Towards Lightweight Tutoring Agents

Steven Ritter

Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213
USA
412-268-3498
sritter@cmu.edu

Kenneth R. Koedinger

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
USA
412-268-7667
ken.koedinger@cs.cmu.edu

Abstract

We describe our efforts to build new learning environments that incorporate tutoring elements into pre-existing software packages. Two systems are described, one which provides tutoring support in the Geometer's Sketchpad and the other which supports students using Microsoft Excel. Although the implementation of these two systems was somewhat different, they share many basic components. An analysis of their similarities and differences allows us to move towards a set of standards for tutoring agents that interact with complex tools. By constructing learning environments in this manner, we can leverage the power of existing workplace software and educational environments to create more powerful learning tools.

Topic: Intelligent tutoring systems

Subtopic: Learning environments and microworlds, Authoring systems and tutoring shells

Submitted to the 7th World Conference on Artificial Intelligence in Education, August 16-19, 1995, Washington, DC, USA.

Introduction

Intelligent tutoring systems have traditionally been large-scale software systems that are costly to produce and difficult to adapt to curricular objectives other than those of the original designers. There are clear pedagogical benefits of well-designed intelligent tutors as has been demonstrated by numerous studies (cf. Anderson, Corbett, Koedinger, Pelletier, in press; Lajoie & Lesgold, 1989; Koedinger & Anderson, 1993a; Mark & Greer, 1991). And indeed such benefits justify the production cost for subject-matter domains for which there is a wide-spread need, for example, algebraic symbolization and tool use (e.g., tables, graphs, and equations). However, as increasing technology use is changing the landscape of necessary skills in the workplace and academics (SCANS, 1991), there is regular evolution in curricular objectives (NCTM, 1989). The rising use of symbolic calculators, for example, will reduce the need for paper-based symbolic manipulation skills, just as numeric calculators have reduced, or even eliminated, the need for paper-based algorithms like finding the square root of a number. The need for reasoning skills, like algebraic symbolization or how to write a computer program, will remain and even increase as workers need to construct inputs and interpret outputs of modern computational tools and, here, traditional intelligent tutors are justified. The state of technology is such that we can now begin to build lightweight tutoring agents that can be embedded within workplace software tools or other educational environments. By leveraging the power of existing tools, we can reduce our development cost, and the resulting systems can provide better tools for the workplace and better learning environments for the school.

