Note: this is a DRAFT.
For the final version see the Journal of the Learning Sciences,
Vol 7, No 1, 1998, pp. 5-64.

Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design

Tom Murray

Center for Knowledge Communication

University of Massachusetts

Amherst, MA, USA

tmurray@cs.umass.edu, www.cs.umass.edu/~tmurray/

Abstract

While intelligent tutoring systems (ITSs), also called knowledge based tutors, are becoming more common and proving to be increasingly effective, each one must still be built from scratch at a significant cost. We have developed domain independent tools for authoring all aspects of a knowledge based tutor: the domain model, the teaching strategies, the student model, and the learning environment. In this paper we describe these tools, discuss a number of design issues and design tradeoffs that are involved in building ITS authoring tools, and discuss knowledge acquisition and representation issues encountered in our work. We also describe how we plan to use these tools (collectively called Eon), including "ontology objects," as a meta-authoring tool for designing special purpose authoring tools tailored for specific domain types.

Table of Contents

1. Introduction

Intelligent tutors. Intelligent Tutoring Systems (ITSs) are computer-based instructional systems that have separate data bases, or knowledge bases, for instructional content (specifying what to teach), and for teaching strategies (specifying how to teach), and attempt to use inferences about a student's mastery of topics to dynamically adapt instruction. ITS design is founded on two fundamental assumptions about learning. First, that individualized instruction by a competent tutor is far superior to classroom style learning because both the content and style of the instruction can be continuously adapted to best meet the needs of the situation [Bloom 1956]. Second, that students learn better in situations which more closely approximate the situations in which they will use their knowledge, i.e. they "learn by doing," learn via their mistakes, and learn by constructing knowledge in a very individualized way [Bruner 1966, Ginsburg & Opper 1979]. Individually paced instruction and frame-based computer aided instruction (CAI) comprised early attempts to provide adaptive instruction, and, though successful for some types of learning, fell short because their learning environments were too contrived and their ability to adapt was limited to branching between static screens. ITSs, also called knowledge based tutors, use techniques that allow automated instruction to come closer to the ideal, by more closely simulating realistic situations, and by incorporating computational models (knowledge bases) of the content, the teaching process, and the student's learning state [Wenger 1987].

The need for ITS authoring tools. In the last half decade ITSs have moved out of the lab and into classrooms and workplaces where some have proven to be highly effective as learning aides [Shute and Region 1990]. For example, students working with an Air Force electronics trouble shooting tutor for only 20 hours gained proficiency equivalent to that of trainees with 40 months (almost 4 years) of on-the-job experience [Lesgold et al., 1990]. In another example, students using the LISP tutor [Anderson 1990] completed programming exercises in 30% less time than those receiving traditional classroom instruction and scored 43% higher on the final exam. While intelligent tutoring systems are becoming more common and proving to be increasingly effective, each one must still be built from scratch at a significant cost. Little is available in terms of authoring tools for these systems. Authoring systems are commercially available for traditional CAI and multimedia-based training, but these authoring systems lack the sophistication required to build intelligent tutors. Commercial authoring systems excel in giving the instructional designer tools to produce visually appealing and interactive screens, but behind the presentation screens is a shallow representation of content and pedagogy. Some [Youngblut, 1995] say there are too few ITS's to make informed design decisions about ITS shells and authoring tools. There is certainly a grain of truth to this, but it is also true that so few ITSs exist for evaluation and generalization because they are so difficult and expensive to build. Some of us must build chickens, and some of us must build eggs, all artifacts limited by the nascency of the field, if the field is to mature.

Our approach to developing ITS authoring tools is motivated by issues of pragmatics and usability. Rather than starting with a laboratory-based AI tutoring system and asking how it can be generalized to produce more generic shell, as is the case in the design of some ITS authoring shells, our design base-line is commercially available and widely used authoring systems for traditional computer-based teaching, such as Authorware and Icon Author. Our goal was to extend rather than replace the capabilities afforded by such systems, to preserve the level of usability and some of the authoring methods instructional designers have become familiar with, and add additional tools, features and authoring paradigms to allow more powerful and flexible tutors to be built.

In this paper we will first describe how the knowledge based approach to computer based instruction differs from the more traditional approach. Then we will describe a suite of authoring tools we have developed for building knowledge based tutors. Next we will discuss design tradeoffs involved in building authoring tools. Finally we will summarize what was learned about representing pedagogical knowledge in knowledge based tutors and how these ideas have become part of the design of the underlying representational scheme of authoring tools.

2. From CAI Story Boards to ITS Knowledge Bases

There is a large established user population using COTS (common off the shelf software) to create instructional software. We will call these users instructional systems designers (or instructional designers, though in other contexts the term may have a more specialized meaning). We would like to facilitate cost-effective ITS production by these instructional designers, as well as by those who have traditionally built ITSs from scratch (primarily those in academic and industrial research labs). Empowering instructional designers to build these more powerful systems requires new tools *and* a paradigm shift in the way many of them conceptualize instructional systems. However, we want this shift to be accessible, incremental, and evolutionary. For this reason, on the surface many of our tools have a look and feel similar to COTS tools, yet allow additional levels of abstraction, modularity, and visualization necessary for producing an ITS.

Specifically, we proposed that moving from CAI authoring to ITS authoring involves a fundamental paradigm shift from "story board" representations of instructional material to more powerful and flexible "knowledge based" representations. The basic concept is not new; in fact, it is fundamental to all AI work. Our contribution is in fleshing out how the knowledge based paradigm can be best presented to empower instructional designers.

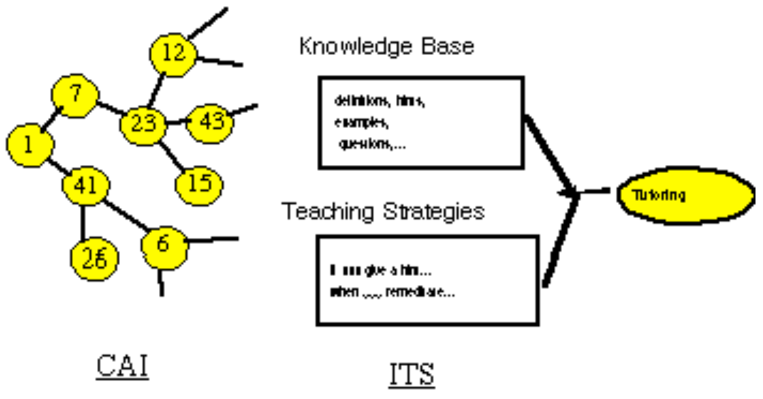


Figure 1: CAI story board vs. ITS knowledge base

Commercially available authoring systems assume and support a representation of instructional content and instructional flow that is explicit, non-modular, and fairly linear. At a conceptual level, the instruction is specified like this (see Figure 1): "Bring up screen # 41; If the user clicks on button A, then go to screen # 26." Though branching is allowed, each branch must be explicitly specified. Adding a new topic or question involves explicitly encoding the branches to this content. Designing new content requires a duplication of efforts. We call this paradigm for designing instructional systems "story boarding" because it is based on enumerating all of the screens and the explicit links from each screen to the next. In contrast, in a knowledge based tutor the instructional content is separated from the specifications of how and when that content is presented to the student, so that the content can be used and re-used in multiple ways. Specifying how and when the content is to be presented is done with generic, reusable teaching strategies. For example, a CAI system may be programmed to give two hints for wrong answers to exercises. If the author later realizes that three hints are necessary, he has to go back and change every link associated with giving hints. In contrast, in the knowledge based model, there is one strategy specifying how and when hints are given, so that changing from 2 to 3 hints is a matter of making one change.

Stretching COTS to the limit. Actually, many "power users" of commercial authoring tools such as Authorware have already begun this paradigm shift. They have built layers, shells, or macros on top of the existing authoring systems that capture the repetitive or modular format of their instructional application, so they don't have to repeat the same work with every new topic or question. But these additional layers are usually large and awkward patches that result in increased authoring efficiency, but at a loss of generality, since they are created for a specific application. The powerful features of the authoring tool are compromised, because commercial packages allow, but do not support, this type of abstraction. For instance Authorware has an edit-in-place feature that allows the designer to pause the tutorial, click on text or graphics, and edit right on the screen. This is extremely useful, because the screen may contain a number of text and graphic items that were brought up at different times and specified in distant portions of a large curriculum control structure. With edit-in-place the user does not have to search through this structure to find each piece. But when a shell incorporating more general procedures that access a database of questions is built on top of Authorware, the edit-in-place feature is lost. When power users pause in such an augmented system to correct, say, a spelling error in a question, the display shows a variable called "the-current-question," rather than the actual text of the question sought. To edit the actual text they have to switch to a different program (the database program), search for it in the data base, then edit it and return to the original program to see if their change resulted in the desired effect on the screen. This is one of many ways in which the powerful features of CAI authoring tools are compromised or lost when they are coerced into a form which allows separation of subject matter and instructional strategy.

2.1 Benefits of the Knowledge Based Approach

The knowledge based approach includes the following design principles:

Represent instructional content and instructional strategies separately.

Modularize the instructional content for multiple use and re-use.

Create generic teaching strategies that can be used with different instructional content.

Explicitly represent abstract pedagogical entities (such as "topics").

Design at the pedagogical level, as opposed to the media level, when possible.

Designing tutoring systems in this way has many advantages over the traditional CAI design paradigm:

1) The behavior of the tutor can be easily modified. As shown above, to change when hints are given, only a single "give hint" strategy needs to be changed and this effects how hints are given through the entire curriculum.

2) The content of the knowledge base is modular and can be used for several purposes. For example, a "topic" object in the knowledge base can contain information about how to teach itself, summarize itself, give examples of itself, introduce itself, and test the student's knowledge of itself. The same topic can then be used in many parts of the tutorial, for example, giving a summary at some point and teaching itself later on.

3) Instruction can be much more learner-centered, since modularization allows students to navigate to the topics they want to learn, and to ask for hints, examples, etc.

4) The authoring tools for a knowledge based system can be built to provide multiple and abstracted views of the instructional content. Instructors need be able to easily view, inspect and navigate through these knowledge bases.

2.2 Designing at the Pedagogical Level

In traditional CAI instructional actions are encoded using building blocks at the level of the media: text, pictures, button clicks, etc. In contrast, knowledge based tutors can facilitate the design of instructional actions using pedagogically relevant building blocks. For example: "give a hint," or "teach the prerequisites". Designing instruction using building blocks such as "hint," "prerequisite," "if-confused," "known," and "summarize" is much more powerful than designing instruction at the level of "show video," "present picture" or "wait for the button click" The instructor can conceptualize the curriculum at a more natural level of abstraction. An instructional strategy in the intelligent tutor might be: "if the current topic is conceptual and the student is doing poorly, give several examples." Alternate strategies can be created, so that the appropriate strategy can be used according to the needs of the student (e.g. learning style or mastery of the current topic) or the pedagogical characteristics of the content being taught (e.g. whether it is procedural or conceptual information).

2.3 Preliminary Research

This work is an outgrowth of an earlier investigation of tools for the acquisition of ITS domain and teaching knowledge which culminated in a dissertation thesis and a system called KAFITS (Knowledge Acquisition Framework for ITS) [Murray 1991]. The problem being addressed by this work was the gap between the ITS research community and the educational research community, as ITSs of increasing complexity were being developed without tools that would allow practicing educators to participate knowledgeably in the design process [Clancey & Joerger 1988]. Several tools were created, including a semantic network editing tool for visualizing topics the their relationships, and an editor for creating instructional strategies in the form of Parameterized Action Networks (similar to ATNs but replacing states with actions to create a planning rather than parsing formalism). Our sixteen month case study of three educators using the tools to build a 41 topic tutor for high school Statics (representing about six hours of on-line instruction) was reported in [Murray 1993, Murray & Woolf 1992a, and Murray & Woolf 1992b]. Figure 2 shows the productivity data for the domain expert (physics teacher), the "knowledge base managers" who did most of the data entry, and the knowledge engineer who did most of the knowledge representation work. Figure 2 gives an indication of the relative amounts of time spent on different activities:

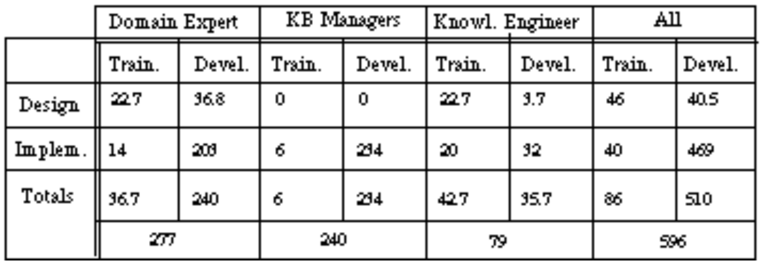
An analysis of productivity indicated that it took about 100 person-hours of development time per hour of instruction (596 total hours for 6 hours of instruction, or about 15 hrs per topic);

The above data was interpreted cautiously but we did note that it compared favorably with the 100 to 300 hours often given for building traditional (non-intelligent) CAI;

The domain expert invested a significant time in designing and debugging the tutor, 47% of the total, while the knowledge based managers worked 40% of the total time, and the knowledge engineer only worked 13% of the total time;

Design constituted abut 15% of the total time, and implementation the other 85% (time spent on formative evaluation is not included in the data); and

Training totaled about 15% of the total time (vs. 85% for development).



	Domain Expert		KB Managers		Knowl. Engineer		All	
	Train.	Devel.	Train.	Devel.	Train.	Devel.	Train.	Devel.
Design	22.7	36.8	0	0	22.7	3.7	46	40.5
Implem.	14	203	6	234	20	32	40	469
Totals	36.7	240	6	234	42.7	35.7	86	510
	277		240		79		596	

Figure 2: Person-hours vs. Participant Role in building the Statics Tutor

We also studied the design process itself, and well as several usability and representational issues. We learned of the importance of 1) providing clear visual representations of the underlying concepts and structures of the system; 2) providing features which reduced cognitive load on working memory (by reifying information and structure) and long term memory (by providing remindings); 3) facilitating opportunistic design by not forcing the user to make decisions in a rigid order; and 4) allowing quick movement and iteration between testing and modifying what is being built.

In sum, this work, combined with experience the author later gained using COTS authoring tools to build training systems in industry, formed the conceptual foundation for the Eon authoring tools project. The productivity data, though only from a case study of building one tutor, have given us some guidelines for estimating productivity figures on other projects, and also yielded the encouraging suggestion that intelligent tutors can be built with human resources comparable to building CAI.

3. The Eon ITS Authoring Tools

Next we will describe the authoring tools by showing how they were used to build a tutor which teaches how a refrigerator works. In later sections we will discuss knowledge representation issues and make comparisons with other ITS authoring projects.

