N91-21949

Integrating Knowledge and Control into Hypermedia-based Training Environments: Experiments with HyperCLIPS

Randall W. Hill, Jr.
Jet Propulsion Laboratory
4800 Oak Grove Drive
M/S 125-123
Pasadena, CA 91109

Integrating Knowledge and Control into Hypermedia-based Training Environments: Experiments with HyperCLIPS

Randall W. Hill, Jr.
Jet Propulsion Laboratory
4800 Oak Grove Drive
M/S 125-123
Pasadena, CA 91109

Abstract - In this paper we discuss the issues of knowledge representation and control in hypermedia-based training environments. Our goal is integrate the flexible presentation capability of hypermedia with a knowledge-based approach to lesson discourse management. We represent the instructional goals and their associated concepts in a knowledge representation structure called a *concept network*. Its functional usages are many: it is used to control the navigation through a presentation space, generate tests for student evaluation, and model the student, to name a few. We have implemented this architecture in HyperCLIPS, a hybrid system that creates a bridge between HyperCard®, a popular hypertext-like system used for building user interfaces to databases and other applications, and CLIPS, a highly portable government-owned expert system shell.

1. Introduction

Hypermedia-based learning environments provide two potential benefits that other computer-based instruction has typically lacked: (1) a tremendous variety of ways to present information, and (2) a relatively easy means of navigating through a non-linear information space. The advantages of being able to present instructional information in different media forms are obvious: it accommodates a wide range learning styles, it reduces boredom, and it potentially integrates information in ways that couldn't be accomplished by text alone. The second benefit, the ability to navigate through a non-linear information space, is clearly an improvement over lock-step instructional systems that are typically used in institutional training settings. This enables the student to take a self-guided tour through the hypermedia-based instructional domain without being constrained by a pre-defined control script. The problem, however, is that unconstrained exploration is not always the most efficient way to learn a task or concept. The student may never actually obtain the intended instruction due to getting lost or distracted in a confusing maze of hyper-connections. The same navigational freedom that makes hypermedia so powerful, also means there is less control over the actions of the student. How do we balance the need for instructional control with the need to explore the instructional domain?

The purpose of this paper is to discuss an approach to incorporating knowledge-based control into a hypermedia-based training environment so as to have the best of both worlds: we provide navigational freedom within a carefully selected subset of the instructional domain.To this end, the paper is organized into three sections. In section 2 we discuss the use of HyperCLIPS to manage the interface between hypermedia and a knowledge-based system. In section 3 we describe the use of the *concept network* as a knowledge representation scheme for reasoning about the instructional curriculum, the student, and the presentation. And in section 4 we describe the architecture for integrating knowledge-based control with hypermedia-based presentation.

2. HyperCLIPS = HyperCard® + CLIPS

This section of the paper discusses some practical matters concerning how we created a bridge between a knowledge-based system and a hypermedia system via a tool we have developed called HyperCLIPS. We had a strong motivation for building HyperCLIPS: we

wanted to avoid having to develop either an inference engine or a hypermedia tool from scratch, hence, we sought existing tools that we could marry together via some sort of interface. Consequently, we chose HyperCard®, a hypertext system for the Apple Macintosh®, and CLIPS (C Language Integrated Production System), a highly portable government-owned expert system shell. We do not belabor the particulars of HyperCLIPS in this section: it has already been reported elsewhere [1],[2]. Rather, the purpose of this description is to highlight some of the issues involved in integrating a knowledge-based system with hypermedia, in particular, how control is exercised, and the separation of knowledge, data, and display.

2.1 Building the bridge

We built HyperCLIPS as a bridge between HyperCard and CLIPS. In theory, the interface between HyperCard and CLIPS is natural. HyperCard was designed to be extended through the use of external commands (XCMDs), and CLIPS was designed to be embedded through the use of its I/O router facilities and callable interface routines. As it turns out, there are some limitations to an XCMD: it cannot have global data and it can be no larger that 32K bytes. HyperCLIPS overcomes these limitations with an XCMD called "ClipsX", which when added to HyperCard gives it access to the CLIPS routines: *clear*, *load*, *reset*, and *run*. In addition, an I/O router was added to CLIPS to handle the communication of data between CLIPS and HyperCard.