Examples of Combining Tutors and Existing Software Tools

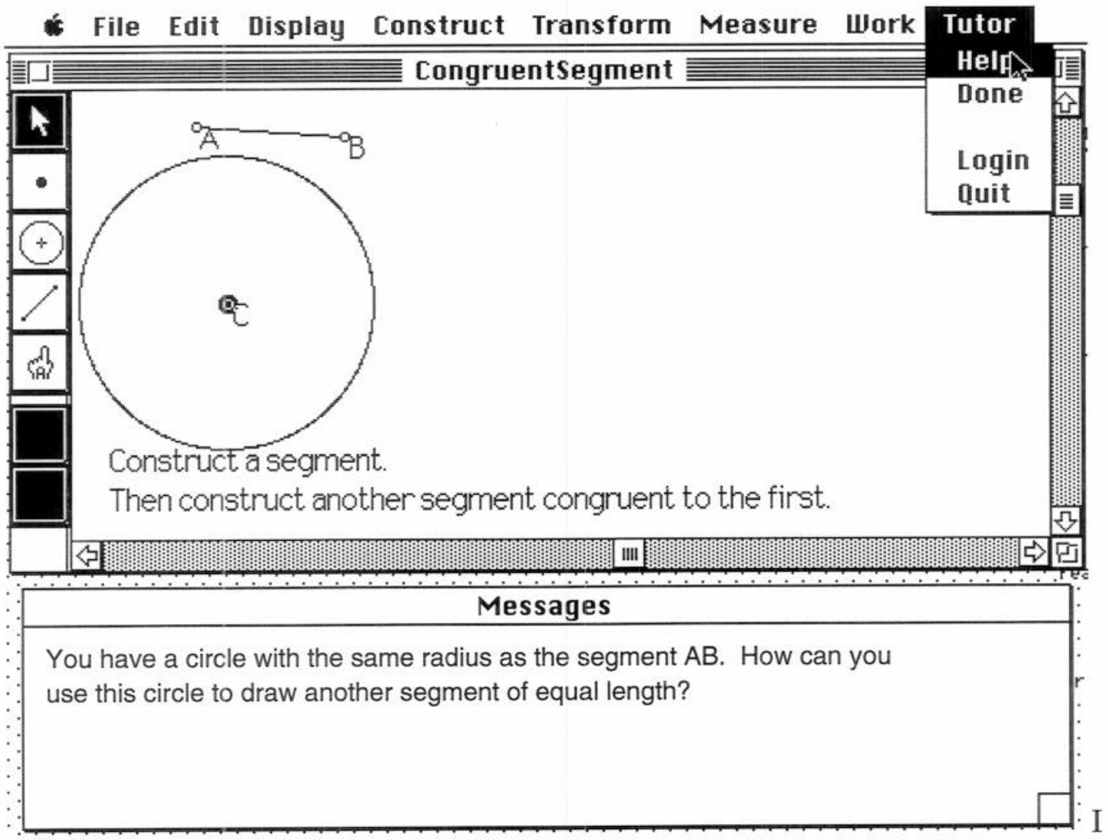
Our first steps towards lightweight tutoring agents have been the use of interprocess communication (e.g., Apple's AppleEvents) to create two prototype learning environments that use an existing software tool as the interface to an intelligent tutor. The tutoring components of these systems are cognitive tutors (Anderson, et al., in press) implemented using the Tutor Development Kit (Anderson & Pelletier, 1991). Building such tool-tutor marriages has allowed us to investigate the nature of a tutoring agent independent of the workspace or tool environment in which the student performs actions. In particular, we want to characterize the communication protocol between tool and tutoring agent and move toward a standard that will reduce the courting time for future tool-tutor marriages. After describing our two prototype systems, we step back to generalize their properties. We present an initial specification of a lightweight tutoring agent and the communication protocols for interacting with other tools in a complete learning environment.

A Tool-Tutor for Geometric Construction Using Geometer's Sketchpad

The Geometer's Sketchpad (1989) is a commercial software tool for creating geometric constructions and dynamically investigating them. It functions both as an educational microworld for exploring and discovering properties of geometric figures (cf. Schwartz, Yerushalmy, & Wilson, 1993) and also as a tool for mathematical research. In previous work, Koedinger and Anderson (1993b) proposed a learning environment that would integrate tutor-supported geometric conjecturing with existing tutor support for conjecture evaluation (Koedinger & Anderson, 1993a). The first step in geometric conjecturing is to construct figures to investigate. We built a prototype tutor for geometric construction that uses Sketchpad as the student's workplace.

Figure 1 shows the screen as it looks to a student in the middle of a problem. The problem statement appears as text in a Sketchpad file that is loaded when the student starts the problem -- see the text "Construct a segment ..." just below the circle. This problem asks the student to draw a segment and then construct a second segment to equal to the first. The student has drawn the first segment (AB), drawn one endpoint of the second segment (C), and taken the crucial step of constructing a circle whose radius is equal to the first segment. At this point the student asks for halp and tutor decides, based on the student's progress so far, what it would do next and gives an initial hint. As with other cognitive tutors, further help requests yield more specific hints.

From the user's point of view, the system looks almost identical to the Geometer's Sketchpad application. The only differences are that there is an additional menu labeled "Tutor" which students can use to start and quit the tutor, to request help, and to select the next problem. There is also a "Messages" window used by the tutor to give provide hints and give feedback.



Construct a segment.
Then construct another segment congruent to the first.

