International Journal of Artificial Intelligence in Education (1997), 8.71-96

A Multimedia Authoring System for Crafting Topic Hierarchy, Learning Strategies, and Intelligent Models

Wing-Kwong Wong
Dept, of Electronic Eng., National Yunlin Institute of Tech., Touliu,
Taiwan, wongwk@el.yuntech.edu.tw

Tak-Wai Chan
Dept, of Information Eng., National Central University, Chungli, Taiwan.
chan@src.ncu.edu.tw

Abstract: Intelligent computer-assisted learning environments usually involve complex strategies to help students learn a subject domain, possibly with simulated or human companions. The design and implementation of such systems rely much on the powerful features of multimedia authoring systems. Traditional authoring systems can present instructional content in dynamic ways with hypertext links, which implement the instruction strategies (or learning strategies from the student's perspective). However, they generally lack tools for organizing the instructional content and for viewing and editing the organization. Although some systems can use a network model to organize the content materials, this approach embed explicit hyperlinks in the materials. We, as well as some other researchers, argue that authoring tools should organize instructional content and build dynamic learning strategies independently. Based on this observation, we have designed and implemented an authoring system whose pyramid architecture separates the learning strategies from the content organization so that modifying one would not affect the other. Moreover, the system's script language is LISP, which is powerful for implementing intelligent models used in learning systems. The system is called GETMAS—Goal Episode Tree Multimedia Authoring System.

Introduction

Face-to-face instruction such as lecturing is an important means of transferring knowledge to students but it has some drawbacks. For example, the students might have very different background and the lecture might be trivial for outstanding students or too difficult for poor ones; it might take too much preparatory work to demonstrate some physical behavior because of its equipment requirement or dangerous consequences. Fortunately the technology of computer-assisted learning systems can help solve some of these problems. In a well-designed computer-assisted learning environment, a student can navigate the learning materials based on her needs. If she is familiar with a topic, she can skip it and jump to more interesting topics. The environment could allow the student to supply the values of some parameters and run a simulation whose process and results can be viewed on the monitor screen. If the environment has more intelligence, then the student's cognition could be modeled and misconceptions diagnosed. These design features are made accessible to educational professionals with the advancement of multimedia authoring systems. To understand the strengths and weaknesses of some of the currently available technologies, we have studied some research and commercial authoring systems, including Macintosh's HyperCard and Allegient's SuperCard, Asymmetrix's Multimedia ToolBook Version 3.0 for Windows, and Macromedia's

Authorware Professional 2.0 for Windows. To solve some of the problems which are not addressed by these and other similar tools, we have designed and implemented a multimedia authoring system called *GETMAS.* Of course, there exist more updated versions of the software mentioned above and some high-end commercial and propriety software, such as Sybase's Gain Momentum and SK8 of Apple's Advanced Technology Group, (http://trp.research.apple.com/SK8_ Information/default.html) which are not covered in this paper and which might provide their solutions to these problems.

The next section provides some background on currently available commercial multimedia authoring technologies. Against this background, the issues of management of instruction strategies and organization of instructional content are then explored in two sections. Then arguments for separating strategies from content are presented. The next three sections presents the architecture of GETMAS, its script language and its system implementation. The next section describes two sample applications in order to show some of the capabilities of GETMAS. Then other related research systems are compared to GETMAS. The final section suggests several directions we can take to improve the system and draws some concluding remarks.

HyperCard, SuperCard, and Multimedia ToolBook

HyperCard, developed by Apple for the Macintosh personal computer, is one of the earliest commercial multimedia authoring tools. Its multimedia capability is strong because it can play and manipulate various types of multimedia resources. A HyperCard application has a simple architecture with two levels of organization. The upper level is the root node called the *stack,* which has a number of children nodes called the *cards* (Figure 1). The stack is actually a file that stores the application's data. A card is the basic unit for editing and displaying the multimedia objects in a window on the monitor. HyperCard's script language is called HyperTalk. This is a high-level language resembling natural language and is easy to learn but its speed is slow since it is interpreted rather than compiled.

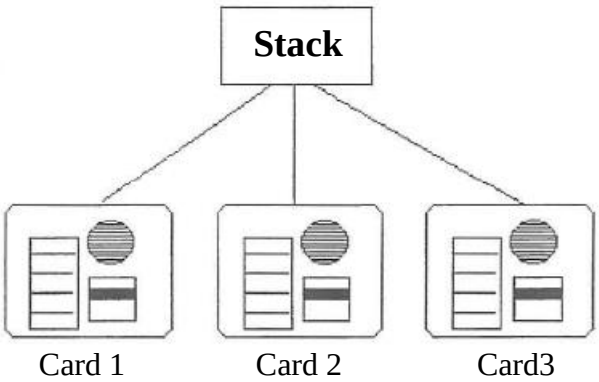

Figure 1. HyperCard applications architecture

The bi-level architecture of HyperCard obviously does not support hierarchical organization of the domain knowledge, whose hierarchy can have any number of levels in general. Consider the partial topic hierarchy of the subject Physics shown in Figure 2.

Each node contains the learning materials for the topic labeled at the node. This tree is similar to the table of contents in an introductory Physics textbook. With HyperCard's bi-level architecture, the four-layer curriculum tree would be forced to flatten into two layers, as shown in Figure 3.

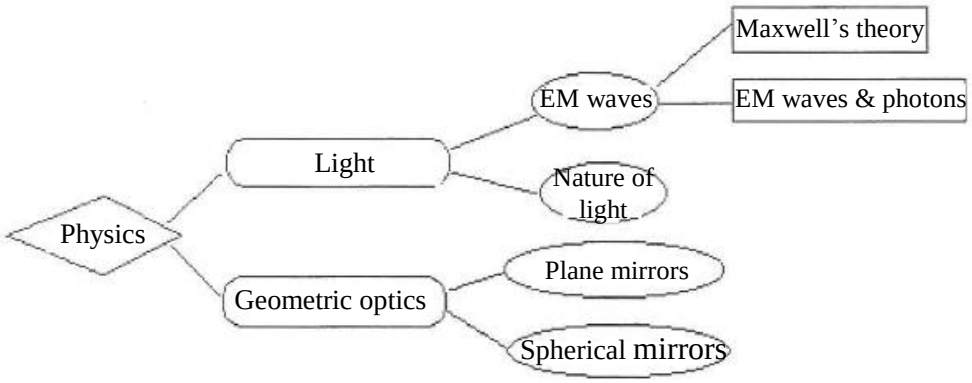

Figure 2. Partial hierarchy of topics in introductory Physics

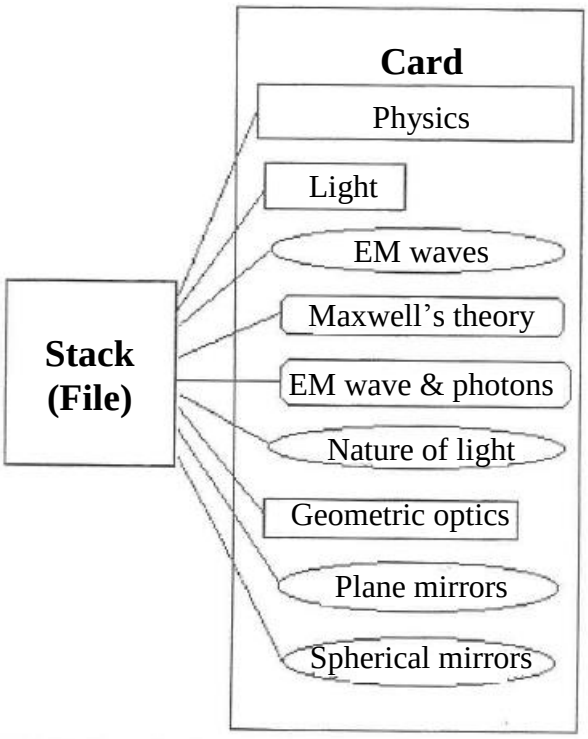

Figure 3. Organization of the Physics tree on HyperCard

Allegiant's SuperCard, an extension of HyperCard, was released in 1989. SuperCard extends HyperCard's bi-level architecture to a three- level one. It inserts a level of window between the top level and the bottom level (Figure 4), meaning that each card has its own window. A SuperCard application's root node is called the *project*, whose children are called *windows*. Each window has one or more cards under it. Each window and its children cards form a *stack*. SuperCard's script language is called SuperTalk, which is similar to HyperTalk. Though an improvement over HyperCard, SuperCard's

three-level architecture is still not sufficient to support a general hierarchical organization of domain knowledge with four or more levels.

