TOM MURRAY

Chapter 17

AN OVERVIEW OF INTELLIGENT TUTORING SYSTEM AUTHORING TOOLS:

Updated Analysis of the State of the Art

Abstract. This paper consists of an in-depth summary and analysis of the research and development state of the art for intelligent tutoring system (ITS) authoring systems. A seven-part categorization of two dozen authoring systems is given, followed by a characterization of the authoring tools and the types of ITSs that are built for each category. An overview of the knowledge acquisition and authoring techniques used in these systems is given. A characterization of the design tradeoffs involved in building an ITS authoring system is given. Next the pragmatic questions of real use, productivity findings, and evaluation are discussed. Finally, I summarize the major unknowns and bottlenecks to having widespread use of ITS authoring tools.

1. INTRODUCTION

Intelligent Tutoring Systems (ITSs) are computer-based instructional systems with models of instructional content that specify *what* to teach, and teaching strategies that specify *how* to teach (Wenger 1987, Ohlsson 1987, Shute & Psotka 1996). They make inferences about a student's mastery of topics or tasks in order to dynamically adapt the content or style of instruction. Content models (or knowledge bases, or expert systems, or simulations) give ITSs depth so that students can "learn by doing" in realistic and meaningful contexts. Models allow for content to be generated in real time. ITSs allow "mixed-initiative" tutorial interactions, where students can ask questions and have more control over their learning. Instructional models allow the computer tutor to more closely approach the benefits of individualized instruction by a competent pedagogue. In recent years ITSs have moved out of the lab and into classrooms and workplaces where some have been shown to be highly effective (Shute and Regian 1990; Koedinger et al. 1997; Mark & Greer 1991; Person et al. 2001; Rosé, et al. 2001). While intelligent tutors are becoming more common and proving to be increasingly effective they are difficult and expensive to build. Authoring systems are commercially available for traditional computer aided instruction (CAI) and multimedia-based training, but these authoring systems lack the sophistication required to build intelligent tutors. Commercial multimedia authoring systems excel in giving the instructional designer tools to produce visually appealing and interactive screens, but behind the screens is a shallow representation of content and pedagogy. Researchers have been investigating ITS authoring tools almost since the beginning of ITS research, and over two dozen very diverse authoring systems have been built. This paper summarizes the contributions of these systems and describes the state of the art for ITS authoring tools.

This article is written for two types of readers. First are research and development personnel who are building ITS and/or ITS authoring tools. They might ask the question "what methods and designs have been used, and how successful have they been?" in their efforts to build the next generation of systems. The second type of reader is the developer or purchaser of instructional software (intelligent or otherwise) who might ask the question: "what is really available (or soon to be available) to make ITS authoring cost effective?" I hope both readers will find this article informative. For those needing an "executive level summary": 1) In the last few years there has been significant progress in the development of ITS authoring tools and in the understanding of the key issues involved. 2) The development efforts to date represent many diverse approaches, and it is still too early to get a sense for which approaches will prove to be the most useful (or marketable). 3) In general, ITS authoring tools are still research vehicles which have demonstrated significant success in limited cases, yet have not been made robust enough to be placed and supported in production contexts or commercial markets. However, it is encouraging that several systems have been released as products or are approaching productization, and some relatively large scale evaluations have taken place. Significant progress has been made since this first version of this paper was published in 1999.

The paper is organized according to four broad questions that readers might have concerning ITS authoring tools:

- What types of tutors can be built with existing authoring tools?
- What features and methods do the tools use to facilitate authoring?
- Have the tools been used in realistic situations; have they been evaluated; are they available?
- What have researchers learned about the process of authoring and the tradeoffs involved in designing an authoring tool?

The Sections of this paper are sequenced to answer these questions. I first describe the types of ITSs that have been built with ITS authoring tools. Next I describe the interface, knowledge representation, and knowledge acquisition techniques that have been used to allow non-programmers to build ITSs using authoring tools. Then I report on the pragmatic aspects of ITS authoring in order to locate current work in the research-to-application spectrum. Finally I discuss a number of general issues and lessons learned (for example "who should author ITSs?"), and discuss tradeoffs between power, usability, and fidelity among authoring tools.

2. A CLASSIFICATION ACCORDING TO TASKS AND TUTORS

Any discussion about authoring tools would be too abstract without some context describing the tutors that they have been used to build. ITS authoring tools have been used to build tutors in a wide range of domains, including customer service, mathematics, equipment maintenance, and public policy. These tutors have been targeted toward a wide range of students, from grade school children to corporate trainees. However, the key differences among ITS authoring systems are not related to specific domains or student populations, but to the domain-independent capabilities that the authored ITSs have. In this Section I present a classification of authoring tools based on these capabilities. But before describing a number of ITS authoring tools I need to mention a related area of work that will not be directly addressed.

Shells vs. tools. An ITS "shell" is a generalized framework for building ITSs, while an ITS "authoring system" (or authoring tool) is an ITS shell along with a user interface that allows non-programmers to formalize and visualize their knowledge. Inspired by goals of elegance, parsimony, and/or cost effectiveness, software designers seem naturally driven to write software that is general and reusable. Thus there have been many papers published describing ITS "shells" that consist of software architectures, code libraries, or conceptual frameworks that make ITS construction more efficient for programmers. Though some of these systems include form-based data entry to support authoring tasks, most of them are either content acquisition shells or instructional planning shells. For examples, see Goodkovsky et al., 1994 (Pop ITS shell), Ikeda & Mizoguchi, 1994 (FITS), McCalla & Greer, 1988 (SCENT-3), Goodyear & Johnson, 1990 (TOSKA), Anderson & Pelliteir, 1991 (TDK), McMillan et al., 1980 (SIPP), Wasson, 1992 (PEPE), Winne & Kramer, 1989 (DOCENT), Jona & Kass, 1997 (GBS architectures). This paper focuses on authoring tools only.

A bags of tricks vs. a shelf of tools. Over two dozen ITS authoring systems have been built. They differ by the types of domains and tasks they are suited for, by the degree to which they make authoring more easy or efficient, and by the depth and fidelity employed to represent the knowledge or skill being taught. Not all of the designers of these systems would describe their systems as being "ITS authoring systems." But I include computer-based instruction authoring systems that use AI representation techniques such as rules and semantic networks, and those that include models of content and/or teaching strategies. These systems seem to populate the space of authoring tool features almost uniformly, making it difficult to cluster them into discrete groups in an effort to summarize the field. In fact, every system I will describe in one category has important elements from at least one other category. Since the field is still in a formative stage, this paper is intended to help the reader envision the next generation of authoring tools, more than to select an existing one to use. Therefore its organization is more like the description of a "bag of tricks" that can be mixed and matched to create an authoring tool than a description of a shelf of completed authoring tools.

Early ITS authoring systems fell into two broad categories: those based on a traditional curriculum (or courseware) metaphor and those geared toward device simulation and embodying a "learning environments" instructional metaphor. The majority of current authoring tools fall similarly into two broad categories: pedagogy-oriented and performance-oriented (Murray 1997). Pedagogy-oriented systems (categories 1, 2, 5, and 7 in Table 1) focus on how to sequence and teach relatively canned content. Most of them pay special attention to the representation of teaching strategies and tactics. Performance-oriented systems (categories 3, 4, 6, and sometimes 5, in Table 1) focus on providing rich learning environments in which students can learn skills by practicing them and receiving feedback. Most of them pay special attention to the representation of human problem solving skills or domain-specific processes or systems (either man made ones such as electrical components, or natural ones such as the meteorology). In general, performance-oriented systems focus on feedback and guidance at the level of individual skills and procedural steps, while pedagogy-oriented systems pay more attention to guidance and planning at a more global level, looking at the sequence of topics and the need for prerequisite topics. Some systems have elements of both categories, but those that do tend not to have very complete or deep representational systems for both pedagogy and domain processes. Though it is possible to create a system with full capabilities in both areas, none yet exists. This is probably due to a combination of the fact that supporting each type of authoring is difficult (and each project has limited recourses to spend) and supporting each type of authoring requires a very different kind of expertise.

Table 1: ITS Authoring Tools by Category ([brackets] refer to Chapter numbers)

CATEGORY	**AUTHORING SYSTEMS**
1. Curriculum Sequencing and Planning	Swift/DOCENT, IDE, ISD Expert, Expert CML
2. Tutoring Strategies	REDEEM (& COCA) [8], Eon [11], GTE
3. Simulation-Based Learning	SIMQUEST [1], XAIDA [2], RIDES[3], DIAG [5], Instructional Simulator [7]
4. Domain Expert System	Demonstr8 [4], DIAG [5], D3 Trainer, Training Express
5. Multiple Knowledge Types	XAIDA [2], DNA [6], Instructional Simulator & IDVisualizer [7], ID-Expert, IRIS [9], CREAM-Tools [10], ,
6. Special Purpose	IDLE-Tool/Imap/Indie [12], LAT [14], BioWorld Case Builder, WEAR
7. Intelligent/Adaptive Hypermedia	InterBook [13], MetaLinks, CALAT, GETMAS, TANGOW, ECSAIWeb

Table 2: ITS Authoring Tool Strengths and Limitations by Category

CATEGORY	STRENGTHS	LIMITS	VARIATIONS
Curriculum Sequencing and Planning	Rules, constraints, or strategies for sequencing courses, modules, presentations	Low fidelity from student's perspective; shallow skill representation	Whether sequencing rules are fixed or authorable; scaffolding of the authoring process
Tutoring Strategies	Micro-level tutoring strategies; sophisticated set of instructional primitives; multiple tutoring strategies	(same as above for most systems)	Strategy representation method; source of instructional expertise
Device Simulation and Equipment Training	Authoring and tutoring matched to device component identification, operation, and troubleshooting	Limited instructional strategies; limited student modeling; mostly for procedural skills	Fidelity of the simulation; ease of authoring
Domain Expert System	Runnable (deeper) model of domain expertise; fine grained student diagnosis and modeling; buggy and novice rules included	Building the expert system is difficult; limited to procedural and problem solving expertise; limited instructional strategies	Cognitive vs. performance models of expertise
Multiple Knowledge Types	Differential pre-defined knowl. representation and instructional methods for facts, concepts, and procedures, etc.	Limited to relatively simple fact, concepts, and procedures; pre-defined tutoring strategies	Inclusion of intelligent curriculum sequencing; types of knowledge/tasks supported
Special Purpose	Template-based systems provide strong authoring guidance; fixed design or pedagogical principles can be enforced	Each tool limited to a specific type of tutor; inflexibility of representation and pedagogy	Degree of flexibility
Intelligent/ Adaptive Hypermedia	WWW has accessibility & UI uniformity; adaptive selection and annotation of hyperlinks	Limited interactivity; limited student model bandwidth	Macro vs. micro level focus; degree of interactivity

Proponents of constructivist learning theories (e.g. Jonassen & Reeves 1996) often criticize pedagogy-oriented tutors and the instructional design theories behind them as being too "instructivist." Such critics contend that these systems ignore important aspects of learning such as intrinsic motivation, context realism, common misconceptions, and social learning contexts. Actually these factors are acknowledged by most instructional design theorists (Merrill 1983, Gagne 1985, Reigeluth 1983), but are either seen as not being as important or as being too complex or incompletely understood to incorporate into instructional planning and knowledge representation.[1]

Table 1 enumerates seven categories of ITS authoring systems, grouped according to the type of ITSs they produce.[2] Table 2 describes the strengths and limitations of each category, along with a summary of how systems within the category differ. The categories are described below in their particular sequence because some build upon concepts developed in previous categories.

2.1. Curriculum sequencing and planning

Authoring systems in the Curriculum and Course Sequencing category organize instructional units (IUs, or "curriculum elements") into a hierarchy of courses, modules, lessons, presentations, etc., which are related by prerequisite, part, and other relationships. The instructional units typically have instructional objectives. Some systems include IUs that address misconceptions or remedial material. The content is stored in canned text and graphics. These systems are seen as tools to help instructional designers and teachers design courses and manage computer based learning.

Intelligent sequencing of IUs (or content, or topics) is at the core of these systems. To the student, tutoring systems built with these tools may seem identical to traditional computer-based instruction. Screens of canned text and pictures are presented, and interactions tend to be limited to multiple choice, fill-in, etc. Of course, the difference is that the sequencing of the content is being determined dynamically based on the student's performance, the lesson objectives, and the relationships between course modules. Because domain knowledge is not represented in a very "deep" fashion, any arbitrary domain can be tutored (just as a textbook can be about any domain). But the depth of diagnosis and feedback in tutors built with these authoring tools is limited by the shallowness of their domain knowledge representation. This makes them more appropriate for building tutors

[1] Historically, instructional design theories were ignored by most ITS researchers in favor of cognitive learning theories, but in the realm of ITS authoring tools instructional design was a primary basis for the early systems. Thus I believe the authoring tools research community was instrumental in promoting the more balanced merger of instructional design and cognitive theories that we increasing see in recent years.

[2] In the case of two relatively large-scale ITS authoring system projects, MITT-Writer and ICAT, there was insufficient published material for me to include them in my analysis (these systems are mentioned in an overview of US government sponsored ITS research (Youngblut 1995)).

that teach conceptual, declarative, and episodic types of knowledge, and less pedagogically powerful for building tutors that teach procedural or problem solving skills. Authoring systems in the Curriculum Sequencing category are, generally speaking, the most "basic," or minimally functional (though each system in this category has certain very evolved signature features or capabilities). All of the systems in this category except Swift are "historical" in that they were early examples that inspired other projects. In particular, the systems in categories 2 and 5 build upon the functionalities of category 1 systems. Swift is a commercial product that calls itself an authoring tool for "minimalist" ITSs.

2.2. Tutoring strategies

Systems in this category excel at representing diverse teaching strategies. They tend to be similar to the Curriculum Sequencing systems described above, in that content is stored in canned text and graphics and domain knowledge representation is shallow. Systems in this category go beyond Curriculum Sequencing systems to encode fine-grained strategies used by teachers and instructional experts. Systems in the Curriculum Sequencing category tend to focus on the "macro" level of instruction--i.e. the sequencing of topics or modules, while systems in this category also address the "micro" level of instruction. Instructional decisions at the micro level include when and how to give explanations, summaries, examples, and analogies; what type of hinting and feedback to give; and what type of questions and exercises to offer the student. Systems in the Tutoring Strategies category have the most sophisticated set of primitive tutorial actions, compared with systems in other categories. In addition some systems in this category represent multiple tutoring strategies and "meta-strategies" that select the appropriate tutoring strategy for a given situation (e.g. REDEEM and Eon). The availability and intelligent interjection of small grain sized components such as explanations, multiple levels of hints, and analogies can make the tutor appear quite responsive, at times even conversational (as in Socratic strategies).

In GTE and COCA authors create instructional rules. In Eon teaching strategies are defined using procedural flow lines. In REDEEM authors specify strategy parameters using sliders.

2.3. Simulation-based learning

For tutors built by authoring tools in this category, instruction centers around a simulation of a man-made process or phenomena. In RIDES, XAIDA, DIAG, and Instructional Simulator the student is shown a piece of equipment and is asked to identify its components, perform operating steps, perform maintenance steps, or diagnose faulty device behavior and fix or replace the implicated parts. These authoring tools have been used to build instructional simulations of mechanical, electrical, and hydraulic systems. These types of skills are relatively widespread and generic, so authoring tools that specialize in this area should be widely usable. The expert knowledge for component locations and operational scripts is straightforward to model. Performance monitoring and instructional feedback is also straightforward (e.g. "That is not the Termination Switch," and "You should have checked the safety valve as your next step"). Thus authoring tools can be built which closely match the needs of the author (and student).[3] The most difficult authoring task with these systems is building the simulation. But once the simulation is authored, much of the instructional specification comes "for free." Component location and device behavior "what if" activities can be generated automatically. However, the device operation procedures must be authored. The four systems mentioned differ in the complexity of the simulation models that they can author. While these systems focus mostly on learning procedural skills, SimQuest focuses more on conceptual knowledge and physical principles. It has features for providing explanations of phenomena, exploratory and hypothesis generation learning activities, and instructional sequences based on a "model evolution" paradigm (White & Frederiksen 1995). SimQuest simulations are based on a set of equations which constitute a model of a physical process. Though they are not covered in this paper, there are other simulation-based and model-based educational software projects that incorporate authoring tools for building models (see a brief summary in Murray, et al. 2001).