You have a circle with the same radius as the segment AB. How can you use this circle to draw another segment of equal length?

Figure 1: Geometer's Sketchpad and a tutoring agent for geometric construction

A Tool-Tutor for Algebra Problem Solving Using Microsoft Excel

The Excel Algebra Problem Solving Tutor is a variation of a learning environment that we have been using to teach word problems in ninth-grade algebra. In this system, students are given a word problem such as: "You are planning to make money shoveling snow after the next snow storm. You know that your supplies will cost $50 and that you can earn $8 for each walk you shovel. How much money will you earn if you shovel 5 walks? How many walks do you need to shovel to break even?" Students answer these questions by completing a table that contains two columns (one for the number of walks shoveled and the other for the profit) and two rows (one for shoveling five walks, one for breaking even). In addition to completing the specific questions, students are asked to come up with an algebraic expression for the profit, to indicate the unit used to measure each of the variables and to create a graph of the line that relates walks shoveled to profit.

The original version of the learning environment used a tool that we created specifically for this task, but it is clear that the graphing and table functions are similar to those that we might find in a spreadsheet. In addition, there is a clear mapping between the algebraic expression for profit and (at least one version of) the Excel formula used to calculate cells in the profit column. For this reason, we defined a system which uses Excel 4.0 for the Macintosh in place of our own spreadsheet window.

From the user's perspective the system looks much like Excel would look for any user. There is an additional "Tutor" menu from which the user can ask for help or exit the tutor. When the system starts up, an Excel worksheet is opened containing the problem statement. On a separate worksheet, borders are drawn around cells to create a table similar to the one they would see in our own table tool.

Users have access to all of Excel's capabilities, although the features needed to perform their task are quite limited. They fill in the table by typing the appropriate headings, numbers and algebraic expressions. In the "profit" column, users may enter an Excel formula to calculate the profit, given the number of walks shoveled. When the tutor needs to display a message to the user, an Excel dialog appears.

Basic Elements of a Learning Environment

These systems share some basic architectural features that define the kind of learning environments we are creating. An important consideration in a learning environment is whether it is aware of the user's goals in performing some action. There are some systems (such as the "Tip Wizard" in Microsoft Excel 5.0) that monitor sequences of user actions and inform the user of shortcuts. For example, such a system might detect that a user has moved through several levels of dialogs to perform an action when the same action could be accomplished by pushing a function key. Such systems are limited to giving advice on manipulating the interface, since they have no prior knowledge of the user's goals, and it is unreasonable to expect to be able to discover such goals by observing the user's actions in a complex environment.

Our tutoring systems present the user with a problem, which becomes the user's highest-level goal. Since we know the user's goals, we can direct our assistance towards accomplishing them. We will limit further discussion to goal-directed learning environments. The learning environment need not require that the goal be accomplished in a particular fashion, only that it be accomplished within the environment. A particular learning environment may or may not wish to comment upon user actions that are irrelevant to or in conflict with the goal.

Our learning environments contain four basic objects: a tool, a tutoring agent, a curriculum agent and a control center.

Tools

A tool is a piece of hardware or software that can be used to perform some work in a particular domain. The design of a tool typically involves a trade-off between generality and ease of use. For example, the programming language C can be used to create programs that are useful in many domains, but creating such programs is time-consuming and prone to errors. A spreadsheet has a more limited (but still broad) range of applicability and is somewhat easier to use than a programming language. An application like Quicken is (in part) a spreadsheet that has been specialized for use in tracking personal finance. Although it is limited in its domain of application, it is much easier to use than the more general spreadsheet.

Our proposal is applicable to all of these types of tools, but the advantages to be gained are greatest with the most general (and difficult to use) tools. Both Excel and the Geometer's Sketchpad are of this type.

Since we want to be able to build learning environments using pre-existing tools, it is important that we not be too strict in our requirements of these tools. Nevertheless, we do require that the tool be appropriate for the learning environment, be able to communicate user actions in an appropriate fashion and be able to create, delete and change objects upon request.