Asymmetrix's Multimedia ToolBook for PC is very similar to HyperCard. It has a bi-level architecture and a high-level script language called OpenScript, which is interpreted during execution and is not targeted for AI programming. Nevertheless, the script language feature allows the option of adding extension components with software methods such as DLL (dynamic linked library) and DDE (dynamic data exchange). A research system which adopts this approach will be described in the related work section below.

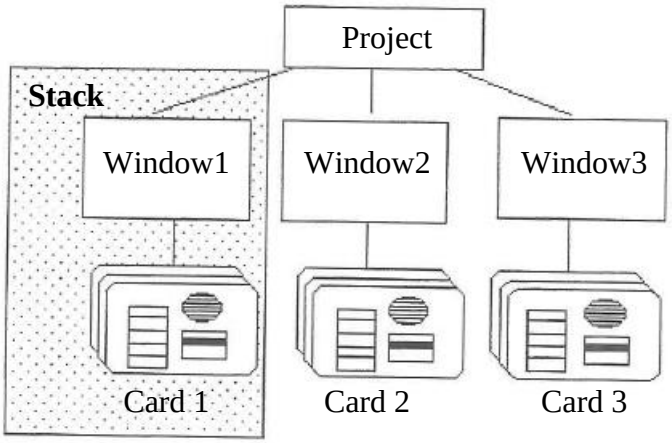

Figure 4. The architecture of a SuperCard appEcation

HyperCard, SuperCard and Multimedia ToolBook are indeed powerful multimedia authoring systems. They can present fancy multimedia materials and implement complex navigation sequences. However, they lack tools for organizing the presentation materials and for viewing such organization and lack a centralized map of all navigation links. It should be noted that they allow an author to manually organize presentation materials in a hierarchy. Yet these authoring tools still provide no tool to view the entire hierarchy in any abstraction form. Similarly, the navigation links are scattered through all the cards and pages of the application, making it hard for the author to manage the navigation complexity. These issues will be further explored in the following sections.

Management of Hypertext Links

Tool for constructing hypertext links to navigate content materials is an essential feature for a multimedia authoring system and all authoring systems we encounter have this feature. In order to implement an intelligent computer-assisted learning system which stresses personal and adaptive learning for students, hypertext link is a must. When the student is already familiar with a topic, she can choose to skip it by jumping to the next topic. When the student fails a test, she can jump to review the topics that are covered by the test. When there exist other human or simulated learning companions, the interaction protocol for all involved agents can be viewed as a set of navigation links—an agent would take certain action depending on both the previous action taken by another agent

and the current state of the interaction. Consider a learning environment where a tutor is monitoring the progress of a student, when the student poses a question, the tutor is obliged to respond to her. For examples of social learning systems, see [Chan 95]. Thus, hypertext links can be viewed as an implementation of one form of teaching strategies. If we take the student's perspective, hypertext links define possible learning strategies for the student.

As a learning environment gets larger when its instructional content grows, its learning strategies can become quite complex and the management of navigation links is a serious problem for the author of the system. The problem is worse if the navigation strategies are tentative and subject to revision based on the evaluation of learning effectiveness when the system is repeatedly tested and laboriously and skillfully improved (this is why this paper's title says learning strategies and other features of learning environment are *crafted*, instead of simply *built*)—which is often the case for most learning environments. For traditional authoring systems, the navigation links are specified in scripts that are attached to user-interface objects such as a push button in a window. Since the interface objects are scattered all over the system and the navigation codes are embedded in scripts that are attached to the instructional content, this often produces *spaghetti networks* of information, similar to the use of the *goto* statement in early programming ([DeYoung 90]). As a result, it is difficult for the author to keep track of all the navigation links and the learning strategies they implement.

The authoring system Authorware offers a nice *visual* solution for managing the complexity of navigation links (e.g. see [Koegel & Heines 93]). When the author is developing a computer-assisted learning system, she does so by constructing a visual network to model the navigation control of the system (Figure 5). A node in the network represents a unit of learning materials or a decision point where navigation to one of several destinations is triggered by the previous user input or system message. Authorware offers a number of node types for manipulating different types of media and for controlling navigation. To help manage the size of a network, Authorware allows a subnetwork to be abstracted as a node so it is convenient for the author to construct a subnetwork small enough to fit a single screen at each level of abstraction. For example, Figure 5 shows a sample application provided by the commercial software Authorware Star, which is a smaller and more economical version of Authorware Professional. The application includes at least two networks, one on the left and one on the right. The network on the left, labeled "present.asw", defines the top-level (level 1) control flow of the presentation. The network on the right, labeled as "Off", is an explosion of the "Off" node on the top-level network and so is marked as "Level 2". Thus, at any point during the system development process, the author can view the global shape of the network at different levels of details. This visualization helps the author to understand some of the learning strategies the system currently provides.

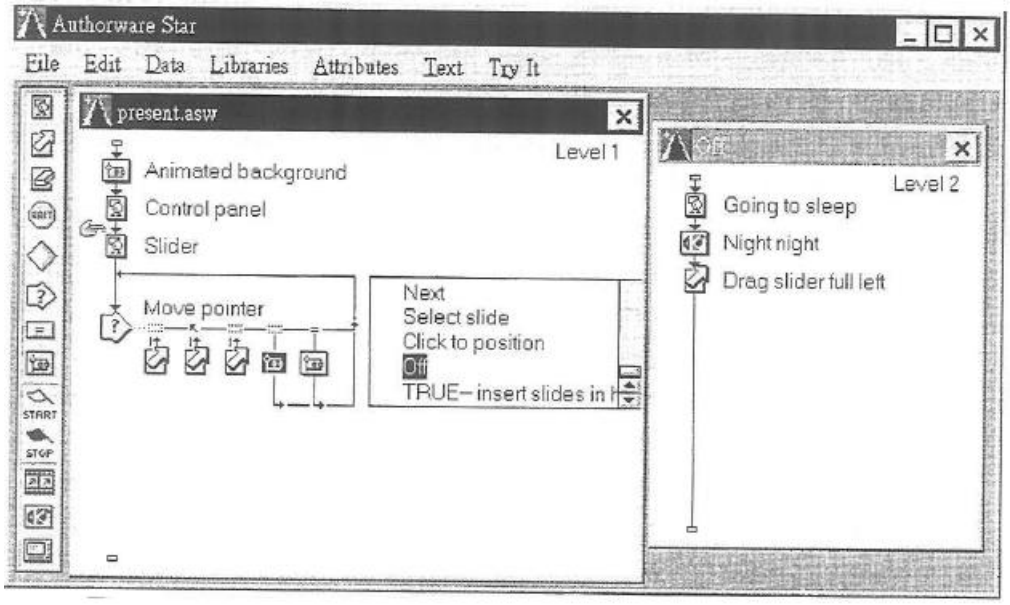

Figure 5. A Network of Presentation Materials in Authorware Star

Management of Hierarchical Topic Organization

We take it for granted that almost all text books are arranged hierarchically, with chapters, sections, subsections, paragraphs, etc. This indicates that when there is a large amount of information, a natural way to organize the information is to do so hierarchically. It helps the author to manage the organization when producing and organizing the learning materials. It also helps the reader to understand the global structure of the domain knowledge and to locate the information the reader needs. When an author develops a computer-assisted learning system, tools for organizing the learning materials hierarchically would make the development a much more pleasant task.

