Web-based Intelligent Tutors Derived from Lecture-based Courses

A Dissertation Proposal Presented
by
Mia K. Stern
December 5, 1997

Abstract

In this proposal, we present the work on an intelligent tutoring system for the World Wide Web. In this work, we will focus on the research issues of domain representation, student modeling, reducing network delays, and code reusability. The framework for this research is MANIC (Multimedia Asynchronous Networked Individualized Courseware).

MANIC courses originate from existing video-taped courses. These courses provide an initial set of material, including audio, video, and class notes. However, the course structure is initially linear, which is not suitable for a WWW presentation. Therefore, we will investigate presenting the content non-linearly. Furthermore, the original class material is directed towards the entire class, and does not take into account the learning differences in individual learners. As a result, we must devise a *student model* that tracks each learner and adjust the course for that learner's needs.

We plan on using these student models to extend the currently existing course material in three ways: (1) guiding students through the material on their optimal path, (2) altering the content of the course material students see, based on their student models, and (3) generating quizzes most appropriate for the students' levels of ability and learning. Additionally, we propose to investigate how these student models can be used to *prefetch* parts of the course material, reducing the delays seen by students.

Furthermore, creating intelligent tutoring systems is a costly endeavor, both in time and in money. We would prefer to develop shells in which more tutors could be build. Therefore, in creating one intelligent tutoring system for the WWW, we hope to develop general techniques that can be used for other courses.

Contents

List of Figures

List of Tables

1 Introduction

The purpose of this thesis is to advance the theory, development, deployment, and assessment of Web-based intelligent instructional systems technology. We focus on the research issues in computer networks and intelligent user modeling. The computer networks issues include reducing network delays and monitoring network behavior. The intelligent user modeling research questions include knowledge representation, dynamic construction of a student model, and using this model to dynamically generate course material. We are also interested in exploring the possible reusability of such an intelligent tutoring system framework. This work will be presented in the context of MANIC (Multimedia Asynchronous Networked Individualized Courseware), a shell for Web-based intelligent tutors, whose course content originates from existing video-taped courses [47][48].

Many Web based educational systems do not customize the material to meet individual student needs. These systems are usually static and do not reason about the student to personalize the interaction. On the other hand, MANIC, since it is an intelligent tutoring system delivered over the Web, can customize the learning experience for each student. MANIC synchronizes audio/video and HTML slides as well as interactive quizzes. With the MANIC project, existing technologies (such as WWW browsers and plugins, like RealPlayer) are augmented with specialized WWW servers and proxies to provide a more individualized learning experience.

Although the audio/video and slides for a MANIC course are taken from existing video courses, MANIC courses are not simply direct translations of those video courses. By designing the course to be delivered over the WWW, it is considerably more interactive, thus allowing students to take more control over their learning.

The domain for the first course developed with MANIC is UNIX network programming. The content for this course was taken from an existing six hour course taught by Jim Kurose through the National Technical University. This course was used as part of a 1-credit course taught during the Fall of 1996. The second MANIC based course was based on Computer Science 453/653: Computer Networks. This course was taught in the Spring of 1997, with 150 registered students. We are currently converting three additional courses to be used with MANIC, including a course on art history.

Three main research topics will be explored in this thesis. The first involves building an intelligent multimedia tutor on the World Wide Web. We are restricting the tutoring system to be courses composed of HTML slides, audio, video, and animations.

We are concentrating on these aspects since the goal of the MANIC project is to be able to "replace" traditional lectures by putting them on-line. One advantage of using MANIC for learning the lecture material over the traditional lecture experience is the availability of an on-line course. Students can access the material at any time and from anywhere. Another advantage is the student can study at his own pace, choosing to review material multiple times if necessary, without altering the pace for the rest of the students. Furthermore, the instruction can be individualized for each student, using MANIC, which it cannot be in a full lecture course. However, lectures themselves have advantages. The primary one is the ability to ask questions of the instructor. But as the goal of MANIC is to replace traditional lectures, and not to replace the instructor, students will still be able to meet with a human instructor during regularly scheduled lecture times to ask those questions. The lecture time would be available for more constructive learning.

Essentially any existing video courses can be converted with the MANIC system, in which case the domain of the course begins completely linear in nature and the course content is directed towards all the in-class students, who may learn at different speeds. We intend to make the learning experience more enjoyable for students, by adjusting the domain which should not be linear, and providing multiple ways for presenting the material. There are four questions that we will explore while developing intelligent tutoring in the MANIC context:

- How should the domain be organized?
- What kind of student model can we have in these systems?
- How can the student model be used to affect the way material is presented to the student?
- How can the student model be used for testing the student?

The second part of the project involves reducing the network delays experienced by students. Because we are using various forms of multimedia, there will be some difficulties providing the highest quality of service to all users. As a partial solution to this problem, we will investigate a mechanism in which part of the material can be *prefetched* to the client's site before it is explicitly requested. If this prefetching is accurate, the delays seen by the user will be considerably smaller, if not non-existent. Within the MANIC system, we will investigate how to make the prefetching accurate by using the student models to best predict what to prefetch.

The last part of the project involves demonstrating the reusability of the MANIC software. Since creating instructional material has proved to be so difficult and time consuming, one of the goals of this project is to develop general student modeling techniques that can be used with any video-taped course being converted to the Web.

1.1 Motivation

Web-based training systems provide a good mechanism for distance learning but the technology is currently poorly understood. The Web makes interactivity difficult and provides a limited view of the student's activity. It is not possible to be aware of every action the student takes. For example, in a standalone ITS, if the tutor wants to track all answers the student gives, it can do that by responding as soon as an answer is given, before even an "ok" button is clicked. However, on the Web, the student must submit his answer before it can be evaluted by the tutor.

Other obstacles of a Web-based system include the latency experienced on the Web, the lack of customization of material for each user, and the inability to monitor student progress and reason about responses. In this section, we discuss some of the benefits of Web-based teaching systems, as well as some of their problems.

1.1.1 Benefits offered by a networked system

Distance learning is becoming more and more popular and prevalent and delivering learning systems over a computer network is more practical than distributing the software to all sites via CD-ROM. Specifically, each learning site will not be required to have copies of the system for on-demand usage and will not require users to wait for the system to be delivered. Network-based systems are available at any time and at any place around the globe.

Because educational material can change rather frequently, using CD-ROMs for instruction is not cost-effective, since it requires redistributing the CD-ROMs every time the material changes. With a network-based system, instructional designers can continuously upgrade and augment the material without any users being explicitly made aware of such changes. Therefore, networked educational systems allow for more flexibility and extensibility than traditional, static CD-ROMs.

1.1.2 Obstacles of a Web-based system

However, there are some difficulties with Web-based delivery of course content. First, Web sites may be unavailable, either temporarily or permanently. Second, multimedia can be delivered with a greater quality of service (i.e. greater bandwidth) using a CD-ROM than using the Web. But even with these disadvantages, the flexibility of Web-based instruction makes it a worthwhile endeavor.

Furthermore, many users claim, when using the Web, that information retrieval is too slow. One way to reduce these latencies is to prefetch, or download to the client ahead of time, material to the user. If the prefetching is accurate, the delays at the client side will be minimal.

However, most prefetching work, such as that in [38], involves using the global Web as the test bed. On the other hand, the MANIC system is a closed environment, providing us a limited number of documents to prefetch. We can therefore consider reducing network latencies in a controlled environment, before approaching the problem of the global Web community.

Furthermore, this project provides a way to examine the issues in networked multimedia (e.g. quality of service guarantees and admission control) in a setting in which users will provide actual data on how the algorithms should function. Many software studies on these network issues are based on simulation; our system (student model and prefetching schemes) can be based on data from actual users. We can examine the way students use the material, and design a student model from that data.

1.1.3 Benefits offered by an intelligent tutoring system

It is becoming very evident that more efficient training systems are needed by all organizations undergoing rapid technological change. Lecture-style training and traditional instructional systems are unable to keep up with the number of people who require training. New technologies are needed to reduce the increasing cost and burden of education and training.

On the other hand, effective multimedia intelligent systems often include substantial multimedia components, making these systems memory and computer intensive. Because educational computational resources are typically limited, cross-platform delivery is essential. Thus we again look to a Web-based solution.

Properly designed computer-based tutoring systems have proven highly effective as learning aides. Intelligent tutoring systems have been shown to teach twice as quickly as traditional classroom methods [45] and to produce increased skill retention with fewer mistakes [29].

An intelligent tutoring system consists of five components: student model, pedagogical module, domain knowledge, communications module, and expert model [2]. In MANIC, we are concentrating on how student modeling can be done most effectively on the Web, and how this student model can then be used to affect the pedagogical decisions.

Computer-based training systems are extremely effective in training because they can be adapted to an individual student's needs and idiosyncrasies, thus increasing motivation and significantly improving learning by tracking student abilities and altering instruction accordingly. Individualized instruction has proven extremely useful in improving the education of students [6]. Intelligent tutoring systems can provide this individualized instruction that cannot be achieved in a lecture-style class.

Intelligent tutoring systems have been used to teach a variety of skills, from Lisp programming [1] to treating heart attack patients [20]. For training people in technical fields, educational systems have been developed that provide realistic working environments [28]. These systems include simulations of complex and dangerous machinery, thus providing opportunities to learn that otherwise would not be available.

Although intelligent tutoring systems have proven highly effective for teaching, there are very few systems available over the WWW. The lack of instantaneous interaction between the tutor and the student has been a hindrance to the development of truly intelligent Web-based tutors.

1.2 Goals and Contributions

The research goals for this work fall into five main categories. In this section, we discuss the research focus of the MANIC system and the unique contributions of this work.

1.2.1 Network latency

Many users on the World Wide Web complain of latencies experienced due to downloading a new page [38]. One way to reduce these delays is to prefetch the new page before it is explicitly requested. In a completely linear setting, this would not be a problem, since the next page is known. However, with a MANIC course, as with most Web settings, students are not limited to a linear traversal of the material.

One option is to prefetch large sections of the course, or even the entire course, ahead of time. This prefetched material would be stored on the client's machine in a temporary cache for easy access. However, in reality, we cannot do this. There is never infinite cache space nor network bandwidth. Thus we cannot prefetch the entire course, nor can we always prefetch more data than we expect the student will use. For these reasons, we need to selectively and intelligently prefetch only small portions of the course.

To accomplish this task, we use the student model to predict what actions the student will take and prefetch that part of the material. This prefetching, if accurate, can reduce the delays seen by the user, since the requested information will be available before it is requested. However, prefetching that is not accurate will waste network bandwidth and cache space, by storing pages not requested by the user. One contribution of this work, then, is to identify what can and should be prefetched, based on the student model.

Another contribution is to decide when not to prefetch. Inaccurate prefetching will reduce the amount of bandwidth and processing available for real-time requests. Real-time requests should have priority over inaccurate prefetching. We will investigate how to use both the student model and empirical data collected by the tutor to decide when and how to slow and/or terminate the prefetching.

A further difficulty with MANIC prefetching is the dynamic course content generation. Since there are no static slides for users to see, it is not possible to prefetch far into the future.

1.2.2 Dynamic student modeling

One of the research goals of this work is to determine the type of student model that can be employed in the MANIC system. Student modeling is a difficult subject, in standalone tutors as well as web-based tutors. However, in order to have an *intelligent* tutoring system, the system must maintain a student model.

In MANIC, we model all actions the students take, even if they use the history list or the browser's navigation buttons. We are able to do this since all of our web pages are common gateway interface (CGI) scripts, which must be reloaded each time. Therefore, the HTTP server is contacted whenever the student loads a page. In section 4.3, we discuss exactly how the actions can be tracked.

In intelligent tutoring systems, quizzes provide the tutor with the most useful, direct information about the student's knowledge. However, in MANIC, the quizzes are optional, since some people may use a MANIC course without being registered. The intelligent features should still be available in this case, even when quizzes are not required. However, quizzing provides the tutor with a lot of information about the student's knowledge, so it would be beneficial, from a user modeling standpoint, for students to take quizzes. Quizzes generated by the tutor allow it to have a degree of expectation for student's answers. The questions are directed to test knowledge about which the tutor is lacking sufficient information.

Since quizzes in MANIC are optional, we cannot base our student model on being able to collect these data. The main interactions with students are through their viewing slides and listening to audio. Thus, we must devise a student model that uses this information as its basis for judgment.

This is very tricky, though. It is not always possible to accurately judge a student's knowledge based on the slides he has seen and the audio he has heard (as has been discussed in [10]). If a student views slides 15 through 20, does he sufficiently know the material? What if he starts on slide 15 and skips the first 14 slides? Should the tutor suggest he go back to that initial material? Also, if a student does not listen to the associated audio, does he know the material as well as someone who has (we are assuming "no")?

For all of these reasons, designing a student model to use in the MANIC framework is an open research question that we address in this work. We will then explore how to use the model for individualizing instruction and for reducing network delays.

1.2.3 Overcoming Web-imposed difficulties

As we have mentioned, the Web reduces the level of interactivity in an intelligent tutoring system. In MANIC, we are examining how to overcome some of the difficulties imposed by this restriction.

The first problem we must face is being able to record the student actions that make up the student model. This is problematic on the WWW, since the HTTP protocol [4] is stateless, which means there is no long term connection between the client and the server. Every time the client makes a request of the server, a new connection must be established. Thus the server cannot maintain information about the client, since the communication, from its perspective, is one time only.

Many solutions to the statelessness problem have been proposed and implemented, so that is not the focus of this discussion. Our solution, using cookies [24], is presented in section 4.3. We have chosen cookies because they are easy to implement and are a very simple mechanism. However, cookies have one severe drawback. A student can choose not to accept the cookies by turning on an option within his Web browser. With this option on, the student can view the cookie, and elect not to accept it. If this occurs, there is no way for MANIC to track students' actions.

Another difficulty with web-based student modeling is students do not have to use the software as intended. In MANIC, if a student wants to jump to another part of the material, we expect and want him to use the provided index. However, some students do not do this. Rather, they simply change the slide number in the URL [1]. By changing the slide number, the page displayed may not be correct, since the URL also contains information about which buttons should be active (e.g. whether the "next" button is enabled). Only if the index or navigational buttons are used can the slide be accurately displayed.