The Refrigeration Tutor. The Refrigeration Tutor was designed to be used in a new UMass course called "Engineering, the Human Enterprise," a sort of "engineering for poets" class that has as its goal to give non-technical students a sense for the history, concepts, processes, and wider social aspects of engineering and design. A large section of the class will be based upon the evolution of the needs for and engineering/scientific responses to societies needs for refrigeration. The class is scheduled to be first taught in the Fall of 1997.

The Refrigeration Tutor teaches about the operation of the refrigerator, an a general understanding of the relationships between temperature, pressure, volume, energy, and phase so that students can grapple with such questions as:

- How is it that we can cool something off by putting energy *into* it?

- Why is the refrigerant boiling in the *colder* part of its cycle?

- Why is it that if you leave the refrigerator door open in the kitchen the room gets hotter?

- Why would a fluid such as water not work as well as Freon and other materials used as refrigerants?

The Refrigerator Tutor is relatively simple (its interactions are mainly multiple choice and point-and-click interactions) and it does not fully utilize much of Eon's functionality, but it forms a good example case for describing the tools due to this simplicity. Since our goal is to describe the authoring tools, not the tutor itself, we will only describe enough of the tutor to show all of the tools at work.

3.1 Design Steps and Authoring Tools

We will present the tutor design process in an order which best suits the tools description, but in fact the design of the tutor happened with several stages taking place in parallel, and in a much more opportunistic fashion. The Eon tools can be used in any order that is logically consistent. Top down, bottom up, and opportunistic design approaches are possible. Eon does not walk the author through a series of design steps, nor does it have "wizards" that instruct authors in certain steps (though these would be useful). As explained later, Eon is expected to be used by design teams with at least one person having a moderate level of training (though in Section 6 we discuss methods for scaffolding the process). The Refrigeration Tutor was authored primarily by two members of our lab, who worked closely with the domain expert, who did not have the time to learn to use many of the tools (except the Contents Editor).

Design of a tutor can start from a concrete, bottom up orientation, designing the screens and interface the student will use, and sketching out story boards of typical interactions. However, in our example of authoring the Refrigeration Tutor we will start top-down, with the most abstract components of the tutor, i.e. the topics and the topic network which comprise the domain model in Eon tutors. Following that we will jump down to the concrete level and describe tools for authoring the student screens and learning environment. Next we will describe how the student model is authored. Finally we will describe how instructional strategies are authored using flowlines. Figure 3 shows the relationship between the knowledge bases in Eon (domain model, teaching model, interface specification, and student model) and the authoring tools used to build them.

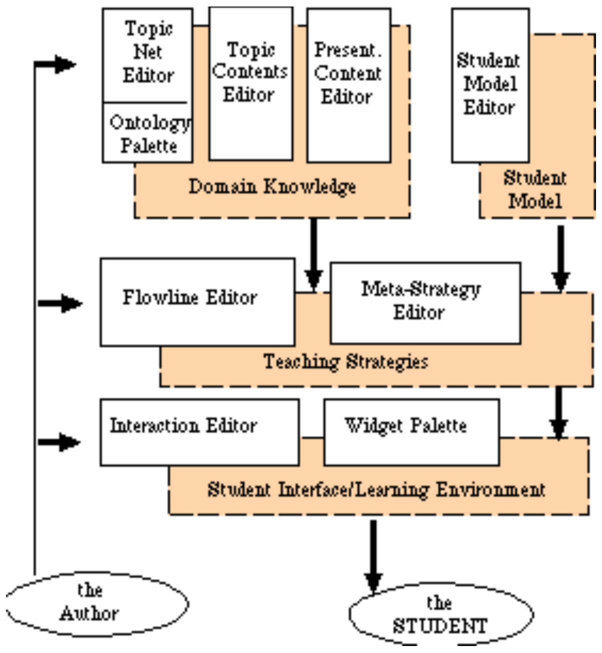


Figure 3: Eon Authoring Tools and Knowledge Bases

3.2 Authoring the Domain Model

A major difference between ITSs and conventional CAI systems is that ITSs contain an inspectable model of domain expertise. This expertise can be either runnable (usually a rule-based expert system) or non-runnable. The domain model also contains two types of information: performance information, which represents knowledge about the subject matter and problem solving in the domain, and pedagogical information (information relevant to learning or teaching the content---we discuss the need for representing pedagogical knowledge at length in Section 5.1). The domain model in Eon consists of a semantic network representation of the units of knowledge (called topics) that the tutor is designed to teach. Although this semantic network may represent domain expertise such as part-whole relationships and sequences of steps, the focus is on pedagogical information, i.e. links such as part-of and prerequisite which can be used in sequencing the instruction.

The Topic Ontology Palette

The first step in building a tutor is to map out the learning goals or topics and their relationships in the form of a topic network, using the Topic Network Editor. However, prior to this we must define the "topic ontology," which specifies the types of nodes and links allowed, and also the types of properties topics can have. Figure 4 shows the Topic Ontology Palette which is a visual representation of the topic ontology and is used to draw topics onto the topic network. The topic ontology defines a number of "topic types" (or knowledge types), which are shown in a pop-up menu on the palette, with each topic type having its own shape. For the Refrigeration Tutor we defined these types: Fact (square), concept (pentagon), principle (triangle), physical component (circle), and unspecified (oval). The topic ontology also defines a number of "topic properties." For the Refrigeration Tutor we defined "importance" (with allowed values one to five associated with the topic node's color) and "difficulty" (with allowed values "easy, moderate, difficult" associated with the topic node's border color). The topic ontology also defines "topic link types," and for the Refrigeration Tutor we defined "prerequisite" (red), "part-of" (black), "is-a" (blue), and "context-for" (green). The topic ontology also defines "topic levels," which we will discuss later, and which are not shown graphically on the Ontology Palette or on the topic network.

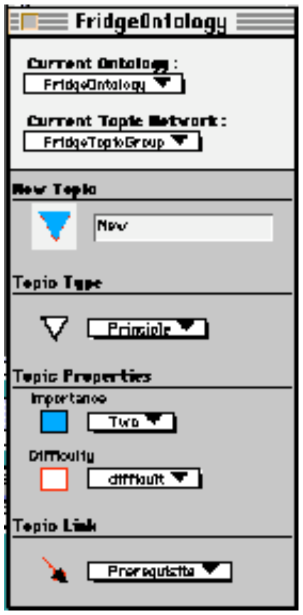


Figure 4: Topic Ontology Palette

Eon does not yet contain a tool for defining the topic ontology itself, and this must be done via a text file that contains lines like this:

```
NewTopicOntology <name>

NewTopicLinkType <name>

NewTopicProperty <name> <allowedvalues>

NewTopicLevel <name>

NewTopicType <name> <allowedLinkTypes>
<allowedProperties> <allowedLevels>
```

Additional information is included in the text file to define how graphic properties (shapes, colors, etc.) are associated with these semantics.

When this file is loaded it defines a topic ontology, and the Topic Ontology Palette (also called simply the "Topic Palette") shows a visualization of the ontology. To create a new topic the author selects the desired topic type and property values. The appropriate graphic attributes are shown in the "new topic" node on the palette. The author types the desired name for the new topic to the right of this new node. Finally, to instantiate this new topic the author drags the node onto the Topic Network Editor and drops it at the desired location.

The Topic Network Editor

Once the topic ontology is defined the author can use the topic palette to draw out the topic network. Figure 5 shows the Topic Network Editor and the topic network developed by our domain expert. New topics are created as described above, and they can be repositioned on the screen by dragging them. To create links between the topics the author selects the link type at the bottom of the ontology palette, then clicks the "link creation tool" button on the Topic Editing Tools Palette, shown to the bottom left in the figure, and defines a segmented line from one topic to another. In the figure some lines are "part of" links and some lines are "prerequisite" links (they are black and red, respectively, on the computer screen).

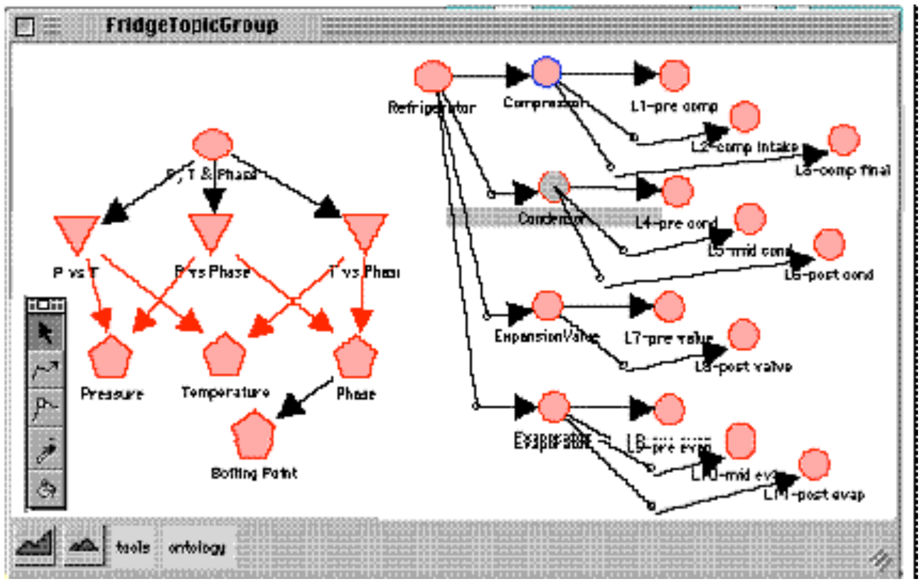


Figure 5: Topic Network Editor

The flow of instruction in the Refrigeration Tutor is organized around the parts of the refrigeration cycle. The student is taken on several trips around the cycle, each one having more difficult information and questions. The main components involved are the compressor, which compresses the gaseous refrigerant and heats it up, the condenser, which cools down the gas and turns it into liquid (fanning the heat to the outside), the expansion valve, which cools down the refrigerant, causes a drop in pressure, and a partial phase change back to gas, and the evaporator, which absorbs heat from the inside of the refrigerator, causes the refrigerant to boil and become completely gaseous again.

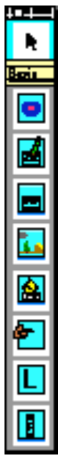
The domain expert has identified eleven important locations to focus on in this cycle, and these are shown in the topic network as sub-parts of each of the components (labeled Ln-XXX, to the far right in Figure 5). The main concepts being learned are also shown in the topic network, including understanding the relationships between pressure, temperature, and phase. The student model (explained later) is used to infer the student's mastery of each of these topics. In some of the tutors built with Eon, the topic network is essential to sequencing the material that the student sees. In such tutors (the Statics tutor, for example) the sequencing of topics, and thus instructional material, is determined dynamically using a network traversal procedure defined by the author (using flowlines, described later). However, in the Refrigeration Tutor, sequencing is fairly simple, as the eleven locations are cycled through three times giving questions with more difficulty each time.

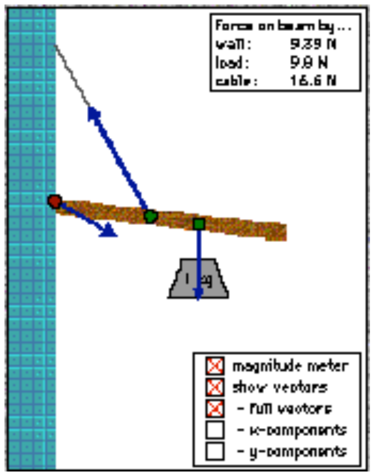
3.3 Authoring the Learning Environment

The aspect of most ITS authoring shells that is most sorely lacking, in relation to COTS CAI authoring systems, is student interface design, which for ITSs is also called learning environment design. Eon allows authors to completely customize the student interface to create highly interactive learning environments.

The Interaction Editor and Interface Extensibility

Eon's Interaction Editor tool contains a hierarchical pallet of user interface components called widgets. There are simple widgets such as buttons, text, pictures, sliders, movies, and hot-spots, and more complex widgets such as multiple-choice dialogs, tables, and graphs (there are a total of 26 widgets in six categories: Basic, Controller, Geometrical, Complex, Special, and Custom). Widgets are selected for drawing onto the interaction screen using the Widget palette (see Figure 6, which shows the Basic widgets) .





Force on beam by...
wall: 9.39 N
load: 9.8 N
cable: 16.6 N

☒ magnitude meter
☒ show vectors
☒ - full vectors
☐ - x-components
☐ - y-components

Figure 6: Widget palette **Figure 7:** Crane Boom Widget

The Widget Palette is extensible via the "Custom" category. Arbitrarily complex widgets can be programmed outside of Eon, and "dropped in" as needed for particular domains. These custom widgets can be device simulations or whole learning environments. For example, our Statics Tutor has a "crane boom" widget (see Figure 7) which lets students manipulate positions of objects and cables and observe the resulting static forces. In order for custom widgets to inter-operate with the rest of the Eon tools, they must adhere to a simple protocol which involves specifying the "parameters" used to set a widget's properties, and the student "events" that the widget recognizes. The events can be simple as "button-pushed," or require some processing as in a multiple choice widget's "correct answer" event, and can be arbitrarily sophisticated, as in a "student has moved the load past the tension limit of the cable" event in the crane boom widget.

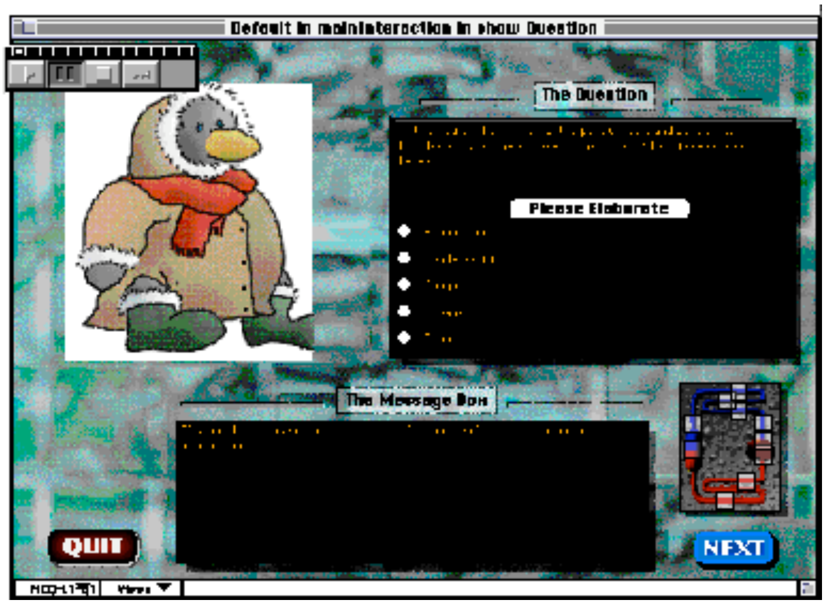
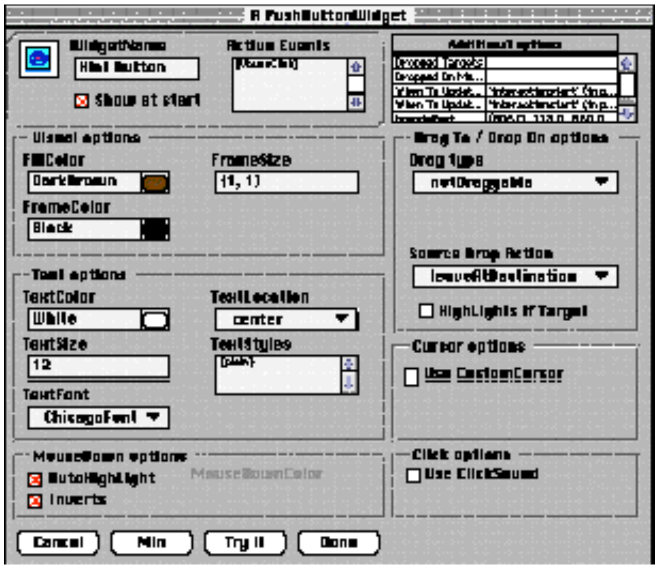


Figure 8: Interaction Editor

Widgets are selected from the Widget Palette and drawn onto the Interaction Editor (see Figure 8). Figure 8 shows the screen design for the Refrigeration Tutor's multiple choice questions. It includes a picture, text areas for the question and answer choices, a text area for hints and an "elaboration," an icon-ized map of the refrigeration cycle (lower right) onto which a "you are here" indicator is placed. This is a reusable template screen which was built using text, graphic, multiple choice, and button widgets from the widget palette.