All authoring tools allow manual hierarchical organization of the learning materials. That is, the author can first produce the hierarchical design with pencil and paper, implement the design by grouping the relevant learning materials, and store the implementation in a hard disk. However, since the learning materials are under development and so are constantly updated electronically, the emerging hierarchy would soon be different from the original design. Unless the pencil-and-paper hierarchy design is updated manually and synchronously with the actual electronic hierarchy—a discipline that is painful for any author to keep, the only way to find out the current status of the organization is to go through the learning materials page by page on the monitor screen. This is a serious problem when the learning materials grow to a big enough size. So this manual, pencil-and-paper approach is not a good solution. [Mayer et al. 93] addresses this problem for the ToolBook system, which is an early version of Multimedia ToolBook, by automating the construction of a table of contents which lists the section titles that are specially marked in the learning materials.

This management problem for organizing instructional content can be solved with an explicit electronic representation of the hierarchy. This is like turning the paper design into an electronic version and requires the system to make the hierarchy representation reflect the updated organization of the learning materials. In this way, the author can

understand the current organization by just looking at the hierarchy representation without going through the actual learning materials at each node of the hierarchy. Moreover, a subtree can easily be removed from its parent and attached to another node. This makes development and reorganization much easier. We call this solution an *editable representation of the material organization.*

Nonetheless, Authorware can claim to solve this problem indirectly. The previous section has described the network architecture of Authorware for modeling the navigation control of a' learning system. The same network model can be used to organize the learning materials hierarchically. Each node can refer to a unit of learning materials and can have links to zero or more children nodes. Also, each subtree can be abstracted as a single node. In short, even though the network model of Authorware is intended for dynamic navigation control, the navigation network can be viewed as a static organization of the learning materials.

Why Separate Instruction Strategies from Instructional Content

Though Authorware[1] s navigation network can be constructed hierarchically and treated as an organization of the learning materials, we believe that such treatment is not natural. Since the learning materials form a subject domain, they can generally be grouped as a static hierarchy, like the chapters and sections in a common textbook. On the other hand, the teaching strategies that an author uses are of a dynamic nature. For example, skipping familiar topics, reviewing unfamiliar topics, and communication protocol for learning companions are strategies that should be independent of the static organization of the learning materials. Therefore, we believe that the dynamic navigation control and the static organization of learning materials should be handled independently. Below, we review similar and additional arguments that are presented in related research.

Traditional multimedia authoring tools such as those described at the beginning of this paper use the "story board" approach—it mixes the instructional content with instruction flow ([Murray 96]). Inserting a new topic as a branch flow means explicitly encoding the branch to the content. Thus, powerful features such as "edit-in-place" are compromised or lost when the tools are forced into a form which separates instructional content and instructional strategies. Murray argues for the separation of the instructional content (or knowledge base) from the hyperlinks that navigate the materials and calls this method the "knowledge based" approach. It puts forth four advantages of the knowledge based approach. First, if the author wants to change when hints should be given, she needs only change a single rule, which is checked whenever the user reads across any material where hints can be given. A sample rule can say that if the user failed to a couple of test items that involve concepts occurring in the current exercise, hint should be given. Second, the content of the knowledge base can be used for different purposes, e.g., summarization, introduction, or testing of a topic. Third, the instruction is more learner-centered, since students are more free to choose what topics to learn and how it should be presented, e.g., summary, introduction, exercise. Fourth, the authoring tools for a knowledge based system can provide multiple and abstracted perspectives of the instructional content. Beginner students and advanced students can pick the perspective that suits their needs while authors can easily view, inspect and navigate through these knowledge bases.

[Stubenrauch et al 93] proposes future hypermedia systems to be large, distributed, dynamic, linkless, and structured. It distinguishes two types of links: structural and referential. Structural links are for organizational purpose—they connect nodes structurally in a certain context. For example, they can be the hierarchical structure of chapters and sections, etc. like the table of contents in a book, or the alphabetically ordered keywords or authors like the index at the back of a book. Referential links are like all the possible learning paths a user might take in reading the contents of the hypermedia documents. Our approach adopts this distinction—it organizes the instructional content hierarchically and it constructs all possible learning paths with hyperlinks independently.

Pyramid Architecture for Navigation and Organization

The above arguments for separating instructional content and strategies motivate the architecture of the multimedia authoring system that we have designed. The system is called GETMAS—Goal Episode Tree Multimedia Authoring System. With GETMAS, an author can build the knowledge hierarchy of a subject domain and then design the learning strategies to control how a student navigates through the hierarchy to study the instructional content.

GETMAS provides a tool for the author to organize the instructional content hierarchically. This hierarchy is called the *goal tree* (Figure 6), which consists of a unique root node and other nodes. Each node has zero or more children nodes. The child-parent relation is graphically represented by a line linking the parent node to a child node below it. Each node is called a *goal* node, representing a learning goal for a student to achieve. Besides having children links, each goal node also has an *episode* node attached to it (Figure 6). Each episode contains a number of *cards* attached to it. It is the cards that contain the learning materials, i.e., texts and other multimedia resources, to be presented to the students. In this way, an episode groups together a small collection of closely related learning materials.

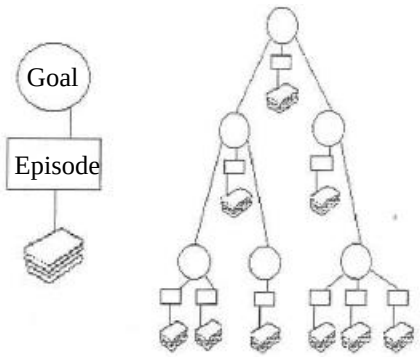

Figure 6. Static topic organization in GETMAS

Figure 7 shows the goal tree of a multimedia demo application. It has a root node whose unique ID is 200. It has four children nodes, each of which has a card to demonstrate different multimedia capabilities of GETMAS.

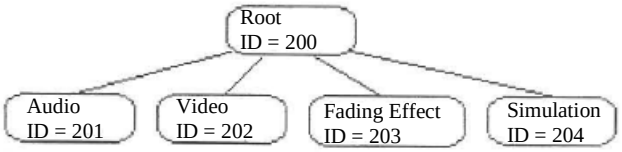

Figure 7. The goal tree of a demo application

GETMAS provides a dialog box to let the author do several things to a goal node. First, she can add and delete children nodes of the goal node. Second, she can browse the episode cards under the goal node. Finally, she can edit the script of the node to control its behavior, including possible transition from its episode to another episode of another node.

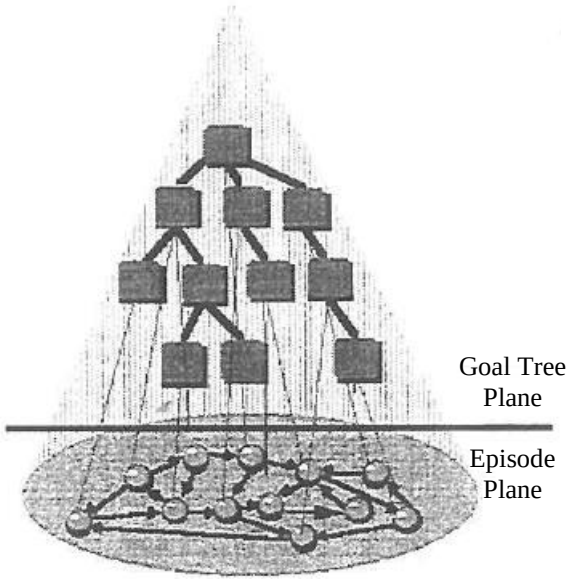

Figure 8. Pyramid architecture for a GETMAS application

The architecture of a GETMAS application can be depicted as a pyramid (Figure 8). The upper part is the goal tree while the bottom layer is a collection of episodes with hypertext links connecting the episode nodes, which are attached under the nodes of the goal tree. There are two types of hypertext links in GETMAS: links connecting episodes *(episode links)* and links connecting cards *(card links*) in an episode. No links are allowed to connect a card of an episode to another card of a different episode. This is consistent with the organization principle that closely related learning materials are grouped under a single episode. In any case, transition between episodes can be made via episode links.