Furthermore, even though we have provided navigational buttons, students do not have to use them. Web browsers have their own "back" buttons and history lists that students can use to traverse the material. If students do not use our buttons, the audio playback does not stop (it does when the MANIC buttons are used). Since the audio is still playing, the slide will change when a synchronization point is reached. However, the student may not want the slide to change.

[1]Each slide in the initial version of MANIC had a number, corresponding to where it fit within the lecture course.

Thus using the web browser's navigation buttons can lead to unexpected behavior from the user's perspective.

Another problem faced in the MANIC system is the use of audio. We have provided a "stop" button to terminate the play back of the audio. When students press this button, the tutor knows how long the audio has been playing, which it adds to the student model. However, many students use the RealAudio controls to stop the audio. There is no way to inform the tutor when the audio stops if this method is used. However, this problem is being fixed in the next version of MANIC.

Therefore, when designing web intelligent tutoring systems, it is not possible to assume that students will use the software as it was designed. They will always find a way around the interface [].

1.2.4 Knowledge representation

The goal of this work is to be able to convert existing courses to be standalone courses on the WWW. The courses that we are converting start off completely linear, since they are derived from linear lectures. However, if MANIC is used as the primary method of providing instruction, and lectures are not necessary, there is no reason to adhere to the linear nature of the existing course. One of the contributions of this work is being able to begin with lectures, but represent the domain in such a way as to provide a non-linear traversal of initially linear material. We therefore hope to develop a methodology for converting lecture based courses to be on-line, making the process easier for other instructors who would like to deploy an intelligent MANIC course.

1.2.5 Adaptive and generative course material

Another contribution of this work is using the student model to individualize the course material for each student. In lecture-based courses, there are many students, each with a different level of ability and knowledge. The instructor cannot individualize the teaching for each student in the class. Therefore, the instruction may be too slow for some students and too fast for others. This is illustrated explicitly in the course that we are working with, since both graduate and undergraduate students are enrolled.

We are developing alternate ways of presenting slides and exams. We must now use the student model to choose the correct material. This involves changing the course content (the slide material and the progression through the topics) as well as providing quizzes appropriate for each student.

The content of the slides the student sees should be detailed enough to provide all the important information, but not be too detailed to provide unnecessary information. We must therefore develop methods in which to decide how slide content can be constructed accurately.

Furthermore, since the domain of a MANIC course is not linear, we must develop techniques to help guide students through the course material in an effective way, to prevent the "lost in hyperspace" problem [19]. We use the student model and a semantic network of course content to help with this process. The student is not forced to learn the topics the tutor chooses. Rather the student is presented with suggestions from the tutor, and he can choose to either take those suggestions or to ignore them.

Finally, students are given the opportunity to take quizzes to test their skills. These quizzes should test the correct topics and be at the correct level of difficulty. The quizzes are constructed dynamically, based on the student model, which considers both of these factors.

1.2.6 Code reusability

Developing intelligent tutoring systems is a long and costly process. Therefore, one of the goals of this work is to develop techniques that do not apply specifically to one course, but to many domains. Any domain in which lectures are a main form of instruction can be taught using MANIC. The goal, though, is not to develop general instructional methods. That problem is too big, and will not be addressed in this work.

However, an approachable goal is to convert any existing video-taped course to be delivered via the Web. The domain organization and student modeling is not specific to any particular domain. The techniques that we will develop can be applied both to computer networks and art history. The only requirement for using MANIC to develop on-line intelligent courseware is that the material must include lecturing with slides and audio/video as the basic method for teaching.

2 Related work

In this section, we discuss research that is closely related to the work presented in this proposal. In particular, we describe other WWW educational systems and other caching and prefetching systems.

2.1 Distance learning systems

Distance learning is quickly becoming a very popular method of teaching and learning. These systems come in two varieties: synchronous and asynchronous. With synchronous systems, all participants must use the system simultaneously, while with asynchronous systems, this requirement is lifted [33]. As a consequence, synchronous systems allow for more interactivity amongst the users while asynchronous systems allow for a more self-paced approach to learning.

Many synchronous systems revolve around a video conferencing metaphor. These systems are similar to traditional lecture courses, with the exception that the participants are geographically dispersed. Clearly the limitations for such systems are the current video conferencing technology and the lecture nature of the interactions. Some examples of these systems can be found in [23] and [40].

Other synchronous distance learning systems allow users to work collaboratively, which allows for much more student interaction. One such system, Belvedere [50], allows students to construct scientific arguments collaboratively. Another system, The Electronic Classroom [21], provides students with a shared screen on which they can "paint". The connections are maintained using standard telephone lines and modems. The Telemethea system [7] allows a tutor and a trainee to have audio communication over the network as well as to jointly edit text documents and still images.

Asynchronous systems, as we have noted, do not require that all students participate concurrently. Some of these systems, such as that described in [41], are designed to be repositories of course material and are strictly text based. The material in the system in [41] is used as a supplement to an already existing lecture course. Other systems, such as that in [43], allow for asynchronous collaboration. This differs from the synchronous collaboration work we have just discussed in that, while students work collaboratively, the participants do not have a prearranged work time.

The MANIC system is an asynchronous system, so students can access the material at their convenience. However, adding synchronous support, such as discussion sessions, would provide more support for the student's learning experience.

2.2 WWW tutors

Very few tutors have been deployed using the WWW. Most of those that do exist have not been developed or deployed completely. In this section, we discuss these systems, as well as indicate their similarities to our system.

Most web-based educational systems use quizzes implemented with HTML forms and common gateway interface (CGI) programs to determine a student's knowledge. The results of these forms are then used to update the student model. This is similar to traditional intelligent tutoring systems in that they are question/answer based systems.

One such system is designed to teach users about the road regulations in Quebec [36]. This system uses a curriculum component, a planner, and a tutor to present material to the student. Automatically generated tests are used to capture the student's abilities and the results of these tests are used to construct the student model. The system dynamically decides what pedagogical resource (problems, critiques, simulations) should be presented to the student. To determine the actions the user has taken, the History list from Netscape is perused.

Another WWW-ITS is ELM-ART [14][52]. This system teaches Lisp programming by providing an intelligent interactive integrated textbook. The examples given to the student are those that are most relevant during problem solving. Furthermore, when a page is displayed that includes a problem, links to the course material are included that correspond with that problem. The tutor also suggests the material most appropriate for the student. But if the student attempts material for which he is not quite ready, the tutor will provide prerequisite links for him to explore, if he so chooses. However, the system in only text-based and the student model is rather shallow (an overlay model [15]).

ELM-ART dynamically generates all the HTML pages based on the student model and the stored domain knowledge. The tutor provides feedback when a student is trying to solve a problem, and adapts the curriculum to the student's abilities. However, all of the data is still gathered through questions and answers, which is only a small part of how MANIC gathers data.

When presenting links to the student, indicating pages in the material to examine, ELM-ART "suggests" those pages for which a student is ready. In this way, the student can either take the suggestions of the tutor or explore on his own direction. We would like to incorporate a similar mechanism in the MANIC system. We feel it is not advisable to strictly direct students through the course. Students should be given the choice of what material on which they would like to concentrate. We have decided to take this approach since some students may use this material as a reference guide. If this is the case, the tutor cannot know for certain the state of the student's knowledge. If the tutor then restricts what the student sees, based on the highly faulty student model, the student will be frustrated and will be forced to see material inappropriate for him. ELM-ART successfully allows students to progress through the material as they choose, but provides sufficient suggestions so they have good guidance.

A more general framework for adaptive electronic textbooks is that of InterBook [12][13]. This system allows for incremental interface learning by adding features to the interface over time. The student model in this system is an overlay model, giving scores to concepts if a student has read a page and evaluating how the student has performed on quizzes concerning those concepts. This is similar to the student model we are using in MANIC. However, we do not assume, as is done in InterBook, that if a student reads a page he understands the associated material. Rather, we evaluate how a student progresses through a topic, and use this information to judge a student's knowledge. The InterBook system uses both adaptive navigation support [26] and adaptive presentation [9] techniques, as we are in MANIC. The adaptive navigation support is provided by indicating to the student which concepts in the hyperbook should be explored and in what order.

The adaptive presentation is accomplished by incrementally adding interface features (such as a glossary button) to the hyperbook layout whenever the student is ready.

Lin, Danielson, and Hergott [31] have designed an adaptive instructional system for the WWW to teach introductory programming in C++. This system also uses true/false, multiple-choice, matching, and short-answer questions and answers to gain information about the user. Simple essays and procedures to be performed and evaluated are also used. The tutor includes four modules, all of which are on the HTTP server: expert, instructor, student, and interface. The system keeps track of the topics the student has seen, the knowledge of the student, the inferences about misconceptions, and the inferences about possible plans the student has used to solve problems. Currently there is no way to adopt different materials and methods to deliver a question/answer form, but that is the main focus of their research.

None of these systems that we have described use any form of multimedia. They are simply text-based intelligent tutoring systems that have been converted to work via the WWW. Our system, however, is being converted from a lecture course, which implies the use of video and audio. Also, the main form of interaction in these other systems is through test questions, which provides the system direct information on how to update the student model. Since this is not the primary way of gathering information in the MANIC system, we cannot use the same student modeling and curriculum planning techniques that these tutors utilize.

One WWW tutor, CALAT [34][35] does include a considerable amount of multimedia. This tutor includes interactive simulations, exercises, and explanations. The architecture is similar to that of MANIC since there is one intelligent tutoring server for each student taking the course. However, it is also somewhat different in that the CALAT system includes its own client for the interactive simulations. By doing so, the user must simultaneously view multiple windows, which can become confusing. In MANIC, we have, so far, avoided any client side development. Additionally, research in CALAT is exploring how to reduce the download times experienced by users. The solution in CALAT is to *pipeline* the data transfer and the playback. Caching is also used in a proxy to reduce delays if a user revists a page.

2.3 Web-based user modeling

Not all user modeling on the web is designed for tutoring systems. For example, the system described in [46] is designed to tailor web pages to users' needs. This is accomplished using new hypertext markup language (HTML) tags that can be parsed and ignored by standard web browsers. However, an "agent" can parse these tags, and based on the user's model, dynamically construct the content of the page.

Electronic shopping is another area in which user modeling techniques have been attempted. In [25], an initial design of a prototype on-line shopping mall is discussed. The goal of the system is to provide the same kind of in-person shopping experience, but while using an on-line catalog. To this end, the system adapts the presentation of products as well as the navigation through the catalog.

2.4 Caching

Traditional caching (e.g. for cache memory accesses) involves storing data a user has just seen in the hopes that the data will be needed again in the near future. Then if the data is needed, it can be retrieved from the cache, thus reducing the latency and delays seen by the user. However, if the data is not in the cache, it must be fetched and stored, causing something currently in the cache to be evicted. Some typical cache replacement policies are least recently used (LRU), first in first out (FIFO), and least frequently used (LFU) [51].

Caching on the WWW is similar to traditional caching in that the cache is used to store data that has recently been seen by the user. When a client makes a request of the server, the cache is checked first to see if the document is already available on the client side. If so, then the document can be displayed without contacting the server, thus reducing the load on both the network and the server. WWW browsers themselves have caches, but it is also possible to add additional caches by using *proxies* [32]. A proxy resides between a WWW client and a WWW server. The proxy forwards requests from the client to the server, and returns the response to the client. By using this type of architecture, the proxy can also cache documents that are requested by clients.

The WWW offers us additional ways to study and implement caching. Most caching techniques examine a single user's access patterns, such as those in operating systems. However, with so many people accessing the same URLs, we can study how entire populations access the data and then we can determine caching techniques for multiple users. In the usual case, WWW proxies are shared by multiple users, and thus the cache can store documents for an entire population [32].

Caching on the WWW has been shown to be very effective. Some initial work by Glassman [22] has shown that 30-50% of all requests can be serviced from the cache, and the time to service a cache hit takes 1.5 seconds while to service a cache miss requires 6 to 9 seconds. Further work by Pitkow and Recker [39] has shown that an LRU cache replacement policy on the WWW is very effective for providing users with cached material with a hit rate of (at best) 67%.

The Harvest system [8] includes a hierarchical object cache. In this system, multiple caches are used to store data. Cache misses are resolved by contacting neighbor and parent caches, as well as the home machine of the requested object. Whichever site returns the fastest *hit* is the one from which the object is retrieved. If all the caches miss, and the home sight is slow to respond, the object is retrieved through the fastest parent to respond. Unlike the work in [38], these caches are shared by all users, rather than having one cache for each client. This work differs from the cache we have implemented in MANIC since it is hierarchical. In MANIC, we have only one single proxy and cache to handle all data requests.

This caching work we have just described differs from the work we have been doing in MANIC in that it does not use any intelligent techniques for adding and removing items from the cache.

2.5 Prefetching

Prefetching is clearly not a new idea in computer science. Operating systems and compilers have been using prefetching to speed up execution for years [16]. Prefetching in both file systems and in the WWW are extensions of this work.

Lei and Duchamp have explored file prefetching as a way to improve access times. The work done in [30] indicates that by analyzing the file access patterns of many users, it is possible to fairly accurately predict the next file to be accessed. In this work, the prefetcher remembers file access patterns, and tries to match the current pattern to a stored one. If there is a match, prefetching the next file is trivial. If the access pattern does not match with previous patterns, a new access pattern is created to be used for future file accesses with which other users' patterns can be compared. This is similar to our work in MANIC because it is trying to analyze user's access patterns. However, we have "pre-stored" patterns that the system tries to match, while in [30], new patterns are created based on what previous users have done.