Eon distinguishes two types of widget attributes: "options" and "properties." Options comprise most of the widgets attributes, will usually remain as initially set during the tutorial, and are set using the widget's options dialog. Figure 9 shows the options dialog for a push-button widget, and gives an indication of the complexity of Eon's widgets. Widget *properties* are the small set of the most important widget attributes that deal with student input or instructional content, for example, the text of a text widget, and the picture in a graphic widget. Properties are likely to change during the course of the tutorial. The widget properties of the multiple choice widget are: the question text, the answer choices text, the correct answer index, and the answer selected by the student (which is set at run time by the student, not the author). Both options *and* properties can be manipulated dynamically at run time (using a scripting language), allowing for dynamic screens with content generated on the fly. Properties, however, have additional flexibility as described below.



WidgetName: Hint Button
[✓] Show at start
Action Events: (MouseClick)
Dropped Targets:
Dropped On Me...:
When To Updat...: 'InteractionStart' (in p...
When To Updat...: 'InteractionStart' (in p...
FillColor: DarkBrown
FrameSize: (1, 1)
FrameColor: Black
TextColor: White
TextLocation: center
TextSize: 12
TextStyles: (plain)
TextFont: ChicagoFont
Drag type: notDraggable
Source Drop Action: leaveAtDestination
[ ] HighLights If Target
[ ] Use CustomCursor
[✓] AutoHighLight
[✓] Inverts
[ ] Use ClickSound

Figure 9: Options Dialog for a Push-button Widget.

Content Generation and Re-use

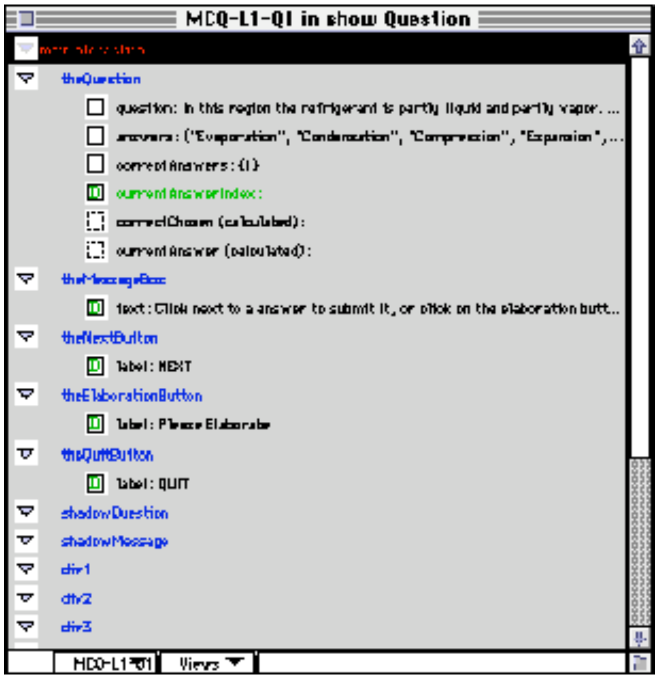
Widget properties, such as text and graphics, can be set in several ways:

1. Single value. E.G. a graphic widget always has the same picture.

2. Calculated value. The value can be attached to a script. E.G. each time the picture is shown in the tutorial, the script is evaluated to produce a new value (e.g. a pointer to a new picture).

3. Template. Screens containing widgets can be specified as "templates" which get reused. In such cases the properties of the widgets are stored in "Contents" objects (described below).

Using template interactions the author can create re-usable screen formats, and then create "Content" objects to fill in the template. Figure 8 shows one of the template-based interaction screens authored for the Refrigeration Tutor. The figure shows the default contents of this template. About fifty Contents objects have been created to fill in this template, each with its own question, picture, explanation, and hints. As the author adds widgets to the screen a data base entry form called the Contents Editor is created showing the properties of the widgets. Figure 10 shows the Contents Editor for the multiple choice template screen, with the contents object MCQ-L1-Q1 shown. The Contents Editor shows all of the properties of all of the widgets on the student screen. Some properties are designated "D" to use the default value (a "C" would indicated that the value is computed dynamically, i.e. attached to an author-defined script). The author can edit widget properties directly from the Interaction Editor (which is WYSIWYG), or she can edit the properties using the Contents Editor (which is like a database form). In the bottom left of the Contents Editor and the Interaction Editor (of template interactions) is a pop-up menu listing all of the contents that have been created for this template (MCQ-L1-Q1 is one of them), an item "default" to view the default contents, and an item "New" to create a new contents item. The "Views" button to the right of this button is for easy navigation between the three tools which constitute three views of this domain content: the Interaction Editor, the Contents Editor, and the Flowline Editor (see below).


MCQ-L1-QI in show Question

theQuestion
question: In this region the refrigerant is partly liquid and partly vapor. ...
answers: ("Evaporation", "Condensation", "Compression", "Expansion", ...
correctAnswers: (1)
currentAnswerIndex:
correctChosen (calculated):
currentAnswer (calculated):
theMessageBox
text: Click next to a answer to submit it, or click on the elaboration butt...
theNextButton
label: NEXT
theElaborationButton
label: Please Elaborate
theQuitButton
label: QUIT
shadowQuestion
shadowMessage
div1
div2
div3

Figure 10: Contents Editor showing widget properties

To summarize, content reuse is facilitated by allowing an author to create interactive screens as templates. The widgets are drawn once using the Interaction Editor, and then numerous Contents are created using the Contents Editor. This facilitates "canned" material to be entered into the domain knowledge base. Additional flexibility is available by attaching widget properties to scripts (as opposed to canned material) so that interactive screens can be generated on the fly.

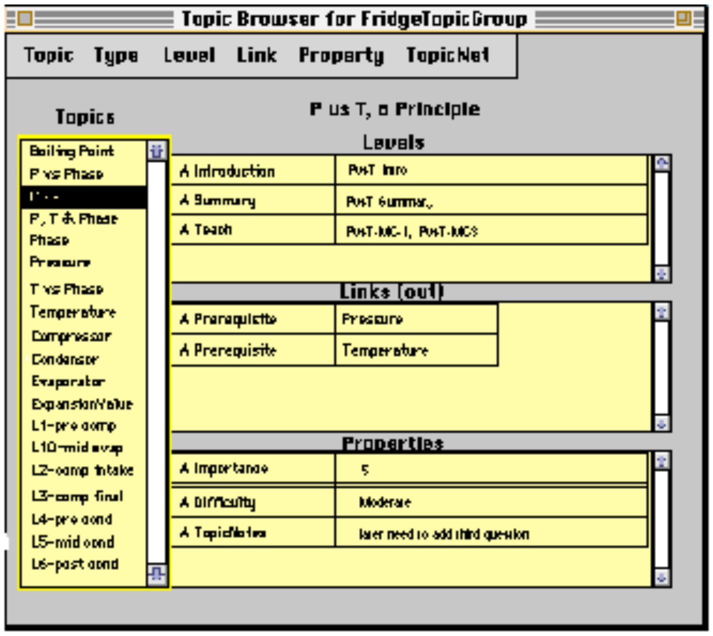
3.4 Connecting Topics to Contents

Thus far we have shown how the author works at the abstract level of the curriculum, i.e. in mapping out the topic network, and at the concrete level, i.e. designing the interactive screens and (if they are template screens) filling in data base forms to define the Content objects that fill in the screens with domain content (the questions, hints, explanations, etc.). In this Section we will describe how the abstract level is linked to the concrete level. Basically, Content objects are assigned to the topics, but it is a bit more complex than that, and to explain how, first we must add one detail about topics left out of the earlier discussion: Topic Levels.

In Eon topics need not refer to a single monolithic entity, but have an extra level of internal structure called Topic Levels. Each topic has one or more Topic Levels which can specify different aspects or uses for the topic, for instance: introduction, summary, teach, test, beginning level, difficult level, etc. In the Refrigeration tutor the topics levels "Introduce," "Teach," and "Summary" were defined. A list of Content objects are assigned to each Topic Level for each topic. Thus when we want to "introduce a topic" we give the Contents in its Introduction level, and when we want to teach a beginning level of a topic we give the Contents listed in its Beginning level. The list of Contents in a Topic Level is usually thought of as a sequence of contents to present, but it can also be a *set* of applicable Content objects, and selection and sequencing of these can be left to the teaching strategy (for example choosing randomly from the set). The topic levels used in a tutorial are defined in the Topic Ontology object, described previously.

Thus far we have described the curriculum contents as it is stored in modular, declarative units: as Topic and Content objects. Later we will discuss teaching strategies, which are used to determine how this declarative information is used: i.e. how the topics are sequenced, when each topic level is given, how the Content objects are sequenced, when the interactive screens are shown, and how student behavior is responded to. For now, we will only remind the reader that all of the domain information discussed thus far is strategy-independent. It can be sequenced in arbitrary ways, as specified in the teaching strategies.

The Topic Contents Browser. The Topic Contents Browser is an authoring tool that shows another view of the topic space, a tabular one as opposed to the graphic view given in the Topic Net Editor. Figure 11 shows the Topic Contents browser for the topics in the Refrigeration Tutor. The Topic Contents Browser shows a list of all of the topics, and the links and Properties of the selected topic, and, unlike the Topic Network Editor, it shows the Topic Levels and the Content objects associated with each level.



Boiling Point
P vs Phase
P, T & Phase
Phase
Pressure
T vs Phase
Temperature
Compressor
Condensor
Evaporator
ExpansionValue
L1-pre comp
L10-mid evap
L2-comp intake
L3-comp final
L4-pre cond
L5-mid cond
L6-post cond

A Introduction	PvsT Intro
A Summary	PvsT Summar,
A Teach	PvsT-MC-1, PvsT-MC3

A Prerequisite	Pressure
A Prerequisite	Temperature

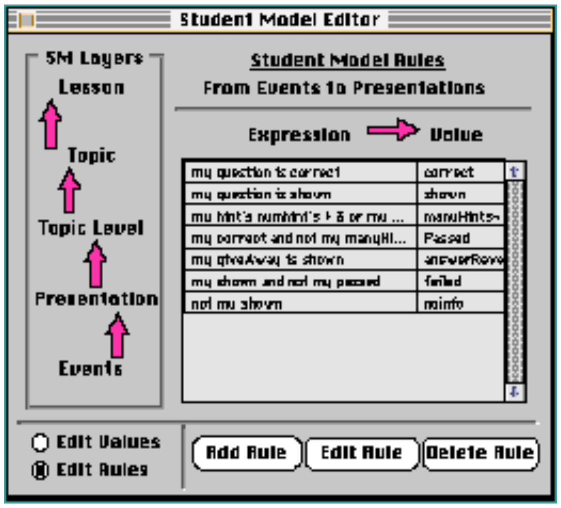
A Importance	5
A Difficulty	Moderate
A TopicNotes	later need to add third question

Figure 11: Topic Contents Browser

3.5 Authoring the Student Model

The student model is the component of the system that keeps track of student behaviors and makes inferences about what the student knows. Eon uses a variation of an "overlay student model" [Wenger 1987] in that mastery values are calculated to correspond with each topic. The Eon student model can also be used as a "bug library," since topic types for "mis-knowledge" such as misconceptions and buggy procedures can be defined to keep track of known classes of common deficiencies. We call our student model a "Layered Overlay Student Model" because values are inferred a several "decision layers," as shown in Figure 12. Most overlay student models assign values to topics only. In Eon values are assigned to objects at several decision layers: Lesson, Topic, Topic Level, Presentation Contents, and Events. Objects at each layer are composed of objects at the next lower layer, for example, running a lesson will invoke the teaching of a number of topics, teaching a topic will run a number of its topic levels, and each topic level consist of a number of Contents (representing student interactions or blocks of information). Within a Content, a multiple choice question for example, a number of low level student and tutor events will occur, such a selecting an answer or asking for help (a student event), or giving a hint (a tutor event). As shown in the Student Model Editor

(Figure 12) objects at each level can have a value. The value of objects at any level is determined by the Student Model rules written for that level. These rules specify how the value of an object depends on the values of the objects at the next lower level.



Expression	Value
my question is correct	correct
my question is shown	shown
my hint's numhint's > 3 or my ...	manyHints
my correct and not my manyHi...	Passed
my giveAway is shown	answerGiven
my shown and not my passed	failed
not my shown	noinfo

Figure 12: Student Model Editor

Figure 12 shows the rules for mapping from the Events level to the Presentation Contents level for the Statics Tutor. For the Refrigeration tutor, some example rules are:

Topic Level to Topic: `my Teach Level is KNOWN OR all of my Parts are KNOWN ==> KNOWN my Teach Level is SHOWN OR my Introduce Level is SHOWN OR my Summary Level is SHOWN ==> SHOWN`

Presentation to Topic Levels: `greater than 80% of my Presentation Contents are CORRECT ==> KNOWN any of my Presentation Contents is SHOWN ==> SHOWN`

The student model is used to make the tutorial adaptive to the student's inferred state. This is accomplished by referring to student model values in decisions in teaching strategies (and meta strategies), for example, decision branches predicated upon whether a topic is "mastered" or whether a Contents has been "shown" already. The refrigeration Tutor uses the student model to ask fewer easy questions if a topic is near mastery, and to give more hints if the topic is far from mastered.