Suppose an author wants to jump from card A to card B when the user presses a button in card A. The author needs to take two steps. First, she builds a button on card A. Then she writes a script for card A that tells the application to jump to card B when the button is pressed. An example is given in Figure 9a. Here we have three cards, A, B and C. For the Next button of card A, the script *"On-MouseUp (go-card card-B) End-MouseUp"* says that if the mouse is up when pointing at the button, control will pass from card A to card B. Similarly, when the Next button of B is clicked, control will pass to card C; when the Next button of card C is clicked, control will pass to card A. The card transition is

represented as the finite state transition diagram in Figure 9b.

Since the cards under an episode are closely related in their contents and there are usually a small number of cards under a single episode, the card links are often quite simple, with transitions like *next, previous, first,* and *last.* In contrast, episode links implement learning strategies, which can be very complicated. Therefore, a design decision is made for GETMAS that a single centralized *episode transition table* is used to store all the transitions among episodes. This is important because a centralized transition table presents a clear *navigation map* which helps explain the embedded learning strategies. Since good learning strategies can only result from repeated experimentation, it is "author friendly" to provide a handy tool for depicting and editing the current strategies.

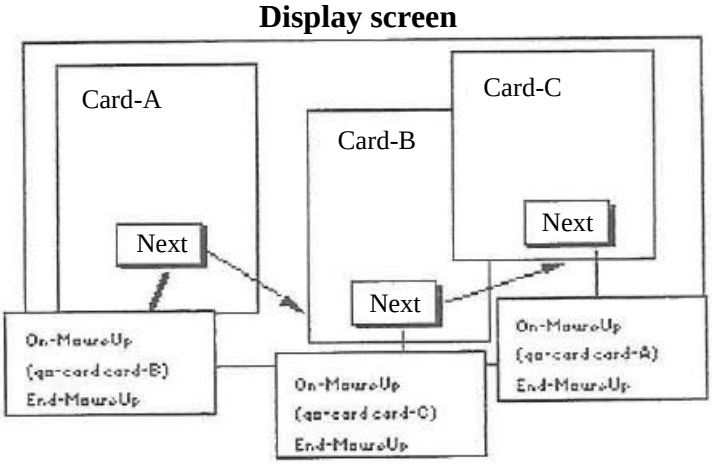

Figure 9a: Scripts that control card transition

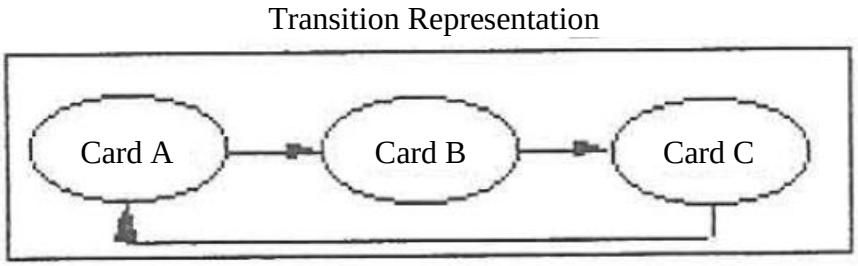

Figure 9b: Card transition 'diagram

The following two figures demonstrate the transition table of a sample application. Figure 10a shows the five transitions for the application. The first entry says that Episode 1 transits to Episode2 when the Next message is received while the second entry says that Episode1 transits to Episode4 when the Previous message is received. The delivery control of the Next and Previous messages is specified in scripts associated with the corresponding episodes. Figure 10b represents the transition table as a state transition diagram.

Start Episode	Transition Message	End Episode

Episode1	Next	Episode2
Episode1	Previous	Episode4
Episode2	Next	Episode3
Episode3	Next	Episode3
Episode4	Next	Episode1

Figure 10a. An episode transition table

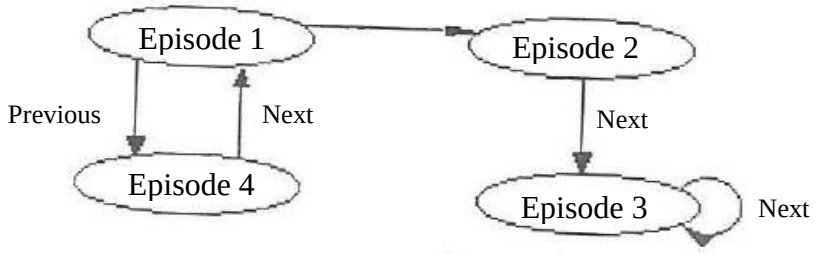

Figure 10b: An episode transition diagram

The origin of this pyramid architecture can be traced back to the proposal of *curriculum tree* [Chan 92], which is a knowledge-based architecture for building intelligent tutoring systems. The non-leaf nodes of a curriculum tree are called *scheduling nodes,* which are similar to GETMAS' goal nodes, and the leaf nodes are called *episode nodes.* Each node contains rules about the domain knowledge and the protocol strategies for controlling navigation among the nodes and for coordinating actions of a student and other simulated agents. A node inherits rules from its ancestor nodes so similar rules need not be duplicated in every relevant node. On the other hand, the rules are scattered throughout the nodes and so are not as easily managed as the teaching strategies implemented with GETMAS' transition table.

AI Script Language

For instructors who want to design intelligent computer-assisted learning systems, they need an authoring system with a script language that is targeted for AI programming. Common AI techniques used in some classical learning systems are inferencing (e.g. SCHOLAR [Carbonell 70]), natural language processing (e.g. SOPHIE [Brown et al. 82]), tutoring heuristics (e.g. WHY [Steven et al. 78]), and student model (e.g. GUIDON [Clancey 79]). They are usually implemented with LISP, Prolog, or other expert system shells or knowledge representation languages. Since AI programs are usually built as prototypes to demonstrate the feasibility of AI ideas and short development time is desired, languages such as Basic and C are not good candidates for such purpose. Unfortunately, most authoring systems use script languages that are like Basic and so are not intended to do AI programming. To solve this problem, GETMAS uses LISP as its script language. This is a natural choice since GETMAS itself is implemented with LISP and using LISP as the script language does not compromise the efficiency of the system. Though there is a myth that LISP is not an efficient language, the myth is not true—LISP can be nearly as efficient as C [Norvig 92] and GETMAS runs almost as fast as SuperCard according to our experience.

System Implementation

GETMAS is developed on the Macintosh platform with Macintosh Common Lisp (MCL) Version 2.0. MCL provides a set of multimedia objects in the Common Lisp Object System (CLOS) class library. Two major objects are *view* and *dialog item*. Besides using these objects, GETMAS also calls some low-level trap functions for displaying graphics objects on screen and uses some low-level C functions. Based on the CLOS objects, GETMAS develops more multimedia objects, such as radio button, scrolling-field, polygon, and arc. Sometimes, we need to rewrite some methods of the CLOS objects to provide the behavior we want. In GETMAS, objects can be classified in two types. The first type includes the editable user-interface objects such as graphics, buttons, and text fields. The second type includes the objects concerned with the organization and navigation of the application such as project, goal, episode, background, and card.

GETMAS' system architecture can be viewed at two levels: design time and run time. There are six design time components, as shown in Figure 11.