In [17], the authors explore whether prefetching in the context of the WWW is even a worthwhile endeavor. Their initial study indicates that prefetching all documents that the user will see during a session at the start of a session produces worse network characteristics than if no prefetching had been done. The reason for this result was that the requests were no longer bursty, as they all occurred at the beginning of the session. Thus the network was flooded with downloading documents all at once, instead of spread out over time. However, when the authors considered smoothing out the prefetching, i.e. fetching only one document at a time while also using rate controlled prefetching to further smooth the traffic, the performance with prefetching was considerably better than the performance without prefetching. Thus applications using prefetching can increase the network utilization and decrease delays.

In [38], the authors discuss a technique for prefetching material on the WWW as a way to improve latency. The prefetching in this paper is based on the fact that an HTTP server sees how an entire population of users view the material at the web site. From this information, the server can develop a population model and use this to predict what a new user will do in similar situations as those already observed. The server will only suggest that the client prefetch those documents that have a certain probability of being used in the near future. This probability is calculated from prior accesses of the documents from the current state. For example, if document B is always accessed after document A, then the probability that a user viewing A will view B is 1. The results of the study indicate that this prefetching technique does indeed reduce the latency in downloading a web document. When adding 20% more network traffic, the access times for retrieving a document were lower with prefetching than if the bandwidth had been increased by 20% with no prefetching. Currently, in the MANIC system, we are not analyzing general population data to help with our prefetching algorithm. However, this is an issue we will be researching in the future.

This work in [38] incorporates "intelligent" prefetching on the WWW. However, it is very similar to the work in [30] on file prefetching, in that general population patterns are explored to help the prefetching. In the future, we will also include population models to help with our prefetching. However, the work just described differs from our work in MANIC in that we are using *individual* user models to aid in the prefetching decisions. We are examining each users' patterns and using this to predict what they are likely to see next.

In [18], the authors explore individual user models to help determine which documents to prefetch. In order to accurately prefetch, user modeling is used to determine what documents the user will view next. Users are categorized at two extremes: a net surfer and a conservative accessor. Conservative users will access the same pages multiple times, while net surfers will explore the net without revisiting many documents. If users can be classified as conservative, it is possible to prefetch documents. The techniques in this paper are mostly concerned with determining whether a user will revisit a previously seen page, which is not the focus of the prefetching prediction in MANIC. Rather, we are concerned with where the student will go next when venturing into unchartered territory, as well as whether the student will visit previously seen pages.

There has been some work in prefetching educational material. One such system [44] is designed to support focused network exploration, or guided paths. Since the paths in this system are linear, the next page in the exploration can be downloaded before it is requested by the user. The architecture of this system employs a path server that acts between the WWW browser and the HTTP server. This path server allows for the precaching of path material.

This work is very similar to our work, currently. The material in the MANIC system, at this point, is fairly linear. However, in the future, we will be adding various paths that students can explore, making the courseware non-linear. Thus we will not be able to prefetch along a singular, predescribed path, since one will not exist.

Similar to prefetching is the work on server side speculation [5]. In this work, the server sends additional documents to the client whenever a request is received. The additional documents sent are those that have a sufficient probability of being viewed by the user. These probabilities have been determined a priori by examining user access logs for a period of time. Therefore, in order for this method to work, some learning period, where no speculation can be done, must exist. This is different than the work we are proposing as we are not intending to view population accesses to see how users use the system. Rather, we are concentrating on individual user models to predict the next documents to prefetch.

3 Student's interactions with a MANIC course

In this section, we describe how a student can currently interact with the MANIC system. We discuss the different modes in which the student can use the software as well as some of the additional features we have built into MANIC. Figure 1 illustrates the typical screen layout for the MANIC system.

Our goal is to encourage students to be more interactive in their learning. Therefore, we have divided the course into "topics" (see section 5.1 for a description of the domain organization). Students are encouraged, when learning the material for the first time, to explore the course in terms of these topics.

To help with this, we have provided a way in which students can view the domain structure (both topics and slides) of the course, which enables them to jump from topic to topic. For this reason, we have added a table of contents, which presents the list of topics to the student. Additionally, the course is indexed by keyword, allowing the student to jump to any slide containing the desired word.

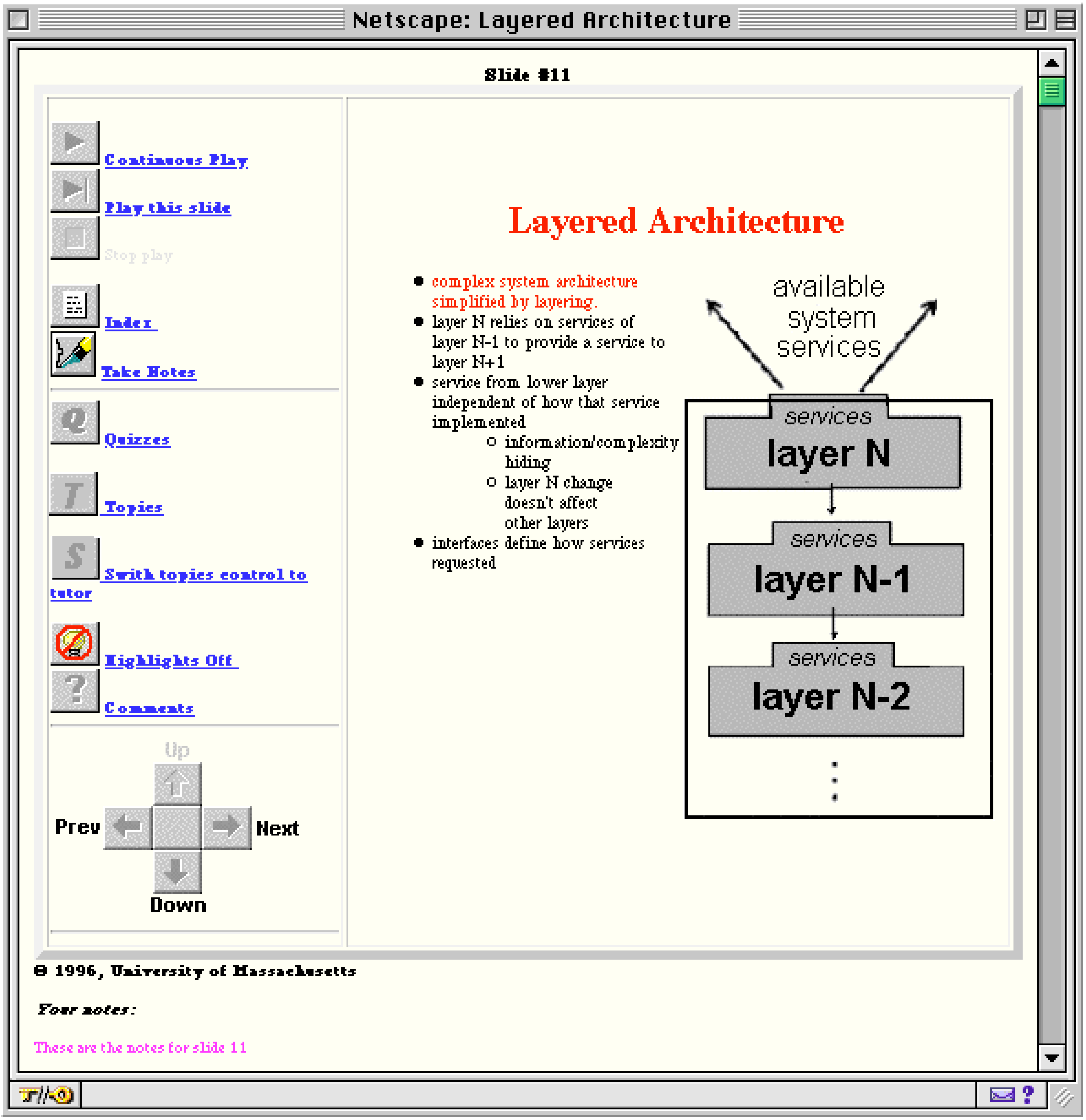
Students can listen to the audio associated with a given topic. If this option is chosen, the audio for the topic is synchronized with the slides[2]. This means that as the audio plays, the correct corresponding slide is displayed to the user. Optionally, students can browse through the slides at their own pace. This browsing is aided by the addition of "previous" and "next" buttons, which point to the previous and next slides within the topic. Since some students may not want to hear all of the audio for a topic, we have added a feature by which they can also listen to the audio for a single slide at a time.

When the end of the topic is reached, the student should start a new topic. There are two options for selecting a new topic: have the student select it or have the tutor select it. The student chooses which option he wants. If he wants to select the topic, when the "next" button is clicked on the last slide of the topic, the list of topics is presented. If the tutor selects the next topic, when the "next" topic is clicked, the topic the tutor has chosen is automatically started. The student has the option of changing who controls the topic selection process at any time, as there is a button on each slide which, when clicked, makes the change.

When we first designed the on-line course, we noticed that simply having the audio and the slides was rather dull. Therefore, when the audio is playing, the corresponding part of the text on the slide is highlighted in red, bringing the user's attention to the relevant text. However, if the user does not want to see this highlighting, he has the option of disabling it. The slides themselves are still synchronized to the audio.

For ease of use, we have decided that all user controls should be contained within the browser window. The other option is to have the navigational controls in the browser window with the controls for audio playback in the RealAudio plugin window. One reason for this decision is the RealAudio controls do not provide a convenient way to index into the course material. For a student to move to another part of the material, he would have to use the timer control on the RealAudio client (i.e. go to time 3:47 of the playback). Our table of contents provides a much more convenient way to jump to other parts of the material. However, using the table of contents to move to another

[2]We use the term "slide" very loosely in this section. See section 5.1 for a more detailed discussion on what is meant by a "slide".



Netscape: Layered Architecture

Slide #11

Continuous Play

Play this slide

Stop play

Index

Take Notes

Quizzes

Topics

Swith topics control to tutor

Highlights Off

Comments

Up

Prev

Next

Down

Layered Architecture

- complex system architecture simplified by layering.
- layer N relies on services of layer N-1 to provide a service to layer N+1
- service from lower layer independent of how that service implemented
  - information/complexity hiding
  - layer N change doesn't affect other layers
- interfaces define how services requested

© 1996, University of Massachusetts

Your notes:

These are the notes for slide 11

Figure 1: A screen shot of MANIC course material.

part of the course does not allow students to move in a fine-grained manner. Students may want to fast forward through some of the audio without having to jump to the next bullet point or the next slide. However, currently, there is no way to have this kind of fine-grained audio control.

As we have noted, this course was converted from an already existing video course. Thus the material was initially very linear in nature. However, the WWW provides us with an easy way to make the presentation of educational material non-linear, and to not do so would not be taking advantages of the power of the web. Thus, we have added "hyperlinks" within the material. These links provide supplementary material concerning the information in the slides. These links can either be to material within the course, or outside URLs (such as Request for Comments on various network protocols).

We must note that when a student elects to view a hyperlink or chooses to use the table of contents, a separate browser window is spawned. The reason for this is that students, when viewing the supplementary material, should not lose the original context of what they were learning [49].

Students also have the opportunity to take on-line "quizzes", which are designed to test a student's knowledge on the course material. Currently, the quizzes have three kinds of questions:

- Multiple choice
- True or False
- Short answer

The answers to the short answer questions must be only a few words, since we do not have a natural language interface for grading the answer. Currently, we are simply doing a straight string comparison to grade the answer.

Whenever the student elects to take a quiz, one is generated dynamically to meet his needs and skill abilities (see section 5.4 for more information on how this is accomplished). The questions are designed to test the student on appropriate material. In the future, the questions will also be chosen to be at the right level of difficulty for the student, based on his model.

It should be noted that these quizzes are optional, i.e., the student does not have to take them in order to move on to other material in the course. We have already discussed the reasons for this policy. However, the benefits of having mandatory quizzes may outweigh the flexibility of not having them. We will explore whether it is possible to construct an accurate student model without the use of on-line quizzes.

An additional facility that we have provided for students is the ability to take notes on each page viewed. Whenever the student views the same page in the future, his notes will appear at the bottom of the screen. It should be noted that the notes are per page, and not per bullet point or graphic. Furthermore, each student can only view his own notes and notes the instructor makes available for the whole class.

Students have the option to either register for the course or not. If a student registers for the course, then all of his actions will be recorded. For example, each slide the user views, each time the table of contents is brought up, each time the audio is started or stopped, the action is recorded. Not only will this help in updating the student model, but it will also help in designing the MANIC system to be more efficient based on students' behavior when using the system.

Non-registered students will not have their actions tracked. As a result, none of the intelligent features in MANIC can be used for these students. These students will see the original slides, taken straight from the lectures. Furthermore, we will develop default quizzes to give to those students, but these quizzes may test topics the student has not yet viewed. Without any user tracking, this is unavoidable.

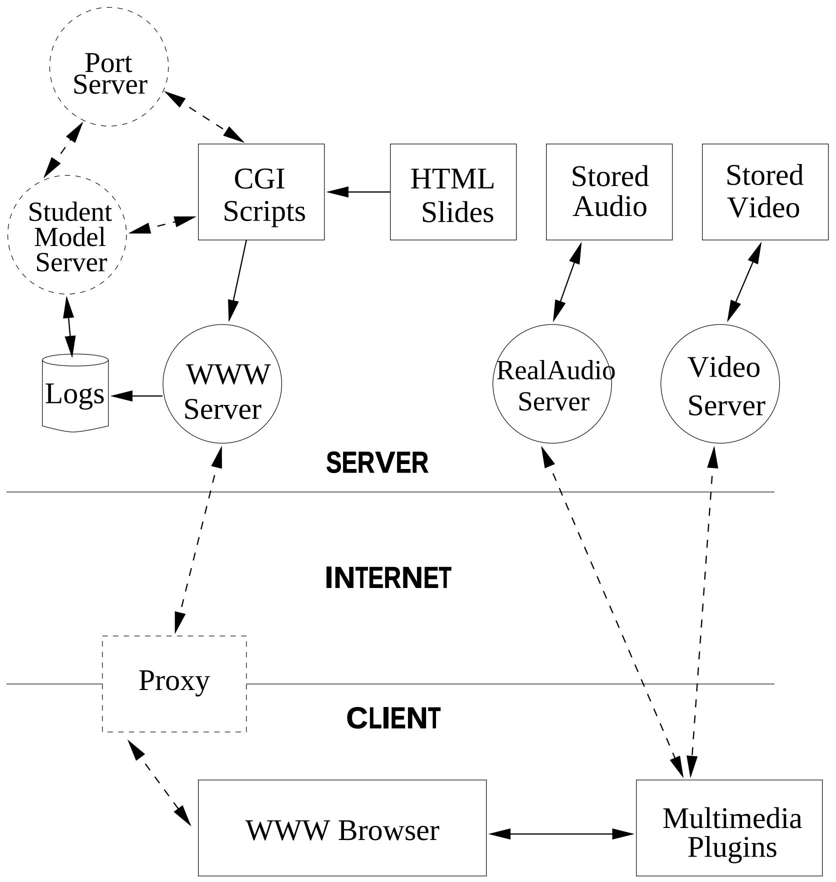


Figure 2: The architecture for the MANIC project

4 Current system architecture

There are five basic components to MANIC:

- Client side software (Web browser with RealPlayer plugin)
- Proxy
- HTTP server (running CGI scripts)
- Port server
- Student Model server

Figure 2 illustrates the system's architecture. In this section, we describe the details of each of these parts.