In contrast to the previous two categories of authoring tools, students using tutors built with tools in this category will be "learning by doing." It is usually assumed that students have a basic familiarity with important concepts and procedures in the domain before using the tutor and students immediately start to practice skills. Specific feedback is given for each skill step, and task difficulty is increased as students progress.

A major differentiating factor among systems in this category is the depth and fidelity of the simulation. Authoring tools range from those supporting static expression-based relationships between device components (XAIDA, which also supports domains other than device simulation, see the Multiple Knowledge Types category), to those supporting runnable but shallow simulation models (RIDES), to those supporting deeper, more causative or cognitive models of how the device works (SIMQUEST).

2.4. Expert systems and cognitive tutors

An important class of intelligent tutors are those that include rule-based cognitive models of problem solving expertise. Such tutors, often called model tracing tutors (Anderson & Pelletier 1991), observe student behavior and build a fine-grained cognitive model of the student's knowledge that can be compared with the expert model. Authoring tools have been prototyped for such tutors. I also include in this category authoring tools which use traditional expert systems (built to solve problems, not to teach) and produce "value added" instruction for the encoded

[3] Equipment diagnosis tasks ("troubleshooting") are more complicated, less standard among types of equipment, and thus more difficult task to model and teach than operation and maintenance steps.

expertise. These systems are similar to model tracing systems, except the expert system is based on performance competency, rather than cognitive processes. Some systems include buggy or novice-level rules that capture common mistakes, allowing the tutorial to give feedback specific to those errors.

Students using these systems usually solve problems and associated sub-problems within a goal space, and receive feedback when their behavior diverges from that of the expert model. Unlike most other systems described above, these systems have a relatively deep model of expertise, and thus the student, when stuck, can ask the tutor to perform the next step or to complete the solution to the entire problem. Authoring an expert system is a particularly difficult and time-intensive task, and only certain tasks can be modeled in this manner.

Demonstr8 allows authors to build an expert problem solver through demonstration of an example problem solution. D3 trainer and Training Express add a tutorial component onto an existing expert system (authored with an expert system shell). DIAG includes an expert model of fault diagnosis strategies.

2.5. Multiple knowledge types

Instructional design theories classify knowledge and tasks into discrete categories, and prescribe instructional methods for each category. They tend to be limited to types of knowledge that can be easily defined, such as facts, concepts, and procedures. Though the knowledge types and instructional methods vary for different theories, they typically prescribe instruction similar to the following. Facts are taught with repetitive practice and mnemonic devices; concepts are taught using analogies and positive and negative examples progressing from easy prototypical ones to more difficult borderline cases; procedures are taught one step at a time, with particular attention payed to branching decision steps. Instruction for these knowledge types includes both expository presentations of the knowledge, and inquisitory exercises that allow for practice and feedback. Straight-forward instructional strategies for how to sequence content and exercises, and how to provide feedback, are defined separately for each knowledge type (for example see Merrill 1983). The pre-defined nature of the knowledge and the instructional strategies is both the strength and the weakness of these systems. Domain knowledge for each knowledge type can be easily represented for authors, who fill in templates for examples, steps, definitions, etc. (depending on the knowledge type). The tools support the decomposition of complex skills into elementary knowledge components. Links between knowledge components can be authored and used in instruction (e.g. the concepts or facts that support a procedure or a concept that helps explain another concept). Since instructional strategies are fixed and based on knowledge types they do not have to be authored. Of course, not all instruction fits neatly into this framework, but it has significantly wide applicability (problem solving and other higher order cognitive skills can not be represented in this way).

Authoring systems in the Multiple Knowledge Types category are diverse in many respects, but they all use a knowledge/skill classification scheme and represent and instruct differentially based on knowledge type. Also, they all cite classic instructional design literature as part of the basis for their pedagogical approach. For the student, tutors built with these systems are similar in character to those in the Multiple Teaching Strategies category. The main difference is for the authors, whose task is more constrained, and thus both easier and less flexible.

2.6. Special purpose systems

In this category are authoring tools that specialize in particular tasks or domains. Systems in the Device Simulation and Multiple Knowledge Types categories are also for particular types of tasks, but systems in the Special Purpose category focus on more specific, less general tasks. There is a rough principle that authoring tools tailored for specific tasks or instructional situations can better support the needs of the student and author for those situations. Systems in this category were designed by starting with a particular intelligent tutor design, and generalizing it to create a framework for authoring similar tutors. Authoring is much more template-like than in other categories of authoring tools. One potential problem with special purpose authoring is that once a task and its instructional approach have been codified enough to become a template, the resulting system reflects a very particular approach to representing and teaching that task; one that may only appeal to a limited authoring audience. On the other hand, preferred design and pedagogical principles can be strictly enforced, since the author has no influence over these aspects. Bell (this volume) calls this approach using a "strong task model" to build "knowledge rich" authoring tools.

Since the only thing that systems in this category have in common is that they support particular types of tasks or domains, we can not say anything in general about the types of tutors built or about students' experience using the tutors. The IDLE-Tool/Imap/Indie systems build tutors for "investigate and decide" GBS (goal based scenario) learning environments. LAT builds tutors to train customer service representatives how to answer product questions. Both of the above mentioned projects involve learning by doing in role playing environments. BioWorld is a learning environment for promoting scientific reasoning in the medical domain. Its authoring tool is a template-based tool allowing teachers to create new medical cases. WEAR builds tutors in algebra-related domains.

2.7. Intelligent/adaptive hypermedia

As adaptive hypermedia systems and web-based tutors become more sophisticated, they increasingly incorporate methods and models from the field of intelligent tutoring. Since these systems and their authoring tools are becoming more predominant, I have created a separate category for them. The functions of these systems overlap with those from the Curriculum Sequencing and Tutoring Strategies categories above (depending on whether the focus is on instruction at the macro or micro level). The level of interactivity and fidelity available to the student is low for tutors built with these authoring tools. They are HTML-based, and do not yet incorporate highly interactive features available through programming languages such as Java and Flash. Unlike systems in the other categories, these systems must manage the hyperlinks between units of content as well as the form and sequencing of the content itself. The links available to the student can be intelligently filtered, sorted, and annotated based on a student model or profile (Brusilovsky 1998). Link filtering can be based on prerequisites, cognitive load, topic appropriateness, difficulty, etc. Brusilovsky's Chapter 13 in this Volume has a more complete overview or adaptive hypermedia authoring tools.

3. HOW ARE THE PARTS OF AN ITS AUTHORED?

Having described ITS authoring tools in the concrete terms of what types of tutors they can build, I will move on to describe features of the authoring tools themselves. ITSs are often described as having four main components: the student interface, the domain model, the teaching model, and the student model. Though this categorization is not always sufficient to describe an ITS, the functionality of ITS authoring tools can best be described in terms of authoring these four components.

3.1 Authoring the interface

Interface design is the one area where traditional multimedia authoring tools excel over ITS authoring tools. This is probably because building an interface construction kit is quite time consuming. Since basic graphics authoring is a "solved problem" most ITS authoring researchers have not prioritized the effort need to build full graphics construction tools. However, the experience of our research team has indicated that customizing the tutorial's interface is a priority for authors (Murray 1998). Also, constraining the student interface to pre-defined screens and layouts severely constrains the types of tasks and interactions that an ITS can have with the student.

Three of the authoring systems, RIDES, SIMQUEST, and Eon, allow authors to construct the tutoring system's interface completely from scratch, using interface objects such as buttons, text, sliders, imported graphics, movies, and low level "drawing" objects. The interface objects in these systems are "live," in that they can be scripted to respond to user and program generated events, and their properties (color, position, etc.) can be set to depend on other values in the tutor. With the RIDES system the author can define components, sub-components, and physical connections such as wires and pipes. In the Eon system authors define graphical screen 'templates' and the system automatically creates a database for holding the template contents. For instance if a screen containing a movie, a question, and an explanation was authored, the author could use a data entry tool to easily fill in the text and movie names for dozens of these interactive screens.

Features that actively assist the author in designing an ITS interface, for example by analyzing the interface design for clarity and usability, have not yet been included in ITS authoring systems. The vast majority of authoring systems assure reasonable interface designs simply by pre-defining the student interface--i.e. by not providing interface design features at all.

Authoring the interface, though more flexible, has the negative side effects of freeing authors to design poor interfaces, and adding to the list of skills that an author must have. Building the interface for "user friendly" software can take from 50% to 90% of the entire project resources. The MetaLinks system addresses this trade-off by allowing the author to customize the layout by selecting from two menus. The first menu lists purposes of the page (e.g. glossary, explanation, chapter introduction, etc.) and second menu lists layout types (e.g. pictures in upper right; first picture above the main text with other pictures half size at the bottom, etc.). MetaLinks combines these two parameters to determine the overall layout and look and feel of the page. The REDEEM system allows users to import interactive screen created with an off-the-shelf multimedia authoring tool (ToolBook).

Systems in the Intelligent Hypermedia category offload the job of displaying the interface to the web browser. Though no interface authoring is allowed (or needed) for these systems, the layout capabilities of web browsers make it easy to generate web pages with a fair degree of adaptability with little effort.

3.2 Authoring the domain model

ITSs contain representations of curriculum knowledge, simulation models, and problem solving expertise. Authoring tools have been built for each of these domain model categories, as described below.

Models of curriculum knowledge and structures. Several authoring systems include tools for visualizing and authoring content networks (including IDE, Eon, RIDES, and CREAM-Tools). These tools help the author visualize the relationships between curriculum elements (such as topics, courses, concepts, and procedures) and allow a bird's eye view of the subject matter. Some tools are limited to strict hierarchical representations of courses, modules, lessons, topics, etc., but most allow more free-formed network representations.

Curriculum knowledge can include knowledge about the pedagogically relevant properties of topics, such as their importance and difficulty. Almost all of the authoring tools in the Curriculum Sequencing, Tutoring strategies, and Multiple Knowledge Types categories include the ability to author topic properties. Several systems (including IDE, IRIS, and Cream-Tools) provide tools for authoring instructional objectives separately from topics.

Simulations and models and of the world. The level of fidelity of a simulation is constrained by the representational formalism used for domain modeling. Among the systems we have mentioned, SimQuest has the "Deepest" modeling framework. It models continuous physical processes using systems of algebraic equations and differential equations that can interact as "functional blocks." The next most sophisticated is the RIDES system which models device component behaviour determined using event-based based methods (e.g. when the user performs some event on the interface an action is triggered) and constraint-based methods (the value of an object property is determined by a mathematical expression). Next in sophistication are state-based simulation models, as used by XAIDA and Instructional Simulator. Components have different states depicted graphically and simple branching procedures are used to determine what state a component is in. Instructional Simulator uses a PEAnet (processes, entity, activity network) representation which includes a simple rule base describing how properties and events affect elements of a simulated process. This allows authors to easily build simple cause-and effect simulations.

RIDES and SIMQUEST include sophisticated WYSIWYG tools for building models of devices and other physical phenomena. In RIDES authors combine atomic components, such as switches, levers, pipes, electronic black boxes, etc., each of which have properties (such as color, voltage, on/off state) and the ability to connect with other components (via input and output connections). Components are joined to form larger components. Rules and constraints are authored to specify how each component affects others (e.g. how a pressure meter value effects a pneumatic valve position).

While simulations in RIDES are based on device components (and properties) and their connections, simulations in SIMQUEST are based on an authored model, i.e. a set of equations. Devices and other physical phenomena are constructed in SIMQUEST using simple graphical objects, and the properties (size, location, color, etc.) of these objects are linked to variables in the model. While SIMQUEST is more cumbersome that RIDES for authoring devices with many parts, SIMQUEST can more easily model natural phenomena such as in physics and meteorology.

Models of domain expertise. Domain expertise can include several types of knowledge, including problem solving expertise, procedural skills, concepts, and facts. Authoring systems in the Curriculum Sequencing, Tutoring Strategies, and Multiple Knowledge Types categories allow authors to represent simple facts, relationships, and procedures. Facts and relationships are stored as associations (e.g. the color of X is Y, A is the capital of B). Simple procedures are stored as a sequence of steps, and some systems have the ability to author sub-procedures. Most systems that use content networks incorporate domain information into the model of curriculum structures. For example, a topic network can relate concepts to sub-concepts (with Is-a links) and procedures to sub-procedures. This type of information is both content (i.e. the student should learn the sub-steps of a maintenance procedure, and the fact that a mushroom Is-a type of fungus) and curriculum specification (since the teaching strategy may teach about siblings before parents and sub-steps before general steps).

Procedural expertise for device operation and other procedures is represented using simple script-like representations with steps, sub-steps, and limited decision branches. More complicated procedural skills and problem solving skills require the production-rule-based representation used in expert systems (or a similarly complex formalism, such as constraints). The PUPS and Demonstr8 systems facilitate the authoring of production rules (Demonstr8 uses an example-based method described later). D3 Trainer re-uses the production system rules authored using the D3 expert system shell (Training Express uses a similar method). Both of these systems provide authoring support for associating hints and explanations with each production rule. LEAP allows users to author dialog grammars, which are similar in complexity to production rules.

As is the case with other AI systems, the authoring of facts, relationships, and simple procedures is relatively straightforward. For domain expertise modeled with rules, grammars, or constraints, the authoring is much more demanding. Authoring tools for these types of representations require programming or knowledge engineering skills. (Knowledge engineering is the skill of formalizing knowledge or procedures into computationally-relevant forms.)

Domain Knowledge Types. As mentioned above, systems in the Multiple Knowledge Types category distinguish different knowledge types and have different knowledge representation schemes and different teaching strategies for each knowledge type. This structure guides and constrains authoring. The DNA system is used to author symbolic (factual), conceptual, and procedural knowledge. These knowledge types are related by "what, how, and why" links. For example, an author creating a curriculum unit for "standard deviation" is prompted to create additional content describing "how" to calculate it (procedure), "why" it is important (concept), and "what" it is used for (fact). DNA uses a single representational framework for its three types (symbolic, procedural, and conceptual): a semantic network that includes GOMS (goals, operators, methods, and selection rules) inspired link types to represent procedural and rule-like information as well as more common is-a, part-of, and causal relationships between knowledge elements.

XAIDA provides maintenance training in four areas: the physical characteristics of a device, its theory of operation, operating and maintenance procedures, and troubleshooting. Tools for the last two categories are only partially complete. Semantic networks are used to represent physical characteristics and operation/maintenance procedures; causal reasoning schemes are used to represent theory of operation, and fault trees are used to represent troubleshooting expertise. CREAM-Tools and IRIS use more elaborate systems. CREAM-Tools use different vocabularies for learned capabilities vs. behavioral objectives. It uses Gagne's five categories of learned capabilities, Bloom's six-level classification of learning objectives (further divided into 31 terms), and a large vocabulary of relationships between these elements. IRIS uses different vocabularies for the pedagogical description of domain knowledge vs. the performance specification of domain knowledge. It also uses both Gagne's and Bloom's descriptive vocabularies.

3.3 Authoring the tutoring model

Tutoring strategies specify how content is sequenced, what type of feedback to give, when and how to coach, explain, remediate, summarize, give a problem, etc. A variety of representational methods are used to model tutoring expertise, including procedures, plans, constraints, and rules. However, the vast majority of ITS authoring tools include a fixed, i.e. non-authorable, tutoring model. Eon, COCA, REDEEM, IDE, and GTE allow authoring of the pedagogical model. COCA uses a rule-based representational method, and the author uses pull-down menus to specify the right and left-hand components of IF-THEN rules. Eon uses a flowline-based graphical programming language that allows the user to author arbitrary instructional procedures. For both of these systems the flexibility comes at the price of ease of use, and no guidance is given to help the author create effective tutoring strategies. REDEEM has a fixed rule set defining the pedagogical behaviour, but authors can define their own "teaching strategies," which are settings for key pedagogical parameters such as "amount of student choice," "preference for specific (vs. general) informationfirst," and "amount of help." For example, a strategy called "Advanced learners " might have high student choice, start with general information before teachingspecific information, and help only available on error.

