International Journal of Artificial Intelligence in Education (1999), 10, 183-197

Deploying Intelligent Tutors on the Web: An Architecture and an Example

Sherman R. Alpert, **Mark K. Singley** and **Peter G. Fairweather** *IBM T.J. Watson Research Center, POB 218, Yorktown Heights, NY USA, {alpert, ksingley, peterf}@watson.ibm.com*

Abstract. We describe the conversion of a standalone intelligent tutoring system (ITS) to one that operates on the World Wide Web. First, we focus on one product of this effort, a Web-enabled architecture as extension of a widely-used standalone ITS architecture. The advantages of the architecture we have chosen are discussed. We then describe the specific Web-based ITS that uses this architecture, highlighting features of the tutor for supporting and enhancing problem solving. Both the architecture and many features of the tutor that support students' problem solving activities should be generalizable to other ITSs.

Keywords. Intelligent tutoring systems, World Wide Web, intelligent tutoring system architectures, Java, user interface design, problem solving tutors, algebra tutors.

EVOLVING AN INTELLIGENT TUTORING SYSTEM TO THE WEB

Empirical studies have demonstrated that effective one-on-one tutoring is the most powerful instructional manipulation, with a 2-sigma beneficial effect compared with standard classroom instruction (Bloom, 1984). What would our educational system be like if everyone had his or her own private tutor? Logistically and financially, however, one-on-one tutoring for all students is impossible. So we're building intelligent tutoring systems (ITSs) in an attempt to bring the personal tutor experience to a broader audience. The World Wide Web offers an unprecedented opportunity to move closer to this goal.

The domain we've chosen to focus on is elementary high school algebra. This decision is due in part to algebra's importance in the math career of many students: algebra is the point at which they "hit the wall" and continue no further. To date, we have two works in progress: a symbolic equation solving tutor and a word problem tutor. In this paper we discuss the former, focussing in particular on the overall architecture of the Web-based version of this tutor, features that support students during problem solving, and the tutor's user interface design and rationale thereof. Our work in these three areas should have implications for problem solving tutors in other domains.

The AlgeBrain equation solving tutor provides an environment for practicing algebraic skills learned in a separate—typically classroom—instructional setting. We position the tutor as a pedagogical *partner* to the classroom instructor—we wish not to replace this role but to offer an environment that acts as an adjunct to the classroom. Few educational principles have emerged so clearly to investigators as the finding that the performance of cognitive skills is related to practice as a power function (Anderson, 1982; Newell and Rosenbloom, 1981). Effective practice has proven to be a reliable way to improve the performance of a cognitive skill. AlgeBrain offers practice with the benefit of a coach providing advice and remediation, keeping students on track toward solution while minimizing their level of frustration. The tutor focuses specifically on the skills and knowledge involved in transforming and simplifying equations—that is, the selection of algebraic operators and associated operands—toward the goal of solving for a particular variable.

We began by building a standalone tutor—that is, a single executable running on a single machine. At a bird's-eye level of abstraction, our standalone equation tutor conformed to a well-established intelligent system architecture, comprising a tutorial component, a student model for

keeping track of student progress over time, a cognitive simulation of an expert problem solver, and the code underlying the user interface (see Figure 1; Burns & Capps, 1988).

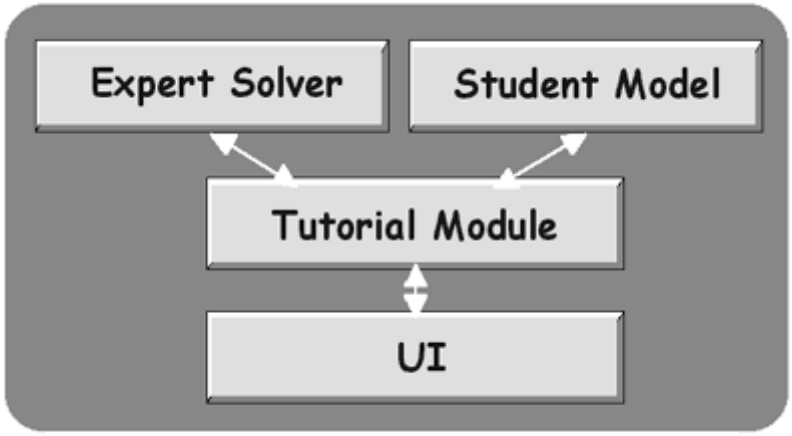

Figure 1. Standalone intelligent tutor architecture.

Having constructed a working tutor, we faced the commonly encountered problem of how to get it in front of real students. Initially we had intended to distribute the software to local school districts that expressed interest in using the tutor for their algebra students. But we realized there was now a way to have a greater impact by reaching a much larger audience: by building a Web-enabled version, users could have access to individual tutoring from a standard Web browser.

The rest of this paper is comprised of two main components. First, we describe the conversion of the standalone tutor—and more importantly the extension of a common standalone ITS architecture—to one capable of operating over the World Wide Web. Second, we discuss the resultant Web-enabled tutor itself, in particular focussing on features that support problem solving and ease of use. In both portions, many of the ideas we present are applicable to ITS construction in general.

A WEB-ENABLED ITS ARCHITECTURE

To deploy a Web-enabled ITS, there are a number of architectural paths from which we might have chosen. Here are just three examples:

1. Java-only solution: The tutor—that is, the entire program that interacts with the student—resides in a Java applet which is downloaded by visiting a specific URL and executes on the client (student's) machine. All tutorial behavior resides on the client;
2. HTML-CGI architecture: The user interacts with HTML entry forms in a Web browser; information entered by the user is sent to the Web server which forwards it to the CGI (Common Gateway Interface) program which then replies with new HTML pages (the CGI approach is described in greater detail later). All tutorial functionality resides on the server side (in the CGI program) but the user interacts with it using a standard Web browser,
3. Distributed client-server architecture: A downloadable Java applet contains the user interaction portion of the tutor and communicates directly with a server application using a socket connection or other inter-program communication mechanism—some of the tutorial behavior resides in the client, some in the server.

This list is by no means exhaustive (at the very least, there are a number of distributed implementation options, some involving heavyweight middleware componentry). Nonetheless, the list clearly demonstrates there are many decisions to be made when attempting to implement

a Web-enabled application. Further, each of these adds layers of complexity on top of the skills required to build a standalone tutor.