4.1 Client software

At the client side, the student uses a web browser, such as Netscape Navigator or Microsoft Internet Explorer. Additionally, students need the RealPlayer plugin so they can listen to the audio and view the video associated with each HTML slide.

By using the RealPlayer technology, we are able to synchronize the audio playback with the slides being displayed. RealPlayer provides for timing stipulations, such that at a certain point in the playback, the HTTP server is contacted to send a new slide to the Web browser. For more information on the workings of the audio/slide synchronization, see [27].

All of the software needed by a student is free and easily available. One of our goals when starting this project was to avoid client-side development as much as possible. By using commonly available web browsers and multimedia players, we have accomplished this goal.

4.2 Proxy

A proxy is an application that runs between a web client and server [32]. All requests from the client are processed by the proxy before being sent to the server. Similarly, server responses are processed by the proxy, and are then sent on to the client.

We have included a proxy as part of the architecture solely so we can do prefetching. Without a proxy, we would have to add data to the Web browser's cache, which is not practical. Therefore, the client must employ its own cache, which is done through a proxy.

The proxy that we are developing is a general purpose proxy, in that it will be able to work seamlessly between any client and any server. However, when the proxy needs to contact the MANIC HTTP server, it will behave differently, by adding information to the header of the request. The proxy will also cache course material, and if the student's requested page is in the cache, then the request does not go to the HTTP server.

Whenever the student makes a request for slide material, however, some information is sent back to the HTTP server, for logging purposes, even if the request can be fulfilled from the proxy's cache. This information is piggybacked with a normal HTTP request. There are three types of requests that the proxy sends to the HTTP server. First, if the user's request can be serviced from the cache, the proxy simply informs the HTTP server that the student has viewed the slide (so that the student's log is updated).

Second, if the request cannot be served from the cache, the HTTP server will be asked to provide the information, as well as update the log. The HTTP server, in response, will send prefetch and delete advice when either of these two requests are received. This information is sent as part of the header of the response, and the MANIC proxy knows how to parse this header to retrieve the information. Other proxies will simply ignore the headers.

Once the header is parsed, the proxy can send the third type of request – to prefetch a document. In this case, the HTTP server must be contacted to send the data, but the request is not recorded in the student log, since the student has not requested the document yet. Exactly what to prefetch is a decision made by the student model server (see Section 5.5).

The proxy is not obligated to follow the advice of the student model server concerning prefetching. Future versions of this software will include the ability for the proxy to record information on network congestion. Clearly, only the proxy can record this, since it knows the time a request is sent to the HTTP server and it can record the time that request is fulfilled. If the network is too busy, the proxy can decide not to prefetch, to avoid adding additional load to an already saturated system.

We have omitted many implementation details concerning the proxy. The interested reader is referred to [37] for a more detailed discussion.

4.3 HTTP server

This part of the system is a basic HTTP server which services HTTP requests. The server accepts requests from the client and sends responses back. We will not describe the HTTP protocol; see [4] for details on this protocol. However, the HTTP server has been augmented with common gateway interface (CGI) scripts.

One role of these CGI scripts is to dynamically creates the HTML pages that the user sees. This is done to update the highlight and buttons displayed on the page. Otherwise, we would have to make multiple copies of the slide, one for each highlight.

HTTP is a stateless protocol, and in order to do student modeling, we need to maintain information about the user. Traditionally, student models have been implemented in situations in which the tutor has a stateful interaction with the user. The WWW, however, provides us with challenges not seen in stand-alone systems, such as maintaining state while using the stateless HTTP protocol.

Our solution to this problem involves "cookies" [24] to maintain a user's state. A cookie is a header which is set by the HTTP server and sent to a client, and contains one or more name-value pairs. For example, a cookie may consist of "name = Mia". Whenever the client contacts the server, the cookie is sent as part of the request. In this way, the server "knows" who the client is. Using the previous example, whenever the "name = Mia" cookie is sent, the server knows it came from the client whose name is Mia. Thus when the client contacts the server, the server can record a log of what the student is doing, given that the cookie indicates who the student is.

The use of cookies has helped us track the mode in which the student is using the software, i.e. whether the slides are highlighted and whether the audio is playing. If the user does not want to see the highlights, a cookie is set that indicates this. When the HTTP server generates the slide, the cookie is checked to determine if the slide should include highlighting or not.

A cookie also records if the sound is being played. As we have noted before, students have the option of browsing the material, without using the audio. If the user is browsing, a cookie is set to indicate this mode. This was necessary for one simple reason – the RealAudio player does not accept signals from the Netscape viewer. And since we wanted to have all the controls in the Netscape window (see Section 3), the RealAudio player must be told that the audio is no longer playing. To do this, when the user clicks on any button that stops the audio, a silence file is sent to the RealAudio client to play, thus halting the previous playback. At this time, the user's cookie is sent to indicate that the audio is not playing. Then when the user clicks on another control button that would stop the audio, the silence file need not be loaded, since the audio is not playing.

A new implementation of MANIC aims to fix this problem. This new solution involves embedding the RealPlayer controls directly in the browser window. Java is then used as the intermediary between the client and the server.

4.4 Port server

Once the HTTP server knows who the student is, it must make decisions about the educational material the student should see, based on that student's model. There are a few options for which part of the system should make these decisions. The first one is to allow the HTTP server to make all of the decisions with CGI programs. However, this is not a viable option since, in order to save time and processing, the student model decision engine should maintain the student model in memory. If the HTTP server were to itself maintain the student model, it would have to reconstruct the model on each request, due to the lack of state.

A second option is for the proxy to make the decisions rather than the server. We discarded this option for several reasons. First, it is possible that users will run their own individual proxies. In this case, their computers may not be powerful enough to handle the student model processing. Second, for consistency, we want to store the student logs at a central site. Every time the student uses the software, the log would have to be transported to the proxy, taking a possibly significant amount of time for a large log.

For these reasons, we have decided to use a separate *server* to control the student model. In this way, the student's model will be maintained in a running process, and will not have to be reconstructed on every request. Currently we spawn one student model server for each student registered for the course. To do this, we assign a "port" to each student model server on which it can be contacted. One way to do this is to have the HTTP server be responsible for the creation of the student model servers. However, since HTTP is a stateless protocol, the HTTP server could not remember which ports were in use without having to reconstruct that information. Another option, the one we have chosen, is to use a *port server* that can maintain in working memory the ports that are in use.

When a student registers for the first time, the HTTP server contacts the port server in order to get a port number for this student. The port server then creates a new student model server to listen on that port. The port assigned to the student is also written to a log, for use when the student logs in again. With this architecture, the port server and the student model servers run on the same machine, but they do not have to execute on the same machine as the HTTP server. With this architecture, the same CGI scripts on an HTTP server can contact different student model servers and port servers to teach different classes.

A consequence of this architecture is that once all ports are in use, there is no way to add students to the system. Also, once a port is assigned to a student at the beginning of a session, it can only be taken away when a timeout occurs, since there is no explicit log out procedure. Timeouts occur when the student model server has not received any connections for one hour. At this point, the server assumes that the student has "logged out", and thus the port can be reclaimed.

Once the student has registered, the port server takes that port out of availability, and returns the port number to the HTTP server to transmit as part of the user's cookie. This is necessary so the HTTP server knows which student model server to contact while the student is using the course material.

When a student logs in after having registered, the HTTP server looks in the log file to find out the port number for the student. It then contacts the student model server on that port, sending the name of the student. If the student model server is not servicing that student anymore or a student model server does not exist on that port, a negative acknowledgment will be sent, and the HTTP server will then have to contact the port server, as it does when a user first registers. But if the student model server is waiting for the student who just logged in, the port server need not be contacted.

After the student logs in (either the first time or repeated), the HTTP server can contact the student model server directly, using the port number for the student (which can be retrieved from the cookie). Thus the port server will only be responsible for keeping track of port numbers and spawning new student model servers.

4.5 Student model server

The student model server is responsible for making decisions based on the usage patterns of each student. As previously noted, there is one student model server for each student who is logged into the course. When the student model server is created by the port server for a student who had previously used the course, that student's model is reconstructed by reading the file recording all of his previous actions. Thus the student model is reusable between sessions.

Whenever a student takes any action (e.g. views a slide, takes a quiz, or views the index), that action is reported to the student model server. A student's server then keeps track of the slides the student has seen, the way he has been viewing the slides (e.g. playing the audio continuously or just browsing), and the quizzes he has taken. The server is then used for all the intelligent decision making processes, including helping the student navigate the course, deciding the content of the slides the student sees, deciding which documents to prefetch, and creating the quizzes a student is "qualified" to try. The decisions made at the student model server will be described in more detail in Section 5. Each action the student takes and each decision the student model server makes is recorded in the student's log file, which more permanently records his student model.

The HTTP server is the only part of the system that initiates communication with the student model server. The HTTP server sends the following list of commands to the student model server:

- display the welcome page for the student
- set the control of choosing the topics
- get the last slide the student has seen
- generate a new slide
- generate a quiz
- grade a quiz
- give the topic list
- save the notes written by the student
- retrieve the notes written by the student

When any of these requests are made, the student's name is sent as part of the request. In this way, the student model server can check that the request is for the student it is serving. If not, the HTTP receives an error code saying the request could not be fulfilled. In this case, the HTTP server must contact the port server in order to set up a new student model server. Then the HTTP server can contact that student model server to handle the request. If the name is correct, the student model server simply fulfills the request.

According to the directive sent by the HTTP server, the student model server will perform the correct action. The results of the student model server's processing is sent back to the HTTP server to be sent directly on to the client's Web browser.

5 Proposed Implementation

In this section we discuss the proposed futher implementation of the MANIC on-line tutoring system.

5.1 Domain Organization

The domain will be structured in terms of "topics," with each topic consisting of a set of subtopics. These subtopics are smaller pieces of knowledge within a topic, and are used to give us finer grained details for the knowledge base. We may need even further subdivisions, but for now, these two seem sufficient.

The topics are arranged in a semantic network [42]. In this network, topics that have some relationship to other topics (e.g. prerequisite topics or related topics) will be linked together, with the value on the link the weight of this relationship. The prerequisite links are directional; they originate from prerequisite topics. For example, if topic A points to topic B with a weight of 0.75 (on a 0 to 1 scale), then topic A is a fairly important prerequisite to topic B. The related topic links are bidirectional.

Topics that are closely related will have higher weights on the links, while those that are not closely related will have lower weights. The weights are determined a priori by a domain expert. Unlike traditional tutors that determine what students see (for example, the LISP tutor [1] and the Cardiac tutor [20]), the MANIC system does not strictly impose presentation of prerequisite topics before a new topic is started.

Furthermore, it is often necessary to add course material, i.e., supplemental topics to the existing material. This might be done for two reasons: (1) material is added at branch points so the course is non-linear and (2) remedial topics are added on background information to help a student with deficient knowledge. In either case, a domain expert will be required to supplement the already existing material. This supplemental material is necessary since instructors have limited time in lectures to cover all the topics to the depth to which they would like. However, with an on-line version of the course, there is no time limit, and the additional topics should be added.

Furthermore, we will be adding supplemental material in the form of hypertext links. Each hypertext link will have a label indicating the type of material it contains. Some examples of the types of hypertext links we will include are links to related material, links to prerequisite material, and links to more detailed information. These labels are important as they provide the student model with more information on the student's preferences and abilities, by analyzing which types of links the student follows. For example, a student who elects to see related material, but not necessarily material required for the class, will most likely be an enthusiastic learner.

Additionally, we will be extending the existing course by providing alternative ways to present essentially the same material. This is necessary since in all courses, students have different levels of ability and knowledge. In a lecture-based class, the instructor cannot individualize the teaching for each student in the class. Therefore, the instruction may be too slow for some students and too fast for others. This is illustrated explicitly in the course that we are working with, since both graduate and undergraduate students were enrolled. The instructor of this course (James Kurose) has noted that there are times when graduate students are bored because something must be explained to undergraduates. Ideally those students should be able to skip the parts they know, or just see a review.

These multiple ways of teaching must provide the correct amount of detail for each student. Some explanations provide more details on background information, while others assume that knowledge is known. For example, when discussing the TCP checksum, 1's complement is discussed. One version of the material should describe, in detail, the definition and functionality of 1's complement. Another version should just use the term without defining it. However, in the second case, there would be a hypertext link to the definition, in case the student did want to look at it after all.

There are three ways that we have considered doing this. The first is to have multiple versions of slides, say three: for novice, average, and advanced students. The version for novice students would contain more background information and more detail, while the advanced version would assume this information is known by the student.

However, this approach has some significant drawbacks. First, a great deal of work is required to create these different versions of the slides. Either the instructor or another domain expert must determine the material that should be included on each version of the slide. So instead of simply constructing one slide, three must be made, each with slightly different information.