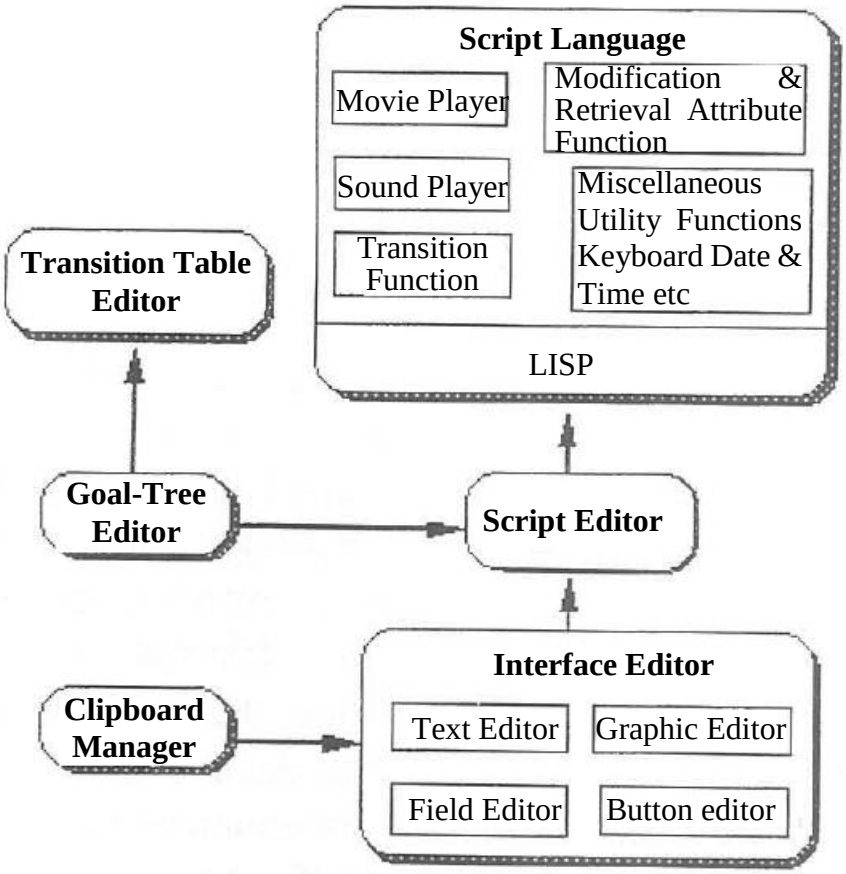

Figure 11. GETMAS authoring tools for the author

1. *Script language:* LISP is the script language. This language is suitable for quick development of system prototypes and for implementing AI techniques. Moreover, the computation speed of LISP is quite fast when compared with SuperCard. In addition to the LISP functions and libraries, GETMAS also provides some multimedia functions such as video and sound playback.
2. *Script editor:* All LISP scripts are edited with a script editor.
3. *Clipboard editor:* This editor mainly takes care of data exchange among applications and the operating systems. The most common data for exchange are graphical images.
4. *Interface editor:* This editor provides tools for editing texts, geometric objects, buttons, and text fields. It imports graphical images and resources through the clipboard. GETMAS provides 24-bit true color screen. This feature slows down the system during display.
5. *Transition table:* GETMAS provides a global episode transition table and for each episode a card transition table.
6. *Goal-tree editor:* With this editor, the author attaches episode to or deletes episode from a goal node and inserts cards to or deletes cards from an episode.

The run-time architecture of GETMAS is similar to other event-driven systems and has five components:

1. *Event manager* is responsible for delivering messages to the intended objects in the systems. The messages are invoked by external events such as user's button click or events internal to the system such as navigation control messages.
2. *Object:* When an object in GETMAS receives a message from the event manager, it will call the runtime interpreter to interpret the message with respect to the object's script. This might trigger the object to send messages to the event manager or other objects.
3. *Runtime interpreter* is responsible for interpreting the script of an object to match messages for the object. If it runs across a function that deals with card or episode transition, it asks the transition manager to help. If it runs across a function that deals with network communication, it asks the network communication manager to help. For other tasks, it can complete by itself.
4. *Transition manager* switches from a card to another or from an episode to another.
5. *Network communication manager* sends out messages to objects through the Apple Event module, which connects a number of Macintoshes in a local area network.

Sample Applications

Three's Company

An example we have developed with an early version of GETMAS is known as the *Three's Company*, which is an intelligent computer- assisted learning system with two simulated learning companions. When using the system, the user goes through three phases: introduction, test, and regular phases. During the introduction phase, the user watches a short video describing some achievements of the research of neural networks. In the test phase, the user is presented with five questions about the video. The system uses the results to set up the initial proficiency level of the user. If the user gives four or five correct answers, then the user's level is superior. Two or three correct answers

implies the mediocre level. Zero or one means inferior. Depending on the user's proficiency level, the system sets up some initial attributes such as confidence and challenge for the student with the following rules.

Rule Group 1:
If (user is superior)
then (user has high confidence & faces medium challenge).
*If (*user has high confidence & faces medium challenge)
then (increase challenge for user).
If (increase challenge for user)
then (increase companions' relative performance & make question more difficult).

Rule Group 2:
If (user is mediocre)
then (user has unknown confidence & faces unknown challenge).
If (user has unknown confidence)
then (increase confidence for user).
If (increase confidence for user)
then (make question easier).

Rule Group 3:
If (user is inferior)
then (user has low confidence & faces high challenge).
If (user has low confidence)
then (decrease challenge for user).
If (decrease challenge for user)
then (make question easier).

The regular phase comes after the test phase. This phase presents six questions to the user. After the user answers each question, the learning companions react in a way depending on the correctness of the user's answer. For example, if the user's number of correct answers is greater than that of a companion, the companion answers the question correctly. Otherwise, the companion uses its past correctness rate to determine whether to answer correctly or incorrectly. More specifically, if its past correctness rate is 40%, then it generates a random number between 1 and 100. If this number is less than 40, then it gives a correct answer. Otherwise, it gives an incorrect answer.

Learning Lisp Recursion

The second sample GETMAS application is an intelligent learning environment for students to learn recursive programming with LISP. The learning environment consists of three main components: Three's Company, Petal, ITS-Petal. This Three's Company is a modification of the Three's Company system discussed above. This Three's Company provides two simulated learning companions and a simulated teacher to interact with the student. It offers three successive stages of learning: competition, debugging, and fading. During competition, companions compete with the student. Such competition is done with intention to increase the student's motivation to learn. Then the student will be asked to take some tests. If the student gets less than 60%, then the system will jump to an

earlier episode for the student to review the relevant LISP concepts. Otherwise, the student proceeds to the stage of debugging. During debugging, the teacher presents some LISP programs with bugs and ask the student to find the bugs. If the student gets stuck, the companions will suggest hints. If the student cannot find the bugs even with the given hints, the teacher then suggests the student to jump to review earlier chapters. During the fading stage, the companions and the teacher will offer less and less help to the student. This forces the student to take up more and more responsibility as a problem solver. Similarly to the previous stages, when the student performs poorly, the teacher will ask the student to review earlier episodes. A typical screen for the Three's Company component is shown in Figure 12. The teacher, who is facing the students, asks the three students who invented the Lisp programming language, with the multiple choices of answers McCarthy, Minsky, Newell, or none of the above.

The second major component of the learning environment is Petal ([Bhuiyan et al. 92]). Petal is a tool scaffolding system that a student can use to minimize syntactic errors when writing LISP programs. It provides a number of primitive LISP functions such as "car", "cdr", "null?" that the student can use to write simple recursive functions. To choose these functions, the student needs only to push a button. Also, the display shows how many arguments the function needs so that the student will not provide less or more arguments. After the student completes a program, the system evaluates the program with sample inputs and determines whether the program computes the wrong results.