Three overarching goals guided our architectural decisions. One factor was the fact that we already had a working standalone system. As much as reasonable, we wanted to reuse the existing infrastructure and code while simultaneously deploying a viable and sensible Web-enabled solution. Further, we realized there were advantages to having portions of the tutor reside in a centralized location common to all users. At the same time, we kept the user in mind—we wanted the user interaction component of the tutor to be interactive, responsive, and engaging (we discuss these issues in greater detail below). We thus decided to rewrite the user interface portion of the tutor as a downloadable Java applet, making it available to anyone with a connected Web browser, and convert the rest of the standalone tutor to act as an application server that would communicate with the client applet. Very briefly, at the highest level, this involved the following steps:

- eliminate the user interface component from the standalone application,
- add server socket communication capabilities to the standalone tutor,
- redesign and reimplement the user interface in a Java applet executable in a standard Web browser, and also incorporating multithreaded socket communication capabilities,
- design a language for communication between application server and client, that is for client requests and application server responses,
- implement socket communication between the distributed components,
- and code an HTML page incorporating the Java client applet, to be retrieved by students using a browser anywhere on the World Wide Web.

The result is a "new" high level architecture for intelligent tutoring systems—now deployed on the Internet. As the diagram in Figure 2 portrays, our Web-based client-server tutor architecture is an adaptation and extension of the standalone infrastructure. In essence, this model distributes the components of a classical ITS architecture between the client and server while adding a communication channel between the separated elements. The implication of the fact that this distributed architecture is in a real sense an extension of a widely used standalone architecture is obvious—it offers a path for the Web-enablement of other existing ITSs.

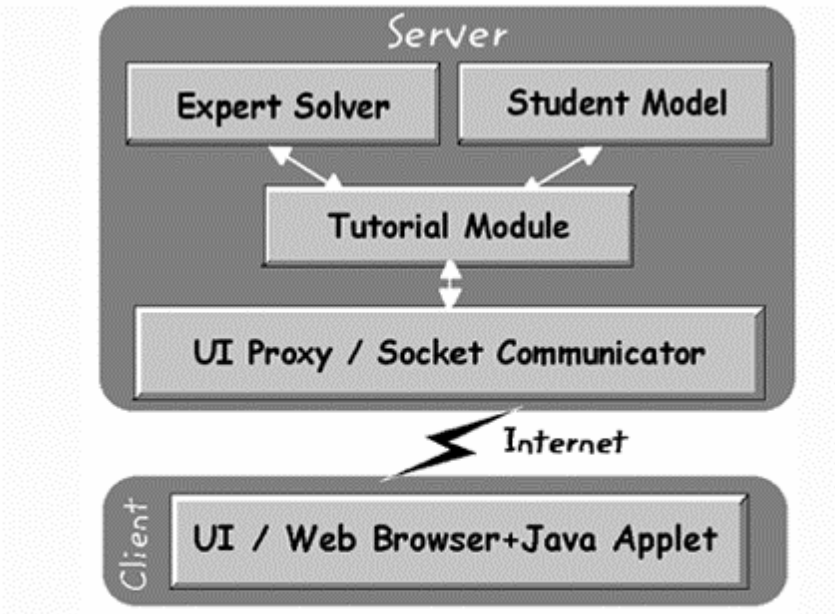

Figure 2. A "new" (Web-enabled) architecture for intelligent tutoring systems.

In the server application, we "plugged in" a new module to replace the user interface component, and the other tutor components had to be modified minimally or not at all. (A side benefit of this architecture was that it allowed us to test components of the system—such as new rules in the Expert Solver—by running the standalone tutor as we were simultaneously implementing the Web version.) The Tutorial Module communicates with a "UI" object just as in the standalone architecture, but now that object is a *surrogate* for the actual interface with which the user interacts. Rather than displaying information on the screen and obtaining input from the user directly, this UI proxy sends and receives information over a socket connection to a separate program on a client machine somewhere on the Internet. The real user interface now resides in that client application.

Figure 3 offers another view of the Web-enabled tutor's architecture. The server application in fact fulfills two distinct server roles: a generic HTTP server[1] and the tutor-specific application server containing the socket communicator, expert problem solver, student model, and tutorial components. A user points a Web browser to the appropriate URL (Web address; e.g. http://www.my.server/algebra.html) and the HTML document is served up by the HTTP server component. The HTML code includes tags specifying the Java applet that acts as the client portion of AlgeBrain. When the client applet is downloaded and executed by the Web browser, it establishes a socket connection with the AlgeBrain application server for all subsequent client-server communication. For each client connection, the server spawns a communication thread dedicated to that client.

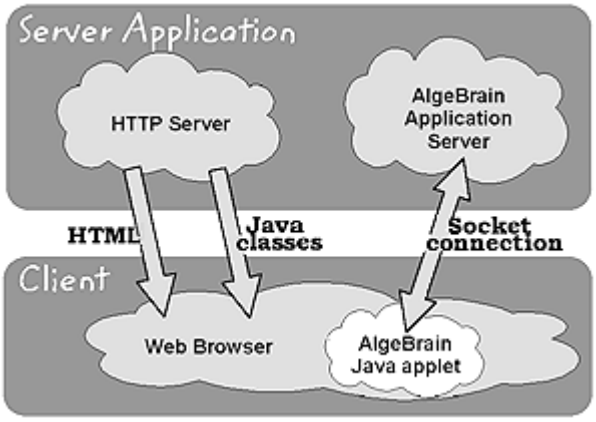

Figure 3. Alternative view of our Web-enabled tutoring architecture.

The AlgeBrain application server is independent of the HTTP server component—the former may be executed with or without the latter. So we may run the tutor's application server on a server machine that already supports a generic Web server for other purposes and users. Hence, because of its modular architecture, AlgeBrain may be executed in any of three runtime modes: standalone, client-server including the "built-in" HTTP component on the server side, or client-server in coordination with a separate Web/HTTP server. Even without the intrinsic HTTP server component, our architecture offers an unusual flexibility, allowing an ITS to be run standalone or over the Web.

Why the Web?

We've already mentioned the primary motivation for investing significant effort porting our standalone tutor to a Web-based implementation, namely the ability to reach greater numbers of students. If users can access a centralized tutor server from a standard Web browser, they need not own a copy of a standalone program, and the logistical problems of distributing software to

[1] We implemented only the subset of HTTP server functionality required for the AlgeBrain client, specifically the HTTP *GET* method for serving up files of various types, including HTML documents, Java classes, and media (e.g., image and audio) files.

individuals are eliminated. Carrying this notion further, students are no longer constrained with respect to where or when they can receive tutoring. They can interact with the tutor at school, home, public libraries, community centers, collaboratively at the homes of friends, anywhere they have access to the Internet. Hence, we move a step forward toward providing the tutor experience to a broader population. Further, students are no longer limited to tutorial interactions during school hours only, and those with Internet access from home may use the tutor whenever, as often, and for as long as they wish or need.

A Web-enabled tutor—at least one with an architecture involving components on the server side—yields another major advantage that is specific to ITSs: wherever you are, your student model is there. The student model for all users resides on the centralized server rather than on a single machine. Thus, though students may log on from anywhere, the tutor always knows each student's current knowledge level and position in the curriculum and can use this information to inform problem selection and help content. This feature provides further rationale for the architecture we've chosen: a possible alternative would be to implement all of the tutor's functionality in a Java applet that is downloaded from a server and runs standalone on the client machine. Having a student model database for all users on a central server argues against this architecture.