Second, there is not much flexibility in the teaching, since all of the material is hard coded. There are three ways to individualize the instruction, but only those three ways. This is too restrictive for an intelligent tutoring system.

The second approach to developing multiple versions of the course material is to have only one version of the slide hard coded, the advanced version. At certain points in this hard coded slide content, supplemental material could be added if necessary. Using the 1's complement example again, for a novice student, the slide would be constructed with all the relevant information on 1's complement added. However, for the advanced student, the slide would be constructed without this information. This would reduce the work required to put the slides on line, since only one version of each slide would need to be written. But the places for supplemental material would have to be identified, and additional details written.

This alternative would also increase the flexibility in teaching. The tutor would not be restricted to three versions of the material. Rather, it could add information only when necessary, further specializing the course for each student.

The final, and most promising approach to developing alternate ways of teaching is to abandon the slide format all together. Rather, we would construct a database of *content objects*, which are of every piece of text, image, animation, and video that could be shown to a student. Each content object would have an associated degree of difficulty, indicating how detailed and descriptive it is. Also, associated with each object is some audio and video, which could be played when presenting the object to the user.

Furthermore, each content object would have an associated *instructional type*. These instructional types include definition, example, description, graphic, and animation. Each subtopic in the domain would have at least one content object of at least instructional type. Figure 3 shows how the domain will be arranged in this way.

The order of presentation of the subtopics within a topic is linear. However, how the subtopic is presented is decided by the student model server. For each subtopic, the tutor, using the student model, has two decisions to make. The first decision is which instructional types should be used. The second decision is which content objects should be displayed for the chosen instructional types. Choosing the content objects appears to be the easier decision, as the decision is based on the student's model and the level of difficulty of the object. The tutor will pick the content objects that are at the right level of difficulty for the student.

Choosing which instructional types to use is a much harder decision. There is no easy way to judge which types work better for each student, and under what circumstances. An initial strategy we are considering is to use more instructional types for students with lower scores on a topic and less for students with higher scores. However, we would like to investigate techniques in which we would be able to judge a student's learning style preferences and use these to choose which instructional types should be used.

The advantage of this method is the system's flexibility in determining what to display to the user. All of the user's interactions would be customized based on the student model, and two students would almost never see the same page. For example, for a given subtopic, one student may see content object 1 from instructional type 2 and content object 2 from instructional type 5, whereas another student may see content object 3 from instructional type 1 and content object 1 from instructional type 2.

However, the disadvantage is the increased complexity in representing the domain knowledge and in the engine required to create material to show to the user. The workings of the intelligent engine will be driven by the student model, and will be discussed in the next subsection.

Another difficulty with this solution is the student cannot jump to other subtopics within a topic. We are investigating a way to show the subtopics to the user so that he can choose what parts of

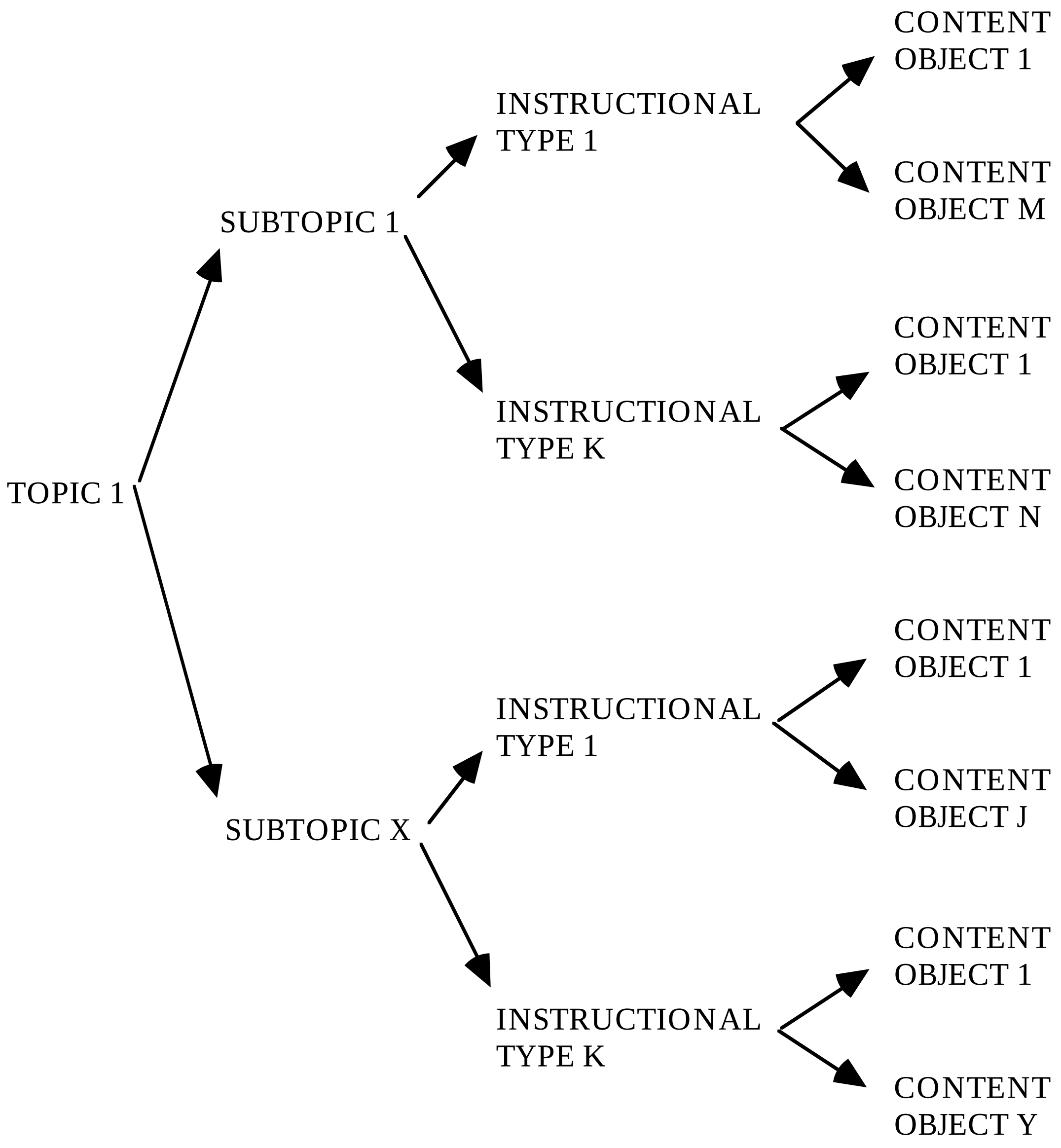


Figure 3: The content object structure for a generic topic

the topic to view.

With each of these solutions, the student model server must keep track of the actual page displayed to the student. Each time the student navigates through the course material, the student model server is contacted to generate the content of the page. And this is the behavior we want when the student is proceeding forward in the course. However, when students want to return to previous material that they have just seen, using either our provided "previous" button or the web browser's back button, a new page should not be generated. In this case, the student model server should simply return the exact page the student had seen, without generating new content.

To do this, a unique code is generated for each page displayed to the user. This code is included in the URL when the page is displayed. This unique code is also included in the URL which is referred to by the MANIC previous button. If the student model server sees this code in the URL, it knows to simply display the page associated with the code. The next button pointers and the pointers from the index do not contain this code, so when those links are followed, the student model server knows to generate a new page, choosing the appropriate content.

5.2 The student model

In this section we describe the representation of the student model and discuss how it enables the tutor to individualize the course for each student and prefetch parts of the course material before they are requested by the student.

5.2.1 The basic student model

The student model in MANIC is essentially an overlay model [15]. In an overlay model, the student's knowledge is considered to be a subset of the expert's. The goal of the system, then, is to expand the student's knowledge until it is equivalent to the expert's. No buggy knowledge is tracked, as the system simply attempts to guide the student to learn the desired knowledge.

The MANIC student model tracks a student's "ability" on each topic within the domain. This ability is represented as five scores. The first score records how much of the topic has been viewed. This is determined by how many subtopics a student has seen on the topic. It is also influenced by how long a student spends on each page. For example, a student who only spends 1 second per page cannot learn the material as well as someone who spends 20 seconds. Accurately recording this kind of timing information is impossible, since there is no way to distinguish someone staring at a page for 30 minutes from someone who has gone to eat dinner.

The second score is determined by which version of the material the student has seen. If the student sees a more advanced version, his ability on the topic is more highly rated. This score also tracks if the student sees the correct version. For example, if the student sees the more advanced version of the material, but this presentation is in error, the student model records this information. In this case, the tutor judges if the presentation is in error based on whether the student clicks on the hyperlinks to provide more details on the subtopics.

The third score records the student's access patterns for the pages. If he chooses to review some of the material within the topic, then he probably does not understand it as well as someone who chose not to review. Also if he did not listen to the associated audio, he may have missed some of the information, thus reducing his presumed comprehension of the material.

The fourth score records whether the student has followed any of the hypertext links on any of the pages to review the topic. If so, then he possibly did not fully understand that background topic.

Finally, the fifth score is the student's quiz performance on questions associated with the topic.

How many questions the student got right and wrong, and their levels of difficulty, indicate directly how much the student knows about the topic.

We have elected to use (at least) these five scores in order to retain as much information as possible about the student. Each of these scores records different information about the student, and can thus be used in different ways. For example, one of our prefetching decisions is based on whether a given student follows hypertext links. By having a score for this information, we can easily determine the hyperlink access patterns. Just having a single score on each topic does not provide enough information about the student's learning styles and abilities.

It is possible that the student model we have just described will not be sufficient for the MANIC system. Attempting to discern a student's understanding based on his access patterns may turn out to be too hard. We may discover that the only way to accurately be able to judge a student's knowledge is to base the student model mostly, if not solely, on quiz performance.

5.2.2 Keeping track of the student's past performance

In addition, the tutor needs to keep track of how the student's performance changes over time. For this reason, we are using a history-based student model. Each action a student takes is recorded. Since the domain is organized in terms of topics, student actions are associated with the topic currently being studied. When a student leaves a topic to start another one, the appropriate scores are calculated on the topic being left. Thus when looking at the overall score for a topic, the tutor can return to this action-by-action record to see the trends in the student's learning.

For example, when considering quiz scores, if the student's initial score on a topic quiz is 50 and his next score is 35, then the tutor can determine that the student is having difficulties since his scores are declining. On the other hand, if his initial score on a topic is 10 and it changes to 80, then the tutor might reason the student learned the material very well, and perhaps advance the student through the curriculum at a faster rate.

5.2.3 Using pretests

When a student starts using a MANIC course, the student model has no information on the student's knowledge and ability. Therefore, each student starts out with the same initial model. As has been shown in [3], this a priori approach to student modeling is not sufficient for accurate student modeling. Eventually, if we have designed the student model to update correctly, the tutor will converge on the "correct" student model. However, it may take a long time for this convergence to happen, and before it does, the student is not receiving the best instruction from the tutor.

One solution is to provide a pretest for all students who begin a MANIC course. This on-line pretest would assess the student's basic understanding of the topics within the course and the initial model will be closer to his actual student model than if he started with the default model. As a consequence, the early curriculum and quizzes received by the student will be tailored to his needs and abilities sooner than if the default model had been used.

The student will be advised that this pretest is intended solely to provide information for the tutor to use when helping the student. He is not expected to perform well on the pretest, as the material may be new to him. However, the information on the pretest will help initialize tutor responses based on a more accurate student model.

5.3 Guiding the student through the material

Work in adaptive hypermedia has identified two methods for adapting the material in a course to each individual student: adaptive presentation [9] and adaptive navigation support [26]. The first technique provides different course content for different students, while the second changes the links visible from a given state. We use both techniques in the MANIC system.

When guiding a student through the course, the tutor assumes that the student is viewing the course as we are, in terms of topics. However, this may not be the case. The student may be using the material, for example, as a review. In this case, the student may not view most of the pages for a single topic. Rather, his pattern for viewing the material will be inconsistent and appear somewhat random. Guiding the student in this case may not be possible.

Although we do not necessarily want the tutor to control how a student progresses through the material, we do want it to be able to help guide the process. There are a few ways it can accomplish this, including suggesting topics a student should view and generating the content of the pages the student sees.

In terms of topic suggestions, it is very unlikely that students will want to diverge from the most obvious linear path through the curriculum (as demonstrated by the comments given to us by students after our first test of the MANIC system). Therefore it is likely the student will follow the tutor's suggestions, or even let the tutor decide which topics to learn. However, even though student's may not follow alternate paths through the material on their own, the tutor can still either suggest or provide (depending on who is choosing the next topic) review and remediation when appropriate. Much of the review decisions are based on time between uses of the system. The remediation decisions are based on quiz scores and the use of hypertext links.

On the other hand, regardless of the path taken by the student, the tutor will be able to alter the content of the pages, based on the student model. In this section we discuss how topics are prioritized and how the page content is individualized for each learner.

