International Journal of Artificial Intelligence in Education (1999), 10, 98-129

Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art

Tom Murray Computer Science Dept., University of Massachusetts, Amherst & School of Cognitive Science, Hampshire College, Amherst, MA
tmurray@cs.umass.edu, www.cs.umass.edu/~tmurray/

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.

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). 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 "on the fly." 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 the last decade 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 & Anderson 1995]. 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 tool. 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 five 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 a few systems have just been released as products or are approaching productization.

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.

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.[1] Over the last two decades relatively fewer papers have been published on authoring tools compared to shells. This paper focuses on authoring tools only.

[1] For examples, see Goodkovsky et al., 1994 (Pop ITS shell), Ikeda & Mizoguchi, 1994 (FITS), McCalla & Greer, 1988 (SCENT-3), Goodyear & Johnson, 1990 (TOSKA), McMillan et al., 1980 (SIPP), Wasson, 1992 (PEPE), Winne & Kramer, 1989 (DOCENT), Jona & Kass, 1997 (GBS architectures).

A bags of tricks vs. a shelf of tools.

Over two dozen ITS authoring systems have been built.[2] 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. 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.[3] Since the field is still in early stages, 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.

Table 1 enumerates seven categories of ITS authoring systems, grouped according to the type of ITSs they produce.[4] Table 2 summarizes the sections below, which describe the strengths and limitations of each type of authoring tool, and the differences among authoring tools within that grouping. Table 2 describes the strengths and limitations of each category, along with a summary of how systems within the category differ.

Table 1: ITS Authoring Tools by Category

	CATEGORY	EXAMPLE SYSTEMS
1	Curriculum Sequencing and Planning	DOCENT, IDE, ISD Expert, Expert CML
2	Tutoring Strategies	Eon, GTE, REDEEM
3	Device Simulation and Equipment Training	DIAG, RIDES, SIMQUEST, XAIDA
4	Domain Expert System	Demonstr8, D3 Trainer, Training Express
5	Multiple Knowledge Types	CREAM-Tools, DNA, ID-Expert, IRIS, XAIDA
6	Special Purpose	IDLE-Tool/IMap, LAT
7	Intelligent/adaptive Hypermedia	CALAT, GETMAS, InterBook, MetaLinks

Early ITS authoring systems fell into two broad categories: those geared toward device simulation and embodying a "learning environments" instructional metaphor, and those based on a traditional curriculum (or courseware) metaphor. Even though some recent systems combine aspects of both perspectives, the majority of 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. Performance-oriented systems (categories 3, 4, and 6 in Table 1) focus on providing rich learning environments in which students can learn skills by practicing them and receiving feedback.

[2] 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.

[3] Because my purpose is to characterize the field as a whole, characterizations of systems are approximate and incomplete. The classification of a system into one category is to illustrate its strengths or contribution to the field, and is not meant to imply that it does not also contain features from other categories.

[4] The authoring tools also differ according to the types of authoring features that they provide, as discussed in the next section. XAIDA is purposely in two of the categories. 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)).

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)	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	Clear representation and pre-defined instructional methods for facts, concepts, and procedures	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; particular design or pedagogical principles can be enforced	Each tool limited to a specific type of tutor; inflexibility of representation and pedagogy	Degree of inflexibility
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 (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 systems.[5]

Below I describe the categories listed in Tables 1 and 2.[6]

[5] 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.

[6] The references to articles describing these systems are in a non-standard format. At the end of this paper references for these systems are listed in a table grouped according to the authoring system. Throughout the paper references to these research projects are given by the name of the system (e.g. RIDES) rather than the name of a paper (e.g. Munro et al. 1997).

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 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). Several of the other categories described below contain these minimal capabilities (such as curriculum sequencing or IU planning) and add additional functionality.