The vocabulary of terms used to define the student model values (e.g. "known," "mastered," "suspected misconception") is customizable for each tutorial using the student model editor. We have found the current Student Model Editor to be too restrictive however, since values and rules can only be defined on a per-decision-level basis, i.e. every item in a decision level has the same rules and the same set of allowed values. We are working on an extension to this system to allow rules to be assigned to groups of objects. We would like some topics to use different student modeling rules than others, for example, in the Refrigeration tutor we would like the topics representing the important concepts (to the left in Figure 5) to use a different rule set than the topics representing the components and locations within the refrigerator (to the right in Figure 5).

3.6 Authoring the Teaching Model

To represent teaching strategies we use a flowline paradigm, a graphically portrayed procedural representation of strategic knowledge, which explicitly shows structural and contextual control information. Our Strategy Editor (also called the Flowline Editor) has a look and feel similar to commercially available authoring tools such as Authorware and Icon Author. Eon flowlines, like procedures in programming languages, can have input parameters, local variables, and can return values. All variable referencing and naming is facilitated by menus and drag and drop tools, so the author does not have to memorize or type in references to them.

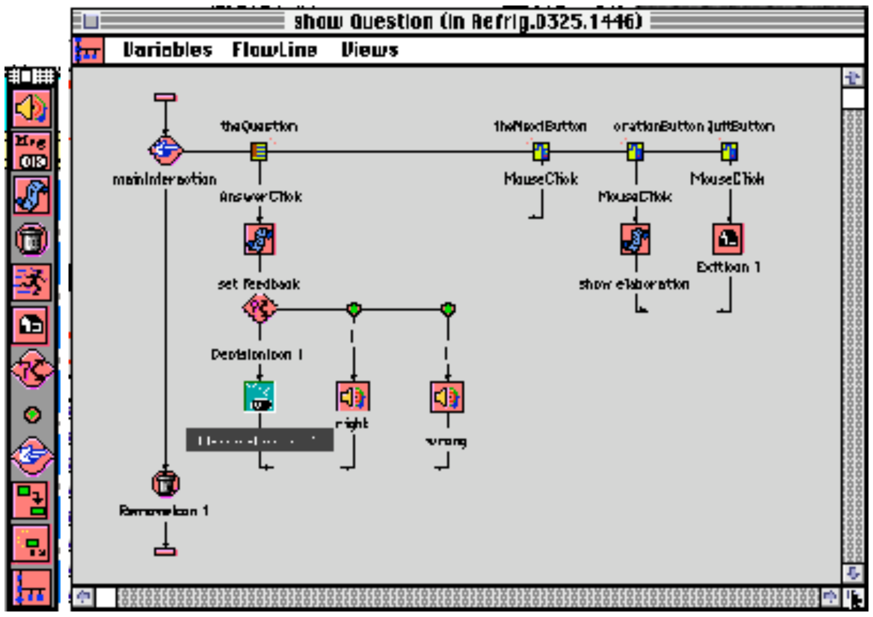


Figure 13: Icon Palette and Flowline

Figure 13 shows the Flowline Editor and the Icon palette used to drag and drop flowline icons onto the flowline. Flow of control travels down the flowline icons, and branches to the right for some icon types. The Run Icon invokes another flowline and the Return Icon is used to exit a flowline prematurely, or to specify a returned value. The icons in the palette are, from top to bottom:

Sound-- to play any type of sound resource.

Message Box -- quick way to show text to the student (without an interaction screen).

Script -- arbitrary scripts which can refer to flowline local variables.

Erase/Remove -- hide or show widgets.

Run -- invoke another flowline, passing input parameters if needed.

Home -- return from a flowline, returning a value if needed.

Decision -- repeat loops, branching, If-Thens, etc.

Branch -- individual decision branches.

Interaction -- bring up an interaction screen and respond to student-generated widget events.

Copy Data-- transfer data from one location to another (e.g. from a Content object to a local variable).

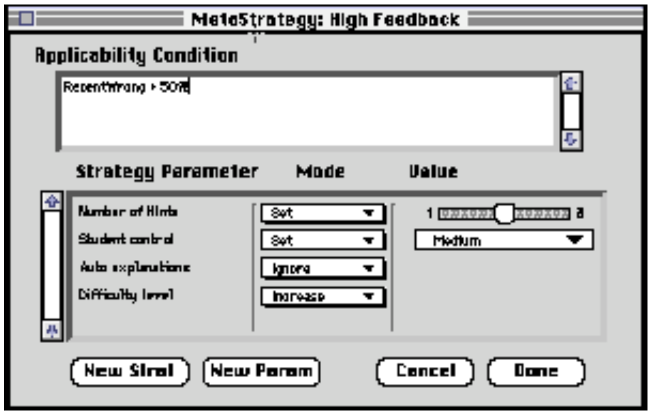
Animate -- animate widgets along an arbitrary path.

Composite -- a sub-flowline.

In the flowline in the Figure 13 the main multiple choice screen interaction is brought up, and branches are defined for giving feedback on student answer selections, for pushing the Next or Quit buttons, and for selecting the Elaborate button.

Meta-strategies

Effective teaching systems need multiple instructional strategies, and meta strategies for choosing among them, to be able to respond to a variety of learning conditions and types of knowledge [Ohlsson 1987]. In Eon we use a flowline parameterization approach. "Strategy parameters" are defined using the MetaStrategy Editor (Figure 14). These parameters are like global variables that can be used in decision points in flowlines. For example, a parameter called "number of hints" can be defined and used in a flowline to specify whether one, two, or three hints are given. A "student control" strategy parameter can be defined and used in a flowline to decide whether to allow students to skip the current question. Unlike normal global variables, which would be manipulated in the flowlines, the strategy parameters are set using meta-strategies. Each meta-strategy includes a set of strategy parameters, and a setting for each of those parameters. Also, each meta-strategy has an "applicability condition" that defines when the MetaStrategy is triggered. For example, in the "High Feedback" MetaStrategy shown in Figure 14, the applicability condition is "Recent Wrong" (a variable in the student model) is greater than 50%. When this is true, the High Feedback strategy is triggered. When it is triggered it sets the values of several strategy parameters: number of hints, student control, auto-explanations, and difficulty level. Since these strategy parameters are used in branch and decision icons in flowlines, the behavior of the tutor will be changed.


MetaStrategy: High Feedback

Applicability Condition

RecentWrong > 50%

Strategy Parameter	Mode	Value
Number of Hints	Set	1 … 3
Student control	Set	Medium
Auto explanations	Ignore	
Difficulty level	Increase	

Figure 14: MetaStrategy Editor

Though the Meta-Strategy Editor is functional, we have not used it yet in any tutors. We hope to include meta-strategies in the Refrigeration Tutor before it is used in the classroom.

3.7 Other Tools and Capabilities, and Implementation

The authoring tools have a WYSIWYG pause-and-edit feature, which allows the author to test run a tutorial, pause it at any point, and easily access the Interaction Editor, Contents Editor, or Flowline Editor, to modify the data bases of teaching strategies. Eon has a number of other tools not described above, including tracing and debugging tools, and a Document Browser which gives a hierarchical view of every object in a tutorial document. The Document Browser is shown in Figure 15. The document browser is an alternate way to view, access, and edit any object or object attribute. Double clicking on an object in the document browser brings up the appropriate editor (e.g. the Flowline Editor if a flowline name is double-clicked).

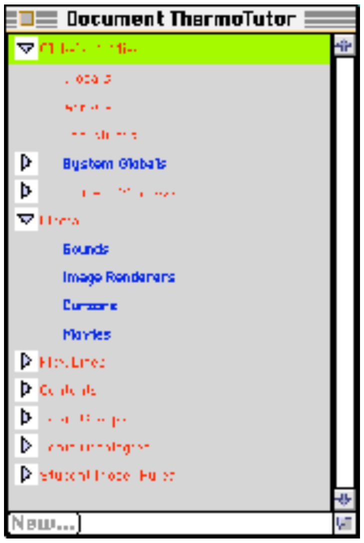


Figure 15: Document Browser

Eon is implemented in a high level programming language called SK8, developed by Apple Computer's Advanced Technology Group (see http://SK8.research.apple.com). Unfortunately, SK8 language development and support was discontinued while the language was still in an alpha stage (it is now available in the public domain). As a result, the Eon authoring tools are relatively slow, take an inordinate amount of memory to run, run only on Quadra-generation Macintoshes, and experience occasional crashes (all of these problems were to be fixed at the SK8 language level, had it continued as an Apple product). The upshot is that the Eon tools are alpha prototypes, useful for demonstrating ITS authoring, and capable of producing ITSs given certain restrictive hardware limitations on the runtime environment. Despite these hindsight problems with using the SK8 development environment, we have been able to build a very large and complex system, with a high degree of interactivity and usability, in a fraction of the time (about 20%) it would have taken to build in a more traditional programming environment. Thus, from a research perspective we made the right choice, but in order for the authoring tools to be a viable ITS development platform, the system needs to be re-implemented in another environment (C++ or Java, for example). In the mean time we are developing a Java-Based runtime engine, which will run Eon-built ITSs over the World Wide Web.

3.8 Other Tutors Built with Eon

Eon has been used to build four other tutors, which we describe briefly below. The tutors are all early prototypes. Some will find their way into instructional situations and evaluation. While most were designed to demonstrate and test specific Eon capabilities, the Keigo and Refrigerator Tutors were designed to address real educational needs, and will be used in college classes.

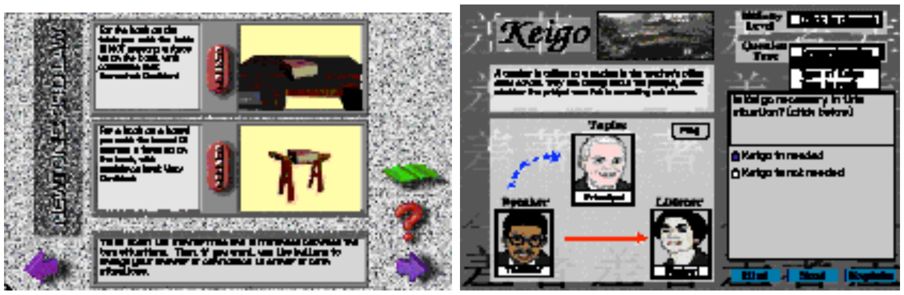


Figure 16: Bridging Analogies Tutor & Keigo Tutor

The **Bridging Analogies Tutor** incorporates a Socratic teaching strategy developed and tested by cognitive scientists to remediate common persistent misconceptions in science [Murray et al. 1990]. This tutor demonstrates the operationalization of a recursive teaching strategy which was researched in human one-on-one tutoring and classroom teaching. The student is presented with a sequence of analogous situations in an attempt to bridge the conceptual distance between their current understanding of the phenomena ("does a flat surface push back up on something that sits on that surface," in this case) and the correct physics interpretation. Occasionally students are presented with screens which encourage them to compare and contrast previous answers, and give them the option of changing a previous answer in light of new conceptual connections (see Figure 16).

The **Keigo Tutor** teaches a part of Japanese language called "honorifics," dealing with the complicated rules used to determine verb conjugation which appropriately honors the listener and topic of a conversation. For this tutor Eon is interfaced with a rule based expert system encoding rules about how to map from surface level features of the conversational situation to linguistically relevant properties of the situation. Students are presented with a variety of situations involving people of varying levels of status and formality, and are asked what types of verb conjugations are needed. Feedback is given based on which rules are applicable. This tutor will be tested in a Japanese Language class in Spring 1998.

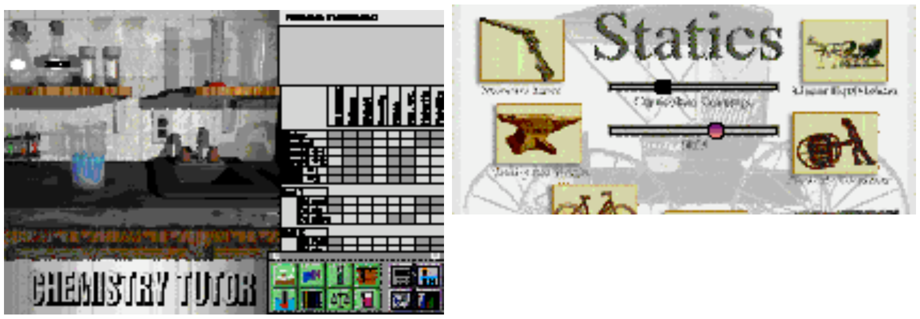


Figure 17: Chemistry Workbench & Statics Tutor

The **Chemistry Workbench** (see Figure 17) is the tutor which demonstrates the most open ended learning-by-doing environment of the tutors built thus far. Students can mix chemicals from the shelf by dragging and dropping them on beakers. Color and precipitates of the resulting solutions are shown, and the instruments in the tool palette at the bottom right can be used to measure the volume, pH, etc. of the results. The goal of the first tutorial is to learn about solvency and chemical reactions by interactively mixing chemicals and measuring the results. The student fills in the table as he accumulates data to answer the chemistry question posed.

The **Statics Tutor** teaches introductory Statics concepts, and includes a "crane boom" simulation widget (see Figure 7). This tutor has the largest topic network and curriculum knowledge base, comprising 40 topic and misconception nodes, and hundreds of presentations with multiple-choice questions. Using topic levels called "Summary," "Teach Easy," "Teach Moderate," and "Teach Difficult," in conjunction with topic link types "Familiarity Prerequisite," "Easy Level Prerequisite," and "Moderate Level Prerequisite," we were able to simulate a "spiral teaching" strategy, in which the same set of topics are taught several times, but at a higher level of difficulty each time.

4. Design Tradeoffs, Usability, and Comparisons with Other Research

Now that we have described the authoring tools and given some sense for how they have been used, we will discuss issues of authoring tool design and use in more depth. In this Section we will discuss our design decisions for authoring tools for all of the major ITS components: interface, domain model, student model, and teaching strategies. We will illustrate areas where usability was desired over power and complexity, and compare Eon with other ITS authoring systems. First, however, we will characterize the space of design decisions available to ITS authoring tool designers, and characterize the intended users of the Eon tools.

4.1 Design Tradeoffs.

A number of ITS authoring systems have been prototyped (see discussion below), and their diversity attests to the lack of consensus in the field on what mechanisms or interfaces are most appropriate. This is true for ITSs, so it is no surprise that it is even more pronounced for ITS authoring systems.

			Domain Model	Teaching Model	Student Model	Learning Environment
	Power/	Scope				
	Flexibility	Depth				
	Usability	Learnability				
		Productivity				

Figure 18: ITS Authoring Tool Design Tradeoffs

This wide variation is due in part to the large and under-constrained design space. There is rough consensus on the nature of the tradeoffs involved, but not on how to balance those tradeoffs to produce useful and usable systems. Designing for power/flexibility vs. usability are usually at odds with each other. Power/flexibility has two aspects: scope (breadth) and depth of knowledge. *Scope* 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. Figure 18 illustrates the factors involved.