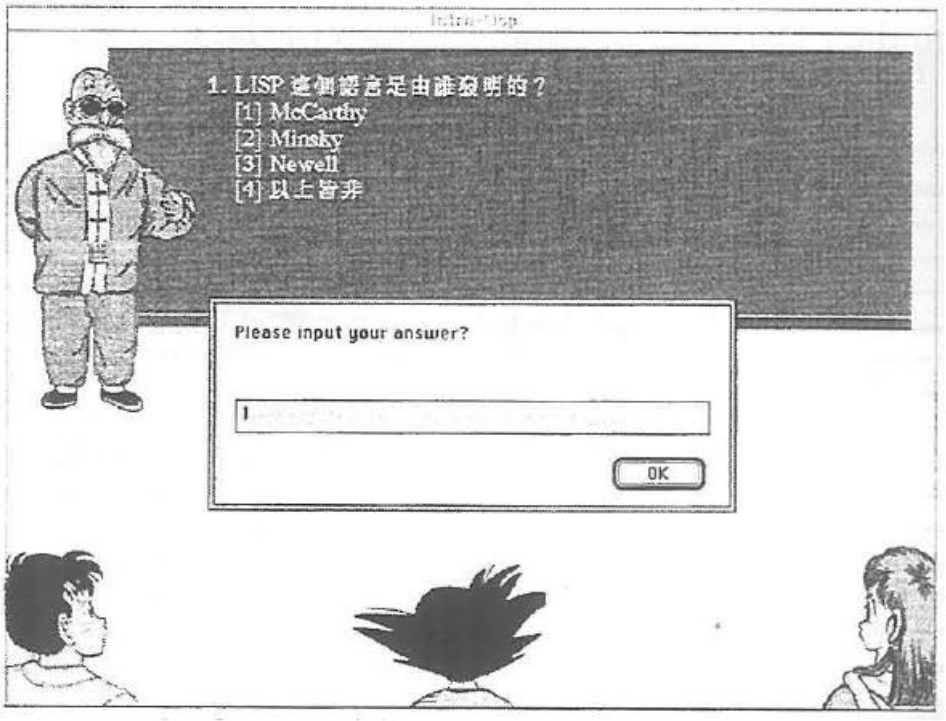

Figure 12. Competition unit in Three's Company

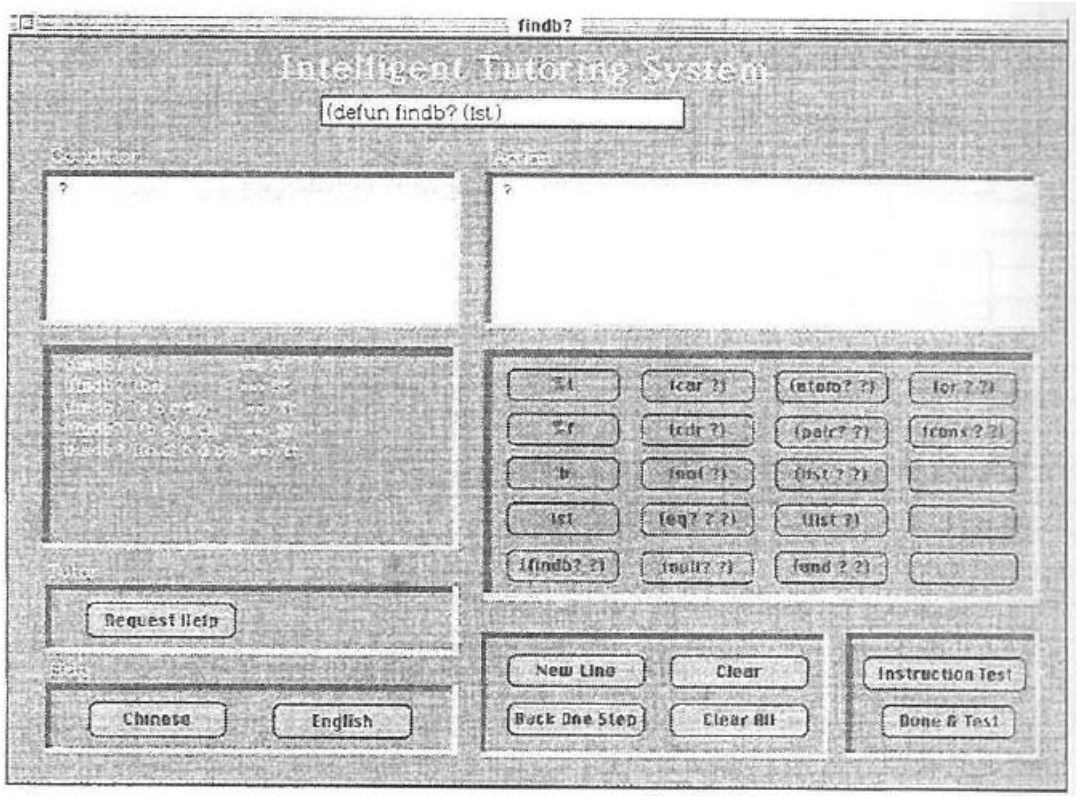

Figure 13. ITS Petals interface for editing LISP programs

The third component of the learning environment is ITS-Petal. In addition to the basic Petal interface and functionality, this system suggests hints to students when she gets stuck during program development. This idea comes from the Reciprocal Tutoring-Kid ([Chan & Chou 95]). When the student runs into some problem, she can ask the tutor for help. The tutor uses a discrimination net to determine the current status of the student's program and gives a corresponding hint. The ITS-Petal tool for constructing LISP program is shown in Figure 13.

What learning strategies are used in this learning environment? Some simple strategies include handy on-line reference materials, friendly user interface and attractive multimedia resources. Navigation strategies are shown in the episode transition diagram of the application (Figure 14).