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. However these systems also encode fine-grained strategies used by teachers and instructional experts. Systems in the Curriculum Sequencing category above 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. Also characteristic to systems in this category is the ability to represent multiple tutoring strategies and "meta-strategies" that select the appropriate tutoring strategy for a given situation.

As in Curriculum Sequencing systems, students using tutors built with these authoring tools will see screens of limited interactivity--they will be learning by reading and thinking, rather than learning by doing. However, 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), compared to tutorials built with Curriculum Sequencing systems.

3. Device simulation and equipment training

For tutors built by authoring tools in this category, 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.[7] 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).[8] The most difficult authoring task with these systems is building the device 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.

In contrast to the previous two categories of authoring tools, students using tutors built with tools in this category will be "learning by doing." Introductory or conceptual instruction is absent or limited, and it is assumed that students have a basic familiarity with important concepts and procedures in the domain before using the tutor. These tutors are learning environments in which to practice skills. Specific feedback is given for each skill step, and task difficulty is increased as students progress.

The major differentiating factor among systems in this category is the depth and fidelity of the device 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). The tools also vary widely in the types of devices and physical processes that they can model.

4. Expert systems and cognitive tutors

An important class of intelligent tutors are those that include rule-based cognitive models of domain 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 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.

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.[9] Though the knowledge types and instructional methods vary for different theories, they typically prescribe instruction similar to

[7] These authoring tools have been used to build instructional simulations of mechanical, electrical, and hydraulic systems.

[8] 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.

[9] Here "concepts" usually refers to the meaning of terms or categories of things, not to the broader meaning in "conceptual understanding" of a domain.

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. 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, and 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.

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.

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. Authors are usually given prototypical examples that help them fill in the blanks. 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.

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.

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). As is the case with most web-based systems today, the level of interactivity and fidelity available to the student is low for tutors built with these authoring tools. 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.

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.

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 (described later). 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.[10]

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.[11] The MetaLinks system addresses this tradeoff 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.

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.

[10] The Eon system also has a fairly elaborate set of interface widgets, including hot spots, graphs, tables, clocks, and hierarchical text.

[11] Building the interface for "user friendly" software of any type can take from 50% to 90% of the entire project resources. This is true for projects where interfaces are built using authoring tools as well as with programmed systems.

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 objects 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. The layout of nodes in a hierarchical representation is automatic, while for network representations authors must position the nodes themselves.

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.

RIDES and SIMQUEST include sophisticated WYSIWYG tools for building models of devices and other physical phenomena. In RIDES authors create 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. Also, while RIDES is geared toward training situations, SIMQUEST is geared toward teaching conceptual understanding. 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).

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 have not been shown to be usable for non-programmers.

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).

XAIDA provides maintenance training in four areas: the physical characteristics of a device, its theory of operation, operating and maintenance procedures, and troubleshooting.[12] 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. 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. 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.

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

[12] Tools for the last two categories are only partially complete.

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 behavior, 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) information," and "amount of feedback." For example, a strategy called "Advanced learners " might have high student choice, low preference for specific information, and medium feedback.

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. (See Major 1995 and Murray 1998 for discussions of tradeoffs among tutoring representation methods.)

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.

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.)

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 overlay models, including fuzzy logic (Goodkovsky et al. 1994) and Bayesian networks (Collins et al. 1996). 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).[13] 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.

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:

1. Decrease the effort (time, cost, and/or other resources) for making intelligent tutors;
2. Decrease the skill threshold for building intelligent tutors (i.e. allow more people to take part in the design process);
3. Help the designer/author articulate or organize her domain or pedagogical knowledge;
4. Support (i.e. structure, recommend, or enforce) good design principles (in pedagogy, user interface, etc.);
5. Enable rapid prototyping of intelligent tutor designs (i.e. allow quick design/evaluation cycles of prototype software).[14]

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 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.

[13] Eon's Student Model Editor, while fully implemented, has been only minimally tested.

[14] Another goal sometimes cited but yet to be achieved 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. 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.