Similarly, having all applications—in our case, an application server program and downloadable client applet—reside in a single centralized location results in students always using the current version of the tutor. Bug fixes and tutor enhancements are immediately available to everyone.

There is also a technical rationale of somewhat lesser importance than the preceding. We wanted the client user interface to incorporate cartoon (in-place) animation. The GIF89a (Graphics Interchange Format, Version 89a) image format permits file-based definition of both static and animated images. Web browsers (and Java virtual machines that execute therein) "understand" the GIF89a format and automatically animate such images. Hence with a Web-based architecture we get animation at the interface "for free"—after creating and including animated-GIFs in the client UI, they will be animated with no additional application-level programming on our part.

Why a Java client?

Given the decision to deploy the tutor on the Internet, we could have implemented an HTML form-based client and a CGI application server containing the tutor application. CGI stands for Common Gateway Interface, a commonly used Web architecture. In the CGI approach, both a Web server (to handle standard HTTP requests) and a server application program run on the server. Each time the Web server receives information from a browser that is intended for a CGI server application, the server forwards these data to the CGI program; in many HTTP servers, the server restarts the CGI program each time it has data for that program. The CGI program responds to users by sending new HTML pages to the Web server which in turn transmits the HTML to the client browser. In all cases, there is no direct communication between the client and the CGI application; all information is routed through the Web server.

Several successful Web-based ITSs have been reported that use this approach (more on these in the next section). We felt that for our purposes, however, there are a number of constraints associated with such an architecture. First—and perhaps most importantly—on the client side we wanted a more highly interactive user experience than HTML alone could afford. We wished to incorporate interface features that would be impossible or highly impractical in HTML, such as intelligent selection of terms directly in equations, immediate feedback via multiple media for each problem solving step, and an interactive graphing component that responded immediately to user actions. To implement these and other UI capabilities, we needed code running on the client, interacting in an immediate fashion with the user independently from the application executing on the server.

Second, we wanted to have direct communication between the client and server code for efficiency and flexibility. We wanted a responsive system, with reasonable response times for user actions. A Java applet can establish a persistent socket connection with a persistently-

executing application server program and thereby offer quicker communication and server responsiveness. A dedicated socket connection between client and server also allows the tutor's to employ a server-push/client-pull communication mechanism. Often in a client-server architecture, client programs send a single request to the server and wait for a single server reply. This is always true in an HTML-CGI model. Instead, our client sends requests to the server without "blocking" or waiting for a reply; at the same time, an independent processing thread in the client program constantly listens for messages from the server. As they are received, this listener thread forwards those server messages to the central applet component for display in the appropriate portion of the user interface.

This communication model allowed us to have the server send multiple messages in response to a single client request. As a simple example, when a new problem is requested by the client, the server sends an equation for display in the equation blackboard and, separately, sends text to be shown in the advice blackboard indicating which variable to solve for. This follows from the standalone tutor where the notion of separate messages to the user interface component was a natural method of having the UI module display different information in different interface components. When the tutor was converted to a client-server implementation, we wanted to strictly minimize modifications to the tutor infrastructure now residing in the application server. Hence, we continued using the many-to-one reply-to-request model.

The server-push model allows for unsolicited messaging from server to client, permitting greater flexibility with regard to *when* the server can offer help or information to users interacting with the client. In particular, it makes possible mixed initiative communication (mixed between server and client and between tutor and student). Thus, for example, the server may offer hints without explicit student request. Although we have not exploited this feature in our Web-ported tutor, our architecture enables it as an option, either for ourselves in future work or other ITSs adopting the architecture. It also allows the server to forward synchronous messages sent by other online users in the context of a multi-user collaborative environment. We are exploring this type of student-to-student collaboration in the context of another (algebra word problem) tutor we have built and are porting to the Web.

The socket connection model also solves a problem faced by servers—including CGI server programs—that support multiple clients. Eliot (1997) discusses this problem of having to associate a request received by the server with a particular client/user. In the architecture used by AlgeBrain, once a particular Java client establishes a socket connection to the server, it is given its own, unique, socket for all subsequent communication—that is, as far as the server is concerned, each client communicates over a different socket. Further, the AlgeBrain application server creates a dedicated processing thread for each client (socket). This thread listens for messages only on that client's socket and in turn transmits responses on that socket alone. Thus, the server has no problem differentiating input from different users.

Other Web-enabled ITS Architectures

A number of previous papers have discussed Web-enabled intelligent tutoring systems. We highlight several here—which we believe to be representative of proposed Web-based ITS architectures—and discuss why we did not choose the identical architecture as those that preceded AlgeBrain.

Several papers have reported Web-based ITSs using an HTML-CGI architecture (e.g., Brusilovsky *et al*., 1996; Brusilovsky *et al*., 1997; Okazaki *et al*., 1997; Ritter, 1997). As enumerated above, we felt this architectural model has a number of constraining features, particularly the lack of immediate interactivity for certain user actions. In Ritter's paper on the PAT tutor he also discusses the problematic nature of the lack of immediate feedback coupled to user input in an HTML client. He suggests a more satisfactory solution would involve rewriting all of the tutor components, other than the student model server, in Java. We specifically wished to avoid, as much as possible, the complete redesign of the infrastructure of the standalone tutor and the recoding of the central components in another application. In particular, our architecture retains the majority of the standalone tutor's architecture and code and merely extends the

architecture in a fashion that gives rise to Web-based deployment. Ritter *et al*. (1998) report on a later Java version of PAT but do not elucidate its client-server architecture.

In CALAT, Nakabayashi *et al*. (1997) go a step further than the HTML-CGI model. They attempt to provide for greater interactivity by employing an animated simulation component that executes on the client in a proprietary viewer or using a plug-in inside a Web browser. Nonetheless, all communication is still routed through the Web server using standard HTTP requests. To invoke the browser plug-in on the client side, responses sent from the application server program possess a specific MIME type associated with the plug-in. There are a number of problems with a browser plug-in approach. First, plug-ins are platform-specific—the platform-independence of a straight HTML or Java client are forfeited. Further, this approach gives up the important advantage cited earlier of not having to distribute code to users. With a browser plug-in or proprietary view, users must obtain and install code on their client machines before they can use the tutor, and further still for a plug-in, configure their Web browsers for the plug-in's MIME type. This problem is iterative when plug-in updates are required for bug fixes or enhancements.

ADIS is another Java-based Web-enabled ITS described by Warendorf and Tan (1997). In ADIS, the entire tutor resides in a Java applet that is downloaded and executed on the client machine. Like ADIS, we wanted AlgeBrain to be highly interactive and so required code running on the client. Unlike ADIS, we wished to have (at least) the student model reside on the server so the same student can access the tutor at different times from different locations and have his student model be up to date, reflecting all previous tutorial interactions and his current knowledge level.