Plan-based systems. Several systems (including IRIS, GTE, IDE, and REDEEM) include plan-based mechanisms with multi level hierarchical representation of instructional objectives, strategies, and tasks (various other terms are used such as goals, events, and actions). For example, the IRIS framework includes three levels: cognitive processes, instructional events, and instructional actions. IDE is unique in allowing authors to specify rationales for each planning rule, so that each rule can be justified by a specific pedagogical theory. The plan rules in some systems are fixed but IDE and GTE allow authors to type in plan rules that define a hierarchy of sub-tasks. For example: "To Teach Functions => 1: Present Function, 2: Teach Linked Processes, 3: Teach Sub-Functions, 4. Present Summary." The item "Teach Linked Processes" may be further defined using another rule. The authoring and visualization tools provided for such systems are minimal, however, and the authoring task requires significant programming or knowledge engineering skills.

Multiple strategies. Some systems include multiple teaching strategies and dynamically choose the appropriate strategy based on content and user characteristics. Systems in the Multiple Knowledge Types and Simulation categories have a handful of relatively simple teaching strategies, one for each type of task or knowledge recognized by the system. For example, a different strategy would be used to teach facts, procedures, and concepts. There is no strategy authoring for these systems. The REDEEM and Eon systems allow authors to define multiple strategies and "meta-strategies" for dynamically selecting among multiple strategies.

Meta-strategies in REDEEM are easily authored. They are defined via a set of sliders that set key pedagogical parameters (such as the depth of hints, and whether prerequisites are required). REDEEM steps authors through a set of multiple-choice questions which determine the conditions under which each defined teaching strategy is used. For example, for the "Advanced Learner" strategy above the author would select the conditions (or triggers) for using this strategy, e.g. when the student is doing well, when the material was previously summarized, and when the content is not very difficult. Eon meta-strategies combine the authoring of meta-strategy triggers, as in REDEEM's meta-strategies, with parameterization values, as in

REDEEM's strategy authoring, with the added flexibility of allowing the author to define which variables appear in the sliders.[4]

Tutorial action vocabularies. Those developing systems in the Tutoring Strategies category (and many ITS "shells") have developed elaborate vocabularies for describing instructional methods. Tutoring strategies or rules are then used to determine the type of action needed at any given time. Example tutorial actions include hint, explain, remediate, summarize, practice, select-a-topic, and reflect-on-exercise. Most such systems have a layered vocabulary in which some actions expand into other actions (e.g. active-prior-knowledge expands into recall-prior-knowledge and/or use-prior-knowledge). The GTE system, for example, has several hundred items in its library of instructional tasks and methods. (See Mizoguchi et a. 1996, Van Marcke 1992, and Murray 1996b for example vocabularies, ans see Section 5.6 on ontologies.)

3.4 Authoring the student model

Almost all of the systems mentioned in this paper use "overlay" student models; i.e. topics or procedural steps are assigned a value based on student performance. XAIDA and Eon allow the author to define misconceptions as well as topics, so that the tutor can evaluate and remediate common errors. Demonstr8 seems to be the only system using a "runnable" student model (i.e. one that can be used to predict and simulate student behavior). Various AI modeling techniques have been incorporated into ITS student models, including fuzzy logic (Goodkovsky et al. 1994) and Bayesian networks (Collins et al. 1996), but none have been incorporated in to ITS authoring systems yet. Eon seems to be the only system that allows the student model to be authored, i.e. it allows the author to specify how the values of topics are calculated based on student responses and actions. A "layered overlay" student model is used, which includes overlay values at several layers: interface events, presentations, topic levels, topics, and lessons (in contrast to other systems that have only one layer for topics). The author specifies simple expressions at each level that define how the overlay values at one level are calculated based on the next lower level.

Some systems use user modelling in unique ways. With RIDES authors can create prototype student models for tutorial testing. The WEAR system contains a model of the *teachers* goals, expertise level, interests, preferences for tool settings, and teaching methods (e.g. a preference for quizzes to be given regularly to students). It uses this model to customize the authoring session by giving the teacher context-sensitive feedback during authoring.

[4] Not all of REDEEMs meta-strategy capability has been implemented and tested with users.

4. WHAT AUTHORING AND KNOWLEDGE ACQUISITION METHODS HAVE BEEN USED?

Next I will discuss general methods used by authoring systems to simplify and automate authoring and knowledge acquisition. These methods are general, in that they could be used to improve authoring for any of the four main parts of an ITS described above, and could be used in an authoring tool for any of the seven categories of authoring tools described earlier.

Authoring tool goals. Before enumerating the authoring methods used, I will summarize the overall goals that motivate these methods. Generally speaking, authoring tools have these goals, in rough order of importance or predominance:

- Decrease the effort (time, cost, and/or other resources) for making intelligent tutors;
- Decrease the skill threshold for building intelligent tutors (i.e. allow more people to take part in the design process);
- Help the designer/author articulate or organize her domain or pedagogical knowledge;
- Support (i.e. structure, recommend, or enforce) good design principles (in pedagogy, user interface, etc.);
- Enable rapid prototyping of intelligent tutor designs (i.e. allow quick design/evaluation cycles of prototype software).
- Another goal is to use the rapid prototyping capability of authoring tools to evaluate alternate instructional methods and add to our inadequate body of understanding of how to match instructional methods with learning scenarios.[5]

Scaffolding knowledge articulation with models
Embedded knowledge and default knowledge
Knowledge management
Knowledge visualization
Knowledge elicitation and work flow management
Knowledge and design validation
Knowledge re-use
Automated knowledge creation

Authoring tools achieve these goals using a number of methods or features. Most of the methods address several of the above goals (for example, a feature that helps the designer articulate a teaching strategy will also decrease the effort and skill

[5] Yet another possible goal is helping the author *learn* something about pedagogy, instructional design, or knowledge representation, and thus become a better author as they use the system.

threshold of building a tutor). I describe eight methods in detail, listed in the box below, and then briefly mention several other methods or capabilities seen in authoring tools.

4.1. Scaffolding knowledge articulation with models

ITS Authoring is both a design process and a process of knowledge articulation. The most significant method that authoring tools employ to allow non-programmers to build tutors is to scaffold the task by incorporating a particular model or framework. Simplification by restricting the universe of what can be built is a somewhat obvious method since that is what all software applications do (e.g. an electronic address book is easier to use than a database application). Providing authors with clear frameworks or templates helps them organize and structure the authored information. Though obvious, it is worth highlighting because one of the major differences between authoring tools is the degree to which their models constrain the product (see the later Section on design tradeoffs).

A significant part of authoring an ITS (or any instructional system) is the systematic decomposition of the subject matter into a set of related elements (usually a hierarchy). Each authoring system provides tools or cues which assist the author in this (usually top-down) process of breaking down and elaborating the content to the necessary level of detail according to a particular model of instructional elements and their relationships. Such "content analysis" can be distinguished from "task analysis" and "cognitive task analysis." Content analysis tools help authors decompose their declarative knowledge of facts, concepts, procedures, and principles. The DNA Chapter (6) describes a system for doing content analysis of factual, conceptual, and (simple) procedural knowledge. Task analysis tools help authors articulate procedures and action sequences. For example tools in Demonstr8, Instructional Simulator, and RIDES allow domain experts to demonstrate a procedure which is then generalized. Cognitive task analysis tools help experts articulate problem solving and other thinking skills. For performance-oriented tutors, the most significant task is in articulating and representing expert performance. Ritter et al. (in this volume) say that "good task analysis is essential in creating an effective learning system [yet] little effort has gone into authoring systems for task analysis." Ritter et al. describe efforts to do cognitive task analysis for model tracing tutors.

Some systems, particularly in the adaptive hypermedia category, minimize the use of authoring tools by allowing authors to enter content using a standard word processing application and a "mark-up" system or language. Characteristics of content such as its place in a content hierarchy, its prerequisites, its difficulty, etc. can be typed into a word processor and tagged in one of two ways (see Brusilovsky in this Volume). The first method is to mark the text (in Microsoft Word) using styles, as is done in the InterBook project. The second method is to mark the text with special symbols (e.g. "<<*difficulty-topic*>>") as is done in the AHA system. This method has the benefit of being simple and no new tools have to be developed or used. It has disadvantages including the need to memorize the mark-up language and being susceptible to typographical errors in the mark-up.

4.2. Embedded knowledge and default knowledge

One way to make authoring easier and more powerful is to embed knowledge right into the system. "Embedded knowledge" means knowledge that is pre-wired and non-authorable. This knowledge can be passive or active. Passive knowledge is knowledge that is implied as part of the structure or constraints imposed by an authoring system. For instance the systems in the Multiple Knowledge Types category have instructional design principles embedded into their structure (e.g. that concepts have necessary and sufficient attributes). Authoring systems that highly constrain ITS design, such as those in the Special Purpose category, contain substantial passive embedded knowledge. (And see the discussion in Section 4.7 on knowledge re-use, which is similar to embedded and default knowledge.)

Active embedded knowledge is runnable and produces some result. For example, REDEEM contains a sophisticated rule-based instructional strategy that the author can effect through the use of strategy parameters, but otherwise can not alter. XAIDA uses embedded expertise to generate 19 different types of practice questions for procedural knowledge. The author can specify when to use each type for a particular instructional module.

Special Purpose systems and several others (including XAIDA and REDEEM) make use of reasonable default values that allow the author to postpone entering some information but still be able to test-run the tutor. Default values in templates can also provide examples of the type of information to be entered, and thus be informative as well as functional. In REDEEM even though the author can modify the teaching strategy, the system has a robust default tutoring method so that a tutor can be authored and run without ever defining teaching strategies. IDLE-Tool scaffolds authoring using a "guided case adaptation" method. Associated with every data-entry template screen is sample input from a prototypical IDLE tutor. Thus authoring is then more like adapting a similar case to fit the needs of a new tutor than starting from scratch. The system, as a special purpose authoring tool, also contains a specific model of the investigate-and-decide task (for example, the investigate task is composed of parts: decide on what sample to take, obtain the sample, determine the analysis method, do the test, interpret the results) and a detailed domain-specific taxonomy of types of samples, tests, and results to help guide the authoring process.

4.3. Knowledge management

ITSs are elaborate systems and authoring them involves managing a large amount of complex information. A number of common user interface techniques are used by authoring systems to assist with knowledge management and organization. Simplifying input through the use of templates, data entry forms, and pop-up menus is quite common. Whenever the range of possible input values can be limited to a finite set, tools should be provided to allow authors to choose rather than type.

ITSs are particularly difficult to author because of the many diverse and interconnected types of information they contain. A primary tenet of ITS design is to have separate representations of content (what to teach) and tutoring strategy (how to teach it), but these can not be made completely independent. Even if a system successfully encapsulates certain aspects in independent modules there are still complex conceptual relationships that the author must be aware of. For example, the structure of the student model depends on the structure of the domain model; the form of the teaching strategies depends on the structure of the domain model; the actions in the teaching strategies depend on the form of the tutor's student interface. Navigation aides that let authors move between various related pieces of information and different representations of the same information have been implemented. For example, if different parts of a tutoring strategy refer to topics, student model rules, and interface components, the author should be able to click on the associated item in the strategy authoring tool and be brought directly to the topic authoring tool, student model authoring tool, or interface authoring tool(this is done in Eon and CREAM-Tools).

Tools that allow authors to zoom in and out between the details and the big picture can help manage large information spaces. Object browsers, which allow authors to scroll through all of the objects of one type and inspect properties and/or relationships with other objects are available in several of the systems (as exist in RIDES, D3 Trainer, IRIS, Eon, SimQuest, and CREAM-Tools). Tools for managing evolving software components and versions are important to most off-the-shelf software engineering design environments, but these features have not yet been incorporated into ITS authoring tools.

4.4. Knowledge visualization

Perhaps the most powerful way to help authors understand and comprehend the large amounts of complexly interconnected knowledge is with powerful visualization tools. Unfortunately, building the user interface is usually far and away the most labor-intensive part of programming any interactive software. The level of interactivity and visualization in ITS authoring tools is still quite primitive as compared with off-the-shelf productivity software (with the exception of RIDES and Eon, which have fairly sophisticated interface design tools).

LAT has tools that help authors visualize conversational grammars. LAT's grammars, which represent how a customer contact employee should respond to service calls of various types, are composed of individual scripts that define the possible actions (things to say to a customer) and decisions points of a thematic unit in the conversation (such as "processing a discounted sales order"). Each script can invoke other scripts. A relatively simple set of scripts can result in a large and complex set of possible conversational scenarios. LAT provides visualization tools that allow the author to see both the static structure of the scripts and the run-time dynamics that simulate possible tutorial scenarios. LAT designers also stress the importance of providing multiple views of authored content.

Topic or curriculum network tools are the most common knowledge visualization tools in ITS authoring. Little currently exists to allow authors to visualize teaching strategies. Eon uses a highly visual flow-line metaphor for authoring tutoring strategies. Strategy authoring in REDEEM consists of setting parameters, so strategies can be easily visualized with a screen of sliders and radio buttons.

4.5. Knowledge elicitation and work-flow management

Knowledge acquisition is widely acknowledged as the limiting factor or bottleneck in building AI systems. A number of techniques have been used for extracting knowledge from experts, most of which are "manual" methods that involve a knowledge engineer interviewing or observing the expert (Hoffman 1987). Software tools have been developed to scaffold or automate some knowledge acquisition (Boose 1988; Shaw & Gaines 1986). Many of the automated techniques use contrived tasks such as sorting or ranking to find conceptual dependencies, logical entailments, or other patterns in the data, which are not generally applicable to acquiring domain or teaching knowledge for ITSs. However, the method of interactive elicitation of knowledge to fill in a pre-structured knowledge base *has* been used in ITS authoring, as described below. As mentioned previously, using an authoring tool to build an ITS involves both knowledge acquisition and design processes. Authors need to be supported not only in filling in a knowledge base, but in the overall ITS design *process*. This includes designing the interface, domain model, and teaching strategies; conceptualizing the interaction among *several* knowledge bases; and the iterative process of user testing and refinement. Interactive prompts and dialogs can help with work-flow management (or "performance support"), as well as knowledge elicitation.

ISD-Expert (an early precursor to Instructional Simulator) led the author through a sometimes excruciatingly long dialog to create an entire course in a top-down manner. The dialog started with general questions such as "what is the title of the module?" and "what is the average motivation for the target audience?" Then a series of questions fleshed out the content and behavioral objectives in a top down fashion, and included questions for each content unit such as "which of the following describes what the student will learn: a. What is it? b. How to do it; c. How does it work?" The potential benefit of this system was that, since the authoring involves responding to specific prompts, the author did not have to make any high-level design decisions (only low level and concrete decisions). But there were two serious drawbacks to the system. First, authors felt too constrained by the fixed sequence of data entry. The design of complex systems usually requires a mixture of top down and bottom up design (i.e. "opportunistic" design). The second problem with such highly constrained dialogs is that the more a system constrains data entry the more essential it is that the underlying model be accurate and complete. But instructional theories are neither entirely accurate nor entirely complete, and each author may have her own style or preferences. It is often better for a system to offer suggestions but allow the author to override the default design decisions. REDEEM's meta-strategy authoring tool uses an automated knowledge elicitation dialog that is less restrictive. First, the dialog is limited to defining the meta-strategy parameters for a particular strategy, and the author chooses when to initiate this dialog. Second, the dialog consists of a series of screens rather than a series of text-based questions. Each screen has a data entry form for several parameters, and includes default choices for these parameters. REDEEM includes an agenda mechanism that keeps track of incomplete authoring tasks and prompts the user to complete them.