1) Scaffolding knowledge articulation with models
2) Embedded knowledge and default knowledge
3) Knowledge management
4) Knowledge visualization
5) Knowledge elicitation and work flow management
6) Knowledge and design validation
7) Knowledge re-use
8) Automated knowledge creation

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 data base application, which in turn is easier than programming from scratch). 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). Systems in the Special Purpose category are the most constraining. IDLE, for example, presents users with a fixed template within which to fill in the blanks.

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.

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.

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.

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 various 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, but are not nearly as common as in off-the-shelf multimedia authoring and CAD software. 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.

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 (including RIDES, D3 Trainer, IRIS, Eon, SMISLE, 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. 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.

The topic or curriculum network authoring tools mentioned previously are the most common knowledge visualization tools in ITS authoring. 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.

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.

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 (a precursor to ED-EXPERT) 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). Also the author did not have to know instructional design theory. 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). Authors need to flesh out content in an order that makes sense to them, and to a depth that makes sense to them. 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.

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 described below). 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. REDEEM includes an agenda mechanism that keeps track of incomplete authoring tasks and prompts the user to complete them.

6. Knowledge and design validation

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 (according to whatever your instructional theory happens to be). One way to allow flexible authoring while maintaining quality is to allow the author to enter what she wants in the way 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.

Inconsistencies in authored information that manifest at run time might be invisible during authoring. REDEEM includes a conflict resolution component that prioritizes instructional rules when they conflict. For example, one rule may look at student mastery and decide to decrease the feedback level, while another might look at topic difficulty and want to increase the feedback level. The system automatically determines which rule takes priority based on a set of instructionally plausible heuristics. [15] RIDES includes a consistency checker : integrated debugger/stepper that allows authors to run and debug their device simulations.

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 using ToolBook, an off-the-shelf multimedia authoring tool. ToolBook 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.

[15] In this example the instructional rules are not really "inconsistent," in that it is perfectly acceptable to create strategies that will result in such run-time conflicts.

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 qualitative failure symptoms entered 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.

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. Though perhaps only two of the authoring tools (RIDES and Electronic Trainer) 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.

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:

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 (this is discussed in a later section).

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."

HOW ARE AUTHORING SYSTEMS BUILT? -- DESIGN TRADE-OFFS.

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 impossible 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 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 or generality of a system comes at the cost of usability.

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.[16]

[16] 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.

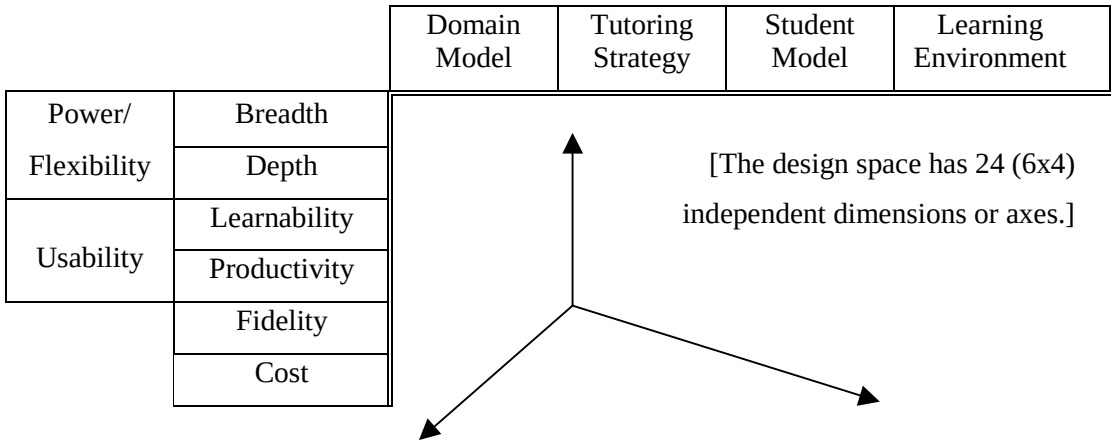

Figure 1: ITS Authoring Tool Design Tradeoffs

Power/flexibility and usability are usually at odds with each other. 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 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.[17] 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. Finally, 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. (See Murray 1998 for more complete discussion of how design tradeoffs effect the authoring of the domain, tutoring, and student models)

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?