Vassileva (1997) asserts a CGI solution is too slow and describes an ITS with a Java client. Again, one of the features of our particular client-server architecture is maintaining the student model for all users in a centralized location (on the server) so the tutor always knows their current knowledge state. Vassileva solved this problem in another, interesting, fashion. When a user downloads the Java tutor application, a local copy of her student model is also copied to the local machine. As the user interacts with the tutor, this local copy of the student model is updated by the Java-based tutor. When the users finishes a session, the local copy is uploaded to the server for persistent storage. Of course, this is still problematic in that if the network link is lost before the student "finishes," the most recent student work and student model updates are lost.

One paper (Eliot, 1997) has reported an ITS system that involves a direct network connection between a Java client and application server. However, the system's architecture is not described beyond merely stating the fact that a program-to-program connection is established.

ALGEBRAIN: DESIGN TO SUPPORT PROBLEM SOLVING ACTIVITIES

We turn our focus now to the specific ITS we converted from a standalone application to one that is Web-enabled, giving rise to the architecture we've described. We discuss the design of, and design rationale for, elements of the tutor. AlgeBrain's design is informed by the sort of principled design for cognitive tutors delineated by Koedinger *et al*. (1997), although AlgeBrain includes elements, supported by principled rationale of their own, beyond those incorporated in their ITS. Our design rationale is expressed in terms of pedagogical implications, ease of use, and how tutor elements support the problem solving activities of users. Along the way, we also describe the use of the Expert Solver's rule system in supporting the Tutorial Module for capabilities other than simply solving equations.

We wanted the AlgeBrain client to have an engaging interface that is visually interesting and incorporates animation and sound effects. Although AlgeBrain tackles the serious task of supporting users' cognitive activities, we do not wish to ignore their affective and motivational needs. As Hoogeveen (1995) asserts, multimedia-enhanced interfaces may improve learning as well as "the entertainment value of systems (i.e., more fun)." Further, the media we use incorporates a modicum of humor. If we want students to spend time using ITSs, making them

more enjoyable to use is a worthwhile goal—time on task is an important ingredient in any learning situation particularly one wherein learning by doing is so important.

Figure 4 shows the Java client running in a Netscape browser. Its UI is an "interactive classroom wall," if you will, with chalkboards that display information dynamically and pieces of "paper" stuck to a bulletin board. These paper notes are actually interactive buttons; when clicked, they appear "scrunched" and the user hears a crumpling paper sound effect. As the user interacts with the tutor, a jazzy tune plays in the background. Animation and other audio effects are central to the user's experience. (Users can turn audio off using the "ear" toggle button in the lower right corner; when sound is off, the ear icon is drawn with an earplug as in Figure 7.)

The goal in the AlgeBrain equation tutor is, of course, to solve an equation for a particular variable. Nonetheless, AlgeBrain is not concerned with simply grading a final answer right or wrong. Instead, AlgeBrain tutors the *process* of solving equations: we're interesting in supporting students' problem solving activities and enhancing their problem solving skills. Though available mathematical software such as *Mathematica* can provide the solution to an equation, it cannot inform us of the process toward that solution. So AlgeBrain incorporates a rule-based expert system that simulates an expert equation solver *and* can articulate the operators and operands for each step in the process. This cognitive model of an ideal student problem solver resides in the Expert Solver component of the server application, solving equations along with the user. The Expert Solver is consulted by the Tutorial Module to (1) provide hints to students, as well as (2) determine the correctness of their individual problem solving steps and offer remedial feedback. The latter is a technique that Anderson has labeled *model tracing*: developing a cognitive model of an expert problem solver and then using that model to interpret student performance (Anderson *et al.*, 1992). Including a cognitive model of an ideal student problem solver with the same skills and knowledge we expect students to use, and applying the technique of model tracing, using the cognitive model to intelligently track and support student progress during problem solving, are two primary ingredients in Koedinger *et al.*'s (1997) notion of principled design for cognitive tutors. Currently, AlgeBrain's Expert Solver can handle linear and quadratic equations, with or without rational polynomial terms.

At each step toward problem solution, users must propose a legal algebraic operation and associated operands given the current problem state. The interaction involves mouse-selection of terms directly *in* the equation and pushing one of the "paper" buttons that represent algebraic operations. Rather than having users type terms in a textual entry field or pick them from a multiple choice list, terms are selected by clicking and swiping in the equation blackboard. We preferred this approach because it eliminates ambiguity regarding term selection ("I mean the `20` on the right side of the equation, not the `20` in the denominator on the left side"; see Figure 4). Further, term selection is intelligent in that it is based on equation structure, not simply contiguous text as in a text editor. For example, when a user clicks on the "`2`" character in the expression `20x(x + 1)`, the atomic term `20` is selected without having to swipe through the "`0`". The selection process in the client applet knows that the "`2`" and the "`0`" are not independent characters but rather parts of a single term. If a student presses the mouse button on the "`2`" and swipes to the right, successive terms `20`, `x`, and `(x + 1)` are selected as soon as the cursor enters each term's "borders." A term can be selected by simply clicking on its primary operator. For example, `(x + 1)` can be chosen by clicking on the "`+`" operator. Parenthesized terms can be selected by clicking on either of the parentheses. An entire fraction can be selected via a single click on the divisor line. And so on. The rationale for all this is simply intelligent selection of terms minimizes user error—we're focussed on tutoring algebra skills and do not want to penalize users for poor selection of equation terms. This ease-of-use feature helps to minimize frustration and enhance the overall user experience in interacting with the tutor.

At each step of the solution process, users can ask for hints by clicking on the animated dancing AlgeBrain agent. The tutor responds with two types of support, as portrayed in Figure 4: (1) generalized "here's what I'm expecting you to do at this point" help text in the lower portion of the advice blackboard on the right of the UI, and above that (2) a hint specific to the current state of the problem regarding how to proceed, that is, regarding a valid operator and

associated operands given the current equation state. Repeated hint requests result in increasingly specific information; this is a technique other ITSs, such as the PAT tutor (Koedinger *et al*., 1997), also employ. For example, a first hint might be a general suggestion such as "try to isolate the variable you are solving for." If the student requires more help, the tutor follows up with something more specific such as, say, "try subtracting something from both sides." Eventually, the last hint helps the student who is "stuck" to move on (with appropriate annotation in the student model). For example, "subtract 4 from both sides."