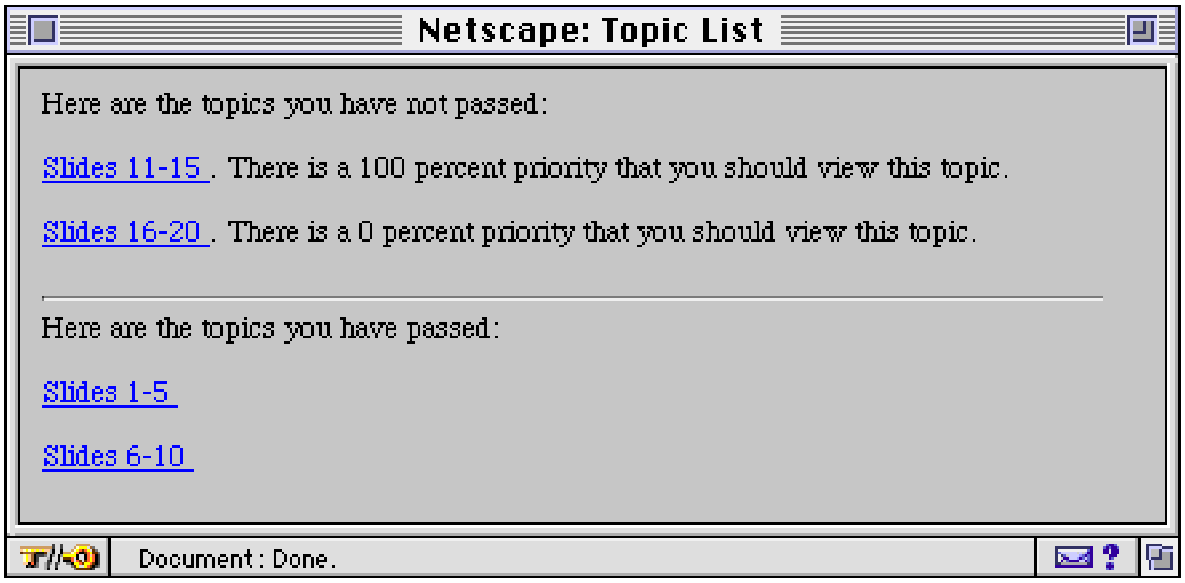
5.3.1 Suggesting topics

When a student finishes a topic, a new topic should be started. There are two options for deciding on a new topic: the tutor's choice or the student's choice. If the tutor chooses, the new topic is started without informing the student of the choice. The tutor chooses the "best" next topic, based on the student model and the topic net. On the other hand, if the student is choosing, he is presented the list of topics, from which he must choose one. The list is annotated with suggestions from the tutor as to which topics the student should study. The tutor thus provides adaptive navigation support. This is similar to the mechanism used in ELM-ART [14] and in the ISIS tutor [11]. As in those systems, the student has the option of either taking or ignoring the tutor's suggestion.

Regardless of who is choosing the next topic, the process of ranking topics is the same. The tutor examines all topics related to the topic just finished and assigns a priority to each. These priorities are based firstly on the based the link weights. The higher the link weight, the higher the priority for the topic. Secondly, the scores of all the prerequisite topics for the possible next topic are considered. The scores that are considered, in order of importance, are the quiz grades on the topic, how much of the topic was seen, how the material was viewed (i.e. whether the student had to review earlier parts of the topic), and which version of the material was seen (i.e. what level of difficulty were the content objects that were presented). If the scores are high on all prerequisite topics, then the priority for this topic is higher than a topic with only half of its prerequisites having high scores.

Another factor to consider is how recently those prerequisite topics have been seen. If they were seen far in the past, a higher priority is given to a review of those topics than to moving on to unlearned material.

As a result of this policy, a student may see a next topic that is highly related to the current



Here are the topics you have not passed:

Slides 11-15 . There is a 100 percent priority that you should view this topic.

Slides 16-20 . There is a 0 percent priority that you should view this topic.

Here are the topics you have passed:

Slides 1-5

Slides 6-10

Figure 4: The topic list as it currently exists in MANIC

topic but whose prerequisites have not been mastered over a topic whose prerequisites have been studied but whose relation to the current topic is not as strong. It is also possible that the opposite effect will occur.

Finally, for remediation, the scores of a potential remedial topics' "next" topics are considered, including those topics' quiz scores. If they are not high, greater priority is given to reviewing this topic. This is of importance for topics that are in the domain, but only as background knowledge. These background topics are not essential to teach, but if a student has shown difficulties learning the topics for which the background knowledge is necessary, the background topics should be given high priority. For example, the *fork* system call is discussed in a socket programming course, but it is not taught. A background topic teaches this subtopic, not as part of the regular curriculum, but as remediation when required.

If the tutor chooses the next topic, once each topic is assigned a priority, the one with the highest priority is chosen and started. If the student is choosing, the list of topics, annotated by the priority, is presented to the student.

Additionally, the student can see the topic list at any time while using the system. The list is again presented with annotations from the student model server, based on the current student model.

The list given to the student is just that, a list. In the future, we hope to be able to present the domain structure in terms of the domain graph, so the student can see where he has been and where he should go. Figure 4 shows the current interface and Figure 5 shows the proposed interface.

It should be noted that unless students take quizzes, the tutor does not have much data on which to judge them. Therefore, its predictive abilities are limited. In this case, the tutor can suggest prerequisite topics that have not been covered, but it will have no basis to judge if remediation on topics not understood is necessary.

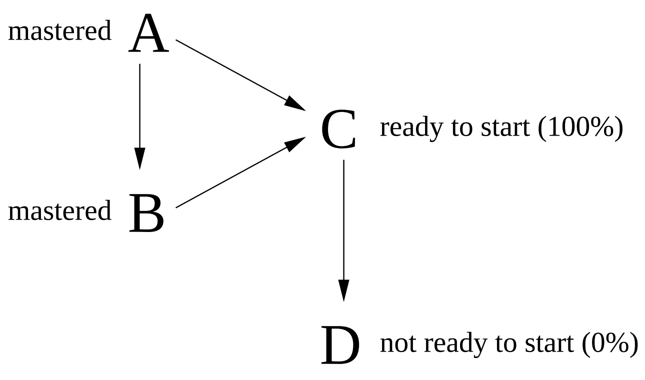


Figure 5: The proposed topic list in MANIC

5.3.2 Determining which content objects to present

The student does not control every aspect of the learning. The tutor decides the content of the pages and the audio, based on his scores on the topics. By doing so, the tutor uses the adaptive presentation technique for customizing the material.

When generating the content of a page, the tutor must make two decisions: (1) which instructional types should be presented and (2) which content objects for the chosen instructional types to include in the final page. Choosing the instructional types should be based on the student's learning styles, i.e. does he respond better to pictures or text, does he need examples, or does he need an explanation? However, in our proposed student model, we have no way to determine such learning styles. We will thus explore how to incorporate this information into the student model and then how to use it to pick the appropriate instructional types.

When deciding which content objects to include in a given page, the tutor examines the student's score on quiz material for this topic, which content objects he has seen in the past, which content objects of the prerequisite material he has seen, and how many prerequisite topics he has studied. Essentially, the tutor is predicting, from past behavior, the student's scores on the chosen topic. For example, if a student has been doing well on all of the quiz questions and he has seen the hard content objects in all prerequisite topics, then he should see the harder and more abbreviated content objects for this topic. On the other hand, if a student jumps ahead to a more difficult topic for which there is no evidence that he is ready, he will be provided with the more detailed content objects, which provide more information on the prerequisite knowledge.

Furthermore, if a student starts a topic as part of remediation, then he should be given an easier version of the material, especially if he had seen the material previously. Remedial work should provide as much support to the student as possible, and giving material that includes more detail accomplishes this goal.

The tutor may not be completely accurate in its assessment of the student's ability and thus it may present the wrong level of detail for certain topics. To compensate for this, the tutor is designed to take a conservative approach. Students will generally see material that is either at the right level of difficulty or easier, with the theory that more detail is better than less.

Even with this philosophy, mistakes will be made. For example, if a student has passed the prerequisite topics for a given new topic, he still might not know all of the material. But if he is given the more advanced versions of the material, without as much explanatory information, there are still links to retrieve those details. Then if the student chooses to view those links, the tutor must reevaluate its judgment of the student, downgrading its view of his ability.

5.4 Quizzing the student

Students have the opportunity to take on-line quizzes at any time while using MANIC, either following the tutor's suggestion or choosing to take a quiz on their own. It may become necessary to require quizzes, as they provide the student model the best source of information. However, currently, quizzes in MANIC are optional.

5.4.1 Types of questions

The MANIC system supports three types of questions: true/false, multiple choice, and short answer. The instructor for the course enters the question text and the answer, and in the case of multiple choice questions, the distractors. Currently the system does not dynamically generate the question text and answers, and we do not see that capability being incorporated any time soon.

Associated with each question is a subtopic that the question is testing as well as a "level of difficulty." For example, a multiple choice question with obvious distractors is easier than a short answer question (generating the answer is usually harder than being able to "guess" the right answer).

5.4.2 Creating a quiz

The tutor dynamically decides, from a question database, which questions should be asked. The tutor reasons about: (1) selecting questions for topics the student has just seen (and their prerequisites) and (2) selecting questions for review. The factors which influence the tutor's decisions are concerned with the subtopics describing the topic on which the student is currently working. These factors include:

- Which content objects have been seen - If the student has seen the less detailed and difficult content objects, then he should receive harder questions than someone who has seen the easier version.
- How frequently the material has been viewed - If the student is viewing the material for the second or third time, based on the suggestion of the tutor, then it is unlikely he has fully mastered the subtopics. Therefore, the questions he receives should not be at the highest level of difficulty. However, if he is simply reviewing the material on his own, then he should be given harder questions, due to his exposure to the material.
- The student's scores on prerequisite topics - If the student has understood the prerequisite information, then there is a higher probability that he understands the subtopic being tested. Therefore he should receive harder questions.
- Other questions on this subtopic - Has the student answered other questions on this subtopic, either correctly or incorrectly? This should clearly impact which questions the student should receive.
- General test performance - If the student, in general, does not perform well on these quizzes, then he should not be challenged as much.

Each of these factors is considered when choosing questions to ask on a particular subtopic. The tutor gives higher priority to those questions that have not already been answered correctly. However, if there are no new questions on a given subtopic, one question is chosen as review.

The tutor may also choose to ask questions about topics on which the student has not demonstrated sufficient mastery. In this case, the tutor looks at the scores for all topics. If the quiz score on a topic is below a certain threshold, then the student is asked questions about the subtopics within that topic. The tutor decides which questions to ask using the factors given above, with the additional criteria that they be easier, if possible, than the questions asked previously. The student will be asked review questions, both on material on which he has been quizzed, and on material not yet quizzed but viewed some time in the past.

Likewise, the tutor may elect to do some review work on topics on which the student has done well. This review is planned in the following manner. If the previous quiz score on a topic is above a certain threshold, the tutor then looks at how long it has been since the topic had been part of a quiz. If the time has been sufficiently long, then the topic is retested. The time variable is a function of the topic's score: the higher the score, the longer the time interval allowed. If the score on the topic is very high, there is a good chance the student knows the material, and will thus need less frequent reminders. When testing a topic that has a high score, the tutor tries to choose questions that are harder than those asked in the past. If none exist, those questions that have not been asked for the longest period of time are chosen.

It should be noted that within a topic, it is possible that not all subtopics will be tested. This will happen if it seems that the student has a good grasp of some subtopics, and it is not time to review that subtopic.

An interesting side effect of our strategy is that it is unlikely a student will see the same questions on two subsequent quizzes. As soon as the student enters his answers, the student model is updated. If he then goes back to try to take the quiz again, a different quiz will be generated, based on the new information in the student model. Thus we do not have to counteract possible "cheating" by students.

5.4.3 Grading a quiz

The student is graded on his quizzes, with correct answers leading to increasing his scores on the corresponding topics. If the question is answered incorrectly, the tutor concludes that the student did not understand the subtopic nor the corresponding topic. Then the student's assessment on the topic is lowered and the question is marked as being answered incorrectly. Furthermore, the tutor suggests that the student review the material corresponding to the subtopics he did not understand. If the student elects to see these pages, he will see an easier and more complete version, since his knowledge of the topic and subtopic have been downgraded.

Additionally, we are considering the possibility for the tutor to judge certain misconceptions that the student has, which would result in an incorrect answer. If the student gives an answer which is classified as a common misconception, the tutor can explain to the student what that misconception is and how to correct it.

5.5 Prefetching

With all the multimedia used in MANIC courses, the network is bound to suffer from difficulties such as saturation and insufficient resources to provide good quality of service. We are interested in exploring solutions to these problems. However, our situation is different from others on the WWW. The MANIC system provides a closed environment in which we can analyze individual user behaviors. This gives us a smaller testing environment for our work than the entire internet.

Many users on the World Wide Web complain of latencies experienced due to downloading a new page [38]. Our solution to this problem is to prefetch material to the client's machine before it is

	Student model server activity	Proxy activity
Server-based architecture all decisions made at server	Quizzes Prefetch decisions Construct pages	display pages request page to prefetch
Proxy-based architecture client side page construction	Quizzes	Display pages Request objects to prefetch Construct pages
Hybrid architecture server side decisions, client side construction	Quizzes Decide objects to prefetch Decide how to construct pages	Display pages Prefetch objects Construct pages

Table 1: The three possible architectures

explicitly requested in real time. As we have discussed, we can only prefetch small portions of the course at a time, due to limited cache space at the proxy.

This prefetching, if accurate, can reduce the delays seen by the user, since the requested information will be available before it is requested. However, prefetching that is not accurate will waste network bandwidth and cache space, by storing pages not requested by the user.

The question we must tackle in the MANIC system is how to make the prefetching accurate. Prefetching has been implemented in both file systems [30] and on the WWW [38]. The work in these papers involves general population models for guiding the prefetching. Our work, however, centers around individual user models to aid with prefetching.

While accurate prefetching would reduce the delay seen by the user, faulty prefetching will add load and be useless, or possibly even detrimental to the quality of service received by all users. In MANIC, and any prefetching system, some probability exists that prefetched data will not be used, so the system will have done extra work that it could ill afford to do, and the work was to no avail.

Therefore, we also must investigate when *not* to prefetch. The worst delays will be seen when the network, server, or proxy are saturated, i.e., the requests are coming in faster than they can be serviced. When this occurs, prefetching should be throttled, since servicing the prefetching requests adds more load to the server, the proxy, and the network, and takes processing power away from real-time requests, which should have top priority.

In this section we explore three prefetching architectures. Table 1 summarizes the differences in these architectures. But first we discuss what can and should be prefetched.

5.5.1 What to prefetch

The prefetching mechanism in MANIC can attempt to predict every action the student will take. However, that is not realistic. Rather, there are three specific instances when MANIC will attempt to prefetch. The first is prefetching while the student is viewing the material on one topic. Prefetching can only be done if the system can determine the student's pattern through a topic. Currently, only a few patterns can be detected within the MANIC system: (1) the student lets MANIC control the display of pages in a topic and (2) the student linearly scrolls through a topic, without listening to the audio. If either of these patterns is detected, the next page in the sequence is prefetched. We are currently analyzing the access patterns from the first student experiences with MANIC to identify other patterns students use for viewing the material. Most likely, however, we will continue to only linearly prefetch within a topic.