DNA uses a semi-structured interactive dialog to elicit domain knowledge from a SME (subject matter expert) (a process called cognitive task analysis). As mentioned earlier, DNA can elicit curriculum elements of three interrelated types: factual ("symbolic knowledge"), procedural, and conceptual. Thus, as the SME is defining a curriculum element he is prompted for the existence of related facts, concepts, and procedures. Questions such as "What is the definition of [a term used in a procedure]" and "Why is [some fact] important" prompt the SME to continue to flesh out the content to include all three knowledge types. The process is called "semi-structured" elicitation because the questions are presented as options on the design screen, allowing the author to choose which one to answer next (and which ones to ignore).

Expert-CML is designed to scaffold the instructional design process by providing advice that is sufficient for the novice user but does not hinder the expert user. A rule-base of over 100 rules provides advice in two forms. The first suggests what to do next (similar to DNA above) and the second points out possible errors in the tutor (as discussed in the next Section). The advice is shown in a status bar at the bottom of the screen, where the author can use it or choose to ignore it. Also, the author can turn off certain instructional design rules to permanently override the system's suggestions.

4.6. Knowledge and design validation and verification

Each authoring system makes different compromises along the spectrum of free-form design to constrained design. More open-ended systems allow for more flexibility in both the form of the content and the sequence of steps taken to design the tutor. However, the more flexibility given to the author the higher the probability that they will enter something inconsistent, inaccurate, or something that is at odds with the principles of good instructional design. Special purpose authoring tools can ensure content validity by tightly constraining what is entered. One way to allow flexible authoring while maintaining quality is to allow the author to enter what she wants in the way that she wants, but to include mechanisms that check the authored information for accuracy, consistency, completeness, and effectiveness. As mentioned above, the Expert-CML system includes an expert system that offers this type of content evaluation. For example, rules exist that inform the author of the following: when the author's estimate of the time to complete a lesson is at odds with the accumulated times given for the component parts; when a summary or introduction might be needed to break up a long sequence of new material; when the objectives of a lesson are not adequately covered by the lesson's instructional components; and when a lesson's general cognitive level (according to Bloom's Taxonomy) does not match with the cognitive levels of the lesson's objectives.

The DNA system deals with content accuracy and completeness by facilitating the process of having several SMEs review the knowledge structures authored by the primary author/SME. The reviewing SMEs can edit and comment on the original knowledge base. The ECSAIWeb system allows authors to simulate learning paths to check the validity of the curriculum model. D3-Trainer is the only system that validates authored information by matching it with a case base of previously authored examples.

Inconsistencies in authored information that manifest at run time might be invisible during authoring. By allowing teachers to match teaching strategies to prototypical student attributes, REDEEM supports teachers in understanding the consequences of the teaching strategies. RIDES includes a consistency checker and an integrated debugger/stepper that allows authors to run and debug their device simulations. It also lets authors create prototype student models for validation testing.

4.7. Knowledge re-use

Authoring tools have the potential to increase the efficiency of building ITSs through re-use of common elements. To date most ITS authoring tools have not been used to build enough ITSs to experience this benefit. Realizing re-use would require a resource library structure, where authored topics, activities, strategies, interface components, and/or domain knowledge could be stored independently from a tutor, and loaded from this library into any tutor. Sparks et al (1999--- the LAT system) discuss the implications of object libraries for work reduction, reduced maintenance effort; and increased consistency among tutors. However, they note that supporting re-use may have considerable costs in terms of system complexity and ease of use.

REDEEM is built to take advantage of courseware libraries. The content and interactive screens of a REDEEM ITS are not authored using REDEEM, but are authored in either html or. ToolBook, an off-the-shelf multimedia authoring tool. This authored content is exported to a library and from there it is imported by REDEEM. Component libraries in SIMQUEST included reusable interface components as well as reusable content objects. D3 Trainer keeps a library of actual cases for re-use, and SimQuest has a library of content from past authors (also see Section 5.7 on interoperability and re-use).

4.8. Automated knowledge creation

Some ITS authoring systems infer or create new knowledge or information from scratch, saving the author from having to derive, articulate, and enter this information. RIDES and Demonstr8 use example-based programming techniques to infer general procedures from specific examples given by the author. RIDES creates a device's operational procedure by recording the author's actions as he uses the device simulation to illustrate the procedure. Demonstr8's method, which generalizes the elements of the authors actions to produce expert system production rules, is more powerful but potentially has more limited application. The DIAG system infers a large body of device fault diagnosis information from a relatively small number of failure consequences it captures by analyzing the device model prepared by the author.

Systems in the Device Simulation and Expert System categories are sophisticated enough to generate new problems and their solutions from general principles or rules, thus saving the author from having to enter every problem and its solution.

4.9 General Authoring Features

Though the authoring systems described in this paper have a variety of features, a number of features have emerged as being generally important to robust usability, as described below.

1. WYSIWIG editing and rapid testing. Authors should be able to easily see and test both the static visual elements of the tutor and the dynamic run-time behavior of a tutor. Easy movement back and forth from editing to test-run modes facilitates rapid prototyping.

2. Flexible, opportunistic design. ITS authoring tools should be designed to work best for those who have had some training. Features that make it easy for a first-time user to get to work should not get in the way of serious longer term use. Tools should allow for a mixture of top down (starting with the abstract curriculum structure) and bottom up (starting with specific screens and content) authoring for different design styles.

3. Visual reification. The conceptual and structural elements of a representational formalism should be portrayed graphically with high visual fidelity if ITS Authoring systems are to be used by non-programmers. Such user interfaces relieve working memory load by reifying the underlying structures, and assist long term memory by providing reminders of this structure. Also, multiple views (visual perspectives) of information are often needed.

4. Content modularity and re-usability. Instructional content should be represented and authored modularly so that it can be used for multiple instructional purposes. Include library structures for saving reusable components. Provide productivity tools that capitalize on repetitive or template-like content. Provide tools that make it easy to browse, search for, and reference content objects.

5. Customization, extensibility, and scriptabilty. A tool cannot anticipate everything a designer will want. All modern design and authoring software provides for extensibility and scripting. Scripting allows authors with moderate skills to create "generative" ITSs that construct problems, explanation, hint, etc. on the fly.

6. Undo! Mundane features such as Undo, Copy/Paste, and Find can be extremely time consuming to build into complex design software such as authoring tools, yet such features are essential components of all robust usable software.

7. Administrative features. Only a few of the authoring tools (including RIDES and SimQuest) have administrative facilities for such things as class rostering, grade analysis and statistics, and generating progress reports, such features may be essential to the eventual adoption of ITSs into mainstream education. CREAM Tools is the only system so far to support collaborative authoring with rights management and access security features.

To the above list I will add the following which, though not as generally accepted, I believe to be important for building authoring tools that are both usable and powerful (these items are discussed in more detail in Chapter 15 Section 5.4):

8. Include customizable representational formalisms. An authoring system will be based on some underlying representational formalism, and any such formalism will satisfy the needs of authoring some types of tutors yet not be appropriate for authoring other tutors. To achieve greater flexibility, authoring tools should include the ability to customize the representational formalism (see Section 5.6 on ontologies and meta-authoring).

9. Anchor usability on familiar authoring paradigms, and facilitate evolution to more powerful paradigms. For those used to building traditional computer-based instruction, building intelligent tutors requires a conceptual shift from "story board" representations of content to more modular knowledge based representations (Murray 1996). It is useful to have some ITS authoring tools have a look and feel similar to common off-the-shelf CAI authoring tools, and to provide features which allow a smooth transition from traditional authoring paradigms to authoring more powerful intelligent tutors.

10. Facilitate design at a pedagogically relevant level of abstraction. Provide tools which allow subject matter experts to author using primitives at the "pedagogical level" as well as the media level (Murray 1996). Media level primitive are "graphic," "button," mouse click, etc. Objects at the pedagogical level have instructional meaning, for example "hint," "explanation," "topic," "prerequisite," and "mastered."

The commercially available and heavily used systems tend to have more of these features. For example, RIDES has multiple levels of undo/redo, copy/pasting, Find, a consistency checker, and an integrated debugger that lets authors step through a simulation execution to evaluate run time bugs. It also includes an instructor administrator module. SimQuest has an online help manual, a context sensitive advice tool (a hypertext on-line manual), authoring Wizards (step by step procedural help), a content re-use library, a scripting language, and administrative reporting features.

5. HOW ARE AUTHORING SYSTEMS BUILT? -- DESIGN PRINCIPLES

In the previous Sections I have characterized the range of tools and methods used by ITS authoring systems. The wide variation among these systems should be evident to the reader. The differences are in part due to different theories and models of knowledge and instruction, but in large part can be attributed to different priorities and design tradeoffs. If one were in the (unrealistic) position of choosing among all of these authoring tools for a specific job, or in the position of intending to design a new ITS authoring tool from scratch, how would one decide which of the tools, methods, and features were most important? Trying to build an authoring tool that incorporates the "the whole enchilada" is practically impossible, not only because of the prohibitive cost and complexity, but because the design decisions these systems are based on are at odds with each other. For instance, increasing the flexibility/power of a system comes at the cost of usability.

5.1 A space of design tradeoffs

Figure 1 illustrates the space of factors leading to design tradeoffs for ITS authoring systems, including breadth, depth, learnability, productivity, fidelity, and cost.[6]

		Domain Model	Tutoring Strategy	Student Model	Learning Envrnmnt.
Power/ Flexibility	Breadth				
	Depth				
Usability	Learnability				
	Productivity				
	Fidelity				
	Cost				

[The design space has 24 (6x4) independent dimensions or axes.]

Figure 1: ITS Authoring Tool Design Tradeoffs

Overall, Power/flexibility and Usability are usually at odds with each other, since simplicity tends to correlate with usability. As shown in the Table, power/flexibility has two aspects: breadth (scope) and depth of knowledge. Breadth is how general the framework is for building tutors for diverse subject areas and instructional approaches. Knowledge depth is the depth to which a system can reason about and teach the knowledge, and the depth to which it can infer a student's knowledge and

[6] The important metric of "instructional effectiveness" is not included in our metrics, because this depends very much on how the authoring tool is used to produce a tutor. However it is also true that the design of an authoring tool can have a tremendous effect on the instructional effectiveness of the systems it is used to build.

respond accordingly. Breadth and Depth are often at odds, because one must often limit the generality of a system to be able to represent deep causal knowledge. Usability also has two aspects: learnability and productivity. Learnability is how easy a system is to learn how to use. Productivity is how quickly a trained user can enter information and produce a tutoring system. Learnability and productivity are often at odds, since a system that is designed to be picked up quickly by novices may not provide the powerful features that experienced users need to efficiently produce large systems. Fidelity is the degree to which a tutor perceptually and operationally matches its target domain.[7] A 3-D immersive environment has more visual fidelity than a 2D simulation. A learning environments that allows the student to directly practice a task has more fidelity than one that merely describes and asks questions about the task. Fidelity can be closely related to depth, since deeper knowledge facilitates more realistic interactions, but it is possible for a system to have shallow knowledge and high fidelity. Cost refers to the amount of resources needed to build the authoring system (for this discussion the availability of personnel, expertise, and time are included in this category). I will not say more about cost except to note the obvious fact that the resources available to a software project greatly effect design decisions. Figure 1 illustrates that all of the design factors mentioned above come into play for each ITS component separately---domain model, tutoring strategy, student model, and learning environment. For example, an authoring system can have a highly usable tool for authoring shallow student models and a not so usable tool for authoring deep teaching strategies. Additional evaluation criterion have been proposed. Ainsworth (Chapter 8 in this Volume) adds: 1) the ability of an authoring tool to successfully represent an expert's teaching strategies (and, by extension, domain knowledge), 2) the effectiveness of the resulting ITSs, and 3) the ability of the tools (through the authoring process) to help educators reflect on, share, and expand their own knowledge of how to teach in their subject area.

The design space for ITS authoring tools is indeed huge. There is rough consensus on the nature of the tradeoffs involved, but not on how to balance those tradeoffs to produce the most effective and usable authoring systems. Questions that must be addressed include:

- How much should the author be constrained to a particular (favored) pedagogical model?
- Who are the prototypical authors who will use the system?
- What types of knowledge and skills should be modeled by the system?
- What is the source of the teaching and domain expertise?

These questions must be answered to make the design decisions and compromises involved in building an ITS authoring tool. Each authoring tool, no matter how general, will embody particular assumptions and models and thus be more

[7] Fidelity, depth, and cost are qualities of the *tutor* that an authoring system produces, while all of the other design factors are about the authoring system itself.

appropriate for building certain types of ITSs than others. Before designing an ITS authoring tool it is best to be as explicit as possible about the nature of the ITSs to be built. I discuss some of the design decisions in more detail in the following Sections.

5.2 General or special purpose authoring systems?

One of the most active areas of disagreement among the authoring tool research community concerns the appropriate degree of generality for an authoring system. For example, the LAT system is designed to only produce tutors to train customer service personnel how to respond to customer inquiries. It is a general tool in that it can be used to create a customer service ITS for a variety of products. An authoring tool that specializes in producing a very particular type of ITS has many benefits. In principle, by severely constraining the universe of what can be designed, an authoring system can have higher usability, higher fidelity, more depth, and be more efficient. More constrained systems make it less likely for authors to enter incorrect, inconsistent, or pedagogically poor content or teaching methods.

IDLE-Tool is designed to build tutors for "Investigate and Decide" learning environments. Bell (1998, and Jona & Kass 1997) proposes a suite of special purpose authoring tools, each for learning a particular type of complex task, such as Investigate and Decide, Run an Organization, and Evidence Based Reporting. In Investigate and Decide environments learners are supported in gathering data about a realistic case or scenario (such as a medical diagnosis), interpreting the data, taking some action based on their hypothesis (such as prescribing medication), and receiving feedback about their action (such as whether it helped the patient). The IDLE authoring tool provides the author with a fixed template for representing the outcomes, feedback, reference informant, and graphics. The task structure and the pedagogy for teaching about the task are predefined based on (presumably) sound instructional principles. A formative study found the tool to be too constraining. Its fill-in-the blanks style did not support the kind of "big picture" view of the instructional scenario that can be important in designing instructional scenarios. Bell and others are continuing to work toward authoring tools that tightly constrain the authoring process but include adequate flexibility.

There are several issues related to tool generality. First, although "Investigate and Decide" seems to be a very general type of task, the authoring system embodies a very specific interpretation of that task and a fixed pedagogical model. Assuming there are a large number of educators (mostly science and technology educators) who would be interested in building (or using) an Investigate and Decide tutor for some topic, it is not clear how many of these would find IDLE-Tool's specific model agreeable. Bell's formative study will lead to more flexible and customizable systems, such as IMAP, which will inevitably require more skilled authors. We can't yet say where the most appropriate generality vs. usability line should be drawn for these types of systems, but every new data point will help.

The LAT system paints a slightly clearer picture. Its conversational grammar approach to customer service training seems more certain to be widely usable than

IDLE-Tool's template-based approach to Investigate and Decide types of inquiry learning. This is partly because LAT has a deeper and more general representation of the task. But it is also because the customer contact task is more easily defined. I think that special purpose authoring tools will find much wider appeal in "training" applications than "educational" applications. With training applications there is wider agreement on the nature of the task and what the behavioral objectives are, but in educational applications (which tend to address higher order thinking and skills) there is much less agreement over exactly what and how to teach. The authoring tools in the device simulation category are a good case in point. The task of training someone how to operate and understand (at a basic level) how a piece of machinery works is very general and there is a fair degree of agreement concerning effective training approaches. Consequently, the RIDES system has seen the widest application of any of the authoring tools.