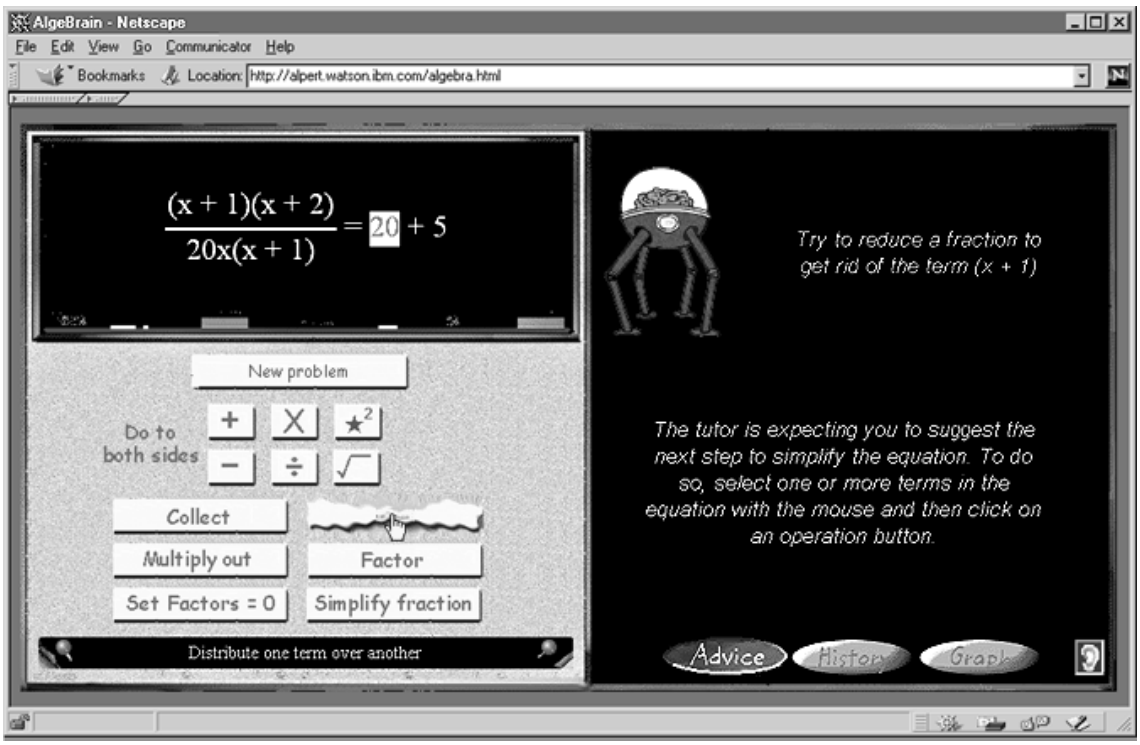

Figure 4. The AlgeBrain client user interface. The term '20' has been selected. The user is clicking the 'Distribute' operation button to the right of the 'Collect' button. The AlgeBrain agent dances to the background music.

The text for hints is template-driven so as to incorporate the actual operands (e.g., "4" in the previous example) that occur in the current equation. In an interesting application of expert system technology, the hints are provided by the rules themselves—more precisely, by the rule instantiations created by the rule interpreter during runtime. In addition to production (if/then) rule information per se, each rule object contains a list of textual templates for hints. Each successive entry in this list represents a more specific hint. Variables in these hint templates reference rule variables found in the production rule portion so that the actual numbers and terms appearing in the current equation are used in the hint text displayed to users. Thus there is no special case code separate from and acting upon rules to produce hints. As new rules are added or existing rules are enhanced with new hints, no application code need change, making for an easily extensible system.

Each time the student proposes an operation, the tutor provides immediate feedback, and does so in four ways. Suggest a correct operation and operands and AlgeBrain animatedly applauds your effort (Figure 5, left). For wrong responses, the AlgeBrain agent scratches its head in "confusion" (Figure 5, right). Student problem solving attempts elicit audio responses as well. Correct proposals are greeted with an enthusiastic "Right!" or "That's correct!" that sound like a television game-show host speaking in an echo chamber. Incorrect moves result in cartoon "sproing" or clown-horn sound effects. Feedback is also provided textually in the advice blackboard and, in the case of a correct operation proposal, by the resultant new equation written to the equation blackboard. Finally, when the solution is reached, a group of "fans" cheer "Wow!" via audio.

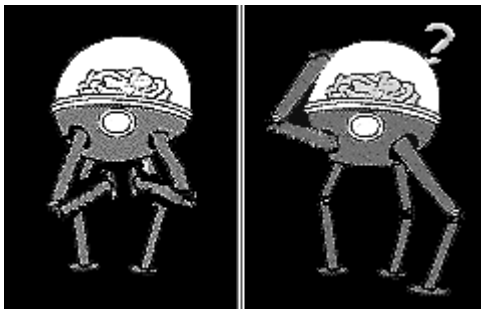

Figure 5. AlgeBrain's animated feedback

The feedback is thus tendered in direct response to user actions—an obvious pedagogical choice and one suggested by many for computational tutors (e.g., Anderson *et al.*, 1984). Nonetheless, the tutor does not immediately provide the correct move to the student. McKendree *et al.* (1992) have determined empirically that users repeatedly express the desire to attempt to repair errors before being told how to proceed. AlgeBrain's feedback fulfills both desiderata. Right/wrong feedback is provided immediately, but students can either retry alternative moves of their own design or choose to ask the tutor for advice. Even in the latter case, because the tutor's hints are provided at first in general terms, users are still able (to a lesser extent with each increasingly specific hint) to infer the correct operation and thus repair errors on their own.

To determine the validity of student operator/operand proposals, the client applet consults the application server. On the server, the "correct or incorrect" decision is made by the Tutorial Module in cooperation with the Expert Solver. Note that for any single equation there may be more than one valid operation that brings the student closer to solution. At each state in the problem solving process, the Expert Solver infers *all* valid operation/operand pairings—that is, its expert system finds all instantiations for every rule whose condition tests match the current problem state. At this point, the expert system suspends processing. Student proposals are compared to this *collection* of valid moves—the student's proposal is compared to the operation and operands represented in each rule instantiation in the expert system's conflict set. If the student proposal matches any instantiation, the proposal is considered valid, and the server sends the "correct" message to the client applet. Hence, AlgeBrain considers all equally beneficial operations to be correct. The expert system then continues processing, firing the rule associated with the matching instantiation. In essence, then, in the case of a valid student proposal the student's selection acts as the expert system's conflict resolution mechanism.

If no match for the student proposal is found among the conflict set's instantiations, the proposal is considered incorrect and an error response is transmitted to the client. In either case, correct or incorrect proposal, the server's response results in the client applet displaying the appropriate feedback in its user interface. Note that for hinting purposes, the tutor wants to use the "best" move represented in the conflict set. Thus for determining the hint text, the expert system performs its normal conflict resolution process to pick the rule instantiation representing the best proposal. The rule associated with the "winning" instantiation is the consulted for the hint text, as described above. The Expert Solver then resets the conflict set back to its prior state—containing the perhaps multiple valid instantiations representing legitimate operations—for purposes of matching subsequent student proposals for the current problem solving step. Hence, although the tutor's hints suggest the Expert Solver's best move, the tutor does not force users to walk a particular path through the problem, accepting instead alternate routes to the solution.