2.2 Crossing the bridge: HyperCard to CLIPS communication

HyperCard and CLIPS do not run concurrently, rather, they pass control and data back and forth to one another. In order for HyperCard to communicate with CLIPS, it uses the *ClipsX* command along with one of four sub-commands specified as the first parameter. The four sub-commands that can be sent to CLIPS are *clear*, *load*, *reset*, and *run*. A typical use of the ClipsX command is shown in the example below, where ClipsX is called with the parameters *"run"* and *empty*.

```
-- in a HyperCard script
-- assumes the CLIPS program
-- has been loaded and is ready to run.
  ClipsX "run",empty
  get the result
  -- process the results returned from CLIPS
  get line 1 of it
  answer it with "OK"
```

HyperCLIPS Example: ClipsX with the "run" command.

Note that ClipsX is embedded as just another command in a HyperCard script, and it is given two parameters: *"run"* and *empty*. When executed, the ClipsX command deactivates HyperCard and switches over to CLIPS. The "run" parameter instructs CLIPS to execute rules that have been activated by assertions in the fact base. The *empty* parameter is sent when no data is being passed from HyperCard to CLIPS. Otherwise, a data parameter could be sent that would be trapped and parsed by CLIPS rules. Here is a summary of the four different sub-command parameters:

clear - the CLIPS command to clear the rules and facts from the CLIPS environment.
load - the CLIPS command to load a file of rules or fact definitions.
reset - the CLIPS command to initialize the rules and facts that have been loaded into the environment.
run - the CLIPS command to initiate the firing of rules that have been activated by assertions in the fact base.

2.3 Crossing the bridge: CLIPS to HyperCard communication

There are two basic commands used for passing control and data back to CLIPS:

fprintout - this is the I/O command used to pass data from CLIPS to HyperCard. The data given to the fprintout command is buffered and routed to a HyperCard variable called "the result", which is parsed and used in HyperCard.

halt - when embedded in the right hand side of a rule, the halt command suspends CLIPS and passes control back to HyperCard.

```
; in a CLIPS program
(defrule get_data
  ?f < - (phase get_data)
  =>
  (retract ?f)
  (fprintout t "need data" crlf)
  (assert (get_data_continue))
  (halt))

(defrule get_data_continue
  ?f <- (get_data_continue)
  =>
  (retract ?f)
  (bind ?data (read))
  (assert (data ?data)))
```

Example: Passing Control to HyperCard from CLIPS

In this example CLIPS sends "need data" to HyperCard and then halts so that control can be passed back also. Once CLIPS is activated again, the *get-data-continue* rule will bind the HyperCard data to a variable, which can subsequently be asserted to the fact base. Note: in addition to the *halt* command, control is also passed back to HyperCard automatically whenever the rules finish firing.

3. The Knowledge Needed to Control Instructional Interaction

3.1 The Role of Knowledge in Hypermedia

Our goal is to provide adaptive tutoring to the student by combining the flexibility of hypermedia to present the information with the representational and reasoning power of a knowledge-based system to model the student and control the interaction. We view the separation of the hypermedia from the knowledge-based system in terms of the functionality that each will implement. The hypermedia environment is used to represent information in human understandable form through the use of non-linear text, graphics, animation, video, audio, and any other available communication medium. The knowledge-based system is responsible for: (1) planning which information to present in order to meet a set of instructional goals, (2) modeling the student's apparent knowledge of the instructional domain through the use of evaluative tests and exercises, and (3) dynamically adapting the lesson to meet the needs of each student based on the modeling performed in (2). The problem, then, is linking the knowledge-based planner and student modeler with the hypermedia-based information presentation system. There must be a mapping between the information encoded in the hypermedia displays and the knowledge used to plan and monitor a lesson. The way in which we have chosen to make this link is through the use of a representation we call a *concept network*.