Research with XAIDA is pushing the generality vs. usability issue in interesting directions. Its target user is much less sophisticated than RIDES', which also teaches about device operation and maintenance. This has lead to a number of design simplifications aimed at usability. For instance, RIDES represents component interdependencies in terms of constraints and event results, while XAIDA uses a less powerful but more felicitous cause-effect framework. Compared to RIDES and the systems in the Domain Expert System category, XAIDA has a relatively shallow knowledge representation (but it is still runnable or executable knowledge---i.e. the system has limited "understanding" of what it is teaching). If looked at separately each of XAIDA's four knowledge types (physical characteristics, theory of operation, operating and maintenance procedures, and troubleshooting) has a relatively shallow representation. Yet its incorporation of four different knowledge types adds a degree of power and generality. Thus XAIDA has been used to prototype tutors in a number of domains, including several that are quite distant from the originally intended domain of equipment operation and maintenance. These domains include algebra, medicine, computer literacy, and biology.

DNA and Instructional Trainer rely on a similar combination of relatively shallow knowledge representation and distinct knowledge types to achieve a high level of generality. The preliminary successes of these systems indicate that the correct level of abstraction for distinguishing different types of authoring tools may be more at the cognitive level of knowledge types (such as concepts vs. procedures) than at the more surface level of task types (such as investigate vs. advise).[8]

Up to this point I have shown how usability is at odds with the power/flexibility of an authoring tool. The only method mentioned thus far that results in both powerful and usable authoring tools is to limit them to particular domains or knowledge types. Another method, that of creating "meta-authoring" environments, is described in a Section 5.6.

[8] It is also possible that an appropriate level of abstraction will similar to the "generic tasks" proposed in the context of expert systems research (Chandrasekaran, B. 1986).

5.3 Who are the authors? --an authoring skills analysis

The main goal of an ITS authoring system is to make the process of building an ITS easier. This ease translates into reduced cost and a decrease in the skill threshold for potential authors. The design specifications of a piece of software are highly dependent on the assumptions made about the intended prototypical user. In the case of ITS authoring tools we must ask:

- What is the assumed skill level in: knowledge engineering?
- In multimedia creation and interface design?
- In instructional design and instructional theories?
- In testing or evaluating educational software?
- In the particular subject matter domain of the tutor?
- How much time does the user have available for training?
- For design and development of the tutor?
- How well does the author know the characteristics of the target student audience?

As this list of questions implies, it will usually require a team of people with different knowledge and skills to author an ITS. But how fast and easy can authoring an ITS be? Of course, in part, the answer to this question depends on what is meant by "an ITS"-- some fairly simple instructional systems have been built that still qualify as adaptive or intelligent. It also depends on how similar are the ITSs that we want to build with the tool, as was discussed in the Tradeoffs Section. Another way to phrase the overall question is: how much of the design process can be scaffolded or automated? If we identify hard limitations in the answer to this question, then this will imply constraints on the amount of a) skill, and b) time required to author an ITS. Below I use a analysis of authoring task complexity to help address this issue. Four levels of task complexity are defined 1) templates, 2) relationships, 3) modelling, and 4) behaviors. For now we ignore tasks that clearly require the skills of a software designer, such as building a plug-in learning environment interface or creating a complex algorithm needed in student model diagnosis.

1. **Templates and forms**. The easiest types of authoring task are filling in template information, selecting items from pre-defined lists, and answering prompted questions. Aspects of an authoring tool that clearly reify the main objects of an ITS and their editable properties through fill-in-the-blanks forms could be accessible to designers and teachers at all skill levels.
2. **Defining object relationships and structures**. Though this task seems easy, defining relationships between objects is a significant conceptual jump over specifying object properties in templates. Some types of relationships are straightforward to create, such as connecting an explanation object with a graphic object, or specifying that the "engine" component is "a part of" an "automobile"

component. But creating object links at more abstract levels is more difficult. The primary example of this is the creation of the topic network. It may be easy for most users to create a concept network that illustrates their informal ideas about how topics are related. But to design a good topic network, one that expresses the important pedagogical relationships in a domain and, when traversed according to some teaching strategy, will result in reasonable and expected tutorial behavior, is difficult and subtle. One inevitably runs into questions about subtle distinctions such as whether one topic is a part-of, a-kind-of, or a prerequisite-of another topic. Questions such as the following arise: are the sub-topics of a topic's prerequisites also the topic's prerequisites? Reasoning through these situations often requires the skills attributed to knowledge engineers. The complexity of the task is compounded by content-strategy interactions, as described in Chapter 15 Section 6. Also, as the complexity of a system of relationships increases, the task looks more like modelling. Software designers have learned that building robust models requires such techniques as data and procedural abstraction, encapsulation, and indirection--concepts they are difficult for many non-programmers to exploit.

3. **Modelling**. I believe that it is not possible to author an ITS without considering the big picture. This includes conceptualizing the intended audience and their needs; reconceputalizing instruction so that it can be delivered flexibly for each student; and decomposing content in a way that maintains coherence and consistence when it is dynamically reconstructed for a student---i.e. ITS authoring will usually involve modeling. It has been said that building explicit models of expertise is at the heart of AI work (Clancey 1986). Questions about building explicit models are also at the heart of ITS authoring. ITSs are complex systems containing embedded models of several types (roughly characterized as domain, teaching, and student models). Understanding ITS authoring requires a conceptual separation of content from instructional method--a reconceptualization of content as flexible and modular. This is very different from designing the screens that a student will see and enumerating the possible paths a student can take, as is done in designing traditional educational software. Building an explicit model of anything is not an easy task, and requires analysis, synthesis, and abstraction skills along with a healthy dose of creativity. Even though an authoring tool might be able to reduce the process to a long sequence of simple atomic steps done in isolation, some degree of holistic understanding and abstract thinking will eventually have to come into play. Authoring tools can significantly decrease the cognitive load involved in various design steps, but it is difficult to reduce the entire design task to low-level decisions.

4. **Defining behavior**. ITSs are user-oriented software systems that behave. Designing good ones involves iterative test trials to make sure they work as intended. Testing software implies that it will be fixed or debugged. This means that the designer has to understand the relationship between non-optimal runtime behavior and information inside the system. Defining and debugging some behaviors is relatively straightforward, for example: "when this button is pushed highlight the hint text," and "if the student gets over 80% of the answers correct, assign mastery to the topic." However, defining control structures, rules, or algorithms proves to be more difficult, even when authoring tools are provided to

make the task easier and at a higher level of abstraction (as is done in LAT and Eon). When the task includes specifying looping/repeating or conditional structures (including IF/THEN rules) the complexity increases dramatically. This is not because it is so difficult for users to understand the meaning and local effects of these structures, but because the emergent global behavior of control structures is complex. No matter how easy we make it, such tasks are essentially "programming tasks" and the skills associated with programming are needed. The main difficulty comes when, inevitably, the system behaves in some way that does not meet the author's expectations, either do to a bug in the algorithm or an unforeseen consequence of the interaction of parts. The task then is one of diagnosing and debugging an algorithm, which takes a programmer's skill. This level of programming skill is usually part of the skill set of knowledge engineers, who may not be skilled enough to write, say, a Java application, but do understand the basics of computer data structures an algorithms.

As mentioned, most ITS authoring will require a team effort. Graphics and multimedia skills will be needed if the multimedia resources are not already available. Interface design skills are not necessary for most authoring tools, because they do not allow the user to design the interface from scratch. As discussed in the XAIDA Chapter (2) and the DNA Chapter (6), the standard model for creating training materials in industry assumes separate roles for subject matter expert and instructional designer. But to simplify our discussion we can temporarily assume that there is one instructor on the team who has both of these skill sets. We will also temporarily ignore the skills needed to formatively evaluate the ITS to ensure that it works as expected. Chapter 15 Section 4.6 describes how the following tasks are particularly difficult for non-knowledge engineers: curriculum representation, strategy representation, ontology design, and student model definition. We can ignore the last two of these as they apply only to very high level authoring situations. Thus we can narrow down the discussion to one primary question: how much knowledge engineering skill is needed? I.E. under what situations can a "regular" instructor or teacher do the bulk of the work in creating the content of an ITS, and when is a trained knowledge engineer needed on the team? In terms of the task complexity levels defined above, we can say the following.

Tasks in level 1 are accessible to most teachers and "laypersons" who are comfortable using computer applications. Tasks at level 2 are already moving beyond what can be expected of most instructors without substantial training, and tasks at levels 3 and 4 require significant training or help from a knowledge engineer. When authoring tools have been empirically evaluated with many users the lessons learned indicate that completing tasks levels above level 1 require significant (initially unexpected) levels of training and skill. Van Jolingen (in this Volume) notes "though SimQuest can take away much of the burden of the authoring process, the conceptual part of authoring in itself requires considerable training for the author [and] training and support are necessary." Ainsworth et al. (this volume) report that in evaluations of REDEEM teachers had the most difficulty in decomposing the material into modular units. Though the benefits of ITSs come in large part by supporting a knowledge-based, as opposed to a storyboard metaphor for content (see Chapter 15 Section 2.2), REDEEM authors tended to create story-board-like tutors. Ainsworth et al. note that the studies have contributed to our understanding of how the average teacher conceptualizes content by highlighting the important role that narrative plays for them. In numerous evaluations of XAIDA researchers noted a reluctance of some trainers to reconceptualize their model of instruction from a linear lesson plan to the more modular one used in the knowledge-based approach. In formative evaluations of the LAT system researchers concluded that they overestimated the level of experience that would be achieved by typical authors. Findings compatible with all of the above studies were found in evaluations of SimQuest and IDLE-Tool. Until an authoring system has been evaluated or used by many users who are not affiliated with the originating research team we should be cautious about claims for its usability.

Some systems, such as XAIDA, and REDEEM, are aimed at authors with very little training (on the order of several hours), so that any instructor could, theoretically, built an ITS. This level of skill is on the order of an intermediate level user of a word processor or spreadsheet program. Other systems, such as RIDES and Eon, assume that the author will be more skilled. My own belief is that, in general, ITSs are complex systems and we should expect authors to have a reasonable degree of training in how to author them, corresponding with a skill level on the order of database applications, CAD software, 3-D modeling, or scripting in Excel or Lingo. This does not raise the bar unrealistically high, since this level of knowledge and creative/analytical skill is used for many common jobs. There are many thousands of people with proficient levels of these skills, as compared to the small number who know how to program an ITS from scratch, so having ITS authoring tools aimed at this level of usability is a substantial improvement. Each company or school could have one person trained in using an ITS authoring tool. Fortunately, it is often reasonable to expect an untrained SME or teacher to enter the bulk of the authored information, and then hand the system over to a more highly trained person for completion. In the case of the XAIDA project and the D3 Trainer project, the goal was to allow SMEs to create tutors by authoring domain knowledge (in XAIDA by describing physical artefacts and demonstrating how they operate, and in D3 Trainer by defining expert system rules). The instructional knowledge is contained within these systems, so instructional designers and teachers may not be needed in the authoring process.

But this design picture is not shared by all. Several Chapters in this volume show evidence that regular instructors can create tutors with very little training (see XAIDA, REDEEM, and Instructional Simulator). Such "easy authoring" seems possible in several situations: 1) when the resulting tutor and content are very simple; 2) for special purpose authoring systems that are highly tailored a particular domain and teaching strategy, and 3) for situations in which the author is essentially customizing an existing tutor, rather than designing one from scratch (and see the discussion of meta-authoring in Section 5.6).

Most of the highly usable authoring systems simplify authoring by providing a fixed and relatively uncomplicated model of the domain that can be authored through templates. We will look at the most usable systems: XAIDA, DNA, Instructional Simulator, IDLE-Tool, and REDEEM. XAIDA, IDLE-Tool, and

DNA contain fairly simple (instructor-friendly) domain models and fixed (non-authoreable) teaching strategies. In XAIDA and Instructional Simulator the domain is composed of parts, their name and location, their sub-components, etc. In DNA the content is described as answering what (facts), how (procedures), and why (concepts and principles) questions. In IDLE-Tool the learning task is divided in to fixed steps: obtain data, analyze data, and interpret results. In all of these cases the system has a fixed teaching strategy. REDEEM takes a different tact. It has essentially no domain model and has a CBT-like open yet shallow content model--just multimedia pages as in a book. But it allows authors to modify the tutoring strategy. It simplifies strategy definition by reducing it to parameter settings (as described above). This puts specifying tutorial behavior at the complexity level of "templates and forms" (reminiscent of the Swift system's "minimalist ITS approach").

The question of how easy authoring can be made, and thus what set of skills is needed by an author is still an open question. We have seen that the amount of knowledge engineering skill needed can be minimized in constrained situations, potentially allowing any teacher or trainer to create or customize an ITS.

I should mention that another important class of potential users are educational theorists. ITS authoring tools should allow theorists to rapidly prototype ITSs and easily modify their teaching strategies and content to experiment with alternative curricula and instructional methods (Winne 1991). Evaluations of SimQuest and REEDEM have included such experimental comparisons of alternate teaching methods or styles.

Finally, another class of authoring tool users is students. In the SimQuest project teachers have created learning experiences for students that involve using authoring domain models and tutors for each other. Arroyo et al. (2001) created an authoring interface for the AnimalWatch tutor that was specifically for students to be able to author arithmetic word problems for each other. Initial trials indicate that this task was quite easy and motivating for them.

5.4 Collaborative authoring

If an ITS is to be created by a team, then the tools should support collaboration. Several authoring projects have directly addressed collaborative authoring. The IDE project supported collaboration by having authors enter the design rationale for each instructional element and decision in the system. Ritter et al.'s vision of tutor agent components (in this volume) is mean to support collaboration among curriculum experts (teachers), cognitive scientists/knowledge engineers, and programmers/database designer s. The DNA project provides tools for a subject-matter expert to work with several domain experts in developing content and check for overlap and differences in what they produce. The WEAR system allows teachers to search for pre-authored problems according to topic and difficulty level. It monitors the creation of a new problem and notifies the instructor if a similar one already exists in the database. In the REEDEM project teachers were observed sharing and critiquing each others tutoring strategies. As mentioned above, CREAM Tools supports collaborative authoring with rights management and access security features.

5.5 Where do the teaching strategies come from?

Another area of active disagreement in the research community is the appropriate source of instructional expertise. The question regards both embedded, fixed tutoring expertise and authored tutoring expertise. There is very little agreement on this front, and the arguments sometimes sound more religious than analytical. Below I present a somewhat hyperbolic characterization of the arguments.

Some emphasize that the power to determine an ITS's teaching strategy should rest with the practicing **teacher** (for example see Chapter 8 on REDEEEM). After all, they are ones working "in the trenches" (the XAIDA project adopted training strategies observed "in the wild"). But others argue that practicing teachers are, first, not very pedagogically adept on the average, and second, not very good at articulating their knowledge. Systems in the Multiple Knowledge Types category rely on **instructional design theories**. ID theories, though primarily prescriptive and lacking in rigorous experimental verification, have stood the test of time in countless on- and off-line industry and government training programs. Though their theories include the most practical and operational prescriptions, some critique instructional design theorists as being "arm-chair" researchers and systems thinkers. Some say that instruction via computers is too new to be able to rely on existing theories, and that we need to rely on empirical evidence from **educational psychologists** to determine which teaching strategies are most appropriate under various conditions. But educational researchers are sometimes criticized for having amassed decades worth of data which has lead to very few generally agreed upon operational principles. Perhaps instructional settings are just too complex to hope for definitive data and analysis. Still others maintain that traditional educational theory and research are based on outmoded behaviorist theories that do not take into account the constructivist and situated nature of learning, and that **cognitive science** might come to the rescue and show our ITSs how to teach. After all, they know how the mind works (!). And finally there are some research groups who simply have the right answer, and reference only their own **home-grown theories** of learning and instruction, ignoring outside empirical or theoretical work.

It is too early to ignore any of the sources of inspiration for ITS teaching strategies: practicing teachers, instructional designers, educational theorists, cognitive scientists, and innovative mavericks. The best models will most probably come from a synthesis of theories from several of these areas.