As an example, in Figure 4 the tutor is recommending simplifying the fraction on the left side of the equation

$$\frac{(x + 1)(x + 2)}{20x(x + 1)} = 20 + 5$$

by eliminating the `(x + 1)` terms in the numerator and denominator. This is not the only valid operation at this point. An equally advantageous move would be to collect like terms `20` and `5`,

or to multiply both sides of the equation by `20x`. A student can propose either of these (or other legal) operations and the tutor will accept the proposal as valid. Successful operations result in a new equation in the equation blackboard; the tutor then infers all legal operations to proceed from *that* equation.

In addition to rules that know how to legitimately move closer to solution from a particular problem state, the server's expert system also includes *buggy* or *mal-rules*. Such rules model well-known or common student errors. The implication for the tutor's interaction with students is this: when a student's proposal matches a buggy rule, the tutor is in the position to offer focussed remediation specific to the error, rather than a simple generic "Sorry, that's wrong." A buggy rule has a left hand side (condition part) that tests for a specific problem state *along with* a specific type of student proposal that we know to be a common error for algebra students given that problem state. Suppose, for example, we have the equation `3(x + 5) = 21` and the student proposes "Subtract 3 from both sides" (by selecting the `3` and clicking the "–" operation button). The Expert Solver includes a buggy rule for this operation choice for an equation whose pattern matches the given equation. This rule contains an appropriate remedial response which is displayed in the advice blackboard: "That won't help simplify the equation. If you want to remove the number from the left side, don't subtract. Divide."

Buggy rules permit the tutor to offer remediation at a more strategic level as well. Specifically, the tutor also uses buggy rules to recognize and disallow clearly sub-optimal operation choices. The tutor's rationale can be understood in the context of a means-ends analysis approach to problem solution. The tutor *does* allow *any* valid operation that brings the student closer to the solution state, but also tries to disallow operations that clearly do not move in this direction. For example, for the equation in Figure 4, the student might suggest distributing `(x + 1)` over `(x + 2)`. Technically, this is a valid algebraic operation, but strategically it's a poor choice. It is legal but counterproductive, leading to a more complex equation to solve rather than simplifying the current one. As with common *errors*, the tutor's Expert Solver includes a buggy rule that recognizes when this *sub-optimal* choice occurs. Based on that rule, the tutor's response to this proposal is: "Don't distribute; the equation can be simplified in its current form." Thus while the tutor allows multiple paths toward solution, it nonetheless attempts to steer students from unequivocally sub-optimal routes.

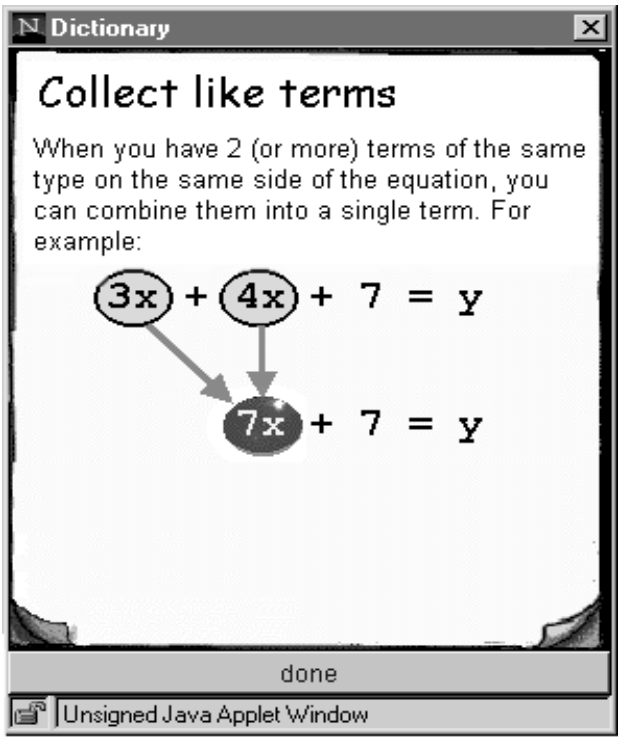

Figure 6. One frame from an animated example in the "collect like terms" dictionary entry .

While the heart of the tutor's domain involves algebraic operations, students may forget the meaning of particular operations or their usage in simplifying equations. Suppose, for example, a tutor hint suggests "collect like terms" but the user doesn't recall how that operation can be applied to an equation. To support users in such a scenario, AlgeBrain incorporates an animated *Just-In-Time Dictionary*. Operation buttons on the corkboard are "self-describing" (Alpert, 1991): students can click the *right* mouse button on any operation button to retrieve the dictionary entry describing the corresponding algebraic operation. A right-button click on, for example, the "Collect" button opens the "collect like terms" dictionary definition (see Figure 6). Users thus may request dictionary explanations *as* they become appropriate for the problem at hand, rather than viewing them in an isolated setting. It has been shown that learners provided skill information in decontextualized milieus have little idea how or when to apply such information during problem solving (Brown *et al.*, 1988). AlgeBrain's dictionary explanations are provided instead in the situated context of actual problem solving

The resulting dictionary window includes declarative information in the form of explanatory text, plus a set of concrete examples of the usage of the algebraic operation in several equations. Hence, the dictionary offers a basic learning by example facility. Ritter *at al*. (1998) have suggested that an ITS should support three modes of instruction: learning by doing, learning from declarative information, and learning from examples. AlgeBrain's dictionary adds the second and third modes to the tutor's learning-by-doing problem solving environment. In AlgeBrain's dictionary, examples are animated to portray a temporal sequence of events, demonstrating how the application of a particular algebraic operation involves several ordered steps, such as the selection of the appropriate operand(s) in the equation, the calculation of intermediate results, and the rewriting of a new resultant equation. The dictionary thus offers animated examples with *how to* information.

As an example, Figure 6 shows one slide of an animation that includes frames displaying (1) an equation, (2) the like terms being selected, (3) the resultant single term, calculated by combining (adding, subtracting) the like terms, shown highlighted, (4) the rewritten equation still highlighting the new term as in the Figure, (5) the derived equation. Several equation examples are included in this dictionary entry. Further, in case a user "misses" any portion of any demo, or simply wishes to review them, the demonstrations play repeatedly until the user closes the dictionary window. The dictionary's animated reification of the accompanying static explanatory text provides for process knowledge in dictionary entries as well as declarative.