3.2 The Concept Network

The *concept network* is a form of a semantic net, a knowledge representation scheme that has been in use for over twenty years [3]. The concept network derives its meaning

from three sources: *nodes*, *links*, and *functions* that work on the network. The *nodes* in the concept network have associations with objects in the hypermedia-based instruction environment. The nodes are divided into types, all of which have a special semantic meaning within the context of the instructional environment. They are *linked* by a set semantic relations that provide structure to the network. The *functions* take the concept network and perform operations that are keyed on the structure and meaning of the network.

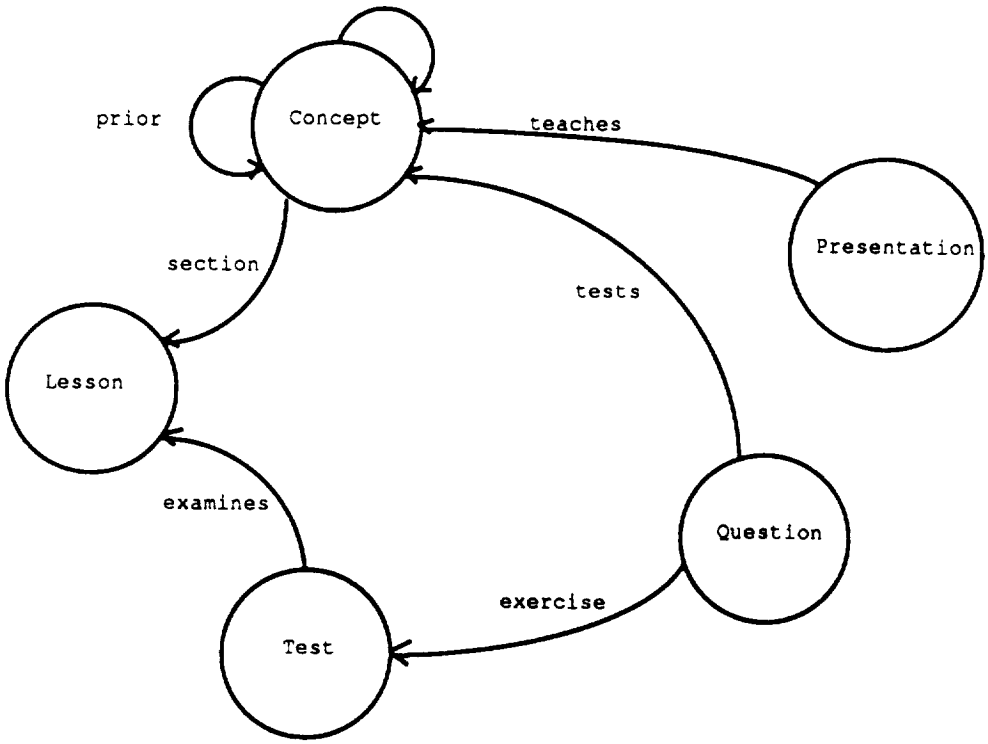

Figure 1: The Concept Network: Node and Link Types

3.2.1 Node Types

We have defined five basic node types: *concept*, *question*, *presentation*, *test*, and *lesson*. (See Figure 1 for a visual representation of the concept network.) Each node, in addition to having a name, may also have attributes associated with it to store information such as the name of a hypermedia identifier associated with the node.

Concept - This is the most commonly used node type in the concept network. A concept is a unit of knowledge or a skill; it can be decomposed into other concepts using the *subconcept* link. If a concept cannot be further decomposed (i.e. it has no subconcepts), then it is generally called a *primitive concept* and it is a self-contained unit of knowledge that must be learned by a student. Clearly, a primitive concept is a terminal (or leaf) node in the hierarchy formed by the subconcept links. Likewise, concepts that have subconcepts are internal nodes in the concept network. A non-primitive concept can be used in two different ways: (a) it can integrate several different ideas represented as concepts, or (b) it can serve as a convenient way of organizing a set of concepts together for teaching and testing.