5.6 Meta-level authoring and Flexible Ontologies

As mentioned above, one method for maintaining both depth and usability in an ITS authoring system is to forgo breadth, i.e. specialize the authoring tool for a specific type of domain or task. Another approach, which has the potential of maintaining depth, breadth *and* usability, is meta-authoring (discussed in more detail in Chapter

15 Section 4.6). By a meta-authoring system I mean a general purpose authoring system is used to build or configure special-purpose authoring systems. For example, ITS authoring shells could be produced for science concepts, human service/customer contact skills, language arts, and equipment maintenance instruction. One problem with current special purpose authoring systems is that so many of them would have to be built to cover a reasonable diversity of tasks or domains. A proliferation of special purpose tools, each with different underlying frameworks and user interface conventions, will be hard to learn. Meta-authoring allows for the proliferation of special-purpose shells with a common underlying structure, so that inter-domain commonalties can be exploited in both content creation and in training the authors. A meta-authoring system requires a relatively high level of skill to use, but relatively few authors would use it. Most ITS authors would be using the more usable special purpose authoring systems that were built with the meta-authoring tools.

Current special purpose systems were programmed from scratch. Yet there are many common features among the diverse authoring tools described in this paper. A topic or curriculum network authoring tool would be useful to almost any authoring tool, as would a highly usable authoring tool for procedures or instructional strategies. Though very few of the authoring systems have tools for constructing the user interface, an interface building tool would be of use to all of them. Eon was designed as a meta-authoring tool, (as well as an authoring tool) but this use has not been realized yet. Jona and Kass (1997) describe an approach similar to meta-authoring, but in the context of ITS shells (architectures) rather than authoring tools.

An important aspect of meta-authoring is the ability to customize the conceptual vocabulary or ontology used to represent knowledge. An ontology is shared vocabulary describing key components, concepts, and properties, along with axioms that define relationships and constraints for these items (Gruber 1991). Many authoring systems use a semantic network representation of content but they differ on the types of nodes and links used. A more flexible approach is to let the author define this vocabulary. Expert-CML has a variety of common taxonomies for knowledge and learning objectives that the author can choose from. The Eon system allows the user to customize the vocabulary of node and link types in its topic network, the topic properties (such as importance and difficulty), and the vocabularies used in the student model and strategy editors.

Mizoguchi, Ikeda, and associates (Yayashi et al. 2000, Ikeda et al. 1997, Mizoguchi & Bourdeau 2000).have been developing ontologies. Mizoguchi et al. have developed such terminological building blocks for describing instructional strategies and actions, curriculum and task components, and learner states. Smart-Trainer is an ITS shell that has been implemented in the domain of electric power systems. It uses and "ontology-aware" approach that allows authors to build a high-level conceptual model of a domain and tutoring system before implementing the details. The system uses the axioms in its ontology in an authoring verification phase to identify discrepancies in the authored model.

5.7 Toward interoperability and re-use

Many have called for more interoperability and reusability in educational software applications and educational e-content. ITSs and ITS authoring tools tend to be monolithic pieces of code that can not communicate with each other (Roschelle et al. (1998) calls them "application islands"). Roschelle et al. (pg. 9) observe that the educational technology community's increasing sophistication in regards to a wide spectrum of teaching and learning theories has lead to the "unforeseen consequence [of] fragmentation of the authoring community around particular tools (rather than learning objectives), with limited ability to share innovation across different authoring tools." There is considerable redundancy in the design of many of these systems. Also, it tends to be difficult to extend to new domains or scale up to large courses. In the best of worlds a project should be able to re-use components of another project in a "plug-in" fashion. For example, the best Bayesian student modeler or the most fully functional graphing tool could be re-used by many ITS or ITS authoring tool design teams. Hodgins & Masie (2002) have identified a set of related issues, which I list below with additions from Roshelle et al. (1998) and Ritter et al. (in this volume). Authoring tools should support:

- **Interoperatability** -- can the system work with other systems? The trend is away from centralized client-server architectures toward more decentralized agent-based architectures. Ritter et al. (this volume) note that for applications to interoperate they need to, at the very least, be "inspectable" -- i.e. one application can ask another for specific types of information. A higher level of interoperability involves "record-ability", in which an application is broadcasting data (for example student behaviors) to any application that cares to notice (for example a feedback agent). A further level of interoperability involves "script-ability" which allows one application to control another, for example a tutorial agent invoking a weather simulator with specific constraints on the simulation.
- **Manageability** -- can the overall system track information about the learner and learning session? One method for this is record-ability, as mentioned above. But there are many ways to address this issue.
- **Re-usability** -- can courseware (learning objects) be re-used by applications and by developers different from the context for it was originally designed?
- **Durability and scalability** -- will the item remain usable as technology and standards evolve? Designing software so that it is extensible is one approach. Extensibility can be achieved by using open source coding or by having a plug-in architecture. Authoring tools that work with medium sized-domains or (in the case of on-line systems) with a few users, can experience performance problems when scaled up to large domains or many simultaneous users. Integration with off-the-shelf components such as robust database applications can alleviate this problem.

- **Accessibility** -- can the learner or content developer locate and download the material in a reasonable time? Content repositories, mentioned below, are addressing this issue.

In order to accomplish any of these objectives common standards are needed. In the area of application interoperability, several component software architectures have been developed including OpenDoc (no longer supported by Apple), Active X, and JavaBeans. Though these architectures may make interoperability a reality, so far there have been only a few attempts. Roschelle et al. (1998) describe the EduObject project in which four research groups used OpenDoc to share components of their learning environments. Koedinger et al. (1998) describe a project in which three independently developed educational applications were combined using the "MOO communications protocol" as a communication infrastructure. The applications were Active Illustrations Lab, an open-ended simulation environment (Forbus & Falkenhainer 1995); Belvedere, an environment supporting scientific dialoging and argumentation (Suthers et al. 1997); and a model tracing Tutor Agent that was able to give feedback to students (Ritter & Koedinger 1997). These two projects were feasibility demonstrations that did not see extensive use. Others are working on large scale in-house component based systems (e.g. Cheikes 1995), but a major bottleneck to interoperability of educational components is the lack of shared open standards. As stated in Ritter et al. (this volume) an interoperability framework needs "separate interface and data models which communicate through a fixed protocol [that] should express events in terms of their semantics, rather than user-interface implementation."

Several groups have been working toward and using an emerging set of educational software and learning management system interoperability standards (including IMS & IEEE LOM (Schoeneing & Wheeler 1997) and ARIADNE (Forte et al 1997)). The standards address content meta meta-data, content sequencing, question and test interoperability, learner profiles, and run-time interaction (see Hodgins & Masei 2002). SCORM (sharable content object reference model) is becoming a de-facto reference model for integrating the various standards. There are still many unsolved issues, including how to handle intellectual property rights in an environment of highly shared components and content, and how to include more pedagogically descriptive attributes in the standards (see Suthers 2000, Murray 1998).

Many efforts are underway to develop web-based repositories and catalogues that give educators and learners easy access to a wide range of educational materials (including EDUTELLA (Nejdg et al. 2002); OCW & OKI (Kumar & Long 2002); GEM (Fitzgerald 2001); MERLOT (Wetzel 2001)). These projects have begun to use meta-data standards to converge on common methods for describing generic attributes of learning objects. However, this software is currently designed to facilitates humans searching for educational material. The industry and the research communities are still far from having repositories of interoperable components that can be automatically retrieved and used by intelligent tutoring systems shells or authoring tools.

6. ITS AUTHORING SYSTEM "REALITY-CHECK" -- USE AND EVALUATION

In this Section I address the pragmatic questions of use, evaluation, throughput, and availability of ITS authoring tools. Availability is easy to describe. Several of the systems described here have become commercial products: Electronic Trainer (a simplified cousin of ID-Expert), and SIMQUEST. Neither of these have seen widespread purchasing or use in real educational situations as of yet. In addition some systems have been used by several groups other than the research group that developed the system and may be available through special arrangement.

6.1 Authoring tool use

One measure of the viability of an authoring tool is the number and variety of tutors it has been used to build, and the degree to which the system has been used independent of the lab in which it was developed. Of course, the fact that a system has not seen much use does not indicate that its design is not viable. But, since making usable software requires design iterations based on feedback from user and field tests, it is reasonable to assume that systems that have not been widely used will require significant additional work to become robust. It is also important to note that some systems are the latest in a series of efforts by a particular research group, so a relatively new and untested system may be built with the cumulative expertise from previous generations. For example RIDES, DIAG, and Instructional Simulator are third or fourth generation systems, and REDEEM, SimQuest, and Eon are second generation systems.

Several of the systems are commercial products, mentioned next. SimQuest is in use in a number of school systems. Electronic Textbook has been commercialised as IDXelerator (along with its sister product Instructional simulator, commercialised as IDVisualizer). A simplified version of CREAM-Tools is available under the name "Training Office," which has been adopted by a number of companies. Swift is commercial product, but there is no literature as yet describing its level of use. The D3 authoring tools has been used to produce three commercial training systems in the medical sector. Ritter et al. (in this Volume) describe tools used build Cognitive Tutors, several of which are commercial products being used in over 700 schools.

Table 3 shows a rough estimation of the degree to which various systems have been used. (The significant changes in this table since the original version of this paper in 1995 gives a striking picture of now far the field has progressed.) Category 1 is for early prototypes that are not fully functional authoring systems, and have been tested on a small number of "toy domains." Category 2 contains prototype systems that are complete authoring tools, most of which have been used to build several complete tutors, but the tutors were not used in authentic learning or training contexts. Category 3 systems are a bit more robust or have seen more use than those in category 2, and most have been used outside of the lab where they were built. Category 4 systems have been used to build a dozen or more tutors, have built tutors that have been used in real training situations, and have reached a stage of maturity in robustness and user documentation where they have been used relatively independently of the authoring tool designers.

Table 3: Degree of use of ITS authoring tools

1. Early prototypes and proofs of concept	Demonstr8, Expert-CML, IRIS, Training Express, BioWorld Case Builder, WEAR
2. Evaluated or used prototypes	DNA, Eon, IDLE-Tool, LAT, GTE, MetaLinks, ISD-Expert
3. Moderately evaluated or used	Electronic Trainer, REDEEM, XAIDA, D3 Trainer, DIAG, CREAM-Tools, Interbook, Swift
4. Heavily used (relatively)	RIDES, SIMQUEST, IDE[9], CALAT

- SimQuest has been used to create over 20 applications. including physics (collisions, electricity), biology, chemistry, economics, and geography. These tutors have been used in middle school and high school physics and chemistry classes. SMISLE (SimQuest's predecessor) has been used to author half a dozen systems in various introductory science domains.
- REDEEM has been used to build a Genetics tutor consisting of 12 hours of on-line content, which was used in a high school classroom. It was also used to develop an 8-hour tutorial on "understanding shapes" for 7-11 year old children. and a seven chapter Navy training course on communication and information systems principles.
- CALAT has been used to build over 300 web-based "courseware packages" which are being used for in-house training at NTT, where CALAT was developed.
- RIDES has been used to develop tutors or components of tutors in a number of research efforts, most of which involved exporting the technology to another lab. As well as producing tutors for a variety of types of equipment, these efforts are investigating such issues as immersive VR training, real-time collaborative environments, diagnostic expert systems, and web-based delivery.
- DIAG has been used to create training applications for electronic fault diagnosis for a eight devices including a power supplies, a radio receiver, and a warning system.
- CREAM-Tools have been used in a university biology course and a university mathematics course, and to create tutors in the domains of Baxter pump manipulation, Traffic regulations, problem solving strategies, intensive care unit, and Excel spread sheets.

[9] Though IDE was one of the most heavily used systems, it was also one of the earliest. IDE is now a "legacy system," since it runs on obsolete software (NoteCards) and does not incorporate multimedia capabilities that are now de rigueur.

- As mentioned above, XAIDA has been used to develop tutors in diverse domains, including algebra, medicine, computer literacy, and biology, as well as device operation and maintenance.
- Instructional Trainer has been used to build tutors for HP 5S1 printer operation (this tutorial is distributed commercially with the physical product), ethnographic methods, simple canal theory of operation, and customer service in the telecommunications industry.
- The three commercially available tutorials produced using D3 Trainer have been evaluated via field tests for 4 years. It has been used to build case-based classification tutors in medical domains such as Rheumatology and in other domains such as Flower Classification.
- IDLE-Tool underwent three informal trials: 21 graduate students produced 10 goal-based scenario (GBS) tutors during a six week graduate seminar; 8 primary school teachers produced four GBS tutors over a six week period; and eight graduate students produced GBS tutors in another seminar over a three week period. The INDIE tool has been used to produce 8 goal based scenario tutors, including: Immunology Consultant, Is it a Rembrandt, Volcano Investigator, and Nutrition Clinician.
- Eon has been used to build five prototype tutors covering a wide range of domain types and instructional methods, including: a tutor that incorporates a Socratic teaching strategy to remediate common misconceptions in science; a tutor that teaches a part of Japanese language called "honorifics;" an open-ended learning-by-doing chemistry workbench environment, and a tutor that uses a spiral teaching method to teach introductory physics concepts.
- IRIS has been used to create tutors for power plant operator training, computer program specification, concepts in human resources, and mathematical symbolic differentiation.

6.2 Authoring tool productivity

ITS authoring tools have the potential for decreasing the effort it takes to build instructional systems or, with the same effort, increasing the adaptivity, depth, and effectiveness of instructional systems. A very rough estimate of 300 hours of development time per hour of on-line instruction is commonly used for the development time of traditional CAI. We have indications of the development ratios for some ITS authoring tools. These numbers are very hard to interpret, but give us hope that cost-effective ITS authoring is possible. One reason it is hard to interpret these results is that they usually do not include the time for creating graphics or pre-planning the curriculum design. But what we can say is that there is a strong indication that authoring tools change coding the domain and/or expert knowledge in a tutor from the most labor intensive part of the process to the least labor intensive (as compared with media creation and off-line design analysis)

Many hope to see ITS development times that are an order of magnitude less than the 300:1 CAI productivity ratio. ID-Expert's goal is a 30:1 ratio. The

Instructional Simulator reduced the time to create a simulation from 100s of hours (if programmed from scratch) to a few hours. An informal analysis of Demonstr8 describes a model tracing multi-column addition or subtraction tutor being built in less than 20 minutes. XAIDA's goal is for a 10:1 productivity ratio. Formative evaluations to date indicate that in some situations a first-time XAIDA user can develop a 1-2 hour lesson in 3-4 days, including training.

In three separate studies productivity metrics for the REDEEM system showed authoring to tutorial time ratios between 2:1 and 3:1 for authoring courses that were 8 to 20 hours in length.

A sixteen month case study of three educators using KAFITS, the precursor to Eon, to build a 41 topic tutor for high school Statics (representing about six hours of on-line instruction) resulted in a 100:1 effort ratio. Analysis of time vs. development task and development role yielded the following: 47% effort by the SME, 40 % by the "knowledge based managers", and 13% by the knowledge engineer. Also, design constituted about 15% of the total time, and implementation the other 85%. A similar breakdown of authoring tasks for use of the CALAT system yielded these estimates: planning 10%, design 50%, multimedia material creation 30%, and testing and evaluation 10% of the total time. It was estimated that development time for CALAT tutors was about the same as traditional, non-adaptive instructional systems.

The DNA system supports cognitive task analysis, which usually involves extensive interviews with experts and transcription and analysis of the protocols. This process usually requires a knowledge engineer working with domain experts over many months. DNA attempts to "streamline the bulk of the interview, transcription, and [information] organization process." In evaluations of the DNA system a collaborative authoring process yielded over 50% of the knowledge base in a tiny fraction of the development time.

For authoring tools that build tutors with deeper knowledge, much more time is typically needed. It tool 4 to 9 weeks to use DIAG to build tutors simulating complex electronic systems of up to 150 components. The problems posed by the resulting tutorials are solved on the order of 10 minutes each. Some tools have been developed to make Cognitive Tutors more cost effective but it still takes many man-years to build a cognitive tutor. However, these are among the very few intelligent tutors built to date that cover content from a significant percentage of an entire course.