In order for us to monitor users' actions, it is important for the tool to communicate those actions at an appropriate level of detail. Since we want to monitor the user's actions with respect to some high-level goal, it needs to understand the semantics of the user's action, rather than the details of the manipulation of the interface. It is also important that we receive information in sufficient detail to provide appropriate feedback. Consider a case where the user pushes the mouse button at screen location (100,200) and then moves the mouse and lets go of the button at screen location (148,293). Suppose also that the tool responds to this user action by moving a circle which was centered at (100,200) to screen location (148,293). A low-level description of this action would refer to the actions of the mouse. An intermediate and appropriate description would refer to the way the user is manipulating the circle. At a higher level of description, this event may not be described until later, and then only as one action in service of a higher-level goal.

In general, we need to understand users' actions in terms of the work accomplished, not the user interface actions that were performed. Typically, this description takes the form of a predicate describing the action, the object being acted upon and any other parameters that are necessary to understand the action. In most cases, the predicate will have three places, describing the subject, verb and an additional parameter. For example, if the circle in the above example were named "circle1," our preferred input would be [Subject="circle1"; verb="drag";parameter=(148,293)]. Note that this kind of semantic information can only be provided by the tool, not the operating system, since it refers to tool-specific objects (like the circle). The operating system will know only the lower-level mouse actions. Thus it is essential that the tool, not the operating system, provide this information.

Fortunately, the need for this kind of semantic description is becoming standard in newly-developed applications. In fact, the kind of descriptions that a tutoring agent requires are also required by OLE and OpenDoc in order to support sharing of data and interface elements across applications. AppleScript and other application-scripting systems also require applications to understand actions at this level of detail. To support macro recording, OpenDoc requires that applications, on request, describe all user actions in such terms. A "recordable" application which does so would be a prime candidate for the tool component of a learning environment.

Tutoring Agents

Tutoring agents are pieces of software that monitor users' actions. The purpose of the monitoring may be to determine whether the user is performing a task correctly or to provide advice (either as a response a user's request or due to recognition of some appropriate opportunity). Note that the goals of recognizing errors and giving advice are independent. In this broad definition, tutoring agents may include context-sensitive help systems (which give advice but do not recognize errors) and compilers (which typically point out errors but do not give advice).

Although the tutoring agents in the systems we have built are implemented as model-tracing expert systems, our proposal does not require a commitment to tutoring agents of this form. All it requires is that the tutoring agent be able to evaluate and respond to user actions in some manner.

In many cases, it is desirable for the tutoring agent to be able to control some aspects of the tool. This is especially helpful in giving appropriate feedback to the user. If the tool supports it, we may want to automatically undo an erroneous action. In other cases, we may want to highlight the results of erroneous actions, which may also be accomplished by having the tutoring agent communicate with the tool.

In domains where the problems to be solved take a significant amount of time, it is necessary to be able to tell the tool to restore some previous state, so that the user may quit the learning environment without losing work. This can be accomplished easily in scriptable applications by recording the state of the tool when the user exits the learning environment. When the learning environment is started again, we issue appropriate scripting commands to recreate the state in which the user last left the tool.

In practice, we have found this requirement much easier to satisfy than the requirement that the tool communicate appropriate information to the tutoring agent. Many tools are now controllable through some scripting language (like AppleScript) and, in cases where that language is inadequate, there are usually acceptable alternatives to communicating with the tool.

Curriculum Agent

The curriculum agent is responsible for deciding which task to present to the user. This decision can be as simple as picking the next task on the list (or having the user select a task) or can be as complex as picking a task based on how well its features match a profile of user skills computed by the tutoring agent.

Control Center

The control center handles all communication between the tool and the tutoring agent. When necessary, it translates information from the language of the tool into that of the tutoring agent (and vice-versa). If more than one tool or tutoring agent exists, the control center is responsible for routing messages to the correct component. Since the control center needs to know about the communication requirements of the tool and the tutoring agent, it needs to be customized for each learning environment.