Question - The *question* node is directly related to the concept -- a question is formulated to test the student's knowledge of a concept. It is also associated with a hypermedia object that presents the question, thus, the question node serves to link

concepts with hypermedia objects. Since questions are linked to concepts, we have to carry over some of the assumptions we made about the extent of a concept (i.e. its decomposition). Questions that are associated with non-primitive concepts are assumed to test all of the subconcepts of that node. In other words, if a student answers a question associated with a non-primitive concept correctly, then it is assumed that the student knows everything about the primitive concepts that are descendents of the non-primitive concept. This is an important assumption, as we will show later in the discussion of how to analyze the student.

Presentation - A *presentation* node is similar to a question in that it serves as a link between a concept and a hypermedia object. Thus, when the instructional planner decides that a concept should be presented, it selects one of the presentations associated with the concept.

Lesson - A *lesson* is a set of concepts that must be learned. It is used to drive a lesson presentation -- or the production of a test to cover a lesson.

Test - A *test* is a set of questions that tests a lesson. A test definition node defines a way of generating a test -- the list can be composed in several different ways: a list of questions, a list of concepts, a list of lessons. Naturally, if the test is to be generated from a list of concepts or tasks, then the question selected for each concept will be from the list of questions associated with the concept.

3.2.2 Links

The nodes are connected into a network through the use of various types of links that we define below.

subconcept - The *subconcept* link denotes that one concept is a subconcept of another. In effect, the *subconcept* relation creates a hierarchy of concepts where the top level concepts are the most general and encompassing, while the terminal level concepts are the most specific and basic. This link is used a great deal in generating lesson presentation sequences as well as examinations of the material. A concept can be a subconcept of more than one concept, thus the hierarchy is not a true tree. On the other hand, we do not permit cycles in the network, making the subconcept hierarchy a directed acyclic graph.

prior - Like the subconcept link, the *prior* link is used solely among concepts. It denotes that one concept should be presented before another. Included in the reasons for using this link may be that one concept may be a prerequisite for another, or else it just makes good pedagogical sense to always present a particular concept before another. This lesson sequence generator does not require this linkage to be present, but it does take it into consideration when it is there.

teaches - Presentations are linked to concepts through the *teaches* link. This link means that a presentation, which is directly associated with a hypermedia object, contains information related to a concept. We place no limit on how many presentations can be linked to a particular concept, nor on how many concepts a presentation can teach.

tests - The *tests* link is analogous to the *teaches* link in that it links concept nodes to question nodes, which are individually associated with particular hypermedia objects. More than one question can be linked to a concept using this link. Likewise, a question can be linked to more than one concept.

examines - The *examines* link is the way we link tests with lessons. Thus, this link is a way of connecting a set of questions with a set of concepts.

exercise - Questions are linked to tests using the *exercise* link. A question can be associated with more than one test.

section - Concepts are linked to lessons via the *section* link. Note that it is not necessary to explicitly link every concept covered by a lesson to the lesson node using this link. By linking a high level concept to a lesson using *section*, all of the subconcepts of the linked node are implicitly included.

3.2.3 Functions

So far we have only described the concept network in terms of the meaning that is derived from its structure and its contents. These components are obviously important since they provide the means of semantically organizing the hypermedia to be used for instruction, but there is one more essential ingredient to the concept network that operationalizes its meaning: the *functions* that are used to transform the concept network into an actual lesson, practical exercise, or test.