6.3 Authoring tool evaluation

Because ITS authoring tools are still relatively new, summative evaluations, which ostensibly prove that an entire system "works," may be less valuable than formative evaluations, which give indications of what parts of a system do and don't work and why. A number of qualitative and formative evaluation methods can be used (Murray 1993). In Section 5.3 we mentioned how results from evaluations of SimQuest, REDEEM, XAIDA, IDLE-Tool, and LAT. In Section 6.2 we mentioned system evaluations that had implecations for authoring productivity (XAIDA, KAFITS, CALAT, DNA, and DIAG). A summary of other authoring tool evaluations follows (we do not discuss evaluations of the tutoring systems built by the authoring tools--in fact very few of these have been done).

REDEEM and XAIDA are (by far) the most heavily evaluated authoring systems. REDEEM'S predecessor COCA underwent several evaluations. Ten teachers each using COCA for 2 to 3 hours to build a tutor for the American Revolution. Teachers' attitudes regarding the ability of AI technology to simulate reasonable teaching strategies changed from noncommittal to positive. However, many of the systems features were too complex for teachers, and these problems lead to the design of REDEEM.

REDEEM has been used in 4 major studies (and several minor studies not discussed in Ainsworth et al.). Two studies involved an 8-hour tutorial on "understanding shapes" for 7-11 year old children. One study involved 12-hour secondary school tutorial in genetics, and the final study involved a seven chapter Navy training course on communication and information systems principles.

In two of the studies that looked at how instructors can customize intelligent tutors a novel evaluation method was used. Subjects were given vignettes describing a number of "virtual students," including descriptions of how they performed in the subject area over the previous semester. The author's task was, first, to group the virtual students into similarity classes, and second, to design a tutoring strategy that met the needs of each of these groups (the REDEEM tools directly support such student grouping, strategy creation, and assignment of student groups to strategies). The above method was the primary one used to evaluate the authoring process. To evaluate student learning Ainsworth et al. designed studies that compared learning on a REDEEM-delivered intelligent tutor to leaning with the original non-intelligent courseware d ("vanilla CBT") which served as the content basis for the intelligent tutor.\

In an analysis of usability, two of the studies found that instructors with no prior experience with computer-based learning were able to express, represent, and assess their teaching strategies after only one to 2 hours of training. We mentioned the REDEEM productivity analysis above.

A "virtual student" study of primary school teachers found strong inter-author difference in how they grouped virtual students for differential instructional treatments, and in how they designed teaching strategies for each of these groups. In contrast, the group of Navy training personnel showed mush less differentiation among virtual students and teaching strategies. In both situations authors who used REDEEM to tailor the instructional strategies of tutorial reported more satisfaction than those who did not. When the tutorial was run these authors recognized the existence of their personal classroom-based teaching strategies and styles in the behavior of the tutorial that they authored.

Five separate evaluation studies looked at the effectiveness of REDEEM-authored ITSs, in comparison to similar CBT tutorials. In the first study 86 15 year old students used a Genetics tutor in a laboratory setting. There were no significant difference in outcomes for either high or low achieving students. However, a very similar study done in a classroom context showed significant differences. REDEEM-tutored students improved 16% vs. an 8% improvement for vanilla-CBT-tutored students. An effect size analysis showed that REDEEM led to an average 0.82 sigma improvement in learning outcomes compared to CBT. In a study using the Navy-built tutor REDEEM was again significantly better than the vanilla CBT. REDEEM lead to a 32% pre-to-post test improvement, as compared with a 19% improvement with the vanilla CBT (effect size 0.76 sigmas).

The other extensively evaluated ITS authoring tool is XAIDA.[10] Formative data was taken as XAIDA was used in eight authoring field studies with an average of about 10 participants (mostly military training personnel) in each study. As mentioned, one result was that a 1-2 hour lesson can be developed in 3-4 days. The framework was found appropriate for a wide variety of domains, as mentioned above. In addition to formative evaluations of the authoring tools, there have been 13 studies of students using tutors built with XAIDA. These have indicated that tutors built with the XAIDA framework successfully promote mastery of a wide range of subjects, and that students acquire robust cognitive structures if they are motivated learners. Finally, researchers conducted an in-depth study of 17 participants' attitudes and skills as they learned to use XAIDA (only the physical characteristics shell). Several types of data were gathered, including usability comments, attitudes, productivity metrics, and knowledge base structural analyses. Results indicate that the tool can be used to author ITSs rapidly. However, the training and evaluation focussed on low level authoring skills, and it was unclear how limitations in higher level design and content analysis skills would effect authoring and the adoption of such authoring tools by instructors.

Below we briefly describe other authoring tool evaluation studies:

- In evaluations of IDLE-Tool, conclusions included the need for a higher level view of the curriculum and more conceptually oriented help (as opposed to interface or task oriented help). Practicing teachers using the system differed from the graduate students in their higher pedagogical competence and willingness to work within the limits of the template-based authoring. All users found the example-based help feature very helpful.
- A formative evaluation of LEAP consisted of user participatory design sessions with three authors from a population of target users. The users built a working body of courseware from scratch, and maintained a large body of pre-existing courseware. Anecdotal usability data indicated that the authoring tool was much easier and less error prone than previous text-based methods used for knowledge elicitation.
- A preliminary evaluation of DNA compared the knowledge base of a "measures of central tendency" (statistics) tutor, built using DNA's automated knowledge elicitation, with the knowledge base of a benchmark ITS for the same domain. The benchmark tutor was build from scratch using a lengthy hand-coded cognitive task analysis, which took several months and resulted in 78

[10] Note however that for the most part, only the simplest of XAIDAs four knowledge types was authored in these trials, i.e. physical characteristics.

"curriculum units" (topics). Three subjects used DNA to elicit their knowledge about this domain, and the three resulting knowledge bases were compared with each other and with the benchmark. Analysis of the results showed that the three authors had 25%, 49%, and 23% coverage of the 78 curriculum units, with a combined coverage of 62% over the total of nine hours that the three experts used the system. That a collaborative authoring effort that took nine hours resulted in 62% coverage of a knowledge base that took several months to code by hand indicated that the DNA framework is viable.

Above I mentioned that authoring tools are of significant advantage in evaluating alternate instructional strategies. In addition to REDEEM, this type of studies was done with Instructional simulator and SimQuest.

- Instructional Simulator underwent a series of studies (Mills, Lawless, Drake, and Merrill in press) with elementary students using the canal boat simulation described in Chapter 7. One test evaluated three variations of explanation: at every step, only when a user action resulted in no simulation change, and no explanation at all. The study showed a significant long term retention improvement when either explanation method was included. A second study compared three variations on how simulation instructional demonstrations were given: "Simon says" demonstrations, free exploration demonstrations, and no demonstrations. Students using to the Simon Says mode outperformed the other two conditions.
- The SimQuest system has been used to conduct a number of studies on alternate teaching strategies. The studies looked at variations in learning environments configuration. Variations in the following aspects of the learning environment were evaluated in separate studies (see Chapter 1): model progression, assignment type, and intelligent feedback.

7. FUTURE DIRECTIONS

In 1995 A large scale review of US government-sponsored research and development in intelligent tutoring systems, looking at 47 funded projects, found that one third of the total expenditure was on the 11% of the projects developing authoring tools (Youngblut, 1995). The review concluded that this level of funding might be premature because there were many basic research issues in ITS needing to be resolved before authoring systems were generally applicable. However, ITS authoring tools have matured substantially in the last decade. Several are at or near commercial quality. Though there are of course many unanswered questions in this relatively new research area, it seems that there are three related major unknowns. The first is the extent to which the difficult task of modeling can be scaffolded, as discussed above. The second question, representing the other side of the coin, is the degree to which we can identify instructional situations that can be embodied in special purpose authoring shells that are both specific enough to make authoring template-based, yet general enough to be attractive to many educators. Third is the larger question of whether intelligent tutoring systems will ever be in demand enough to warrant the effort of building authoring tools for them. Those in the authoring tools community see this as a chicken-egg problem, since the demand for ITSs depends in part on their cost, and in part on their perceived effectiveness. Authoring tools certainly reduce the cost, and they also will make it possible to build enough systems so that formal and anecdotal evidence will accumulate regarding ITS effectiveness.

Much more research and development is needed in the field of ITS authoring. We need more empirical testing using multiple authors and domains; more research on authoring student models; more complete and standardized ontologies and meta-data standards; more research on the differential effectiveness of various computationally explicit instructional strategies; and more exploration of open component-based architectures. The future of ITS authoring, like the future of ITSs, depends in part on supply and demand forces. Innovations in software interoperability, web-based applications, and ubiquitous computing provide a "technology-push." Increasing demand for scalability, accountability, personalization, easy access, and cost-effectiveness in computer-based instruction should provide a sufficient "pull" to bring more of these systems into schools and market places.

8. REFERENCES

The references and citations to articles describing authoring systems are in a non-standard format in this Chapter. Citations to the major authoring tool projects are give by the tool name (e.g. RIDES) rather than the author of a paper (e.g. Munro et al. 1997). References are listed in an Appendix table grouped according to the authoring system.

Arroyo, I., Schapira, A. & Woolf, B. (2001). Authoring and sharing worked problems with AWE. *Proc. of Artificial Intelligence in Education* (Moore et al. Eds), pp. 527-529.

Bloom, B. S. (1956). *Taxonomy of Educational Objectives, Vol. 1.* New York: David McKay Co.

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.

Brusilovsky, P. (1998). Methods and Techniques of Adaptive Hypermedia. In P. Brusilovsky, A. Kobsa, and J. Vassileva, editors, *Adaptive Hypertext and Hypermedia,* Chapter 1, pp. 1-44, Kluwer Academic Publishers, The Netherlands, 1998.

Chandrasekaran,B. (1986). Generic tasks in knowledge based reasoning: high-level building blocks for expert system design. IEEE Expert, 1(3), pp. 23-30.

Cheikes, B. (1995). Should ITS Designers be Looking for a Few Good Agents? In AIED-95 workshop papers for Authoring Shells for Intelligent Tutoring Systems.

Clancey, W. J. (1986). "Qualitative Student Models." In Annual Review of Computer Science, pp. 381-450: Palo Alto, CA.

Collins, J.A., Greer, J.E., & Huang, S.H. "Adaptive assessment using granularity hierarchies and Bayesian nets." *Proceedings of the Third International Conference: ITS '96.* Frasson, Gautheir & Lesgold (Eds). Springer, pp. 569-577.

Fitzgerald, M (2001). The Gateway to Educational Materials: An evaluation Study: Year 2. ERIC Clearinghouse technical report.

Forbus, K & Falkenhainer, B. (1995). Scaling up Self-Explanatory Simulators: Polynomial-time Compilation. *Proceedings of IJCCAI-95*, Montreal, Canada.

Forte, E., Wentland, M. & Duval, E. (1997). The ARIADNE Project: Knowledge Pools for Computer-based and Telematics-supported Classical, Open, and Distance Learning. European Journal of Engineering Education 22(1).

Gagne, R. (1985). *The Conditions of Learning and Theory of Instruction*. New York: Holt, Rinehard, and Winston.

Goodkovsky, V.A., Kirjutin, E.V. & Bulekov, A.A. (1994). Shell, tool, and technology for Pop Class ITS production. In P. Brusilovsky, S. Dikareve, J.Greer & V. Pertrushin (Eds). Proc. of East-West International Conference on Computer Technology in Education. Part 1, pp. 87-92. Crimea, Ukraine.

Goodyear, P. & Johnson, R. (1990). Knowledge-based authoring of knowledge-based courseware. In Proc. of ICTE-7, Brussels:CEP Consultants LTD.

Hodgins, W. & Massie, E. (2002). Making Sense of Learning Specifications and Standards: A decision makers guide to their adoption. MASIE Center technical report, Saratogy Springs, NS.

Hodgins, W. et al. (2002). Making Sense of Learning Specifications & Standards: A Decision Maker's Guide to their Adoption. Industry Report by the MASIE Center: Saratoga Springs, NY.

Hoffman, R. (1987). "The Problem of Extracting the Knowledge of Experts From the Perspective of Experimental Psychology." *AI Magazine*, pp. 53-67, Summer 1987.

Jonassen, D.H. & Reeves, T.C (1996). Learning with Technology: Using Computers as Cognitive Tools. In D.H. Jonassen, (Ed.) Handbook of Research on Educational Communications and Technology. New York: Scholastic Press, Chapter 25.

Koediner, K.R, Suthers, D.D., Forbus, K.D. (1998). Component-based construction of a science learning space. In the *Proceedings of Intelligent Tutoring Systems 4th International Conference*, Goettl, Half, Redfield, & Shute (Eds), 166-175.

Koedinger, K., & Anderson, J. (1995). Intelligent tutoring goes to the big city. *Int. J. of Artificial Intellignece in Education*, 8, 30-43.

Kumar, M.S.V. & Long , P. (2002) MITs Open Courseware Initiative (OCW) and Open Knowledge Initiative (OKI). At www.cren.net/know/techtalk/mit.html.

Mark, M.A. & Greer, J.E. (1991). The VCR Tutor: Evaluating instructional effectiveness. In Hammond, & Gentern (Eds.) Proceedings of the 13th Annual Conference of the Cognitive Science Society, Lawrence Erlbaum Asso., Hillsdale, NJ., 564-569.

McCalla, G. & Greer, J. (1988). "Intelligent Advising in Problem Solving Domains: The SCENT-3 Architecture." Proceedings of ITS-88, pp. 124-131. June, 1988, Montreal, Canada.

McMillan, S., Emme, D., & Berkowitz, M. (1980). Instructional Planners: Lessons Learned. In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned.* Hillsdale, NJ: Lawrence Erlbaum, pp. 229-256.

Merrill, M.D. (1983). Component Display Theory. In *Instructional-design theories and models: An overview of their current status,*. C.M. Reigeluth. (Ed). Hillsdale, NJ: Lawrence Erlbaum, pp. 279 - 333.

Murray, T (1998). A Model for Distributed Curriculum on the World Wide Web. J. of Interactive Media in Education 98(5). On-line journal at http://www-jime.open.ac.uk/.

Murray, T. (1993). Formative Qualitative Evaluation for "Exploratory" ITS research. *J. of AI in Education*. 4(2/3), pp. 179-207.

Murray, T. (1996b). Toward a Conceptual Vocabulary for Intelligent Tutoring Systems. Working Paper.

Murray, T. (1997) Expanding the knowledge acquisition bottleneck for intelligent tutoring systems. *International J. of Artificial Intelligence in Education*. Vol. 8 , No. 3-4, pp. 222-232.

Murray, T., Winship, L., Bellin, R. & Cornell, M. (2001). "Toward Glass Box Educational Simulations: Reifying Models for Inspection and Design." In workshop proceedings for External Representations in AIED at AIED-2001. May 2001, San Antonio, TX.

Nejdl, W., Wold, B., Staab, S., & Tane, J. (2002). EDUTELLA: Searching and Annotating Resources within and RDF-based P2P Network. White paper at edutella.jxta.org.

Ohlsson, S. (1987). Some Principles of Intelligent Tutoring. In Lawler & Yazdani (Eds.), *Artificial Intelligence and Education, Volume 1*. Ablex: Norwood, NJ, pp. 203-238.

Person, N. K., Bautista, L., Graesser, A. C., Mathews, E. & The Tutoring Research Group (2001). Evaluating student learning gains in two versions of AutoTutor. In J. D. Moore, C. L Redfield, & W. L. Johnson (Eds.), *Artificial Intelligence in Education: AI–ED in the Wired and Wireless Future* (pp. 286–293). Amsterdam: IOS Press.

Reigeluth, C. (1983). The Elaboration Theory of Instruction. In Reigeluth (Ed.), *Instructional Design Theories and Models*. Hillsdale, NJ: Lawrence Erlbaum.

Ritter, S. & Blessing, S. (1998). Authoring tools for component-based learning environments. *Journal of the Learning Sciences*,. 7(1) pp. 107-132.

Ritter, S. & Koedinger, K.R. (1997). An architecture for plug-in tutoring agents. In *J. of Artificial Intelligence in Education*, 7 (3/4) 315-347.