The advantage of this design is that the tool and the tutoring agent can be fully designed without knowledge of the details of the other. The ability to change the tutoring agent independent of the tool is crucial in domains in which we don't develop the tool ourselves. In many domains, it makes sense to use commonly-available software tools. For example, we might use either Mathematica or Maple as tools for solving calculus problems, and we might use either Excel or Lotus 1-2-3 as tools for a business learning environment. Different applications in a class typically have different interfaces, but they share actions at the level of the semantic event. To the degree that two tools share semantic events, a particular tutoring agent will work equally well with either of them.

Implementation of the Protocol in the Geometer's Sketchpad and Excel

In this section, we discuss the way in which these functions were accomplished in the Sketchpad and Excel tutors.

Sketchpad

Sketchpad has the ability to "demonstrate" a sketch to be drawn on one computer to any other computer in the classroom. This demonstration facility communicates at events near the appropriate grain size. We needed control center functionality to aggregate certain events that came at too small a grain size. In some cases, the demonstration facility did not provide all the communication capabilities we needed, but the Sketchpad developers graciously made experimental modifications to Sketchpad to help us build this prototype. One desired modification was to inform the tutoring agent which object was selected at the time a hint is requested.

Another modification is to include a "Tutor" menu that students can use to start the tutor, to request help, and to select the next problem. The solution here will again be one that has the side-effect of increasing the general functionality of the tool. The Sketchpad developers added an ability to respond to a special AppleEvent that requests the addition of a menu and a list of menu-items. The result of the tutoring agent (or any other application) sending this AppleEvent request will be a new menu appearing in Sketchpad. When an item is selected in this menu, Sketchpad sends out an AppleEvent indicating which item was selected. The tutoring agent (or other application) can respond appropriately.

Excel

We used Excel version 4.0 for the Macintosh, which does not normally provide semantic descriptions of user actions. However, it is fully customizable, so we were able to instruct Excel to send AppleEvents describing all user actions. We also used Excel's customization facilities to create a "tutor" menu. Choosing an item from that menu created an AppleEvent similar to that generated for any other action.

Since a cognitive model and tutor for Algebra problem solving already existed, conceptually all that needed to be done was unplug our home-grown table tool and plug in Excel instead. Most of the implementation work, then, involved creating a control center that could communicate between Excel and the exising tutoring agent. The expert system component of our learning environments expects user actions to be described as a "selection-action-input" triple. This description corresponds closely to the AppleEvent description of a user action. Excel and the tutoring agent use different languages to refer to these elements, however, so it was necessary to translate from one language to another. In our case, the control center translated Excel's references into data structures used by the tutoring agent. For example, "Cell R2C3 in Worksheet 1" (an Excel description of a cell) was translated into a pointer to the working memory element representing that cell.

Communication between the agent and Excel was straightforward. Excel is fully scriptable through AppleScript. For example, if the user entered a value into the table which the tutoring agent determined to be incorrect, it sent AppleScripts instructing Excel to change the font in that cell to outline form. In cases where the user performed some action that was disruptive to the solution (such as sorting the table), we instructed Excel to use its "undo" capability to restore the earlier state. When we wanted to present an error message, the tutoring agent passed the message to the control center, which generated AppleScript code telling Excel to display a dialog containing the message. In response to an incoming action, the tutoring agent informed the control center of any items that should be highlighted or unhighlighted, any messages to display and whether the action should be undone.

Standardization

Based on our experiences in building the Sketchpad and Excel tutors, we are able to take some steps towards standardizing the components of a learning environment. Figure 2 illustrates the basic interaction within the learning environment.

Tool

The tool is required to emit semantic events corresponding to all user actions. We do not commit to any particular format for these events.