The next category of prefetching occurs when the end of a topic is reached. Which new topic should the tutor start to prefetch? Because no clear linear path through the course exists, this work differs from that presented in [44]. If the tutor predicts the student is most likely to choose one particular topic, it can start prefetching that one, ignoring the others. The tutor can begin to verify its choices by seeing how accurate its predictions are for the student's choice of next topic.

However, another possibility is that the tutor will not reason with any confidence about the student's new topic choice. In this case, the tutor can prefetch the first page for each topic, essentially "covering all bases." Once a topic is started, it alone can be prefetched.

A compromise between these two extremes is to prefetch the first few pages of the topics that are most likely to be started by the student. These topics can be determined by viewing how the student progresses through the course and matching this with the weights in the topic net. Those topics with the highest relevance from the current topic are candidates for prefetching. To determine how many pages per topic are prefetched, we use the simple rule: the higher the probability, the more pages will be prefetched.

However, when prefetching multiple pages at once, a possibly severe problem arises, which we discuss in the next section. Because of this problem, it is more prudent to just prefetch one page ahead.

Another category of material that can be prefetched is the hypertext links that appear on some of the pages. The student model tracks which kinds of hypertext links are usually followed. If a link appears which is of a type the student usually follows, the tutor can prefetch that link. The tutor can also judge if the student knows the material to which the link points. If not, then the link content can be prefetched.

There is one instance in which we do not think prefetching will be successful. If the student brings up the table to contents in order to jump to another part of the course, the tutor will most likely not be able to determine which topic the student will choose. However, if it can with a sufficient degree of confidence, then some prefetching may be done.

5.5.2 Server Based Architecture

In this architecture, all of the prefetching decisions are made on the server side. The student model server decides what should be prefetched, constructs the page and then sends it to the proxy. Each page sent to the proxy has a unique identifier (see section 5.1). The proxy then searches its cache for a document with the desired identifier before contacting the server for the material. In this architecture, there is no intelligence at the proxy. This is the current implementation of the prefetching mechanism.

Problems with this architecture

A possible problem arises with the current implementation concerning the use of prefetching. The problem concerns possible staleness of the items in the cache. The tutor must decide the content of the prefetched page, just as it does with all other pages. This content decision is based on the student model. When the tutor creates the content of a page to be prefetched, it does so with the student model it has at the time.

However, when the student actually views the document, his model may have changed, and thus the content of the page would be different based on the now current student model. But there is already a cached version of the document at the client's site, so the request will not be forwarded to the student model server. Therefore, the student will see an inaccurate version of the page, which was constructed based on an old student model.

For example, if the student starts the audio for a topic, but then changes his browsing pattern, the tutor may already have sent additional pages based on the original usage pattern. When the student elects to see those pages that have been prefetched, they may not be accurate, since they are based on a different student model than the one the student currently has.

Specifically, say a student starts the audio for a given topic. Before he reaches the last page in the topic, he stops the audio and goes back to an earlier topic and spends a significant amount of time there. Now when he goes back to see the last page of the original topic, his student model has changed, and the page in the cache may not be accurate for his current state.

There are a few options for solving this problem. The first is to simply ignore it. We may decide that the benefits of prefetching outweigh the benefits of educational content based on an "accurate" student model. Another option is to clean out the cache periodically, i.e. those documents that have been in the cache for a certain period of time without being accessed. However, this option has the disadvantage of possibly evicting a page prematurely from the cache, causing an unnecessary cache miss. On the other hand, this option could be implemented entirely within the proxy, without any intervention from the student model server.

An additional solution involves intervention from the tutor. With this solution, the tutor can track what documents are in the cache, since it sent them in the first place. If the tutor notices that a document that has been prefetched is now stale, it can send a message to evict that document from the cache.

Another possible problem with prefetching is that there is no guarantee that a prefetched document will still be in the cache when the student must see it. This may occur due to some of the solutions just discussed, or because the cache must evict some information. In either case, the student model server needs to record that its prefetching prediction was correct, even if the document had to be deleted from the cache. In this way, the tutor can gain positive reinforcement for its reasoning, since it was correct, even if the document was evicted prematurely. Machine learning techniques may be possible for this reinforcement.

A final problem with this architecture is that documents in the proxy's cache cannot be shared by multiple users. If the proxy is being used by many students, then the amount of cache space allocated to each student decreases. The reason this problem occurs is that for student A, page 53 may be different from student B's page 53. If there was no dynamic content generation, then this problem could be avoided.

When *not* to prefetch

Any solution that involves evicting documents from the cache has the potential to increase network traffic, thus slowing down the prefetching. Furthermore, prefetching which is inaccurate will saturate the network with useless traffic that could be used to satisfy real-time requests. Therefore, we must examine when prefetching is and is not appropriate.

Within the current MANIC architecture, we need a mechanism for determining when the network is saturated enough to indicate that prefetching should be slowed or halted. To do this, we need to consider two factors: (1) the downloading delays and (2) the accurateness of the prefetching.

The only piece of the architecture that can determine the downloading delays is the proxy. The proxy can record the time a request is made and then the time the response is received. However, with this architecture, none of the "intelligent" processing is done at the proxy. Therefore, the proxy returns the timing data to the server.

To do this, when the proxy makes a request, it sends the timing result of the last request made. Thus each student model server will have an update on how fast the previous request was serviced. Each student model server can keep track of the response times for its student. If the responses are taking too long, prefetching can be stopped.

However, this policy gives us limited power to control the prefetching; the prefetching is either on or off. We would prefer a way to keep some accurate prefetching while stopping the inaccurate prefetching. We can accomplish this by using the student models.

With the policy just described, we are not considering if the prefetched material is actually being used by students. In other words, we are not considering the accuracy of the prefetching. Prefetching that is accurate should not be slowed or stopped. On the other hand, prefetched material that is not being used should not be sent.

Therefore, we must consider the individual student models when prefetching. Each student model server calculates a degree of confidence when prefetching. This confidence measure is based on how accurate the prediction mechanisms have been in the past. The student model server considers both this degree of confidence as well as the round trip delays when deciding whether prefetching should be stopped. As the delays increase, a higher degree of confidence is needed before an item can be prefetched.

We must investigate how the network delays interact with the confidence measure. One option is to hard code timing thresholds and their associated confidence measures. For example, if the round trip delay is 500 msec, the confidence value must be 50% before an item can be prefetched. However, this policy is far too limited, since the round trip delays to different parts of the world will be different. A 500 msec round trip time to Japan may be acceptable, but it would be unacceptable to a machine across the hall.

A more reasonable approach is to record relative round trip delays. Each student model server records the round trip delays for its student. Any significant changes in these delays results in a change in the confidence level needed, i.e., if the delay goes up, so does the confidence level. The question to answer now is how much should the confidence level change. For this, we will consider hard coding increases based on changes in round trip times. For example, a 10% change in round trip time results in a 10% change in required confidence level.

5.5.3 Proxy Based Architecture

As we discussed in the last subsection, the current architecture may not be suitable for accurate prefetching. In this section, we discuss an alternate architecture that would allow for accurate and non-stale prefetching.

In this second architecture, all the decisions concerning the content of pages are made on the client side. Furthermore, the documents the student sees will be constructed at the client's site. By having the document construction occur at the client side, we will not be using a stale student model. The decisions on what to display to the user will be made just at the time the page is to be displayed, using the most up to date student model.

A question we must consider is how to construct the page at the client site. Should it be done in the proxy or should we use a Java applet? Both would be able to contact the HTTP server for information. However, with this architecture, none of the intelligence is stored at the HTTP server. Therefore, the slide constructor must be able to contact the student model servers. Java applets, for security reasons, cannot make internet connections to machines from which they were not downloaded. Since it is not a requirement that the HTTP server and the student model servers run on the same machine, we cannot use a Java applet for this purpose. For this reason, only the proxy can be used for the purpose of constructing the slide.

With this architecture, the proxy would prefetch individual content objects rather than whole pages. The proxy would then construct the slide from these content objects. Thus it is highly probable that some of the prefetched material will be used, since there is no individualization in what the objects are. The individualization comes when choosing the objects, and if the objects are at the client's site, then they will be used when constructing the slide. It is also probable that more objects than will be used will be prefetched.

There are two questions we must consider: (1) when to prefetch the objects, and how many to fetch and (2) where to decide which objects to display.

When to prefetch the objects and how many
The first possibility for prefetching the objects is to download all objects associated with a topic when the topic is first started by the student. For students who view entire topics to completion, this policy would be effective. There will be some delay when a topic is started, to prefetch the objects. However, once they are all downloaded, there will no longer be any latency. Obviously this policy will download more objects than necessary, but that is unavoidable.

However, for those students who do not view the material in this way, this policy will prefetch many objects that the user will not see, thus wasting bandwidth and time. Rather, for these students, we should only prefetch the next n objects, where n represents a reasonable set of objects from which the next page could be generated. With this policy, objects will need to be prefetched as the pages are shown to the user. Essentially, we will only be able to prefetch ahead by one page of material. However, in the work by [17], smoothing out the prefetching provided better network performance than prefetching a lot of data at one time. Therefore, the second of these two options may be preferable for increasing network utilization.

Because all students will be different in how they use MANIC courses, the policy for how many objects to prefetch, and at what time, should not be a static one. Rather, the tutor should evaluate how the student is viewing the material and choose an appropriate policy.

Choosing the objects to display
In this architecture, the decisions on what objects to prefetch and what objects to display are both made at the proxy. The advantage of this is that there is no network delay to retrieve the decision. However, the proxy will have to be more sophisticated in order to be able to perform the decision making. The proxy will need to maintain the student model and the intelligent decision making ability. However, the student model will still need to be stored centrally at the server, so there will be some duplication. Furthermore, if quiz decisions and topic suggestions are done at the server, we will need to ensure consistency between the student models in both locations. A possibility, then, is to have all the intelligent decisions made at the proxy. But as the proxy is running on the client's machine, which is most likely slower than the server, the time required to make these decisions may be too large. However, this may not be a bottleneck, as the time required to make these student modeling decisions is relatively small. Furthermore, if the proxy is being used for multiple students, the processing may be drastically slowed by having to compute for all students. However, in the server based architecture, the server is serving multiple clients, and it may also not be able to handle all the necessary processing.

When *not* to prefetch
As we have previously discussed, prefetching more data than will be used may saturate the network with more traffic than it can handle. Even with this new architecture, which has a higher probability that some of the prefetched material will be used, we will see this problem.

There are a few problems we face in the new architecture. The first is that the proxy may not be able to predict where the student will go next, and download objects that do not correspond to the student's destination. In this case, none of the objects downloaded will be of any use. And by the time the student is ready to see the subtopics described by those objects, they may have been flushed from the cache.

Another problem we face in the new architecture is downloading too many objects to the proxy, even though the proxy has accurately predicted how the student will proceed. This would occur if the proxy is not confident on a small enough set from which the next slide could be constructed. In this case, many objects covering the next set of subtopics would need to be prefetched to the client, with only a few of them actually being used.

The proposed solution to this problem is similar to the solutions discussed in the server-based architecture. We can use confidence measures at the proxy to determine how many objects can be prefetched without causing problems. If the proxy can "accurately" predict which objects to prefetch, and this number is under the maximum allowed, they can be prefetched. Otherwise, only some can be sent and the rest must be retrieved in real time.

5.5.4 Hybrid Architecture

This architecture is similar to the proxy-based architecture, except that the decisions about what objects to prefetch and what objects to display are made at the server. The advantage of this is there is no need to duplicate the student model at two parts of the system. Also, all the decision making will be done in one location, by one process, running at the server. However, there will be a network access for every page, and thus a delay, by contacting the server for the information.

5.5.5 A comparison of the architectures

Each of the architectures has their advantages and disadvantages. With architectures B and C, we can avoid the staleness problem, since the cache does not contain any whole pages. Furthermore, assuming infinite cache space, if a student is linearly progressing through a topic, and the next content objects are fetched, they will still be in the cache even if the student has studied other material before those objects are needed.

However, even with architecture A, this is not a severe problem. At any point when the student is making a large jump between course content, most likely what he will see will need to be fetched from the server. The reason for this is it is unlikely that the student model is capable of determining what jump the student will take. And even if he goes back to a topic he has seen recently, fetching from the server, and thus causing a delay, is reasonable behavior.

Another advantage of the proxy-based and the hybrid architectures is the ability to more easily share the proxy's cache. With the server-based architecture, since whole pages are dynamically constructed within the student model server, those pages cannot be shared. However, with the other two architectures, the content objects that are in the cache can be used for all students, and thus can be shared.

One advantage of the server-based architecture is that no proxy is necessary. A student can run a MANIC course without using the proxy at all. In the other two architectures, the proxy is mandatory for using the course.

Overall, it appears that server-based architecture, even with the possible staleness problem, is the best solution. However, more research into this question will be done to determine the appropriate architecture for MANIC.

6 Proposed validation

In this section, we discuss the intended validation of the ideas presented in this proposal. We must validate that our student modeling efforts increase the learning of students. We must also validate that the prefetching mechanism we have described will reduce the delays seen by users. Finally, we must validate that the techniques we have developed are reusable, i.e. they can be used with another course.

6.1 Validating the student model

We will perform both formative and summative evaluations of the MANIC system. During the formative evaluations, we will be able to test the individual aspects of the intelligence as they are added to the system.

In the summative evaluation, we will test the system both with the intelligent features and without. Without intelligent features, the system will use default course content and a linear path through the course material (as it was in the initial lectures). Also, the quizzes will be hard coded, and thus will not be individualized for any student.

We will have two groups of students, one using MANIC with intelligence and one using it without. Each group will be given a pretest to determine their knowledge of the domain material. They will then use the system for the desired period of time. Afterwards, both groups will be given a post test to determine how much knowledge was gained.