[17] 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.

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 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 below.

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 ID-Expert 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).[18]

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 later section.

Who are the authors?

I have alluded to the fact that the nature of intended authors have a critical effect on the design of an ITS authoring tool. Authors may need some skill in several areas in addition to expertise in the subject matter: programming, instructional design, graphic design, and knowledge engineering. Some systems, such as XAIDA, IDLE-Tool, and REDEEM, are aimed at authors with very little training (from several hours to a day), 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 ITSs are complex systems and we should expect users to have a reasonable degree of training in how to use them, on the order of database programming, CAD-CAM authoring, 3-D modeling, or spread sheet macro scripting. 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. Some would like to bring the task of ITS authoring to the average teacher. In contrast, I propose a model in which each company or school has one person trained in using an ITS authoring tool. That person would work on a design team that includes a domain expert, a graphics artist, and an instructional expert. The average teacher should not be expected to design ITSs any more than the average teacher should be expected to author a text book in their field. As with all user-focussed software, an effective ITS design will require several test and re-design iterations to get it right, and all of this requires a significant commitment on the part of the author(s).

It would appear that some believe that by using AI technology we can create authoring tools that build tutors that are an order of magnitude more powerful and flexible than traditional CAI, yet take an order of magnitude less time to develop. To me this seems overly ambitious,

[18] 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).

except for the case of special purpose authoring tools which could use a "mass production" paradigm to easily turn out very similar systems.

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. Authoring tools are designed with certain instructional assumptions and/or knowledge embedded into them, and the design of representations and tools reflect the intended source of the 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**. After all, they are ones working "in the trenches." 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.

Flexible ontologies and meta-level authoring

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 the meta-authoring approach. By a meta-authoring system I mean a general purpose authoring system that could be 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 more usable special purpose authoring systems.

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. Ritter & Blessing (1998) argue for a component-based architecture for ITSs, and champion reusable software modules and standard module communication protocols. From an authoring perspective, it may be possible to build tools that allow authors to put tutors together form libraries of interoperable components.

An important aspect of meta-authoring is the ability to customize the conceptual vocabulary or ontology used to represent knowledge. Many authoring systems use a semantic network representation of content but they differ on the types of nodes and links used. 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. Customizable vocabularies of attributes or primitives are possible for teaching strategies, interface design, and student modeling as well as domain knowledge.

ARE ITS AUTHORING SYSTEMS "REAL?" -- 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. Two of the systems described here have recently become commercial products: Electronic Trainer (a simplified cousin of ID-Expert), and SIMQUEST. [19] 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.

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 usable. 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 is a third or fourth generation system and REDEEM, Electronic Trainer, and Eon are second generation systems.

Table 3 shows a rough estimation of the degree to which various systems have been used. 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. 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

[19] Commercialization requires additional levels of testing, bug-proofing, user documentation, and interface perfection, that, all told, require an order of magnitude more effort that building a solid working prototype. For the most part, the perceived demand for ITSs and ITS authoring tools has not been high enough to justify risky commercialization attempts.

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.[20] SIMQUEST and Training Express are in category 3 because they are robust enough to be commercialized, however there does not seem to be evidence that they have been used to build more than a dozen ITSs.

Table 3: Degree of use of ITS authoring tools

1. Early prototypes and proofs of concept	D3 Trainer, Demonstr8, DIAG, Expert-CML, IRIS
2. Evaluated or used prototypes	CREAM-Tools, DNA, Eon, GTE, IDLE-Tool, LAT
3. Moderately evaluated or used	ISD-Expert/Training Express, REDEEM, SIMQUEST, XAIDA
4. Heavily used (relatively)	IDE, CALAT, RIDES

RIDES is currently the most fully developed system. It 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 (see Munro et al.1997, Towne 1997).