The tool must have accessible content. That is, we must be able to manipulate some objects in the tool (such as geometric objects in Sketchpad or cells in Excel). In order to restore the state of a partially-completed problem, we need to be able to create objects and set properties of the objects. In order to indicate errors without requiring the user to immediately correct them, we need to be able to highlight (and unhighlight) objects in some appropriate fashion. If we wish to disallow some actions in the tool or to immediate remove the user's erroneous steps, the tool must support an "undo" command. We do not require any particular protocol for implementing these actions; any scriptable application should have these capabilities in some form. Although it is sometimes convenient to display messages through the tool, we do not require that the tool be able to display messages. If it can not, the learning environment must be able to display messages in some other way.

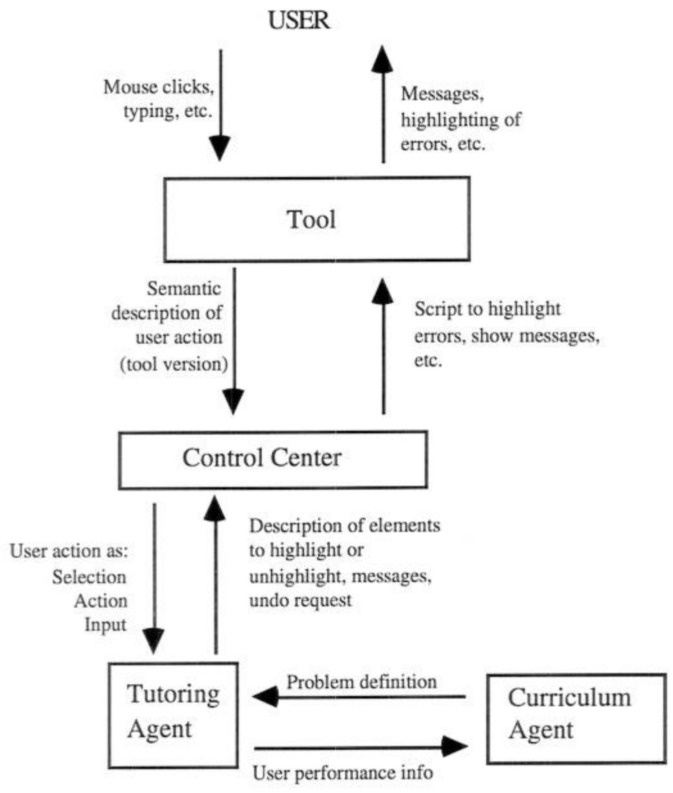

USER
Mouse clicks, typing, etc.
Messages, highlighting of errors, etc.
Tool
Semantic description of user action (tool version)
Script to highlight errors, show messages, etc.
Control Center
User action as: Selection Action Input
Description of elements to highlight or unhighlight, messages, undo request
Tutoring Agent
Problem definition
User performance info
Curriculum Agent

Figure 2: Interaction between tutoring agent and tool

Tutoring Agent

The tutoring agent needs to respond to four messages: *start-problem*, *process-tool-action*, *process-help* and *process-done*.

The *start-problem* message takes two parameters, corresponding to the initial and goal state of the problem. Typically, this message is sent by the curriculum agent.

The *process-tool-action* message takes three parameters, corresponding to the user's input, action and selection. In response to this message, the tutoring agent should decide how to (and whether to) respond to the user's action. If the tutoring agent decides to respond, it calls the *respond-to-tool* method on the control center object, passing a list of messages to display, a list of elements to highlight, a list of elements to unhighlight and a Boolean indicating whether the tool should try and undo the user's action.

The tutoring agent also responds to a *process-help message*, which takes two parameters: the user's selection and input. In response, the tutoring agent calls *respond-to-tool* on the control center.

The *process-done* message takes no parameters. This message is sent to the tutoring agent if the user indicates that he or she has completed the problem. In response to this message, the tutoring agent should call *update-user-performance* on the curriculum agent, passing a parameter describing the tutoring agent's evaluation of the user. In some learning environments, the tutoring agent may detect completion of a problem and automatically move to the next one (without user input). In such systems, the *process-done* message is not used, but the tutoring agent will still need to call *update-user-performance* at the end of the problem.