The concept network nodes and links are stored declaratively as a set of assertions in a CLIPS fact base. The functions that process this collection of semantic information are implemented as CLIPS rule bases, thus, we use a form of logic programming to operationalize our concept network. To date we have implemented functions to: drive the presentation of hypermedia courseware, dynamically generate tests for a selected portion of the concept network, and evaluate the student's test results. Each of these functions are keyed to the semantic relations that exist among the concept network nodes -- much depends on the structure of the network when it comes to generating lesson sequences and tests, as will be discussed in the subsections below. In addition, functions are also influenced by the attributes of the nodes as they pertain to the student model that overlays the network. By sensitizing the presentation planner to the individual student, it can adapt the instruction to fit the needs of the individual rather than giving every student the same lesson.

3.2.3.1 The Presentation Planner - Given a concept network and a set of instructional goals (i.e. a lesson definition), the presentation planner uses chains of subconcepts to identify the set of concepts that must be taught in the lesson. The primitive concepts that must be *covered* by the lesson form a subset of the lesson concepts -- we emphasize the word *cover* because one of our basic assumptions about giving a presentation (or a test) is that in selecting hypermedia objects to present the concept, we must form a *covering set*, that is, a presentation sequence is acceptable if and only if it: (1) directly presents a primitive concept, or (2) it presents an ancestor of the primitive concept. Thus for any given primitive concept there is at least one chain of ancestors from which a presentation can be chosen that will cover it. For each primitive concept we randomly select a hypermedia presentation from the concept's ancestral chain to use for instruction, avoiding the choice of a hypermedia object that has already been chosen for another concept. Once the set of presentations has been chosen, it is necessary to order them into a sequence. The choice of a sequence is influenced by the *prior* link, which creates a partial order among individual presentations -- the rest are randomly ordered.

3.2.3.2 The Test Generator - The test generator works on many of the same assumptions as the courseware driver. Given a concept network and one or more nodes that serve as "roots" to sections of the concept network (which, you may recall, is a direct acyclic graph), the test generator produces a set of questions that will form a *covering set* for the concept subset indicated by these "roots". Note that in the instance where we want to generate a test that covers a particular lesson, we can work from our concept network definition of the *test* node to produce the test. In this case the test is linked to the lesson by the *examines* link, and the lesson has a set of concept "roots" associated with it by the *section* link.

3.2.3.3 The Student Analyzer - A student model is a representation of the current state of the student's knowledge [4]. It is used for the purpose of identifying missing knowledge as well as for diagnosing misconceptions. In the current version of our system, we concentrate on identifying missing knowledge rather than diagnosing misconceptions. Consequently, our approach to student modeling is to use the overlay method with differential analysis [5]. We assume that the concept network represents everything we want the student to know about a topic, hence it is the target model. The student's knowledge is evaluated using tests generated from the target model, and the results are overlaid on the target model. Thus, the function of the student analyzer is to take the test results and generate a model of the student. It isn't quite as easy as just marking the nodes in a concept network as "known" or "unknown" since the test does not explicitly test every concept. In the cases where a concept has not been explicitly tested, we have to infer whether the student knows the concept based on the actual results. We make these inferences by inferring whether each of the primitive concepts is known and then propagating these values backwards from the subconcepts to their superconcepts.

4. An Architecture for Hypermedia-based Instruction

In the last section we defined the structure of the knowledge used to make decisions about how to sequence a lesson, how to generate a test, and how to analyze a student. In this section we will take a step back and look at how the individual components of the knowledge-based system interact with each other and with the hypermedia environment. We will bring the pieces together to show the overall architecture of a hypermedia-based instruction system.

4.1 Preliminaries to Instruction: Analysis and Development

In order to conduct a lesson there are several prerequisites that we are assuming are in place. First, we assume that the concept network for the lesson has already been built by a course developer. Though not described in this paper, we have built a tool to assist in the construction of the network -- it includes a network editor and display capability, and it produces as output the appropriate CLIPS assertions that can be interpreted by the CLIPS rules. Note that this step may be one of the most critical since it involves a careful analysis of the structure of the knowledge to be transferred to the student during the lesson. Second, we assume that once the concept network has been built that the course developer will create a variety of presentations for the concepts in the lesson. If the concepts can be represented in a variety of alternative media forms, then the greater number of choices available to the presentation planner make for a more robust instructional environment. Third, along with the presentations being developed for the concept network, a wide variety of test questions must also be written. It is desirable for each concept to have multiple questions from which to choose when generating a test.