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. Scope and Depth are also often at odds, because one must often limit the generality of a system to be able to represent deep causal knowledge. HyperCard, for example, is an authoring tool with huge knowledge scope and minuscule knowledge depth. Note also that power/flexibility in ITS authoring tools is mostly concerned with the "shell" aspect of the system, while usability is more concerned with the interface and "tool" aspect. Clearly, shell and tool are highly interdependent, since a bad tool can make an important aspect of the representational framework practically inaccessible, and a representational framework that is too ambitious or arcane will lead to tools that are visually and conceptually incomprehensible.

Traditionally, ITSs are described as having four major components or functions [Wenger 1987]. Figure 18 illustrates that the design factors mentioned above come into play for each component separately. The state of the art is such that no authoring system yet excels in all four of these areas, and those that excel the most in one or more areas teach a comparatively limited skill or content area. A number of other generic ITS authoring tools have been developed over the last decade (e.g.: IDE, [Russell et al., 1988], ID Expert [Merrill, 1989], COCA [Major & Reichgelt 1992], GTE [Van Marcke, 1992], RIDES [Munroe et al. 1994], [Nkambou et al. 1996]). We compare these systems with Eon in later sections, in reference to particular issues and design tradeoffs. Below we will discuss some particular design tradeoffs in more detail.

4.2 Defining the User

Many of the design decisions for ITS shells depend critically one the nature of the intended user, i.e. the author. Authors need some skill in three areas: programming, instructional design, and knowledge engineering. The degree of skill needed in each of these areas depends on the authoring system. Some systems [Merrill 1989] walk the user through a pre-defined knowledge acquisition dialog, asking the user a series of questions. This removes knowledge engineering from the design process, making the system much more usable, but such systems tend to be inflexible and tedious to use compared with open ended systems which provide a framework and allow the author to mix top down and bottom up design in an opportunistic fashion. Other authoring systems [Jones & Wipond 1991] allow free-form design but can also critique the design for completeness, constancy, and even instructional validity (if you agree with the instructional model they include).

To build a tutor of any reasonable level of sophistication will usually require the efforts of a design team rather than an individual, and will require a tool with complexity at least on the order of magnitude of Photoshop, AutoCAD, or DBASE. Therefore, for the foreseeable future, we do not expect the "average" classroom teacher or industrial trainer to be able to author an ITS any more than we expect every teacher to author a textbook in their subject area. The "master" teachers or trainers who become ITS authors will have to be able to invest significant time building the systems and invest additional startup time on the learning curve for these sophisticated tools. But whereas now building an ITS is restricted to a few initiates, the right tools could allow every company and every school to have at least one team capable of ITS authoring. These teams could work with teachers, subject matter experts, and graphical artists to rapidly produce ITSs. By providing visualizations of key concepts and components in the ITS, the Eon tools make ITS authoring more accessible. However, we still envision that the Eon tools will be used by teams of people, with one person on the team trained in ITS construction and knowledge acquisition methods. In Section 6 we describe our plans to build special purpose authoring systems to achieve a larger degree of both power and usability, making the authoring tools available to a wider group of potential authors.

Another class of potential users are educational theorists. ITS authoring tools should allow theorists to rapidly prototype ITSs and easily modify their teaching strategies and content to experiment with alternative curricula and instructional methods [Winne 1991]. This is crucial because there is still little known about the form or applicability conditions of instructional strategies for ITSs, and whether such strategies are best acquired from practicing teachers or instructional design theories [Ohlsson 1987, Major 1993].

4.3 Tradeoffs in Interface Design Tools

A large amount of effort was put into Eon's student interface builder, in order to allow authors complete flexibility in the design of the interface. As well as the template feature, making it easy for authors to create repetitive content, all widget properties can be manipulated via scripts, allowing the screens to be composed and modified dynamically during the tutorial. Visual and semantic properties of widgets can be made to depend on each other, allowing simple simulations to be built.

Still, Eon is not as facile at authoring complex learning environments and simulations as tools built specifically for this purpose. For example, the RIDES system [Towne & Munro 1988] allows widgets such as simulated meters, levers, faucets, and motors to be connected by wires or pipes, and represents the interactions between these components in such a way that students can inspect how the device operates. RIDES, like other special purpose authoring tools built to date, has only limited abilities to represent curriculum, content abstractions, or multiple teaching strategies.

4.4 Tradeoffs in Inferencing Power of the Domain and Student Models

As ITSs become more effective and sophisticated, there is danger that the gap between theory and practice will widen, and there is an ever increasing need to fully involve practicing educators in the design, construction, and evaluation of these systems [Clancey & Joerger 1988]. Powerful Artificial Intelligence techniques such as plan recognition, case-based reasoning, natural language understanding, neural networks, fuzzy logic, and rule-based problem solving must be used sparingly and cautiously if the goal is to produce tutors efficiently, since the inclusion of these technologies into tutoring systems dramatically increases the complexity of the necessary authoring tools, the amount of training needed to use them, and the complexity of knowledge acquisition. AI models of expertise are said to be powerful because of their generality and modularity, but in practice, building and expert system is still a "black art," prone to scale-up intractabilities. The amount of complexity and experience needed to build one is at odds with the level of simplicity needed for a truly usable ITS shell.

Research in ITS with deep representations of domain knowledge is still needed, but our work in ITS authoring tools favors usability over domain model inferencing power. We do not support expert system representations of knowledge, i.e. the system does not usually "know," in any deep sense, what it is asking the student to learn. This means that "why" and "how" student inquiries, hints, explanations, and problem solving demonstrations can not be generated from first principles, as in more AI-intensive ITSs (e.g. [Lesgold et al. 1990]).

The lack of a runnable expert model also limits the student model's depth. In general, ITS Student Models come in two flavors: runnable models and overlay models [VanLehn 1988]. Runnable models, such as those used in Model Tracing tutors [Anderson & Reiser 1985] represent student knowledge as a subset of the expert system rules, along with buggy rules, and can compare these rules to the student's behavior in precise ways. Overlay models assign competency (and sometimes certainty) values to curriculum topics according to inferences made during the tutorial. Runnable models are used with expert-system based ITSs, and overlay models are used with non-runnable or curriculum-based ITSs. The same problems in complexity apply to authoring runnable student models as apply to authoring expert systems, as described above. Therefor Eon does not support Model Tracing or similar methods. Authoring tools are being developed for Model Tracing tutors, which incorporate expert systems for relatively simple skills, and maintain fine grained models of student knowledge [Anderson & Pelletier 1991].

Though Eon does not directly support the authoring of expert system models with authoring tools, an expert system can be used with an Eon-built tutor. We have demonstrated this with the Keigo tutor, which has as expert system for determining Japanese verb conjugation in conversations based on the roles and relationships of the participants.

4.5 Tradeoffs in Modeling Teaching Strategies

In addition to limitations in domain knowledge sophistication, there are also limits to what we can reasonably expect in the level of sophistication of tutoring strategies. At the simplest level are CAI systems which have no explicit representation of pedagogy and no generalizations of instructional actions. We wish to encode instructional expertise at as general a level as possible, within limits of practicality and the state of the art. As an illustration of the progression from encoding specific decisions to encoding general rules and deeper principles, consider the following ("English-ized") hypothetical ITS tutoring rules and principles, where each item is intended to be an abstraction or reason encompassing the previous one:

1. If button #1 on screen #5 is pushed, then go to screen # 12.

2. If question-12 is answered wrong, then give explanation-5.

3. If the student gets a question wrong twice, then give a canned explanation.

4. If the student is very confused, then give an additional level of feedback.

5. Give students several opportunities to think about each situation so that they may learn from their mistakes, then scaffold feedback of increasing levels of specificity.

6. Learning happens through an active process of concept formation while trying to account for new information within in the context of previous knowledge.

This progression of hypothetical ITS tutoring "rules" goes from the trivial to the impossible. The first two items illustrate the low-level coupling of state testing and action found in (non-intelligent) CAI. The third item illustrates a type of tutorial reasoning that is typical of today's intelligent tutors. A tutor using this rule must keep a record of the student's behavior, but the reason why the rule is applicable is not explicit. The fourth item is well within the state of the art for ITS, and represents our goal for the Eon authoring system and the tutors built with it. A tutor using this rule must have abstract models of the student's mental state and the tutoring process. A diagnostic strategy must infer the level of "confusion" from student behavior (such as number of times asking for help), and the appropriate interpretation of "feedback" must be inferred based on the current state of the tutorial session. The fifth item states a pedagogical belief or strategy, and represents the principle behind the previous rule. It could be operationalized in a limited way but is not precise enough to be part of a robust ITS (with today's technology). The final item is based on a theory---a psychological assumption. It represents the reason for the previous principle and the purpose for the rule above it. Representing and using knowledge at this level of abstraction is clearly out of the reach of current technology. In the extreme one could ask: "Why go through all the trouble of defining the curriculum and tutoring strategies at all? Why not use a deep causal representation of domain knowledge and its pedagogical properties, and let AI rules infer the relationships and ordering of the subject matter?" The answer is perhaps obvious: the problem is intractable except in very limited cases.

A variety of representational formalisms have been used for control and strategic knowledge in ITS shells. Some employ relatively sophisticated AI techniques such goal-based planning [Russell et al. 1988], black board architectures [W. Murray 1990], agents [Cheikes 1995], task decomposition [Van Marcke 1992], and production rules [Anderson & Pelletier 1991, Major & Reichgelt 1992]. Similarly to the domain knowledge expert system discussion above, no framework or visual editor has yet been devised for any of these formalism which lowers the complexity level sufficiently for our intended user audience. These formalisms are highly modular, but control information elicited from human experts often has clearly defined structure [Gruber 1987], and high modularity can hide the structure of strategic knowledge, obfuscate the context of strategy decisions, and make strategy design unwieldy [Lesser 1984].

Strategy representation in Eon is based on a flowline paradigm for visual authoring, which has proven to be highly understandable and usable. It is not as powerful as the more AI-intensive methods mentioned above, because it does not allow a tutor to search a large space of potential instructional actions, or to reason about what it plans to do (or what it could do) further along in the session.

Multiple Teaching Strategies.

Ohlsson [1987, pg. 220] points out that "in order to provide adaptive instruction, a tutor must have a wide range of instructional actions to choose from." Human tutors have more than one teaching method or style available to them, and likewise, computer tutors should be able to change teaching style depending on dynamic student characteristics. Spensley et al. [1990] describe a shell which allows meta-strategies to choose among pre-defined general strategies, including cognitive apprenticeship, successive refinement, discovery learning, abstraction, practice, and Socratic diagnosis. The strategies themselves are fixed however, and fine grained decisions can not be modified. Major [1995] describes a highly usable authoring tool (REDEEM) that allows teachers to set a number of teaching strategy parameters to customize and select applicability conditions for teaching actions. In this system some flexibility is traded for usability, since the underlying instructional strategies are pre-defined (though parameterized). Van Marcke's [1992] GTE system uses multiple alternative rule sets to carry out the actions of a given tutorial goal. This system is more flexible but difficult to author.

We considered implementing multiple flowlines with the same purpose (e.g. multiple "Give a Hint" flowlines), as a technique for representing multiple strategies in Eon, but this seemed too confusing for users. Eon uses a parameterized approach like REDEEM, but is more flexible since the strategies can be built from scratch. Users define a number of "strategy parameters", for example, "degree of hinting," "degree of interruption," "preference for general vs. specific information," and "amount of information." Using the Meta Strategy Editor authors create meta strategies, which specify combinations of these, e.g. "moderate hinting; give general information before specific; and skim (don't give much information)." These global variable are used in the decisions of teaching strategy flow lines to, for example, take one branch for moderate hinting and another for maximum hinting. The author then specifies conditions for when each meta strategy is triggered. We believe that this method reaches a good balance between usability and flexibility, but we have not yet used meta-strategies in teacher-authored tutors to confirm this hypothesis.

5. Representing Pedagogical Knowledge

In the last Section we discussed the design of the authoring tools. In this Section we first discuss the design of Eon's underlying representational framework (which the authoring tools reify), and then we discuss issues encountered in the *use* of the authoring tools to represent domain knowledge. Representing domain information and representing teaching strategies are the two most labor intensive and knowledge intensive tasks in building an ITS. In our work to date we have not focused yet on representing a variety of instructional strategies, though that is one direction we plan to investigate in the future. Therefore, in Section 5 we will limit our discussion to what we have learned about representing domain knowledge, and since, as mentioned previously, we focus on the representation of pedagogical knowledge rather than performance knowledge, we will further limit our discussion to the acquisition and representation of pedagogical domain knowledge in intelligent tutors.

5.1 The Need for Explicit Representations of Curricular and Pedagogical Knowledge

Although the effectiveness of computer-based instruction can depend heavily upon whether the learning environment is relevant and meaningful, some instructional systems focus exclusively on providing "learn by doing" systems that simulate (with whatever degree of fidelity the state of the art can achieve) the contexts in which the target knowledge will be used, yet neglect to provide adequate structure or guidance to assist students in their learning. Here we argue for the need to provide structure and guidance and thus the importance of representing curricular and pedagogical knowledge.

Instructional systems, since they are designed artifacts, will always have instructional goals for the student (even if the goals are vague, implicit, or open ended), and all but the most motivated, advanced, and prepared students will require guidance and/or structure in achieving these goals. This guidance and structure can take several forms:

1) Intelligent selection and sequencing of topics and tasks (a top down approach),

2) Presentation of feedback, hints, explanations, and other informative reactions to student behaviors and queries, (a reactive or opportunistic approach).

3) Biasing the learning environment to maximize learning (an implicit or covert approach).

Students can not be expected to select learning tasks and topics efficiently in domains they are just beginning to learn about. They can flounder, get bored, or pursue inappropriate goals if they do not receive proper guidance and/or structure (for example, they may run a simulation over and over to simulate some interesting catastrophe, without making the effort to learn about the causal mechanisms involved). In order for the needed guidance and structure to be given, intelligent tutoring systems must contain expertise in the subject being taught (domain knowledge), and expertise in how to teach that material (teaching knowledge). Figure 19 illustrates how what we call "pedagogical knowledge" (sometimes called propeadutic information) is both teaching knowledge (the declarative part, as distinguished from teaching strategies which are usually procedural or rule-based) and domain knowledge (the part that is related to teaching a subject, as distinguished from the knowledge needed to perform in the domain).