Control Center

The control center object needs to intercept all semantic events sent from the tool. It translates these events into selection-action-input triples and calls process-semantic-event, process-help or process-done (whichever is appropriate) on the tutoring agent. Although the translation of the incoming semantic event into selection, action and input depends on the particular communication protocols for the tool and the tutoring agent, we can define a standard variant of the control center which translates an AppleEvent into a selection-action-input form. For example, in response to a SetData AppleEvent, this method would set the selection to the value of the keyDirectObject parameter, the action to "SetData" and the input to the value of the keyAEData parameter. Similar standard translations can be defined for all AppleEvents.

The implementation of the *respond-to-tool* method is dependent on the tool being used. Some tools display messages in a fixed window, others may use a dialog which appears with the message (as did the Excel tutor) and others may use no message display at all (as with the Sketchpad tutor). If the tool does not support message display, the control center must be able to display messages in some other fashion. One possibility is for the control center to display its own window; another is for it to ask the tutoring agent to display a window (as the Sketchpad tutor did).

The implementation of highlighting and unhighlighting elements clearly depends on the tool's ability to display elements in different ways. We allow complete flexibility in this implementation. The Excel tutor displayed text in a different style, although it could just as easily put a border around an errant cell. The Sketchpad tutor did not highlight elements at all.

Curriculum agent

The curriculum agent accepts the *update-user-performance* method. We leave the specifics of the description of the user's performance up to the particular implementation, since this will depend on how the tutoring agent measures performance. In our tutors, we would send a list of skills and, for each skill, the tutoring agent's estimate of the user's progress on that skill. The curriculum manager would then choose a problem which emphasized the student's weakest skills. Other systems might simply take a count of the errors made by the student.

Conclusion

In our development of two systems that add tutoring agents to pre-existing tools, we have begun to define a set of standards that will simplify the creation of learning environments of this type. We believe that technology has advanced to the point where it is practical to think about building systems this way. Furthermore, the resulting systems combine the best elements of workplace tools and educational environments with guided instruction that has proven to be effective.

Although the specifications described here are sufficient for the tutors we have built in the past, we need to consider ways in which they could apply to the next generation of tutoring systems. One way to think about a tutoring agent is as a robot, which has certain senses for monitoring user performance and certain effectors for communicating with others. By separating the tutoring agent from the other components of the learning environment, we can imagine, for example, developing a robotic tutoring agent which interacts with students over the internet in rooms of a MUD. While the standards introduced here define senses and effectors sufficient for the kind of learning environment we have already built, more work needs to be done to generalize the type of interaction for wide variety of environments.

References

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (in press). Cognitive tutors: Lessons learned. To appear in The Journal of the Learning Sciences.

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

Geometer's Sketchpad (1989). Software by Key Curriculum Press, Inc., Berkeley, CA.

Koedinger, K. R., & Anderson, J. R. (1993a). Effective use of intelligent software in high school math classrooms. In Proceedings of the World Conference on Artificial Intelligence in Education, 1993. Charlottesville, VA: AACE.

Koedinger, K. R., & Anderson, J. R. (1993b). A Cognitive Tutor for Mathematical Investigation and Reasoning. Unpublished proposal manuscript.

Lajoie, S. P., & Lesgold, A. (1989). Apprenticeship training in the workplace: Computer coached practice environment as a new form of apprenticeship. *Machine-Mediated Learning, 3*, 7-28.

Mark, M. A. & Greer, J. E. (1991). The VCR tutor: Evaluating instructional effectiveness. In *Proceedings Thirteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum Associates.

National Council of Teachers of Mathematics (1989). *Curriculum and Evaluation Standards for School Mathematics*. Reston, VA: The Council.

Schwartz, J. L., Yerushalmy, M., & Wilson, B. (1993). *The Geometric Supposer: What Is It a Case of?* Hillsdale, NJ: Erlbaum.