers, D. & Schultz, K. (1988). "Knowledge Primitives for Tutoring Systems." *Proceedings of ITS-88*, pg. 491-498. June 1988, Montreal, Canada.

[206] Young, R. M. (1983). "Surrogates and Mappings: Two Kinds of Conceptual Models for Interactive Devices." In Gentner & Stevens (Eds.), *Mental Models*. Hillsdale, NJ: Lawrence Erlbaum.