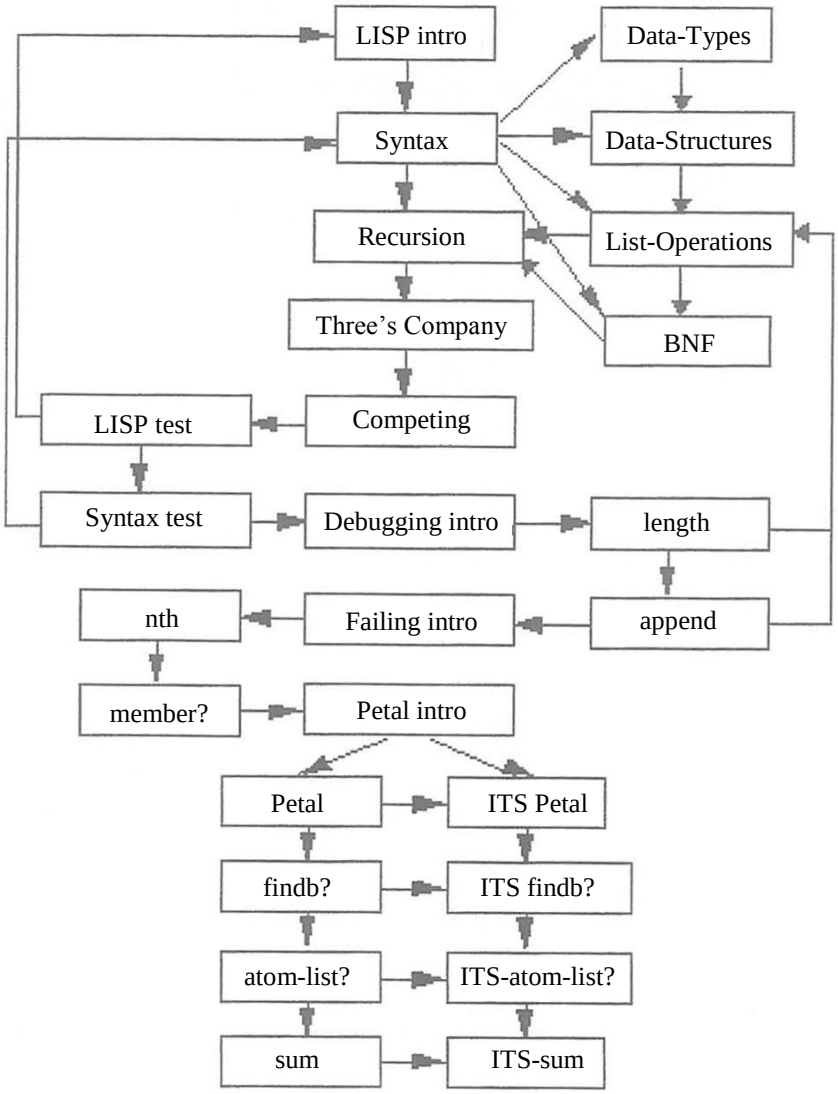

Figure 14. Episode transition diagram for learning Lisp recursion

The equivalent transition table is given in Table 1. For example, consider the goal node labeled as "LISP test" after the "competing" node under the "Three's Company" node. If a user fails a test, the system will jump from the "LISP test" node to the "LISP intro" node which is an episode that explains the background LISP concepts needed for the test. In the Petal mode, if the student performs poorly, the system will suggest the student to jump to use the ITS-Petal system, which offers more help to the student. The navigation can occur in four ways: from "Petal" to "ITS Petal", "findb?" to "ITS-findb?", from "atom-list?" to "ITS-atom-list?", from "sum" to "ITS-sum".

Table 1:
Transition table corresponding to the transition diagram in Figure 14

Start Episode	Transition Message	End Episode
LISP introduction	Next-Episode	Syntax
Syntax	Data-Types	Data-Types

Syntax	Data-Structures	Data-Structures
Syntax	List-Operations	List-Operations
Syntax	BNF	BNF
Data-Types	Next-Episode	Data-Structures
Data-Structures	Next-Episode	List-Operations
List-Operations	Next-Episode	BNF
BNF	Next-Episode	Recursion
Syntax	Next-Episode	Recursion
Recursion	Next-Episode	Three's Company
Three's Company	Next-Episode	Competition
Competition	Next-Episode	LISP test
LISP test	Next-Episode	Syntax
LISP test	Review	LISP introduction
Syntax test	Next-Episode	Debugging introduction
Syntax test	Review	Syntax
Debugging intro	Next-Episode	length
length	Next-Episode	append
length	Review	List-Operations
append	Next-Episode	Fading introduction
I append	Review	List-Operations
1 Fading introduction	Next-Episode	nth
nth	Next-Episode	member?
member?	Next-Episode	Petal introduction
Petal introduction	Petal	Petal
Petal introduction	ITS	ITS Petal
Petal	Next-Episode	findb?
Petal	Help	ITS Petal
findb?	Next-Episode	atom-list?
findb?	Help	ITS-findb?
atom-list?	Next-Episode	sum
atom-list?	Help	ITS-atom-list?
sum	Help	ITS-sum
ITS-Petal	Next-Episode	ITS-findb?
findb?	Next-Episode	atom-list?
atom-list?	Next-Episode	ITS-sum

In summary, the system provides progressive and adaptive help for the student at different stages of the learning process. First, the student grasps introductory concepts with the help of hypertext and reference materials. Then she is involved in learning by discussing with the companions and the teacher. She learns more about her level of understanding during peer competition and proceeds only when her basic understanding reaches an acceptable level. Then in the Petal mode, she learns without worrying about LISP syntax. When facing repeated discouraging failures, she can switch to ITS-Petal where she can request hints from a knowledgeable teacher. The student is expected to achieve a certain level of LISP proficiency after she completes working programs that compute correct answers.

Related Research Systems

As mentioned above, [Murray 96] argues for the separation of instruction strategies from instructional content when authoring computer- assisted learning environments. This insight has led to the development of Eon—an intelligent tutoring system (ITS) shell

for building authoring systems. It includes tools to construct the four functional components of ITS: learning environment, domain model, teaching model, and student model. Eon's Interaction Editor, similar to GETMAS' interface builder, helps to design the graphical user interface of the learning environment. The Topic Network Editor sets the domain concepts and their relationship in a network form. This is similar to the Goal Tree Editor of GETMAS, except that Eon's domain concepts and their relationships are described at a finer detail. The Presentation Contents Editor schedules the default presentation sequence of the course materials that users actually see on the screen. The Topic Browser is a tool to specify which presentations are used to teach selected concepts of the domain. The Strategy Editor, similar to GETMAS' episode transition editor, designs teaching strategies that decides what action to take when student acts in certain way, e.g., jump to review an earlier topic when she answers a question incorrectly. Eon supports overlay models to diagnose students' strengths and weaknesses. This is done with rules that compute the student's competency score for domain concepts. Rules are attached at one level (e.g. lesson, topic, topic level, presentation, transaction) and takes inputs from competency scores at another level. While common commercial authoring tools help prepare presentation that are lively and attractive, they lack the tools to support building intelligence models which provide the depth for intelligent tutoring systems. This "Story Board paradigm" of multimedia CAI is transformed into the "Knowledge Base paradigm" of ITS through tools like what Eon provides.

HM-card uses a *data-model* approach to author computer supported educational packages—it treats courseware units as collections of data units ([Mayrhofer et al. 96]). Each collection contains members that are linked in certain ways. There are four collection types. In an envelope collection, any two members are linked to each other. In a folder collection, members are ordered sequentially. Consecutive members are linked to each other through next and previous links. In a menu collection, a unique member is linked to any other member and vice versa. In a freelink collection, the author specifies links in any way she likes. In addition, each collection has a unique head member. Any access to the collection must start with the head member. In GETMAS' terminology, a collection can be said to specify both the topic hierarchy and the hyperlinks. Also, we can describe the goal tree of GETMAS in the terminology of collection. An episode is like a folder collection so that the pages of the episode are connected with previous/next links. Such episodes are nodes in a goal tree. For the goal tree, each node is like a menu collection, since it is connected to its children nodes. The nodes that are connected with hyperlinks, which stand for the learning paths, in a goal tree are like members of a freelink collection. An advantage of this data-model approach is that author is able to build presentations from existing collections and units without the need to learn a programming or script language. However, from the perspective of intelligent tutoring system, HM-Card lacks an element of interactivity—one that depends on the input behavior and knowledge level of the user. For example, if a user's score is too low in a test, the system can jump to an earlier elementary topic. To extend HM-Card to allow greater adaptation to user's performance should be an interesting research topic.

Hypermedia by itself simply provides knowledge to user statically, i.e., the knowledge is all prescribed and does not adapt to user's behavior when she navigates the hypermedia. HyperTutor, an authoring tool, extends hypermedia to include ITS capabilities such as the control of apprenticeship and system adaptation to student behavior ([Perez et al. 95]). This is done with a two-level architecture: hypermedia

component and the tutor component. At the hypermedia level, hyperlinks are defined to link instructional content, which are node clusters of related topics. At the tutor level, concepts are related with pedagogic expression relationships such as prerequisite, part-of, corequisite, next, abstraction, etc. There is a direct correspondence between these concept relationships and the hyperlinks at the hypermedia level. This direct mapping is in contrast to GETMAS' spirit that the content organization of a domain is independent of the hyperlinks that a student might follow in navigating the topics of the domain. The tutor level also contains both the tutoring module, which determines how instruction should be done in adapting to student's behavior, and the student model, which keeps an updated estimate of student's knowledge.