Descriptions of some other system use profiles follows.

- 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.
- REDEEM: Has been used used to build a Genetics tutor consisting of 12 hours of on-line content, which was used in a high school classroom.
- SMISLE (SIMQUEST's predecessor) has been used to author half a dozen systems in various introductory science domains.
- 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.
- 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.
- 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 (results summarized below).

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

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

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.

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. 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.

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.

Authoring tool evaluation

There have been relatively few evaluations of ITS authoring tools. This is in part because the tools have numerous features and it is difficult to measure the effect of individual features and difficult to create control situations against which to compare the results. Also, it could be argued that it is sufficient to demonstrate that a variety of authors have successfully used an authoring tool to produce a variety of ITSs that were actually used by students. There are few authoring tools that have seen enough use to claim such an "existence proof." In addition, existence proofs, while giving the field greater credibility, are of little help in addressing specific research questions. Because ITS authoring tools are still relatively new, summative evaluations, which ostensibly prove that an entire system "works," may be less valuable that 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). A summary of authoring tool evaluations follows.

A formative evaluation of KAFITS was mentioned above, as were estimates of CALAT's productivity, based informally on its widespread use. IDLE-Tool's informal user trials were mentioned above. Conclusions included the need for a higher level view of the curriculum and more conceptually oriented (as opposed to interface or task oriented) help. Practicing teachers using the system differed from the graduate students in their pedagogical competence and their willingness to work within the limits of the template-based authoring. Users found the example-based help feature very helpful.

REDEEM has not been evaluated yet, but its predecessor, COCA was. 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.

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. However, the authoring tool designers concluded that they overestimated the level of experience that would be achieved by typical authors.

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 "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.

By far, the most extensively evaluated ITS authoring tool has been XAIDA.[21] 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. 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 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.

CONCLUSIONS-- HOW EASY CAN IT BE?

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, as we will see. 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. ITSs are complex systems containing embedded models of several types (roughly characterized as domain, teaching, and student models). How easy could authoring an ITS be? Asked another way, what types of tasks do people find relatively easy--what types of tasks can be done without significant skills or training? And can authoring tools reduce the design process to a (perhaps long) sequence of such easy tasks in such a way that they are independent and the interrelations between these easy tasks does not add an order of magnitude to the cognitive skills required? Entering known information into forms and templates is relatively easy; answering prompted questions is easy; selecting from a list of options is relatively easy. Establishing connections between individual items of information, such as prerequisite relationships between topics, or connecting an explanation object with a graphic object, is a bit more cognitively challenging, but still relatively easy. These all involve low-level information and decisions. But, I would argue, 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 reconstructed and seen by a student, i.e. ITS authoring will usually involve *modeling.* 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.

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

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. 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 that yield a quality product. ITSs are clearly not as simple as date books, so an authoring tool analogous to an electronic 'Rolodex' is not possible. 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 *debugged* or modified. This means that the designer has to understand the relationship between non-optimal runtime behavior and information inside the system. All of this leads to the inevitable conclusion that, although authoring tools can scaffold both the underlying representational structure and the design process itself, the author will still need to have an adequate understanding of both the representational structure and the design process.[22] Even though an authoring tool might be able to reduce the process to simple atomic steps done in isolation, some degree of holistic understanding and abstract thinking will eventually have to come into play.[23] As has been mentioned previously, this does not raise the bar unrealistically high, since this level of knowledge and creative/analytical skill is used for many common jobs. But training and commitment is required to do the job. Fortunately, it is sometimes 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.

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 half 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.