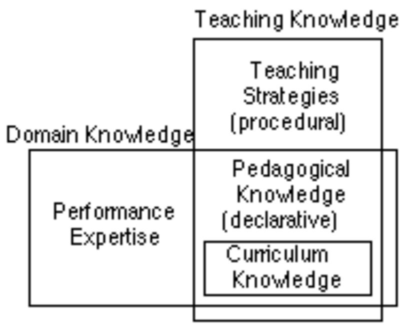


Figure 19. Teaching and Domain Knowledge

Part of pedagogical knowledge is "curriculum knowledge," a term we use to refer to the modular representation of knowledge and learning goals into knowledge units ("topics" in Eon) and the relationships between these units. Clearly, the learning or teaching of different things is done best using different methods [Gagne 1985]. In addition, learning any non-trivial skill involves learning a number of component pieces of knowledge, some of which are dissimilar in their pedagogical requirements. ITSs need to explicitly represent these knowledge pieces, and must be able to distinguish and use their pedagogically relevant characteristics.

To summarize, most students need guidance and structure while using computer based learning environments in order to efficiently reach the learning goals for which the learning environment was built. This guidance and structure can be given external to the computer (as in a teacher looking over the student's shoulder, or a worksheet suggesting what to explore) or from the computer learning environment itself. If an intelligent tutor is going to provide structure and guidance, it needs to represent and reason about modular knowledge units and their pedagogically relevant properties (i.e. pedagogical knowledge, which includes curriculum knowledge, is needed). Some tutoring systems seem to get by without representing pedagogical knowledge, but this is only because they teach something which does not have numerous and/or diverse components, i.e. they teach one thing or a few very similar things.

5.2 Previous Work in Representing Pedagogical Knowledge

Below we will summarize some theories and principles from the literature that are relevant to representing pedagogical knowledge, and use this information as a foundation for a "curriculum object framework." This forms the underlying representational framework for Eon's Topic and Content objects, so our discussion constitutes a justification of Eon's representational framework based on instructional theories.

A. Modular knowledge units. In section 5.1 we argued for the need to represent discrete units of knowledge (or learning objectives) in intelligent tutors, so that the tutor can model and reason about these units in order to provide guidance and structure for the student. (In Section 5.4 we discuss some problems inherent in knowledge modularization.) All of the principles and theories below implicitly or explicitly assume that content can be modularized to organize the learning.

B. Hierarchies and concept learning. Ausubel's "subsumption theory" of learning [Joyce & Weil 1986] focuses on the hierarchical organization of concepts in disciplines, and thus is amenable to computational modeling. He proposes that abstract knowledge (further up in the hierarchy) is more meaningful and useful, and preferred to more specific or rote learning. His Advanced Organizer model prescribes that new information must relate to previous information, and that effective learning paths through the hierarchy of knowledge will differ for each student. He gives strategies for relating new knowledge to existing knowledge based on whether the new knowledge is sub-ordinate, super-ordinate, or co-ordinate with respect to existing knowledge. Web teaching [Halff 1988] similarly requires that knowledge networks be annotated with information about the relatedness of topics (prefer more closely related topics) and generality (give generalities before specifics).

C. Procedural knowledge. The subsumption relationship is valid for conceptual learning, but pedagogical knowledge for procedural or skill learning requires a different treatment. Burton and Brown's BUGGY tutor [1982] uses a skill lattice to represent subtraction subskills. The NEOMYCIN system [Clancey 1982] uses an and/or lattice to represent medical diagnostic procedures. The BIP-II programming tutor [Westcourt et. al 1977] uses a network of subskills related by four links: analogous, harder than, same difficulty, prerequisite.

D. Lessons and instructional goals. Lesgold [1988] points out that the concept of prerequisite is often inadequate, since whether one topic is a prerequisite of another may be a function of the learning goal of a particular session, rather than a static relationship between topics. He proposes a goal lattice structure that captures the different "viewpoints" of a curriculum structure that result from different instructional goals (or perspectives).

Leinhardt and Greeno [1986] distinguish lesson structure and subject matter as the two fundamental systems of knowledge needed for teaching, where subject matter knowledge is used by the lesson structure, the later being in charge of tailoring a session for an individual student. Van Marcke's [1992] GTE framework makes a similar distinction between content and instructional goals.

E. Beyond hierarchies and lattices. Domain knowledge is usually messier than can be represented in a simple hierarchy or lattice. Goldstein's [1982] Genetic Graph includes relationships among procedural rules which represent the way knowledge evolves while a student learns how to master a maze exploration game. The relationships include explanation, generalization, analogy, and refinement, and show how learning can follow knowledge pathways from abstract (simple) to more refined, from deviation to correction, and from specialization to generalization.

F. Knowledge attributes. Bruner's [1966] theory of learning focuses on how we form new concepts, categories, and rules by induction from examples or cases along with the analysis of key features. This indicates that not only knowledge chunks and their relationships, but also their pedagogically relevant properties, need to be represented for instruction. Case-based tutors, such as some of the Goal-Based Scenarios described in [Schank et al. 1994], which use knowledge bases of example objects or situations, search the knowledge base for appropriate cases based on case attributes.

G. Types of Knowledge. All of the work mentioned thus far deals with organizing units of knowledge which are basically of one type (usually concepts or procedural skills). But, as attested to by the diversity of the above instructional approaches, there are many types of knowledge. VanLehn [1987, pg.60], speaking from an AI perspective, says that the popular procedural/declarative distinction is "notorious...as a fuzzy, seldom useful differentiation." We recommend that the procedural/declarative distinction be abandoned for classifying knowledge in instructional systems (except in contexts where it has a precise meaning, as in the ACT* theory of cognition) and that more descriptive and precise schemes be used.

Bloom [1956] and Gagne [1985] were among the first to develop clear classifications of knowledge and learned behavior, and assert that different types of knowledge require different types of learning or instructional methods. Other knowledge typing schemes were later developed which are better grounded in modern cognitive theory and are more operational and concrete for the purposes of computational representation. For example, Merrill's Component Display Theory [Merrill 1983] organizes knowledge in a matrix with *content type*, e.g. fact, concept, or procedure on one axes and *performance level*, e.g. remember, apply, and create, on the other. This matrix scheme is more expressive and intuitive than hierarchical representations. Kyllonen and Shute [1988] propose a more complex multidimensional model which distinguishes knowledge types in a hierarchy which illustrates cognitive complexity, and organizes these types in relation to the level of autonomy of learning and the processing speed needed to perform the task. Reigeluth's Elaboration Theory of Instruction [1983] is another complex knowledge typing scheme. It builds upon Merrill's theory for how to teach individual units of knowledge of different types, and goes further to proposes a theory of how these units can be organized and taught within entire domains, which require knowledge of many types.

H. Representing buggy knowledge. Many of the theories and instructional systems mentioned above include some representation of buggy knowledge and a method for remediating it. Buggy knowledge, such as misconceptions and buggy skills or rules, is usually represented in a form similar to its corresponding performance knowledge, but with additional properties and relationships that allow the buggy knowledge to be diagnosed and remedied.

Implications for ITS frameworks

The above principles lead to a number of recommendations about generic pedagogical knowledge representation formalisms for ITSs. The use of network formalisms containing knowledge units (see item A above) with directed links allows for the creation of hierarchies and lattices and less canonical frameworks (B, C). A number of different types of nodes and links may be needed (B, C), and knowledge units should have pedagocially relevant properties associated with them (F). Learning goals for a tutorial session should be represented separately from instructional content (D). ITSs should be able to distinguish among different types of knowledge (G), and also represent buggy knowledge (H), so that teaching strategies can be predicated on knowledge classes. There is also an indication that exclusive use of a network formalism does not provide enough knowledge structuring, and that more complex data structures will often be needed (E, G).

Based on the above, we conclude that a minimum framework must have the following capabilities: 1. A method for distinguishing the abstract representations of "knowledge" from the concrete media which the student will see, read, hear, and manipulate; 2. A method for referring to discrete chunks of abstract knowledge and the relationships between the chunks; 3. A method for distinguishing the goals and needs of a particular tutorial session from the general domain pedagogical knowledge; and 4. A method (or methods) for assigning instructionally relevant attributes, categories, or purposes to chunks or knowledge or media, e.g. "explanation," "summary," "hint," "difficulty," so that teaching strategies can effectively use the information.

5.3 A Curriculum Object Framework

To address the minimal requirements enumerated above, the Eon system uses the following objects and mechanisms, which have been mentioned in our description of the authoring tools, but are reiterated here with justifications for their use.

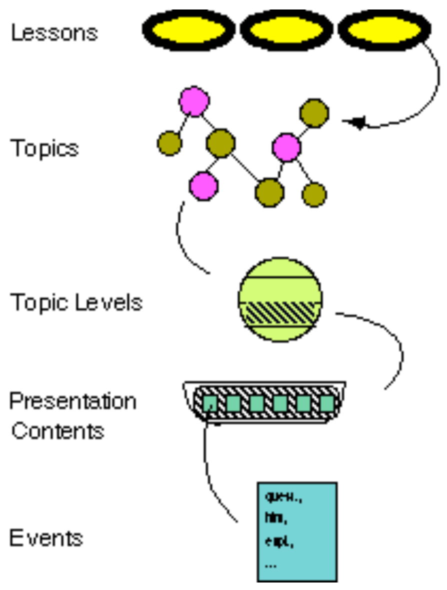


Figure 20: Five-Layer Control Architecture

Topics. Knowledge elements of any grain size are represented by Topics. Since one can define specific types of topics (see below) and create hierarchical links between topics, this simple object suffices for many representational schemes. Topics can have any number of Topic Properties, such as difficulty and importance. The Ontology defines these properties and their allowed values.

Topic Links. The Ontology also defines the types of links (relationships) allowed between topics. Customizable link types allows for the representation of a wide variety of topic networks, including component hierarchies, skill lattices, concept networks, etc.

Presentation Contents. Whereas Topics are *abstract* objects that refer to modular chunks of the knowledge to be taught, Presentation Contents (or just "Contents") contain the *specific* contents that the student will see and manipulate (text, graphics, etc., or the templates or algorithms for generating this content). Contents are expository or inquisitory interactions, usually associated with a specific interactive screen templates which the author designs using the Interaction Editor. For example, a drag-and-match type of interaction screen can be designed and serve as a template. Specific Content objects are created to fill in the contents of that template, e.g. each would specify the labels and pictures of the objects to be dragged, what the correct answer is, and the text to give for an explanation of the correct answer.

Topic Levels. Having only the semantic net to represent all aspects of curriculum structure was found inadequate. Topic Levels allow for distinguishing multiple levels of performance (e.g. memorizing vs. using knowledge), mastery (novice to expert), and pedagogical purpose (summary, motivation, example, evaluation, etc.) for each topic. The Ontology defines what levels are available. Contents are associated with Topics via Topic Levels, as specified in the Topic Contents Browser.

Topic Types. As mentioned, there is general agreement that there are different types of knowledge, each type having its own properties and each type requiring a different instructional method. Rather than have knowledge type be simply a property of Topics, Topic Types are first class objects which all Topics inherit from. Since there are numerous theories of knowledge types, we leave knowledge type definition to the Ontology. Each knowledge type defined in the Ontology has its allowed properties, allowed link types it can connect to, and allowed topic levels. Buggy Knowledge is represented as topics of type Misconception, Procedure Bug, etc., depending on the type of knowledge that is in error.

Lessons. Topic networks, which define pedagogically relevant relationships between topics, do not specify any ordering or starting point for learning sessions, and are independent of the instructional purpose of sessions. Lesson objects are used to specify instructional goals and learning/tutoring styles for a particular group of students. Nominally, the Lesson lists a small number of starting or goal topics, and specifies the default teaching strategy. The teaching strategy then determines how the topic network will be traversed, causing other topics to be taught, to satisfy the goal of learning the goal knowledge.

The basic representational elements described above were included in Eon because some form of all of them (with the exception of Topic Levels) seem necessary for any pedagogical representation. We call this framework a "five layer decision architecture," where the five layers are Lesson, Topic, Topic Level, Content, and Events (see Figure 20). Running a Lesson runs a number of Topics, each of which runs some of its Topic Levels, each of which contains a number of Contents. At the lowest level are the individual Events between the student and tutor, such as a student clicking a button, or the tutor giving a hint, several of which will occur while a Content is running.

Ontology Objects for Customizing the Curriculum Framework

The five-layer decision architecture is fixed in Eon, but the specifics of each layer are customizable using ontology objects. An ontology is a particular way of describing the world (or some domain); it is a scheme for conceptualizing the objects and relationships in a domain [Gruber 1993]. We use the term "ontology object" for a data object which defines a conceptual vocabulary for a part of the system. In our current system Topic Ontology objects specify topic types, topic link types, topic properties, and topic levels. For example, the Topic Ontology for our Statics Tutor defines topic types Fact, Concept, Procedure, and Misconception, and topic links Prerequisite, Generalization, and SubConcept; while a tutor for Manufacturing Equipment might have topic types Safety, Maintenance, Operation, Theory, and Common Failures, and topic links SubPart and SimilarPart. The widget palette and widget properties could be seen as an ontology for interface design. Also, the allowed values for student model levels comprise an ontology for representing the student's state. Strategy parameters, defined in meta-strategies and referred to in flowlines, comprise a vocabulary for describing instructional decisions, are thus are also an ontology. Ontologies tend to be generic and reusable, for example, an ontology developed for one science tutor should be usable (perhaps with slight modifications) for other tutors with similar pedagogical characteristics (e.g. instruction at a predominantly conceptual level).

Most tutoring systems fall into one of a number of loose classes, each addressing specific types of cognitive skills or knowledge types. Example domain classes include: conceptual information, factual information, problem solving skills, design skills, procedural skills (such as maintenance), inquiry and experimentation skills, equipment diagnostic skills, customer contact (and other interpersonal) skills, sensory-motor skills, association and pattern recognition skills, and argumentation/hypothesis generation skills. Ontology objects may be reused across tutors within a domain class. Reigeluth [1983] prescribes that each domain be assigned an "organizing content type:" conceptual, theoretical (principle-like), or procedural, that best fits the characteristics of the domain and the instructional goals. His "elaboration theory of instruction" specifies methods for selecting and sequencing content according to the organizing content type. Others have categorized domains according to whether their structure is predominantly procedural, historical, structural, causal, teleological, inferential, etc. Domain types have characteristic links between topics, for example analogy, physical-part, a-kind-of, etc. Classifying domains according to organizing content type, and creating ontology objects for each organizing content type, would help bootstrap ITS construction.

As one could guess from the variety of instructional approaches described in Section 5.2, trying to design the most general or powerful representational framework for an ITS would, at the least, result in a very complex and obscure system. The design decisions for such a complex system amount to a commitment to a particular perspective on how to organize the world, and though the designer may be able to wield this complex machinery deftly, it is unlikely to have much in common with other equally valid representational frameworks. For example, the Expert CML system [Jones & Wippond 1991] organizes domain knowledge in a hierarchy of objects including Departments, Programs, Courses, Topics, Subtopics, Modules, SubModules, Objectives, and Activities. In contrast, we have tried to identify a minimum underlying object-oriented framework that is neutral regarding domain or instructional theory, and then leave the specification of the remaining complexity to the Ontology design. For example, since we make no commitment to the grain size of an Eon "topic," the ontology can define topic types called Departments, Programs, Courses, etc., and also constrain the types of relationships between them.