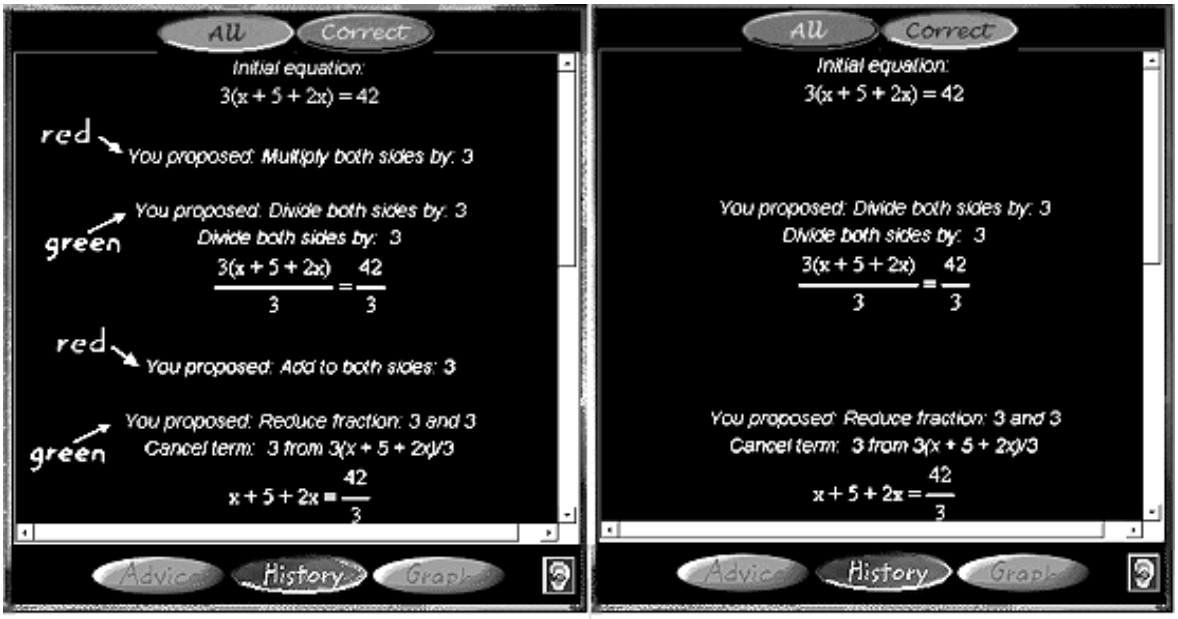

Figure 7. Problem Solving History. The left side shows the (annotated) recap of a student's problem solving activity for the current problem; correct steps are shown in green text, incorrect in red. The right side displays the filtered history: a student clicks the "correct only" button to view valid steps toward solution only.

.Supporting our use of animated demonstrations, with regard to pedagogical rationale several studies have demonstrated that animated demonstrations of a temporally ordered process—in this case the steps involved in applying an algebraic operation to an equation—have a significant learning effect (e.g., Payne *et al.*, 1992; Waterson & O'Malley, 1992). From the affective perspective, Palmiter and Elkerton (1991) found that users enjoyed animated demonstrations and preferred them to text-only explanations

Another feature supporting users' problem solving activities is the *Problem Solving History*. AlgeBrain keeps track of student work on each problem—that is, all of the student's operation/operand proposals and the tutor's responses—and displays this information in the history panel. Correct proposals are displayed in green; incorrect steps are in red. With respect to the design rationale for a reified problem solving history, we've observed problem solvers reaching a solution state but unable to reconstruct the steps they exercised to get there (Singley *et al.*, 1991); the recap of steps toward solution should support recall (e.g., Polya, 1957). We have used a similar technique in earlier work on an ITS for Smalltalk programming (Alpert *et al*., 1995), but there students could review their problem solving only when a successful solution was reached; in AlgeBrain, the history of problem solving steps may be consulted at any point in the process. The history may also be filtered to elide all missteps and display only those moves that brought the student to the solution state—a filtered recap that allows users to reflect on their problem solving without the noise of false moves.

Figure 7, left side, shows the history panel with all of a student's problem solving steps visible. Note that our solution to a (common) screen real estate problem is to have several interactive panels share the same area, the right side of the interface, and let users control their visibility. In Figure 4, the advice panel is visible in this shared space. The history panel is activated by clicking on the *History* radio button in the lower right of the UI. The right side of Figure 7 shows the filtered view of the same history. Now, all red (incorrect) moves are hidden and steps on the solution path only are visible.

Lastly, alternative representations support developing deeper understanding of a problem domain. Of course, in our domain of symbolic equation solving the traditional alternative representation is a graph. Graphing facilitates deeper comprehension by allowing students to make connections between disparate representations of the same mathematical relationships (Vonder Embse & Olmstead, 1993). The AlgeBrain client also incorporates an interactive graphing component. Clicking on the *Graph* radio button in the UI's lower right brings up the graph of the current equation. Students can zoom in on sections of a graph and directly grab and drag graphs using the mouse. As the mouse is moved over the graph surface, corresponding *x*- and *y*-coordinates are displayed, facilitating the use of the graph to visually find the equation's root(s).

CONCLUSION

As the popular saying goes, "There's good news and bad news." The good news is, the World Wide Web presents an unprecedented opportunity for the deployment of intelligent tutoring systems and the democratization of "private" tutoring or coached practice. Further, the Web opens new possibilities, such as synchronous collaboration among students in disparate locations. So students can work on a problem together or consult an human expert online instead of or in addition to the computational tutor.

Clearly, the Web is the platform of the future for computational tutoring. But, it's a mixed blessing: the bad news is, getting there requires much work. Even when constructing standalone intelligent tutoring systems, the expense and skill set is almost overwhelming: knowledge and proficiency in cognitive science, programming, software engineering, rule programming, user interface design, computer art, etc. Now, we must add to that list client-server distributed application construction, multithreaded socket communication, HTML—additional skills associated with Internet development.

On the good news side, as Figures 1 and 2 demonstrate, the Web-based architecture we suggest adapts and extends an architecture for standalone intelligent tutoring systems that is in

common usage. As such it should facilitate the porting of many existing standalone systems to the Web.

We've also attempted to point out features in AlgeBrain's user interface design that are generally applicable for supporting problem solving and providing an enjoyable user experience. Again, many of these features—such as the Problem Solving History that allows after-the-fact recap of and reflection on problem solving activity, and the animated Just-In-Time Dictionary which offers an example-based instructional tool whose use is situated in problem solving—can be adapted to problem solving tutors in general.

Acknowledgments

In addition to naming names, we will—for pedagogical purposes for other developers—mention the tools we used to produce AlgeBrain. The standalone tutor and application server were developed using ParcPlace-Digitalk's VisualSmalltalk®. The client was programmed using IBM's VisualAge® for Java and Sun's Java Development Kit (JDK1.1). Portions of the code were written by Steve Swerling (on the Smalltalk side) and Bret Douglas (Java). UI artwork was created using Adobe® Photoshop®, Microsoft® Paint, and GIF Construction Set for animated GIFs. Some of the art was produced by Cyndi Vincent. One image, the dancing AlgeBrain animated-GIF, is from the *Web Animation Explosion* CD from Nova Development Corp. Some sound clips were created and edited using the GoldWave audio editor; the rest are from another Nova Development CD, *Web Explosion 20,000*. The UI background music was produced by the T.J. Watson Computer Music Center. In the interest of full disclosure: as of this writing (December, 1998) the student model component has not been programmed; *all* other features presented here have been implemented. Java and JDK are trademarks of Sun Microsystems, Inc. Netscape is a registered trademark of Netscape Communications Corporation. Mathematica is a registered trademark of Wolfram Research, Inc.