[Muldner et al. 96] describes the design of an authoring system called NEAT (Integrated Environment for Authoring in ToolBook), which provides tools to extend the capabilities of ToolBook. It adopts the "book metaphor" in building hierarchical organization of topics like chapters and sections in a book, and in building hyperlinks in the texts to refer to materials in other parts of the book. This book metaphor is exactly what GETMAS is implementing. NEAT stresses its ability for the user to customize the learning environment, which is beyond the intended capabilities of GETMAS. User can put notes on the side of each page. She can highlight words on a page, and insert or remove indices that refer to other pages. A history list records all the pages that the user has read, which are also shown as "bread crumbs" (footprints) in the table of contents. For the current version NEAT 3.0, each user is assigned a separate user model, which stores updated information of the user. It also allows authors to construct knowledge base, which consists of units, courses, and curricula, with specification of prerequisite relationship among the units and courses. In addition, NEAT has some limited ITS capabilities in providing question templates for the author to construct questions and their answers, which are used to test the users and provide some data for modeling the user's knowledge level.

[Beaumont & Brusilovsky 95] describes two intelligent learning environments that provide adaptive presentation and adaptive navigation support to adapt to user's general expertise and local knowledge states. The first system Anatom-Tutor, for teaching brain anatomy at university level, estimates the student's general knowledge level, which is either beginner or advanced, with a user model based on their undergraduate levels, why they take the course, special perspectives (e.g. histological), etc. Local knowledge of each student refers to the specific concepts that are covered by Anatom-Tutor's lessons taken by the student. Morphologically descriptive style is used for preparing texts targeted for beginner students and functional style is targeted for advanced students. Contents of the lessons adapt to the student's local knowledge level, e.g., concepts that are already known to the students are not covered. The second system ISIS-Tutor teaches the print formatting language of the information retrieval system CDS/ISIS/M. It consists of two major components: hypertext and tutor. The hypertext component includes a knowledge base of a visual network with nodes of teaching operations connecting all concepts needed for the teaching operations, which include reading a concept, analyzing an example, or solving a problem. Hypermedia pages of instructional content is generated dynamically from internal frame-based representation of domain knowledge. These hypermedia provide adaptive navigation support for users in two ways. First, different colors are used to indicate knowledge states for each concept: has unlearned prerequisites (i.e., ready to be learned), has no unlearned prerequisite, learned. Second, goal concepts

that are supposed to be learned in the current lesson are outlined and non-goal concepts are hidden in the lesson's content. GETMAS provides no special tools for implementing features like what Anatom-Tutor and ISIS-Tutor provide. Nevertheless, an author should still be able to use GETMAS to implement these features with considerable programming efforts.

Discussion and Conclusion

In its current version, GETMAS has several limitations. First, its episode transition table would be difficult to read when there are many entries. Since the table is equivalent to a finite state transition automaton which in its graphical form is more intuitive, a natural improvement of GETMAS would be to show the table as a finite state transition diagram, as demonstrated earlier in this paper. Second, even though the navigation links are gathered at the transition table, the scripts that send messages for navigation control are scattered throughout the episodes. A better solution is to gather these transition-controlling scripts at a single script. This would increase the clarity and manageability of the teaching strategies implemented by the navigation links.

Finally, for an application in GETMAS, the goal tree is embedded in the application and the tree is not used by any other application. In the future, we hope to let course authors share a single goal tree. This tree is constructed by domain experts who are knowledgeable with the domain. Courses are to be designed by instructors who are experienced in teaching classes. Depending on the background of the students and the depth of the course, a course author can build two different learning systems using the same goal tree. For example, non-major students can skip some difficult topics that are needed only by advanced courses for major students; a graduate class can skip the nodes that undergraduates are supposed to know. Therefore, domain experts can build a wide and deep goal tree with GETMAS and store the tree as a file. Then a course author can use GETMAS to retrieve the tree and build the navigation links for a specific class of students. In other words, domain knowledge can be centralized in a single tree instead of being duplicated by different applications that share the same domain knowledge, This should save efforts in building goal trees that share a lot of common topics.

We believe GETMAS is a versatile multimedia authoring system. Compared to other authoring systems, GETMAS has several outstanding features. First of all, it allows a hierarchical organization of the instructional content while not limiting the flexibility of navigation links for implementing the learning strategies. This achievement is made possible with two tools. The first is the tools for building and modifying a goal tree for organizing the instructional content. The second tool is a global episode transition table that helps the author to keep track of the learning strategies used in the application. This helps the author to experiment and improve the strategies. Last but not the least, GETMAS uses LISP as both its system and script languages. Since LISP is a great tool for implementing AI techniques and for quick prototyping, GETMAS is a powerful and user-friendly tool for implementing models of intelligence. In conclusion, GETMAS is made for building large-scale intelligent computer-assisted learning environments.

Acknowledgments

We thank the anonymous reviewers, whose elaborate comments help us improve the quality of this paper. We also thank the National Science Council for supporting this research under contracts NSC85-251 l-S-224-006 CL.

References

Beaumont, I. & Brusilovsky, P. (1995). Adaptive educational hypermedia: From ideas to real systems. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 93-97.

Brown, J. S., Burton, R. R., and de Kleer, J. (1982). Knowledge engineering and pedagogical techniques in SOPHIE I, II, and III. In D. Sleeman and J. S. Brown (Eds.). *Intelligent Tutoring Systems.* London: Academic Press.

Carbonell, J. R. (1970). AI in CAI: An artificial intelligence approach to computer-aided instruction. In H. 0'Neil (Ed.), *Proceedings for instructional systems development.* New York: Academic Press.

Chan, T.W. (1992). Curriculum Tree: A knowledge-based architecture for intelligent tutoring systems. *Second International Conference on Intelligent Tutoring Systems.* New York: Springer-Verlag.

Chan, T.W. (1995). Social Learning Systems: An Overview. In B. Collis and G. Davies (Eds.), *Innovating Adult Learning with Innovative Technologies,* IFIP Transactions A-61, North-Holland, 101-122. Also appeared in T. W. Chan & J. Self (Eds), *A Tutorial on Social Learning Systems in Emerging Technologies in Education.* Charlottesville, VA: AACE, 71-96.

Chan, T.W. & Chou, C.Y. (1995). Simulating a learning companion in reciprocal tutoring system. *Proceedings of the First International Conference on Computer-Supported Collaborative Learning,* 49-56.

Clancey, William. (1979). Dialogue management for rule-based tutorials. *IJCAI6,* AAAI, 155-161.

Conklin, J. (1987). Hypertext: An introduction and survey. *IEEE Computer,* Sept., 81-96.

DeYoung, L. (1990). Linking considered harmful. In Rizk, A., Streitz, N. & Andre, J. (Eds.), *Hypertext: Concepts, Systems, and Applications.* New York, NY: Cambridge University Press.

Koegel, J. and Heines, J. M. (1993). Improving Visual Programming Languages for Multimedia Authoring. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 286-293. Mayer, S., Muldner, T., Unger, C. (1993). NEAT: An integrated authoring environment based upon ToolBook. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 332-339.

Mayrhofer, V., Scherbakov, N., & Andrews, K. (1996). HM-Card: A New Approach to Courseware Production. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 433-438.

Muldner, T., Muldner, K., & van Veen, C. M. (1996). The Design and Evolution of an Authoring Environment and Its Applications. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 503-508.

Murray, T. (1996). From Story Boards to Knowledge Bases: The First Paradigm Shift in Making CAI "Intelligent. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 509-514.

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: Case studies in Common Lisp.* San Mateo, CA: Morgan Kaufmann, 265-269.

Perez, T., Lopisteguy, P., Gutierrez, and Usandizaga, I. (1995). HyperTutor: From Hypermedia to Intelligent Adaptive Hypermedia. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 529-534.

Stevens, A.L., Collins, A., and Goldin, S. (1978). Diagnosing student's misconceptions in causal models. BBN Rep. No. 3786, Bolt Beranek and Neewman, Inc., Cambridge, MA.

Stubenrauch, R., Kappe, F., and Andrews, K. (1993). Large hypermedia systems: The end of the authoring era. *Proceedings of Educational Multimedia and Hypermedia.* Charlottesville, VA: AACE, 485-502.