REFERENCES

Some of the references and citations in this paper are in non-standard format. Citations to the major authoring tool projects are given by the tool name (e.g. RIDES) rather than the author of a paper (e.g. Munro et al. 1997). References to these papers are shown in the table in the Appendix, which is organized according to the authoring tool.

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, pp. 1-8.

[22] Tools can also be provided to assist with testing and debugging, as described earlier, but it is hard to imagine a system that reduced the entire debugging process to simple steps.

[23] I am not arguing against the merit of special purpose authoring tools, but against the idea that untrained authors can create quality ITSs.

Bell, B. (1998). Investigate and decide learning environments: Specializing task models for authoring tools design. *J. of the Learning Sciences*, 7(1) pp. 65-106.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.

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

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

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

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.

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.

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.

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.Koedinger, K., & Anderson, J. (1995). Intelligent tutoring goes to the big city. Proc. of the International Conference on Artificial Intelligence in Education, Jim Greer (Ed). AACE: Charlottesville, VA pp. 421-428.

Lesgold, A. (1988). Toward a Theory of Curriculum for Use in Designing Instructional Systems. In H. Mandl & A. Lesgold (Eds.), *Learning Issues for Intelligent Tutoring Systems*. New York: Springer-Verlag, pp. 114-137.

Major, N. (1995). Modeling Teaching Strategies. *J. of Artificial Intelligence in Education*, 6(2/3), pp. 117-152.

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.

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.

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.

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.

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

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.

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. (1998). Authoring knowledge-based tutors: Tools for content, instructional strategy, student model, and interface design. *J. of the Learning Sciences*, 7(1) pp. 5-64.

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

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.

Schank, R., Fano, A. Bell, B. & Jona, M. (1994). The Design of Goal-Based Scenarios. *Journal of the Learning Sciences*, 3(4) pp. 305-346.

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 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.

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* (this issue).

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.

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 pp. 234-243.

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.

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.

Winne, P.H. & Kramer, L.L. (1989). Representing and inferencing with knowledge about teaching: DOCENT --- an artificially intelligent planning system for teachers.

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.

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	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	Reinhardt, B., Schewe, S. (1995). A shell for intelligent tutoring systems. In J. Greer (Ed) Proc. of the Int. Conf. on AI in Education. AACE: Charlottesville, VA, 1995.
DEMONSTR8 (& TDK, PUPS)	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	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.
DNA/SMART	Shute, V.J. (1998). DNA - Uncorking the bottleneck in knowledge elicitation and organization. *Proceedings of ITS-98*, San Antonio, TX, pp. 146-155.
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.
EON (& KAFITS)	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. 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. & Woolf, B. (1992). Results of Encoding Knowledge with Tutor Construction Tools. *Proceedings of AAAI-92*. San Jose, CA., July, 1992.
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.

ID EXPERT (& Electronic Trainer, ISD-Expert)	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. (1989). An Instructional Design Expert System. *Computer-Based Instruction*, Vol. 16 No. 3, 95-101. Merrill, M. D. (1987). "An Expert System for Instructional Design. "IEEE Expert, Summer 1987, pg. 25-37. Merrill, M. D. & Li, Z. (1989). "An Instructional Design Expert Sys-tem." *Journal of Computer-Based Instruction*, Vol. 16, No. 3, pg. 95-101.
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)	Bell, B. (1999). Supporting educational software design with knowledge-rich tools. Int. J. of Artificial Intelligence in Education (this issue). 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.
InterBook (& ElM-Art)	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	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)	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* (this issue). 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)	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)	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)	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.
TRAINING EXPRESS	Clancey, W. & Joerger, K. (1988). "A Practical Authoring Shell for Apprenticeship Learning." *Proceedings of ITS-88*, 67-74. June 1988, Montreal.
XAIDA	Hsieh, P., Halff, H, Redfield, C. (1999). Four easy pieces: Developing systems for knowledge-based generative instruction. *Int. J. of Artificial Intelligence in Education*. (this issue) 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