5.4 Problems and Solutions in Representing Pedagogical Knowledge

Now that we have described our underlying representational framework for pedagogical domain knowledge, and discussed how Ontology objects are used to customize the representational framework for specific tutorials or domains, we will consider several important issues encountered when we work with instructors trying to explicitly represent pedagogical knowledge in an intelligent tutor.

Human knowledge does not exist in neatly defined, clearly named packages---it is inherently complex, densely connected, fuzzy, and ambiguous. Yet to use knowledge in AI systems we try to represent it in individual units with clear structure. The tension between the organic nature of knowledge and our need to modularize it leads to a number of unavoidable issues for ITS knowledge representation, which we discuss below, along with how we deal with these issues in the Eon system.

Knowledge Structure and Complexity

Most modern epistemological theories assert that the external phenomena that knowledge refers to does not "have" any specific structure, but that we invent knowledge structures to organize and reflect upon our understanding when we communicate or teach. We must avoid the assumption that by organizing knowledge carefully enough we can feed it a student in some logical sequence and expect her to assimilate it (this is sometimes called the "pipeline theory of learning"). Yet it is also true that "the sequence of exercises and examples should reflect the structure of the [knowledge] being taught and should thereby help the student induce the target [knowledge]" [Half 1988]. Our overview of past work illustrates that there are many ways to structure knowledge, and that these can be represented computationally using common data structures such as hierarchies, lists, networks, arrays, and frames, as well as multi-dimensional, multi-level combinations of nested data structures. Though, theoretically, a knowledge space of arbitrary complexity can be represented with just knowledge objects (topics) with their properties and interrelationships (links), very complex structures are unwieldy, esoteric, and difficult to use and maintain. Knowledge authoring must be supported with tools that allow clear visualization of the knowledge structures, and ITS authoring tools must commit to and support (via visualization tools) an underlying structure.

Knowledge is structured in Eon using several mechanisms. First is the layering of basic object classes: Lessons, Topics, Topic Levels, and Contents, mentioned above. The second method is in allowing arbitrary classifications of Topics and Topic Links in the topic network. Given the right Ontology, all of the hierarchies, lattices, and networks mentioned in Section 5.2 can be represented in topic networks. The third mechanism is the Topic Levels themselves.

Topic Levels, though a simple construct, are unique to the Eon framework, and allow a significant improvement in representational sophistication without loss of clarity. For example, we have simulated Merrill's Performance Content Matrix [Merrill 1983] by assigning content types to Topic Types (fact, procedure, skill, etc.), and performance levels to Topic Levels within each topic (remember, use, apply, create, and meta-knowledge). Without Topic Levels, there would be a confusing proliferation of related Topics, one for each level of each Topic, i.e. we would need topics called Gravity-memorize, Gravity-use, etc. As mentioned, levels of mastery can also be encoded, as can different pedagogical functions for a topic. For example, a teaching strategy can tell a topic to "teach" itself, "summarize" itself, "define" itself, and "test" for its knowledge if the Ontology defines these topic levels [Murray & Woolf 1992a].

If additional complexity is needed, the author must resort to "tricks" which are not explicitly supported by the visualization tools. For instance, if an author wanted to represent both mastery levels *and* performance levels of a topic in an Eon tutor, he would have to create topic levels such as Remember-easy, Remember-difficult, and Use-easy. It remains to be seen how wide a variety of domains Eon's current framework can accommodate, without resorting to "hacking" its representational formalism.

Additional analysis of a domain seems to always lead to additional complexity. For example, in our study of authoring the Statics Tutor we discovered numerous different perspectives on the material [Murray 1991 pg. 218], most of which we did not attempt to implement. For example, Newton's' Third Law can be taught by presenting questions which progress from existence (does a force exist here?) to direction (what direction is the force?) to relative magnitude, to quantitative questions. This material can also be taught by showing a progression of example situations in using different surface features, e.g. with objects hanging, falling, rolling, colliding, etc. It can also be taught by dealing with first horizontal forces, then vertical, then both, then rotational forces. Each of these methods has some instructional merit. Yet no system could hope to account for the representational requirements all of these approaches simultaneously.

Topic Modularity and Interdependence

When the knowledge in a domain is organized into modular units, which are then sequenced flexibly according to instructional strategies, a number of unavoidable problems arise. First, it is difficult to encode the knowledge "between" the topics, which can be about how they are related to each other, or the emergent knowledge that comes from understanding topics together. Reigeluth [1983] and Lesgold [1988] refer to this as the "glue" in a curriculum (Lesgold also refers to this as "non-linearities" between topics).

The issue of curricular "glue" is not as much a natural property of the content, as an emergent phenomena that happens when instructional designers break up the content or domain knowledge into discrete chunks. In Eon we deal with this glue in several ways. First, Topics can have Topic Levels such as "Introduction" and "Conclusion," which address how the topic relates to other topics that are likely to precede or follow it in most curriculum paths. Second, Topic Types called Composites and Synthesizers can be used. A Composite topic is one that represents the whole which is more than the sum of its parts. For example, our Static Tutor's Linear Equilibrium (LE) topic had "sub-part" links to LE Intuition, LE Concept, and LE Principle. Knowing Linear Equilibrium involves knowing each of these parts, and also how the parts fit together in an understanding of static situations.

"Synthesizer" is a term used by Reigeluth [1983] for instructional components that interrelate and integrate instructional units. In Eon we can define a topic type called Synthesizer. One possible strategy using synthesizers relates two topics the student has learned: after a topic is taught check whether a synthesizer connects it with another known topic, and if so, teach the synthesis material. Another possible strategy connects new information with existing information before the new information is presented: before a topic is taught, check whether a synthesizer connects it with an already known topic and, if so, present the synthesis material.

Eon has one other mechanism for dealing with curricular "glue:" Lesson objects. Since Lesson objects can specify a sequence of goal topics, they can also be used to insure that certain information is presented between topics, to compare and contrast them.

The second modularity problem is that topics are often interdependent. For example, in our Statics tutor, the student has to know something about Gravity to fully understand Linear Equilibrium, yet some understanding of Linear Equilibrium is prerequisite to learning about Gravity. Topic Levels organized by mastery, combined with levels of prerequisites, allow us to deal with this in Eon. We can assign content to topics at various levels (e.g. easy, moderate, and difficult) and allow prerequisite links such as Familiarity, Easy Prerequisite, and Moderate Prerequisite. Thus we can specify that the easy level of Linear Equilibrium should precede learning the difficult level of Gravity, and that the easy level of Gravity should precede the difficult level of Linear Equilibrium. This method simulates spiral teaching, in which the same topics are taught from successively more difficult perspectives (see [Murray & Woolf 1992b] for more discussion of spiral teaching).

The third modularity problem is that there is a tradeoff between the modular integrity of the units and smooth tutorial transition between the units. It is hard to design a unit whose text makes sense regardless of what comes before and after. In practice, this is dealt with in Eon by providing tools which allow the designer to easily step though many potential paths, check for dialogue consistency, and make adjustments to the knowledge base. Dynamic text generation can ameliorate this problem when it is feasible. In Eon we can assign an arbitrary function to any chunk of text that appears on the student screen. This mechanism could in theory be used to employ natural language generation techniques, but in practice we use it for template-based text generation.

Conceptual Vocabularies

The field needs more in the way of common terminologies (e.g. conceptual vocabularies or ontologies) for describing domain and teaching knowledge in order to better compare systems and share knowledge bases. We are in the process of designing a conceptual vocabulary of primitive tutorial actions, pedagogical parameters, and a knowledge classification scheme that authors can use to organize and codify domain knowledge and teaching knowledge. The goal is to design a conceptual vocabulary for describing the objects, actions, and parameters of instruction which will serve as building blocks or conceptual primitives of representation. We are culling these terms from the literature in instructional design, cognitive psychology, and intelligent tutoring systems, to build a loose taxonomy which, in its first incarnation, will serve more as a paper-based knowledge acquisition tool [Murray 1996b]. Others, including [Mizoguchi et a. 1996] and [Van Marcke 1992] have developed conceptual vocabularies for ITS, and we hope to work with them to develop an integrated set.

The goal is not to build a complete prescriptive model for instruction. It is simply to offer a kitchen-sink style loose taxonomy of terms (with definitions) intended for generality and completeness, not coherence and consistency. A designer can choose from the taxonomy to specify terms which form a coherent model for a particular domain or class of domains. The taxonomy includes lists of topic types, topic links, and topic properties, which can be used to develop Ontology objects. For example, the list of topic links is: a-kind-of, sub-procedure, sub-component; causes, allows, requires, justification, purpose, supports; concrete/abstract, specialization/generalization; prerequisite, critical-prerequisite, familiarity-required, mastery-required; critical-misconception-of; deviation/bug; analogy, bridging analogy, anchoring analogy; synthesizes; comparison.

Also included in our growing conceptual vocabulary are taxonomies for domain types, task types, primitive tutorial actions (for instruction, coaching, and hinting), tutorial decision parameters, student modeling parameters, tutoring styles, and explanation and question types.

6. Meta-Authoring Special Purpose Authoring Systems

We have tried to produce a suite of highly usable tools that have enough similarity to COTS CAI/multimedia authoring tools to support users of these existing tools in making the transition to authoring knowledge based tutors. After using the tools for several tutors and several authors, we find in that the authoring system, though fairly large (the user documentation is several hundred pages long), is composed of tools which are quite learnable and usable. However, the process of starting from scratch to build an ITS is still a formidable one. Not surprisingly, we have consistently run up against the classic knowledge acquisition bottleneck [Hoffman 1987]. Domain experts are usually good at sketching out student interactions, lessons to teach particular topics, and responses to specific student behaviors, but articulating knowledge at a more abstract level is difficult. The following ITS knowledge representation tasks are inherently difficult for most subject matter experts:

1. **Ontology design**. Defining the types of topics, topic links, and topic levels for the Topic Ontology, and also defining the ontology of allowable values for the student model.

2. **Curriculum representation.** Breaking the instructional material and goals up into discrete components (topics) and providing relationships between these components (topic links);

3. **Strategy representation.** Representing teaching strategies in a general way (e.g. how do we recognize that a student is confused, and what is a reasonable general response to student confusion?);

4. **Student modeling.** Defining rules that express when a student knows a topic, and labeling or characterizing the student's level of knowledge.

In our experience, it takes a knowledge engineer, a person skilled in the elicitation and representation of these types of knowledge, to work with the subject matter expert (the teacher) to build these aspects of the system. Highly usable tools such as Eon help, since the teacher will be able to visualize the knowledge represented, and will be able to participate in the knowledge representation once the knowledge engineer has broken the ice and shown how something is done once or twice.

Special Purpose Authoring Tools

One proposed solution to the knowledge acquisition problem is creating special purpose authoring tools, for example authoring tools for building ITSs that teach anatomy, foreign policy, or verb conjugation. Authoring shells which are used to build tutors for specific task types [Jona 1995, Dooley et al. 1995] can, in principle, build tutors with more fidelity and depth than general purpose tools. The depth vs. breadth tradeoff seems to imply that you can have one but not the other, that 1) ITS authoring tools that can build powerful tutors that closely match the pedagogical needs of a domain must have a narrow scope, and that 2) an all-purpose ITS shell, by necessity, must have a shallow knowledge representation and the learning environment it creates will have little conceptual fidelity (in comparison to special purpose tools).

Figure 21: Three Tiered Suite of Authoring Tools

Our approach is to build special purpose authoring systems on top of the generic Eon authoring system, i.e. using Eon as a "meta-authoring tool." This would involve constructing libraries of pre-built components such as topic ontologies, student model rules, and interface screens (see Figure 21). Since some teaching knowledge is general in nature [Van Marcke 1992, Jona 1995], default teaching strategies and meta-strategies will also be incorporated into special-purpose authoring systems. We could then provide sets of these components tailored to facilitate building tutors for classes of domains or tasks. The Eon authoring tools plus these specialized components would comprise a special purpose authoring system. For example, ITS authoring shells could be produced for science concepts, human service/customer contact skills, language arts, and equipment maintenance procedures. Instructional designers could immediately start constructing tutors in an environment that supports and helps structure the knowledge acquisition process. Though we recognize that a special purpose shell programmed from scratch is likely to be more powerful than a shell built with a general meta-shell, the meta-shell approach allows for the proliferation of special-purpose shells with a common underlying structure, so inter-domain commonalties can be exploited in both content creation and in training authors to use the shells.

By using a meta-authoring approach, we hope to achieve a fair degree of fidelity and depth, while maintaining usability and generality. Our eventual goal is to create a three-tiered suite of authoring tools (see Figure 21), at three levels of abstraction [Murray 1996a]. At the first tier is a general purpose ITS authoring system (plain Eon) that requires moderate knowledge engineering and instructional design expertise to use. At the second tier are special purpose ITS authoring systems (built on top of the first tier system) that require minimal knowledge engineering and instructional design expertise. The third tier involves tools for the average teacher using an ITS in her class. At the third tier we will provide a simplified subset of the authoring tools, so that once an ITS is built, *any* teacher can customize it for a particular class or student. For example, by modifying a hint's text, replacing a picture with a more recent version, making a teaching strategy more verbose, or by changing a prerequisite relationship between topics. This is important because some teachers will be reluctant to use instructional systems that they can't understand or adapt.

7. Conclusions and Lessons Learned

In this paper we have argued for the need for authoring tools for intelligent tutors (ITSs, or knowledge based tutors). We summarized foundation research including a productivity analysis indicating that knowledge based tutors can be built with resources comparable to traditional CAI. We described our suite of authoring tools, called Eon, by showing how they were used to build an intelligent tutor for learning about refrigeration, and we briefly described four other tutors built with Eon. We then argued for explicit representations of curricular and pedagogical knowledge, and supported Eon's "curriculum object framework" with instructional theories and principles of pedagogical knowledge representation from the literature. We described a number of ubiquitous knowledge acquisition and representation issues (dealing with knowledge structure and complexity, knowledge modularity and interdependence, and conceptual vocabularies) and discussed how the Eon system addresses each of these.

Our concluding remarks will summarize the remaining major points of the paper by addressing these questions:

What characterizes a knowledge based tutor?

What are the most important features of authoring tools for knowledge based tutors?