References

Alpert, S.R. (1991). Self-describing animated icons for human-computer interaction: A research note, *Behaviour and Information Technology*, 10 (2), 149-152.

Alpert, S.R., Singley, M.K., & Carroll, J.M. (1995). Multiple Multimodal Mentors: Delivering Computer-Based Instruction via Specialized Anthropomorphic Advisors, *Behaviour and Information Technology*, 14 (2), 69-79.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89, 369-406.

Anderson, J.R., Boyle, C.F. , Farrell, R., & Reiser, B.J. (1984). Cognitive principles in the design of computer tutors, In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society* (pp. 2-9). Boulder, CO: University of Colorado.

Anderson, J.R., Corbett, A.T., Fincham, J., Hoffman, & D., Pelletier, R. (1992). General principles for an intelligent tutoring architecture, In V. Shute & W. Regian (Eds.), *Cognitive Approaches to Automated Instruction*, Hillsdale, NJ: Lawrence Erlbaum.

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

Brown, J.S., Collins, & Duguid, P. (1988). Cognitive apprenticeship, situated cognition, and social interaction. Institute for Research on Learning Report No. IRL88-08, June 1988.

Brusilovsky, P., Schwarz, E, & Weber, G. (1996). ELM-ART: An Intelligent Tutoring System on World Wide Web, In C. Frasson, G. Gautier, & A. Lesgold (Eds.), *Intelligent Tutoring Systems, Third International Conference, ITS'96* (pp. 261-269). Berlin: Springer.

Brusilovsky, P., Ritter, S., & Schwarz, E. (1997). Distributed intelligent tutoring on the Web, In the *Proceedings of AIED'97, the Eighth World Conference on Artificial Intelligence in Education*. Also http://domino.psy.cmu.edu/ritter/papers/pat-interbook/pat-interbook.html.

Burns, H.L. & Capps, C.G. (1988). Foundations of Intelligent Tutoring Systems: An Introduction. In M.C. Polson & J.J. Richardson (Eds.), *Foundations of Intelligent Tutoring Systems*, Hillsdale, NJ: Lawrence Erlbaum.

Eliot, C. (1997). Implementing Web-based intelligent tutors, In Proceedings of the workshop "Adaptive Systems and User Modeling on the World Wide Web," Sixth International Conference on User Modeling.
http://www.contrib.andrew.cmu.edu/~plb/UM97_workshop/ Eliot.html

Hoogeveen, M. (1995). Towards a new multimedia paradigm: Is multimedia assisted instruction really effective? In *Proceedings of ED-MEDIA'95*.
Also http://cyber-ventures.com/mh/paper/mth-edu.htm.

Koedinger, K.R., Anderson, J.R., Hadley, W.H., & Mark, M.A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8, 30-43.

McKendree, J., Radlinski, B., & Atwood, M.E. (1992). The Grace tutor: A qualified success, In C. Frasson, G. Gautier, and G. I. McCalla (Eds.), *Intelligent Tutoring Systems: Second International Conference, ITS'92 Proceedings* (pp. 677-684). Berlin: Springer-Verlag.

Nakabayashi, K, Maruyama, M., Koike, Y, Kato, Y., Touhei, H., & Fukuhara, Y. Architecture of an Intelligent Tutoring System on the WWW, In *Proceedings of AIED'97, the Eighth World Conference on Artificial Intelligence in Education.* Also
http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Nakabayashi/Nakabayashi.html

Newell, A. and Rosenbloom, P.S. (1981) Mechanism of skill acquisition and the law of practice. In J. R. Anderson (Ed.) *Cognitive Skills and their Acquisition* (pp. 1-55). Hillsdale, NJ: Lawrence Erlbaum Associates.

Okazaki, Y., Watanabe, K., & Kondo, H. (1997). An ITS (Intelligent Tutoring System) on the WWW (World Wide Web), *Systems and Computers in Japan*, 28(9), 11-16.

Palmiter, S. & Elkerton, J. (1991). An evalutation of animated demonstrations for learning computer-based tasks, In S.P. Robertson, G.M. Olson, & J.S. Olson (Eds.), Human Factors in Computing Systems: *CHI'91 Conference Proceedings* (pp. 257-263). NY: ACM.

Payne, S.J., Chesworth, L. & Hill, E. (1992). Animated demonstrations for exploratory learners, *Interacting with Computers*, 4, 3-22.

Polya, G. (1957). How to Solve It. Garden City, NY: Doubleday.

Ritter, S. (1997). PAT Online: A model-tracing tutor on the World-wide Web, In *Proceedings of the workshop "Intelligent Educational Systems on the World Wide Web," 8th World Conference of the AIED Society*.
http://www.contrib.andrew.cmu.edu/ ~plb/AIED97_workshop/Ritter/Ritter.html.

Ritter, S., Brusilovsky, P., & Medvedeva, O. (1998). Creating more versatile intelligent learning environments with a component-based architecture, In B.P. Goettl, H.M. Halff, C.L. Redfield, & V.J. Shute (Eds.), *Intelligent Tutoring Systems, 4th International Conference, ITS'98* (pp. 554-563). Berlin: Springer.

Singley, M.K., Carroll, J.M, & Alpert, S.R. (1991). Psychological design rationale for an intelligent tutoring system for Smalltalk, In J. Koenemann-Belliveau, T.G. Moher, and S.P. Robertson (Eds.), *Empirical Studies of Programmers: Fourth Workshop*, Norwood, NJ: Ablex.

Vassileva, J. (1997). Dynamic Course Generation on the WWW, Proceedings of AIED'97, the Eighth World Conference on Artificial Intelligence in Education. Also
http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Vassileva/Vassileva.html.

Vonder Embse, C. & Olmstead, E. (1993). *Exploring Algebra with the TI-81*. Reading, MA: Addison-Wesley.

Warendorf, K. & Tan, C. (1997). ADIS-An animated data structure intelligent tutoring system, or Putting an interactive tutor on the World Wide Web, In *Proceedings of the workshop "Intelligent Educational Systems on the World Wide Web," 8th World Conference of the AIED Society*.
http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/ Warendorf/Warendorf.html.

Waterson, & O'Malley, C. (1992). Using animated demonstrations to teach graphics skills, In A. Monk, D. Diaper, and M.D. Harrison (Eds.), People and Computers VII: *Proceedings of HCI'92* (pp. 463-474). Cambridge, UK: Cambridge University Press.