Roschelle, J., Kaput, J., Stroup, W. & Kahn, T.M. (1998). Scaleable integration of educational software: Exploring the promise of component architectures. *J. of Interactive Media in Education*, 98 (6). [www-jime.open.ac.uk/96/6]

Rosé, C. P., Jordan, P., Ringenberg, M., Siler, S., VanLehn, K., & Weinstein, A. (2001). Interactive conceptual tutoring in Atlas–Andes. In J. D. Moore, C. L Redfield, & W. L. Johnson (Eds.), *Artificial Intelligence in Education: AI–ED in the Wired and Wireless Future* (pp. 256–266). Amsterdam: IOS Press.

Schoening, J. & Wheeler, T. (1997). Standards--The key to educational reform. In IEEE Computer, March 1997.

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.

Shute, V. J., and Psotka, J., (1996). Intelligent tutoring systems: Past, present, and future. In D. Jonassen (Ed.), Handbook of Research for Educational Communications and Technology (pp. 570-600). New York, NY: Macmillan.

Shute, V.J. and Regian, J.W. (1990). Rose Garden Promises of Intelligent Tutoring Systems: Blossom or Thorn? Presented at Space Operations, Automation and Robotics Conference, June 1990, Albuquerque, NM.

Suthers, D. (2000). Using learning object meta-data in a database of primary and secondary school resources. *Proc. of International Conf. on Computer in Education*, November 2000, Taipei, Taiwan.

Suthers, D., Toth, E., 7 Weiner, A. (1997). An integrated approach to implementing collaborative inquiry in the classroom. *Computer Supported Collaborative Learning (CSCL-97)*, Toronto, December, 1997.

Wasson, B. (1992) PEPE: A computational framework for a content planner. In S.A. Dijkstra, H.P.M. Krammer & J.J.G. van Merrienboer (Eds), Instructional Models in Computer-Based Learning Environments. NATO ASI Series F. Vol. 104 (pp. 153-170). New York: Sringer-Verlag.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems.* Los Altos, CA: Morgan Kaufmann.

Wetzel, M., & Hanley, G. (2001). Evaluation of MERLOT Tools, Processes, and Accomplishments. Center for Usability in Design and Assessment: Long Beach CA.

White, B. & Frederiksen, J. (1995). Developing Metacognitive Knowledge and Processes: The Key to Making Scientific Inquiry and Modeling Accessible to All Students. Technical Report No CM-95-04. Berkeley, CA: School of Education, University of California at Berkeley.

Youngblut, C., 1995. Government-Sponsored Research and Development Efforts in the Area of Intelligent Tutoring Systems: Summary Report. Inst. for Defense Analyses Paper No. P-3058, Alexandra VA.

APPENDIX

Below is a table of the ITS authoring tools discussed in this paper, with selected references for each.

BioWorld-Case Builder	Lajoie, S., Faremo, S. & Wiseman, J. (2001). A knowledge-based approach to designing authoring tools: From tutor to author. In *Proc. of Artificial Intelligent in Education*, J.D. Moore C. Redfield, L.W. Johnson (Eds). ISO Press, pp77-86.
CALAT (& CAIRNEY)	Kiyama, M., Ishiuchi, S., Ikeda, K., Tsujimoto, M. & Fukuhara, Y. (1997). Authoring Methods for the Web-Based Intelligent CAI System CALAT and its Application to Telecommunications Service. In the *Proceedings of AAAI-97* , Providence, RI.
CREAM-TOOLS	See Chapter 10 in this volume. Frasson, C., Nkambou, R., Gauthier, G., Rouane, K. (1998). An authoring model and tools for curriculum development in intelligent tutoring systems. Working Paper available from the authors. Nkambou, R., Gauthier, R., & Frasson, M.C. (1996). CREAM-Tools: an authoring environment for curriculum and course building in an ITS. In *Proceedings of the Third International Conference on Computer Aided Learning and Instructional Science and Engineering*. New York: Springer-Verlag.
D3-TRAINER	Schewe, S., Reinhardt, B., Bestz, C. (1999). Experiences with a Knowledge Based Tutoring System for Student Education in Rheumatology. In *XPS-99: Knowledge Based Systems: Survey and Future Direction, 5th Biannual German Conference* on Knowledge Based Systems, Lecture Notes in Artificial Intelligence 1570, Springer. Puppe, F., Reinhardt, B. (1996). Generating Case-oriented training from Diagnostic Expert Systems. In *Machine Mediated Learning* 5 (3&4), 199-219. Reinhardt, B. (1997). Generating Case-oriented Intelligent tutoring systems. *In Proc. of AAAI Fall Symposium, ITS Authoring Systems,* November 1997.

DEMONSTR8 (& TDK, PUPS)	See Chapter 4 in this volume. Blessing, S.B. (1997). A programming by demonstration authoring tool for model tracing tutors. *Int. J. of Artificial Intelligence in Education.* Vol. 8 , No. 3-4, pp 233-261. Anderson, J. R. & Pelletier, R. (1991). A development system for model tracing tutors. In *Proc. of the International Conference on the Learning Sciences*, Evanston, IL, 1-8. Anderson, J. & Skwarecki, E. (1986). The Automated Tutoring of Introductory Computer Programming. *Communications of the ACM*, Vol. 29 No. 9. pp. 842-849.
DIAG (& ReAct, DM3)	See Chapter 5 in this volume. Towne, D.M. (1997). Approximate reasoning techniques for intelligent diagnostic instruction. *International J. of Artificial Intelligence in Education.* Vol. 8 , No. 3-4, pp. 262-283. Towne, D.M. (2002). Advanced Techniques for IETM Development and Delivery, Proceedings Human Factors and Ergonomics Society, 46th Annual Meeting, Baltimore, MD, October 3, 2002.
DNA/SMART	See Chapter 6 in this volume. Shute, V.J. (1998). DNA - Uncorking the bottleneck in knowledge elicitation and organization. *Proceedings of ITS-98*, San Antonio, TX, pp. 146-155. Shute, V. J., Torreano, L. A., and Willis, R. E. (2000). Tools to aid cognitive task analysis. In S. Chipman, V. Shalin, & J. Schraagen (Eds.),Cognitive Task Analysis. Hillsdale, NJ: Erlbaum Associates. . Shute, V. J. & Torreano, L., & Willis, R. (2000). DNA: Towards an automated knowledge elicitation and organization tool. In S. P. Lajoie (Ed.) *Computers as Cognitive Tools, Volume 2.* Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 309-335.
ECSAIWeb	Sanrach, C. & Grandbasien, M. (2000). ECSAIWeb: A Web-based authoring system to create adaptive learning systems. *In Proceedings of Adaptive Hypermedia 2000.*
EON (& KAFITS)	See Chapter 11 in this volume. Murray, T. (1998). Authoring knowledge-based tutors: Tools for content, instructional strategy, student model, and interface design. *J. of the Learning Sciences*, Vol. 7. No. 1, pp. 5-64. Murray, T. (1996). Special Purpose Ontologies and the Representation of Pedagogical Knowledge. *In Proceedings of the International Conference on the Learning Sciences,* (ICLS-96), Evanston, IL, 1996. Charlottesville, VA: AACE. Murray, T. (1996). From Story Boards to Knowledge Bases: The First Paradigm Shift in Making CAI "Intelligent.". Proceedings of the ED-Media 96 Conference, Boston, MA, June 1996, pp. 509-514.

EXPERT-CML	Jones, M. & Wipond, K. (1991). Intelligent Environments for Curriculum and Course Development. In Goodyear (Ed.), *Teaching Knowledge and Intelligent Tutoring*. Norwood, NJ: Ablex.
GETMAS	Wong, W.K. & Chan, T.W. (1997). A Multimedia authoring system for crafting topic hierarchy, learning strategies, and intelligent models. *International J. of Artificial Intelligence in Education*, Vol. 8, No 1, pp. 71-96.
GTE	Van Marcke, K. (1998). GTE: An epistemological approach to instructional modeling. *Instructional Science*, Vol. 26, pp 147-191. Van Marcke, K. (1992). Instructional Expertise. In Frasson, C., Gauthier, G., & McCalla, G.I. (Eds.) *Procs. of Intelligent Tutoring Systems* '92. New York: Springer-Verlag.
Instructional Simulator (&Electronic Textbook, IDVisualizer, IDXelerator, ID-EXPERT, Electronic Trainer, ISD-Expert)	See Chapter 7 in this volume. Merrill, M.D., & ID2 Research Group (1998). ID Expert: A Second generation instructional development system. *Instructional Science*, Vol. 26, pp. 243-262. Merrill, M. D. (2001). Components of instruction: toward a theoretical tool for instructional design. *Instructional Science*. 29(4/5), 291-310. Mills, R. J., Lawless, K. A., Drake, L., & Merrill, M. D. (in press). Procedural knowledge in a computerized learning environment.
IDE (& IDE Interpreter)	Russell, D. (1988). "IDE: The Interpreter." In Psotka, Massey, &Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned*. Hillsdale, NJ:Lawrence Erlbaum. Russell, D., Moran, T. & Jordan, D. (1988). The Instructional Design Environment. In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned.* Hillsdale, NJ: Lawrence Erlbaum.

IDLE-Tool (& IMAP, INDIE, GBS-architectures)	See Chapter 12 in this volume. Bell, B. (1998). Investigate and decide learning environments: Specializing task models for authoring tools design. *J. of the Learning Sciences*, Vol. 7. No. 1. Jona, M. & Kass, A. (1997). A Fully-Integrated Approach to Authoring Learning Environments: Case Studies and Lessons Learned. In the *Collected Papers from AAAI-97 Fall Symposium workshop Intelligent Tutoring System Authoring Tools*. AAAI-Press. Dobson, W.D. & Riesbeck, C.K. (1998). Tools for incremental development of educational software interfaces. In *Proceedings of CHI-98*. Qiu, L., Riesbeck, C.K., and Parsek, M.R. (2003). The Design and Implementation of an Engine and Authoring Tool for Web-based Learn-by-doing Environments. *Proc. of World Conf. on Educational Multimedia, Hypermedia & Telecommunications* (ED-MEDIA 2003). June 23-28, 2003, Honolulu, HA. AACE.
InterBook (& ELM-Art, NetCaoch)	See Chapter 13 in this volume. Brusilovsky, P., Schwartz, E., & Weber, G. (1996). A Tool for Developing Adaptive Electronic Textbooks on WWW. *Proc. of WebNet-96*, AACE. Brusilovsky, P, Schwartz, E. & Weber, G. (1996). ELM -ART: An Intelligent Tutoring System on the Work Wide Web. *In Proceedings of ITS-96,* Frasson, Gauthier, Lesgold (Eds.), Springer: Berlin, 1996. pp. 261-269.
IRIS	See Chapter 9 in this volume. Arruarte, A., Fernandez-Castro, I., Ferrero, B. & Greer, J. (1997). The IRIS shell: How to build ITSs from pedagogical and design requisites. *International J. of Artificial Intelligence in Education*. Vol. 8 , No. 3-4, pp. 341-381.
LAT (LEAP Authoring Tool)	See Chapter 14 in this volume. Sparks, R. Dooley, S., Meiskey, L. & Blumenthal, R. (1999). The LEAP authoring tool: supporting complex courseware authoring through reuse, rapid prototyping, and interactive visualizations. *Int. J. of Artificial Intelligence in Education.* Dooley, S., Meiskey, L., Blumenthal, R., & Sparks, R. (1995). Developing reusable intelligent tutoring system shells. In AIED-95 workshop papers for Authoring Shells for Intelligent Tutoring Systems.
MetaLinks	Murray, T., Condit, C., & Haaugsjaa, E. (1998). MetaLinks: A Preliminary Framework for Concept-based Adaptive Hypermedia. *Workshop Proceedings from ITS-98 WWW-Based Tutoring Workshop*., San Antonio, Texas, 1998.
REDEEM (& COCA)	See Chapter 8 in this volume. Major, N., Ainsworth, S. & Wood, D. (1997). REDEEM: Exploiting symbiosis between psychology and authoring environments. *International J. of Artificial Intelligence in Education*. Vol. 8 , No. 3-4, pp. 317-340. Major, N. (1995). Modeling Teaching Strategies. *J. of AI in*

	Education, 6(2/3), pp. 117-152. Major, N.P. & Reichgelt, H (1992). COCA - A shell for intelligent tutoring systems. In Frasson, C., Gauthier, G., & McCalla, G.I. (Eds.) *Procs. of Intelligent Tutoring Systems '92*. New York: Springer-Verlag.
RIDES (& IMTS, RAPIDS, and see DIAG)	See Chapter 3 in this volume. Munro, A., Johnson, M.C., Pizzini, Q.A., Surmon, D.S., Towne, D.M, & Wogulis, J.L. (1997). Authoring simulation-centered tutors with RIDES. *International J. of Artificial Intelligence in Education*. Vol. 8 , No. 3-4, pp. 284-316. Towne, D.M., Munro, A., (1988). The Intelligent Maintenance Training System. In Psotka, Massey, & Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned*. Hillsdale, NJ: Lawrence Erlbaum.
SIMQUEST (& SMISLE)	See Chapter 1 in this volume. Jong, T. de & vanJoolingen, W.R. (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, Vol. 68 No. 2, pp. 179-201. Van Joolingen, W.R. & Jong, T. de (1996). Design and implementation of simulation-based discovery environments: The SMISLE solution. *Int. J. of Artificial Intelligence in Education* 7(3/4). pp. 253-276.
Smart Trainer (& FITS, ontology-based tools)	Ikeda, M. & Mizoguchi, R. (1994). FITS: A Framework for ITS--A computational model of tutoring. J. of Artificial Intelligence in Education 5(3) pp. 319-348. Mizoguchi, R., Sinitsa, K., Ikeda, M. (1996). Knowledge Engineering of Educational Systems for Authoring System Design. In Proceedings. of EuroAIED-96, Lisbon, pp. 593-600. Ikeda, M., Seta, K. & Mizoguchi, R. (1997). Task ontology makes it easier to use authoring tools. Proc. of IJCAI-97, Nagoya, Japan. Mizoguchi, R. & Bourdeau, J. (2000). Using ontological engineering to overcome common AI-ED problems Int. J. of Artificial Intelligence and Education, Vol. 11. pp 107-121. Yayashi, Y., Ideda, M., Seta, K., Kakusho, O. & Mizoguchi, R. (2000). Is what you write what you get?: An operational model of training scenario. Proc. of Intelligent Tutoring Systems 2000.
Swift (& DOCENT, Study)	Winne P.H. (1991). Project DOCENT: Design for a Teacher's Consultant. In Goodyear (Ed.), *Teaching Knowledge and Intelligent Tutoring*. Norwood, NJ: Ablex. Winne, P. & Kramer, L. (1988). "Representing and Inferencing with Knowledge about Teaching: DOCENT." *Proceedings of ITS-88*. June 1988,Montreal, Canada.
TANGOW	Carro, R.M., Pulido, E.., Rodriquies, P. (2002). *An authoring tool that automates the process of developing task-based adaptive courses on the web*. J. of AI and Education.

TRAINING EXPRESS	Clancey, W. & Joerger, K. (1988). "A Practical Authoring Shell for Apprenticeship Learning." *Proceedings of ITS-88*, 67-74. June 1988, Montreal.
WEAR	Virvou, M & Moundridou, M. (2001). Adding an instructor modeling component to the architecture of ITS authoring tools. *Int. J. of Artificial Intelligence in Education* 12(2), pp 185-211.
XAIDA	See Chapter 2 in this volume. Hsieh, P., Halff, H, Redfield, C. (1999). Four easy pieces: Developing systems for knowledge-based generative instruction. *Int. J. of Artificial Intelligence in Education.* Wenzel, B., Dirnberger, M., Hsieh, P., Chudanov, T., & Halff, H. (1998). Evaluating Subject Matter Experts' Learning and Use of an ITS Authoring Tool. *Proceedings of ITS-98*, San Antonio, TX, pp. 156-165. Redfield, C.L., (1996). "Demonstration of the experimental advanced instructional design advisor." In the *Third International Conference on Intelligent Tutoring Systems*, Montreal, Quebec, Canada, June 12-14, 1996