4.2 The Role of Knowledge in Exercising Control

So how does it all fit together? This is an important question since the issue is how much freedom should be given to the student in the presentation of the lesson. At one extreme the presentation sequence being sent to the hypermedia environment (refer to Figure 2) could be equivalent to an entire lesson, and at the other extreme it might contain the identifier for one presentation of one concept. In either case, the question of what the Hypermedia environment should do with the presentation sequence is open. One approach would be to give the student "free play" within the bounds specified by the presentation sequence, with the constraint that partial orderings among presentations be observed (i.e. don't allow free play in a region until the predecessors have been successfully traversed.) Another approach would be to present the information in strict accordance with the presentation sequence -- this approach may be appropriate in some cases, but it obviously takes away more freedom for exploration than the first approach.

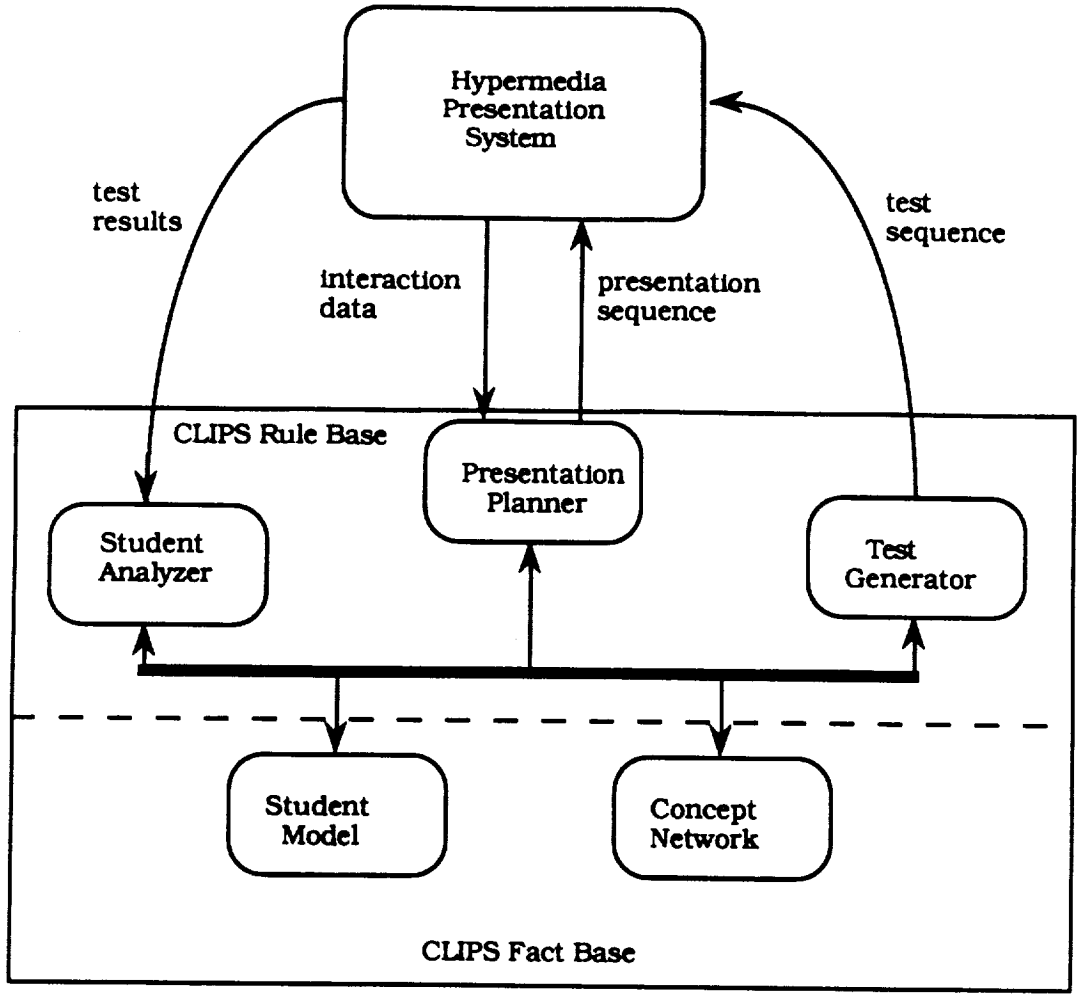

Figure 2: An Architecture for Hypermedia-based Instruction

However one decides to implement the frequency of how often control is passed back and forth between the hypermedia environment and the knowledge-based system, the general flow of control is envisioned as being a cycle:

1) generate a presentation sequence and send it to the hypermedia environment,

2) the hypermedia environment presents the lesson segment and sends user interactions back to the presentation planner,

3) the presentation planner either repeats 1, or it passes control to the test generator,

4) the test generator creates a test to cover the lesson segment just presented and sends it to the hypermedia environment,

5) the hypermedia gives the test specified by the test generator and passes the results back to the student analyzer,

6) the student analyzer takes the test results and updates the student model; it then passes control back to the presentation driver,

7) repeat the cycle until the lesson is complete or the student quits.

4.3 Current Status of Our Work

We have implemented the architecture just described using HyperCLIPS. Due to the modularity of the knowledge-based system we have implemented in CLIPS, we are