Characterizing Knowledge Based Tutors

Our goal has been to provide authoring tools for the creation of intelligent computer tutors, which we also call knowledge based tutors (for reasons given below). The example ITS we used in describing the authoring tools was relatively simple (though we briefly mentioned more sophisticated tutors built with Eon in Section 3.8). Eon does not directly support the authoring of deep causal or expert system representations of domain knowledge, nor does it support intelligent diagnosis of student behavior. This is because not enough is known in general about this depth of inferencing to create generic and highly usable authoring tools for it. Therefore the question "where is the intelligence?" or "what makes a tutor intelligent?" may reasonably arise in the mind of the reader. Yet tutors built with Eon are more sophisticated and "intelligent" than traditional CAI tutors. How do we characterize this difference? Using the less common term "knowledge based tutor" rather than "intelligent tutor" is one way to address the issue, since computational "intelligence" is difficult or impossible to define in a way that satisfies everyone, and the term "knowledge based" highlights the major difference between CAI tutors and intelligent tutors: intelligent tutors explicitly model the processes and knowledge used in instruction (specifically, they model the domain, teaching strategies, and student state). Any ITS authoring system will have a particular underlying representational framework, and therefore will include its own assumptions about what an ITS is and it embodies constraints on the types of ITSs that it can build. We call the types of ITSs that Eon can build "knowledge based tutors." We will define knowledge based tutors as those having the following capabilities and features (this is an extension of our description of the knowledge based approach given in Section 2.1):

An abstract representation of the curriculum or knowledge to be taught (as in a topic network);

Instructional content is modular and separate from instructional strategies;

Generic instructional strategies are used, and they refer to entities at a pedagogically relevant level of abstraction;

A student or user model exists which makes inferences about the student's state;

Instructional decisions can be predicated upon the inferred student state;

Instructional strategies can be predicated upon pedagogically relevant characteristics of the content (e.g. whether a topic is a fact or a concept; whether a topic is difficult or easy); and

Content can be generated and sequenced dynamically.

Since our authoring tools utilizes a curriculum network and do not include rule-based representation of expertise, tutors built Eon will be fairly "curriculum driven," though they can facilitate significant student control in the selection and style of the material. Also, our tools are not well suited for teaching complex procedures or problem solving skills, yet should excel in domains where multiple teaching strategies can be written and predicated on inferred student knowledge. Tools are under development that support other classes of ITSs. Model tracing tutors and other tutors based on runnable models of domain expertise and student knowledge are extremely powerful in the situations where they apply (i.e. where the expertise can be represented at a fine gain size), and authoring tools for these are under development [Anderson & Pelletier 1991]. Another class of ITSs involves teaching about the functionality of equipment and associated diagnostic procedures. Authoring shells that allow the designer to build functioning simulations of equipment and diagnostic strategies have also been built [Towne & Munro 1988].

Tools for Authoring Knowledge Based Tutors

Given the above characterization of knowledge based tutors, the next question is: what features are important in an authoring system for knowledge based tutors?

Authoring tools can have a variety of purposes and intended users, and their design must account for tradeoffs among four overall goals: scope, depth, learnability, and productivity (see Figure 18). We have described our intended user audience (in Section 4.2) and discussed how our system addresses these overall goals for our intended users in terms of each of the four functional components of ITSs. The Eon system has authoring tools for all four functional components of an ITS: domain, teaching strategies (and meta-strategies), student model, and interface. Though designing to maximize any of the four overall goals could compromise the other three, we found certain design principles useful to our goal of maximizing all of them, and we describe these suggested design principles below.

1) Use appropriate representational formalisms. A long standing aphorism in artificial intelligence is that once an appropriate knowledge representation is found for a problem the task of solving the problem, or of programming an intelligent system to solve the problem, is half complete. ITS researchers are still searching for more powerful representational formalisms. So although designing general ITS shells requires many tradeoffs and compromises, additional levels of excellence on all fronts will be realized as better formalisms are developed. The design suggestions below illustrate how an underlying representational structure, along with authoring tools that reify this structure, can achieve both scope and usability.

2) Provide visual reification for the concepts and structure of the underlying representational framework. We believe that 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 without a high degree of knowledge engineering and programming expertise. Such an interface relieves working memory load by reifying the underlying structures, and assists long term memory by providing remindings of this structure. Also, multiple views (visual perspectives) of information are often needed. However, building highly usable interfaces should be done using an iterative user-participatory design process [Blomberg & Henderson 1990], and is unfortunately very costly and time consuming.

3) Facilitate design at a pedagogically relevant level of abstraction. Provide tools which allow subject matter experts to author using primitives that have instructional meaning, for example using objects such as "hint," "explanation," "topic," "prerequisite," and "mastered" (in addition to providing primitives at the presentation and media level such as "graphic," "button," and "mouse click").

4) Design for modularity and re-usability of content. Represent and author instructional content modularity so that it can be used for multiple instructional purposes. Provide productivity tools that capitalize on repetitive or template-like content (see Eon's Presentation Content Editor). Provide tools that make it easy to browse, search for, and reference content objects (as in Eon's Document Browser).

5) Provide features powerful enough to author generative content. Template-based content is important because for many domains current technology can not generate examples, exercises, explanations, or hints in a robust way. But in some cases content and student interactions *can* reasonably be generated on the fly, and tools facilitating this should be provided. Useful features include scripting languages, the ability to attach interface components' attributes to scripts or expressions, and the ability to have attribute values depend on each other.

6) Allow for interface extensibility. Authoring tools should provide a powerful set of interface components (as in Eon's widgets), but should also allow for complex and special purpose components to be programmed separately and linked in to the authoring environment. This feature is called "custom widgets" in Eon.

7) Facilitate an opportunistic design process. ITS authoring tools should allow for top down (starting with the abstract curriculum structure), bottom up (starting with specific screens and content), and opportunistic (switching between top down and bottom up as needed) design of ITSs. WYSIWYG tools that allow easy movement between authoring content and test running the tutorial facilitate the build-and-test iterations needed for rapid production of tutors.

8) Anchor usability on familiar authoring paradigms, and facilitate evolution to more powerful paradigms. For those used to building traditional computer-based instruction, building knowledge based tutors requires a conceptual shift from "story board" representations of content to more modular knowledge based representations. It is useful to have some ITS authoring tools have a look and feel similar to COTS CAI authoring tools, and to provide features which allow a smooth transition from traditional authoring paradigms to authoring more powerful intelligent tutors. For example, Eon's interaction editor and flowline editor have many surface similartities to COTS tools.

9) 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, include the ability to customize the representational formalism. In Eon we do this via Topic Ontology objects, and the capability to customize the conceptual vocabularies used in student modeling and meta-strategies.

10) Create special purpose authoring tools for increased usability and productivity. Generic authoring tools can be used as "meta-authoring tools" to build special purpose authoring tools for particular classes of intelligent tutors. Special purpose authoring tools come with pre-built libraries of components tailored to specific needs. Instructional designers using special purpose authoring tools do not have to start from scratch, but can immediately start constructing a tutor in an environment that supports the knowledge acquisition process. In Eon we plan to combine default teaching strategies, default student modeling rules, default interactive screens, and a topic structure (i.e. an Ontology) which is tailored to a specific type of domain and/or task.

Future Plans

The issues involved in building an ITS from scratch can be subtle (as explained in Section 5), and a trained person may always be needed on the ITS design team. But with appropriate representational formalisms and tools that visually reify the conceptual structures involved, learning how to be a good ITS knowledge engineer can be made accessible to many more people, not just to computer programmers and AI scientists. Also, once a trained person gets the primary structures set up, an instructional designer with much less training can continue to fill in the content.

Our future plans include formative evaluation of the authoring tools, building a number of tutoring systems which demonstrate the breadth of applications possible with the system, generalizing some of these tutors to build special purpose authoring tools and customized ontologies, modularizing the content of the ITSs so that they can be deployed on the world wide web, and using the rapid prototyping capabilities of Eon to facilitate research on alternative instructional strategies.

Acknowledgments

Many thanks to the following people who were crucial in developing the Eon system: Matt Cornell for realizing the ideas in a gazilllion lines of pristine code; Kate Cremer for early versions of the Topic Net Editor; Erik Eide for widget dialogs and the student model editor; Erik Haugsjaa for continued code maintenance and numerous feature enhancements; Ryan Moore for tutor construction and enthusiasm. This material is based upon work supported by the National Science Foundation and the Advance Research Projects Agency under Cooperative Agreement No. CDA-940860.

References

Anderson, J. R. & Pelletier, R. (1991). A development system for model-tracing tutors. In Proc. of the International Conference on the Learning Sciences, (pp. 1-8), Evanston, IL.

Anderson, J.R. (1990) Analysis of Student Performance with the LISP Tutor, in Frederiksen, N., Glaser, R., Lesgold, A.M. and Shafto, M. (Eds.), *Diagnostic Monitoring of Skill and Knowledge Acquisition*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. & Reiser, B. (1985). The Lisp Tutor. BYTE, April 1985, pg. 159-175.

Blomberg, J. L. & Henderson, A. (1990). "Reflections on Participatory Design: Lessons from the Trillium Experience." *CHI '90 Proceedings*, April, 1990, pp. 353-359.

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

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

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

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

Clancey, W. & Joerger, K. (1988). "A Practical Authoring Shell for Apprenticeship Learning." *Proceedings of ITS-88*, pp. 67-74. June 1988, Montreal.

Clancey, W. (1982). Tutoring rules for guiding a case method dialogue. In Intelligent Tutoring Systems, D. Sleeman & J. Brown (Eds.), Academic Press 1982, pp. 201-225.

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.

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

Ginsburg, H. & Opper, S. (1979). *Piaget's Theory of Intellectual Development*. Prentice-Hall: Englewood Cliffs, NJ.

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

Gruber, T. (1987). "A Method for Acquiring Strategic Knowledge from Experts." University of Massachusetts Dissertation.

Gruber, T. (1993). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In *Formal Ontology in Conceptual Analysis and Knowledge Representation* , Guarino & Poli (Eds.). Kluwer Academic Publishers.

Halff, H. (1988). Curriculum and Instruction in Automated Tutors. In *Foundations of Intelligent Tutoring Systems*, Polson & Richardson (Eds.). Lawrence Erlbaum Assoc., Hillsdale, NJ.

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

Jona, M. (1995). Representing and re-using general teaching strategies: A knowledge-rich approach to building authoring tools for tutoring systems. In AIED-95 workshop papers for Authoring Shells for Intelligent Tutoring Systems.

Jones, M. & Wipond, K. (1991). Intelligent Environments for Curriculum and Course Development. In Goodyear (Ed.), *Teaching Knowledge and Intelligent Tutoring*. Norwood, NJ: Ablex.

Joyce, B. & Weil, M. (1986). *Models of Teaching*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Kyllonen & Shute (1988). "A Taxonomy of Learning Skills." Brooks Air Force Base, TX: AFHRL Report No. TP-87-39.

Leinhardt, G., & Greeno, J. (1986). The Cognitive Skill of Teaching. In *Journal of Educational Psychology*, Vol. 78 No. 2, 75-95.

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

Lesgold, A., Lajoie, S., Bunzo, M., & Eggan, G., (1990). A Coached Practice Environment for an Electronics Troubleshooting Job, in Larkin, Chabay and Sheftic (Eds.) *Computer Assisted Instruction and Intelligent Tutoring Systems Establishing Communication and Collaboration*. *Hillsdale,* NJ: Erlbaum.

Lesser, V. (1984). Control in Complex Knowledge-based Systems. Tutorial at the IEEE Computer Society AI Conference.

Major, N. (1995). REDEEM: Creating Reusable Intelligent Courseware. *In Proc. of AI-ED 95,* Washington, D.C., August, 1995.

Major, N. P., (1993). Teachers and Teaching Strategies. *Proc. of the Seventh International PEG Conference*, Heriot-Watt Univ., Edinburgh.

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. Springer Verlag, Berlin.

Merrill, M. D. (1989). An Instructional Design Expert System. *Computer-Based Instruction,* 16: 3, pp. 95-101.

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

Mizoguchi, R., Sinitsa, K., Ikeda, M. (1996). Knowledge Engineering of Educational Systems for Authoring System Design. In *Proceedings. of EuroAIED-96,* pp. 593-600. Lisbon.

Murray, T. (1991). *Facilitating Teacher Participation in Intelligent Computer Tutor Design: Tools and Design Methods*. Ed.D. Dissertation, Univ. of Massachusetts, Computer Science Tech. Report 91-95.

Murray, T. (1993). Formative Qualitative Evaluation for "Exploratory" ITS research. *J. of AI and Education*. V. 4. No. 2.

Murray, T. (1996a). Special Purpose Ontologies and the Representation of Pedagogical Knowledge. *In Proceedings of the International Conference on the Learning Scieces,* (ICLS-96), Evanston, IL, 1996. AACE: Charlottesville, VA.

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

Murray, T., Schultz, K., Brown, D., & Clement, J. (1990). An Analogy-Based Computer Tutor for Remediating Physics Misconceptions. *J. of Interactive Learning Environments* , Vol. 1 No. 2, pp. 79-101.

Murray, T. & Woolf, B. (1992a). Results of Encoding Knowledge with Tutor Construction Tools. *Proceedings of AAAI-92*. San Jose, CA., July, 1992.

Murray, T. & Woolf, B. (1992b). Tools for Teacher Participation in ITS Design. In Frasson, Gauthier, & McCalla (Eds.) Intelligent Tutoring Systems, Second Int. Conf. , Springer Verlag, New York, pp. 593-600.

Murray, W.R. (1990). A Blackboard-based Dynamic Instructional Planner. In *Proc. of AAAI-90*.

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*. Springer-Verlag: Berlin.

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

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

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.

Schank, R., Fano, A. Bell, B. & Jona, M. (1994). The Design of Goal-Based Scenarios. *Journal of the Learning Sciences*, Vol. 3 No. 4.

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.

Spensley, F., Elsom-Cook, M., Byerley, P., Brooks, P., Federici, M. and Scaroni, C. (1990). "Using multiple teaching strategies in an ITS," in Frasson, C. and Gauthier, G. (eds.), *Intelligent Tutoring Systems: At the crossroads of Artificial Intelligence and Education*. Norwood, NJ: Ablex.

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.

Van Marcke, K. (1992). Instructional Expertise. In Frasson, C., Gauthier, G., & McCalla, G.I. (Eds.) *Procs. of Intelligent Tutoring Systems* '92. Springer Verlag, Berlin.

VanLehn, K. (1987). "Learning One Subprocedure per Lesson," *Artificial Intelligence*, Vol. 31.

VanLehn, K. (1988). "Toward a Theory of Impasse-Driven Learning." In Mandl, H. & Lesgold, A. (Eds.), *Learning Issues for Intelligent Tutoring Systems*. New York: Springer-Verlag.

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

Westcourt, K., Beard, M. & Gould, L. (1977). Knowledge-based adaptive curriculum sequencing for CAI: application of a network representation. *Proceedings of the National ACM Conference*, Seattle, Washington, pp. 234-240.

Winne P.H., 1991. Project DOCENT: Design for a Teacher's Consultant. In Goodyear (Ed.), *Teaching Knowledge and Intelligent Tutoring*. Norwood, NJ: Ablex.

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.