In each evaluation, we will also examine subjective measures, such as how pleasurable the learning experience was and how much (or little) the intelligent features helped. If students do not enjoy working with a MANIC course, then are efforts are for naught.

We have done a preliminary study on the first implementation of MANIC. During the Fall of 1996, 15 students used the MANIC system as part of a one credit course on UNIX network programming and Java. The version of MANIC at that time did not include adaptive quizzing, but did include some prefetching techniques. Our main goal was to determine students' likes and dislikes about the software, and to use this information to change the system.

Eight out of nine students expressed very positive feedback about MANIC. These eight students indicated that they enjoyed the on-line course, with the ability to study at their convenience. Six students liked the linear nature of the course, but the other three would have preferred to have been able to choose what material to see next. Also, seven students said they would have taken advantage of on-line quizzes.

However, not everything was perfect. Students complained about the screen layout, indicating that we need to redesign with their comments in mind. Also, since the system is not stable, many students had problems logging in, due to the instability of the servers and the proxy, which caused a great deal of frustration.

6.2 Validating the prefetching

To validate the prefetching mechanism, we will rely mostly on simulations. The reason for this is the proxy needs to be running on the client's machine to get accurate timing data. However, the proxy has been written for a local machine. Furthermore, by using simulations, we can study various prefetching strategies to compare their performance very easily, using the same student data for each study.

Since students will be using the MANIC system for the validation of the student model, we will have log information from those students. We can use this to determine how accurate the prefetching mechanism is. The accuracy of the prefetching will be determined by how many prefetched documents have actually been accessed. The only way in which to do this is to record cache hits. Currently, there is no way to determine if a document is in the cache because it was prefetched or because it was downloaded in real-time. In the future we hope to add distinguishing features that would allow us to know the reason why a document was in the cache. This would enable us to better isolate prefetching hits versus other cache hits.

Additionally, we will investigate the hypothesized advantage of the intelligent prefetching method used in MANIC, compared to a non-intelligent prefetching mechanism. To determine this, we will run the simulations with both an intelligent and a non-intelligent prefetching algorithm. We hypothesize that the intelligent prefetching algorithm will result in more cache hits than the non-intelligent version.

Furthermore, we must study the correct parameters for slowing down or halting the prefetching. To do this, we must add timing delays to the simulations to see if the student model server will ever slow down the prefetching. While doing these simulations, we can investigate the correct parameters needed to make the prefetching as useful as possible. Specifically, we can investigate how the confidence measures should be updated, based on the timing information.

We must also study our updating mechanism for how many documents are prefetched at once. We can examine this by determining how many prefetched documents were evicted from the cache but were then requested in real time. We then must investigate how this policy affects the number of cache hits and the latency time experienced at the client side. We must also investigate how this policy affects how stale documents are.

6.3 Reusability

To validate the reusability of the MANIC software, we will develop another course to be taught via the WWW. We will convert the audio and the slides to be used within the MANIC framework. We will also develop the additional topics and course material, including quiz material, to make the course a full MANIC course.

7 Timeline

The following is a timeline for the work presented in this proposal. The dates given are when the work will be completed.

- October 1997 - Build the domain net for a small course (7-10 topics)
- October 1997 - Build an initial student model
- January 1998 - Implement dynamic topic selection
- January 1998 - Implement dynamic page content generation
- February 1998 - Initial study with small course
- Februay 1998 - Build quiz database
- March 1998 - Implement dynamic quiz generation
- April 1998 - Refine student model to incorporate quizzes
- May 1998 - Start work on another, larger course
- Summer 1998 - Write domain for larger course
- Summer 1998 - Further refine student model
- Fall 1998 - Offer larger course for evaluation by students
- Fall 1998 - Implement prefetching strategies
- Fall 1998 - Investigate the best architecture for prefetching

- Winter 1998 - Use results of study to improve software
- Fall/Winter 1998/1999 - Develop full 3-credit course
- Spring 1999 - Offer full course to students
- Spring/Summer 1999 - Write and defend thesis

8 Conclusions

The work presented in this proposal involves research into three main areas:

- Intelligent tutoring - We plan on demonstrating how existing video taped courses can be converted into on-line courses. Furthermore, we plan on showing that intelligence, in the form of topic suggestion, page content generation, and dynamic quizzing are beneficial to students using this on-line system.
- Prefetching - We plan on demonstrating that prefetching can reduce the delays seen by the students using MANIC. Additionally, we plan on showing that "intelligent" prefetching, based on the student's model, will further reduce these delays.
- Reusability - We plan on demonstrating that the technology used in MANIC is not restricted to one domain. Rather, any existing video-taped course with overhead slides can be converted to be used within MANIC. However, some changes to the course material will be required.

In the future, we plan on using more general population models to aid in the decision making process in MANIC. For example, general population models could be used to alter the instruction of a MANIC course. If all students are having problems with some of the course content, the tutor would be able to prepare for this with future students. With current intelligent tutoring systems, this updating of teaching styles must be done off-line.

Additionally, we will investigate the tradeoffs of using a shared proxy versus using one proxy for each student. Prefetching decisions and cache replacement policies will be affected by how many users are sharing the cache space. Furthermore, this decision may affect the choice of architecture, as sharing the cache is easier in some architectures but harder in others.

References

[1] J. Anderson and B. Reiser. The LISP Tutor. *Byte*, 10(4):159–175, 1985.

[2] J. Beck, M. Stern, and E. Haugsjaa. Applications of ai in education. *ACM Crossroads*, 3(1):http://www.acm.org/crossroads/xrds3–1/aied.html, Fall 1996.

[3] J. Beck, M. Stern, and B.P. Woolf. Cooperative Student Models. In *Proceedings of AI-ED97*, 1997.

[4] T. Berners Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol - HTTP/1.0. *Network Working Group, RFC 1945*, May 1996.

[5] A. Bestavros. Using Speculation to Reduce Server Load and Service Time on the WWW. Technical Report 95-006, Boston University, 1995.

[6] B. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13:3–16, 1984.

[7] C. Bouras, D. Fotakis, V. Kapoulas, S. Kontagiannis, K. Kyriakou, P. Lampas, P. Spirakis, and A. Tatakis. An interactive cooperative teleworking environment - telemethea. In *ED-TELECOM*, pages 37–42, 1996.

[8] C. Bowman, P. Danzig, D. Hardy, U. Manber, M. Schwartz, and D. Wessels. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Univeristy of Colorado - Boulder, 1995.

[9] C. Boyle and A. Encarnacion. Metadoc: an Adaptive Hypertext Reading System. In *In User Models and User Adapted Interaction*, volume 4(1), pages 1–19.

[10] P. Brusilovsky. Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 6:87–129, 1996.

[11] P. Brusilovsky and L. Pesin. Isis-tutor: An Intelligent Learning Environment for Cds/Isis users. In *Proceedings of CLCE'94*, 1994.

[12] P. Brusilovsky and E. Schwarz. User as Student: Towards an Adaptive Interface for Advanced Web-Based Applications. In A. Jameson, C. Paris, and C. Tasso, editors, *UserModeling: Proceedings of the Sixth International Conference, UM97*, pages 177–188. Vienna, New York: Sprinter Wien New York, 1997.

[13] P. Brusilovsky, E. Schwarz, and G. Weber. A Tool for Developing Adaptive Electronic Textbooks on WWW. In *Proceedings of WebNet-96*, 1996.

[14] P. Brusilovsky, E. Schwarz, and G. Weber. ELM-ART: An Intelligent Tutoring System on World Wide Web. In *Intelligent Tutoring Systems*, pages 261–269, 1996.

[15] B. Carr and I. Goldstein. Overlays: a Theory of Modeling for Computer-aided Instruction. AI Lab Memo 406, MIT, 1977.

[16] T. Chen and J. Baer. A performance study of software and hardware data prefetching schemes. In *21st Annual International Symposium on Computer Architecture*, pages 223–232, 1994.

[17] Mark Crovella and Paul Barford. The Network Effects of Prefetching. Technical Report 97-002, Boston University, 1997.

[18] C. R. Cunha and C. F. B. Jaccoud. Determining WWW User's Next Access and Its Application to Pre-fetching. Technical Report 97-004, Boston University, 1997.

[19] D.W. Edwards and L. Hardman. *Lost in hyperspace: Cognitive Mapping and Navigation in a Hypertext Environment.* Intellect Books, Oxford, 1989.

[20] C. Eliot and B.P. Woolf. Multiple Agents Acting in Parallel within an Intelligent Real-time Tutor. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-96*, 1996.

[21] A. Ellis and A. Debreceny. Electronic classroom: Features, users and evaluation studies. In *Educational Media and Hypermedia*, pages 191–196, 1994.

[22] S. Glassman. A caching relay for the world-wide web. In *Proceedings of the First International World Wide Web Conference*, Amsterdam, 1994. Elsevier.

[23] P. Gouzouasis. Video conferencing with preschool children: Mass communications media in music instruction. In *Educational Media and Hypermedia*, pages 229–234, 1994.

[24] G. Gundavaram. *CGI Programming on the World Wide Web.* O'Reilly & Associates, Inc., 1996.

[25] T. Joerding. User Modeling for Electronic Catalogs and Shopping Malls in the World Wide Web. In *Proceedings of the workshop "Adaptive Systems and User Modeling on the World Wide Web", Sixth International Conference on User Modeling*, June 2, 1997.

[26] C. Kaplan, J. Fenwick, and J. Chen. Adaptive Hypertext Navigation Based on User Goals and Context. In *User Models and User Adapted Interaction*, volume 3(2).

[27] J.F Kurose, H.I. Lee, J. Padhye, J. Steinberg, and M. Stern. MANIC: Multimedia Asynchronous Networked Individualized Courseware. Technical Report 96-75, University of Massachusetts, 1996.

[28] S. Lajoie and A. Lesgold. Apprenticeship training in the workplace: Computer-coached practice environment as a new form of apprenticeship. In Farr and Psotka, editors, *Intelligent Instruction by Computer*, pages 15–36. Taylor and Francis, 1992.

[29] S.P. Lajoie. Computer environments as cognitive tools for enhancing learning. In S.P. Lajoie and S.J. Derry, editors, *Computers as Cognitive Tools*, pages 261–288, NJ, 1993. Lawrence Erlbaum Associates.

[30] H. Lei and D. Duchamp. An Analytical Approach to File Prefetching. In *USENIX Annual Techincal Conference*, Anaheim, CA, January 1997.

[31] F. Lin, R. Danielson, and S. Herrgott. Adaptive interaction through www. In *Educational Telecommunications*, pages 173–178, 1996.

[32] A. Luotonen and K. Altis. World-Wide Web Proxies. In *Proceedings of the First International World Wide Web Conference*, Amsterdam, 1994. Elsevier.

[33] R. Mason. Synchronous versus asynchronous technologies for distance education. In *ED-TELECOM 96*, pages 343–346, 1996.

[34] K. Nakabayashi, M. Maruyama, Y. Koike, Y. Fukuhara, and Y. Nakamura. An Intelligent Tutoring System on the WWW Supporting Interactive Simulation Environment with a Multimedia Viewer Control Mechanism. In *Proceedings of Web-Net 96*, 1996.

[35] K. Nakabayashi, M. Maruyama, Y. Koike, H. Touhei, and Y. Fukuhara. Architecture of an Intelligent Tutoring System on the WWW. In *Proceedings of AI-ED97*, 1997.

[36] R. Nkambou and G. Gauthier. Use of www resources by an intelligent tutoring system. In *Educational Multimedia and Hypermedia*, pages 527–532, 1996.

[37] J. D. Padhye. *Can Intelligent Prefetching Make Web Browsing Faster?* Department of Computer Science, University of Massachusetts, 1996. Unpublished Synthesis Project Report.

[38] V. Padmanabhan and J. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. In *ACM SIGComm*, pages 22–36, 1996.

[39] J. Pitkow and M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *http://www.vuw.ac.nz/ mimi/www/www-caching/caching.html*, 1994.

[40] P. Purcell and G. Parr. International collaboration using digital media. In *ED-TELECOM 96*, pages 263–269, 1996.

[41] A. Rebelsky. Improving www-aided instruction: A report from experience. In *ED-TELECOM*, pages 276–281, 1996.

[42] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, Inc., New York, 1991.

[43] J. Sener. Incorporating asynchronous collaborative learning into an as engineering program for home-based learners: Challenges, strategies and tools. In *ED-TELECOM*, pages 314–319, 1996.

[44] F. Shipman, C. Marshall, and R. Furuta. Creating Educational Guided Paths over the World-Wide Web. In *Educational Telecommunications*, pages 326–331, 1996.

[45] V. Shute. Smart evaluation: Cognitive diagnosis, mastery learning and remediation. In *Proceedings of Artificial Intelligence in Education*, pages 123–130, 1995.

[46] O. Signore, R. Bartoli, and G. Fresta. Tailoring Web Pages to User's Needs. In *Proceedings of the workshop "Adaptive Systems and User Modeling on the World Wide Web", Sixth International Conference on User Modeling*, June 2, 1997.

[47] M. Stern, J. Steinberg, H.I. Lee, J. Padhye, and J. Kurose. MANIC: Multimedia Asynchronous Networked Individualized Courseware. In *Educational Media and Hypermedia*, 1997.

[48] M. Stern, B.P. Woolf, and J. F. Kurose. Intelligence on the Web? In *Artificial Intelligence in Education*, 1997.

[49] A. Stinner. Contextual settings, science stories, and large context problems: Towared a more humanistic science education. *Science Education*, 79(5):555–581, 1995.

[50] D. Suthers, A. Weiner, J. Connelly, and M. Paolucci. Belvedere: Engaging students in critical discussion of science and public policy issues. In *Artificial Intelligence in Education*, pages 266–272, 1995.

[51] A. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 1992.

[52] G. Weber and M. Sprecht. User Modeling and Adaptive Navigation Support in WWW-Based Tutoring Systems. In A. Jameson, C. Paris, and C. Tasso, editors, *User Modeling: Proceedings of the Sixth International Conference, UM97.* Vienna, New York: Sprinter Wien New York, 1997.