confident that we could use our current knowledge-based system as a driver in other hypermedia environments. In fact, our HyperCLIPS implementation serves as a prototype and debugging tool so that we can port our efforts to a hypermedia-based instruction system called Tools for Courseware Development, which is being developed elsewhere.

5. Conclusions

In this paper we have discussed an approach to integrating hypermedia with knowledge-based systems for the purpose of conducting instruction. Hypermedia is an attractive way to give information-seekers a way of navigating through a database. It provides freedom and variety, both of which seem to be desirable traits for an information processing system. The problem with hypermedia in general is the ease of taking the freedom to navigate too far -- the user gets lost, off track, distracted, etc. These problems can be even more severe in a training system where the intent is for the user to learn some structured knowledge, be it a task or a related set of concepts. On the other hand, many computer-based instruction systems have been notoriously inflexible and boring, and could benefit from the flexibility and the variety of ways of presenting information found in hypermedia. Thus, we have suggested that by using a knowledge-based system to make decisions and exercise control, one can harness the power of the hypermedia environment without destroying its usefulness.

6. Bibliography

[1] Hill, Randall W., Jr., and William B. Pickering, "Intelligent Tutoring Using HyperCLIPS", CLIPS User Conference, 1990, Houston, Texas.

[2] Pickering, William B. and Randall W. Hill, Jr., "HyperCLIPS: A HyperCard Interface to CLIPS", CLIPS User Conference, 1990, Houston, Texas.

[3] Wenger, Etienne, "Artificial Intelligence and Tutoring Systems", Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[4] VanLehn, Kurt. "Student Modeling". In Martha C. Polson and J. Jeffrey Richardson (Eds.), Foundations of Intelligent Tutoring Systems (1988): 55-78. Hillsdale, New Jersey: Lawrence Erlbaum Associates Publishers.

[5] Wilkins, David C., William J. Clancey and Bruce G. Buchanan. "Using and Evaluating Differential Modeling in Intelligent Tutoring and Apprentice Learning Systems". In Joseph Psotka, L. Dan Massey and Sharon A. Mutter, Intelligent Tutoring Systems: Lessons Learned (1988):257-275. Hillsdale, New Jersey: Lawrence Erlbaum Associates Publishers.