Extending Intelligent Learning Environments with Teachable Agents to Enhance Learning[1]

Gautam Biswas[1], Thomas Katzlberger[1], John Bransford[2], Daniel Schwartz[3], &
The Teachable Agent Group at Vanderbilt (TAG-V)[2]

[1]*Department of Electrical Engineering and Computer Science,*
Box 1679 Station B

[2]*Learning Technology Center,*
Box 45, Peabody

Vanderbilt University
Nashville, TN 37235.

[3]*School of Education*
Stanford University
Stanford, CA 94305.

This paper extends our previous work on simulation-based Intelligent Learning Environments and SmartTools to computer-based Teachable Agents. Teachable Agents have been inspired by our work in classrooms where students have found it very motivating to teach and help others in problem solving tasks. These interactions also helped students to appreciate feedback and the need for reasoning in their learning processes. Our Teachable Agents do not incorporate machine learning techniques, rather, they are computer-based social agents that require explicit instruction to perform well in a given task environment. We describe the design and implementation of Teachable Agents in the domain of solving complex trip planning problems, and justify the design by preliminary experimental studies that demonstrate the effectiveness of this approach.

1. Introduction

For a number of years, our research group has been working on developing Intelligent Learning Environments that support anchored instruction, and constructive and reflective learning [1, 2]. This led to the development of a simulation-based learning environment, AdventurePlayer [3] that was used in conjunction with the videodisc based Adventures of Jasper Woodbury Series [1] for teaching trip planning problems. The Jasper adventures teach complex problem solving skills in the mathematics and science domains. The AdventurePlayer environment enabled students to simulate partial and complete solutions to trip planning problems and receive feedback. We created the notion of SmartTools [4], which allowed students to construct reusable computer-based tools, like distance-rate-time graphs. These tools taught students how to generalize and apply their knowledge efficiently across multiple problem solving episodes.

[1] This work is supported by NSF KDI grant number REC-9873520

A second version of the AdventurePlayer system included coaching agents. These took on the role of personalities in the adventure (*actors*) and helped students to find relevant information and data by responding to known keywords in text questions [5]. Like traditional coaches [6] they also provided help when students generated suboptimal solutions. Experimental studies showed that AdventurePlayer helped middle-school students develop problem solving skills and generate the optimal solution to a complex trip-planning problem [3].

In working to further motivate students we turned to the notion of *learning by teaching.* Our classroom experiences showed that students preparing presentations and tutorials for outside audiences were more motivated in learning subject material than students preparing for a test [7, 8]. In a study that interviewed sixth graders about the highlights of their year as fifth graders, doing projects that helped the community and tutoring younger students received the highest praise from the students [1]. Students preparing to teach also made statements about how the responsibility to teach forced them to gain deeper understanding of materials. Others focused on the importance of clear conceptual organization. Still others talked about how the feedback they received while teaching prompted deeper reflection and better understanding of subject material [9].

Research on mentoring has shown that tutors learn as much as or more than tutees [10], and that lessons in which students tutor each other are beneficial [11], especially if they are well scaffolded (e.g., reciprocal teaching [12, 13]). A recent study by Chi, Siler, and Jeong [14] also provides strong supporting evidence that one on one tutoring helps both student and tutor, because the interaction results in a greater amount of scaffolding, and motivates the learner to take control of their own learning.

In this paper, we introduce Teachable Agents into our Intelligent Learning Environments to enhance student motivation and learning. Our teachable agents are not cognitive agents [15, 16] and employ no machine learning algorithms to actively acquire and generalize the knowledge taught by the student. They are *social* agents, who need explicit and complete instruction in order to do well in a given task. Once taught, the agent will solve problems and display its results to the student. This motivates students to reflect on what they have taught. In the process, students may consult additional resources to improve their own understanding of the material. After students have succeeded in teaching the agent, they can move to problems at the next level of difficulty and repeat the teaching and learning process. In the rest of this paper, we discuss the simulation environment, the SmartTool modules, and the teachable agent system as well as their combined benefits in promoting learning and assessment.

2. Intelligent Learning Environments

The Intelligent Learning Environment, AdventurePlayer helps students to develop plan representations and solve multi-step distance-rate-time problems. Students use the simulation environment to execute their plans, and reflect on the feedback to generate better, and eventually optimal solutions. A typical adventure, which we term a macrocontext [1], can easily overwhelm students, so we scaffold the problem complexity by letting students construct and use a variety of SmartTools, each suitable to solve specific subproblems. We have developed a variety of SmartTools, such as graphs that assist student's distance-rate-time and fuel-consumption calculations, planner tools for creating and representing plans as a sequence of interdependent actions, and timeline tools for sequencing actions in time. Paper and pencil versions of SmartTools, and the motivation involved in creating, testing, and using them are discussed in Bransford, Zech, and Schwartz [17].

The overall computational architecture of the system is modular and each package has its own interface. Therefore, they can be separated into different applets, or integrated into one single learning environment. The environment is implemented using Java Internet Foundation Classes (Netscape) making it accessible remotely via the web.

2.1. The Simulation Tool

This module provides a quantitative, time-based simulation environment where students formulate a solution to a given problem and get feedback on how well a solution works. The simulation is a realistic, game-like visualization and animation of a mathematical problem.

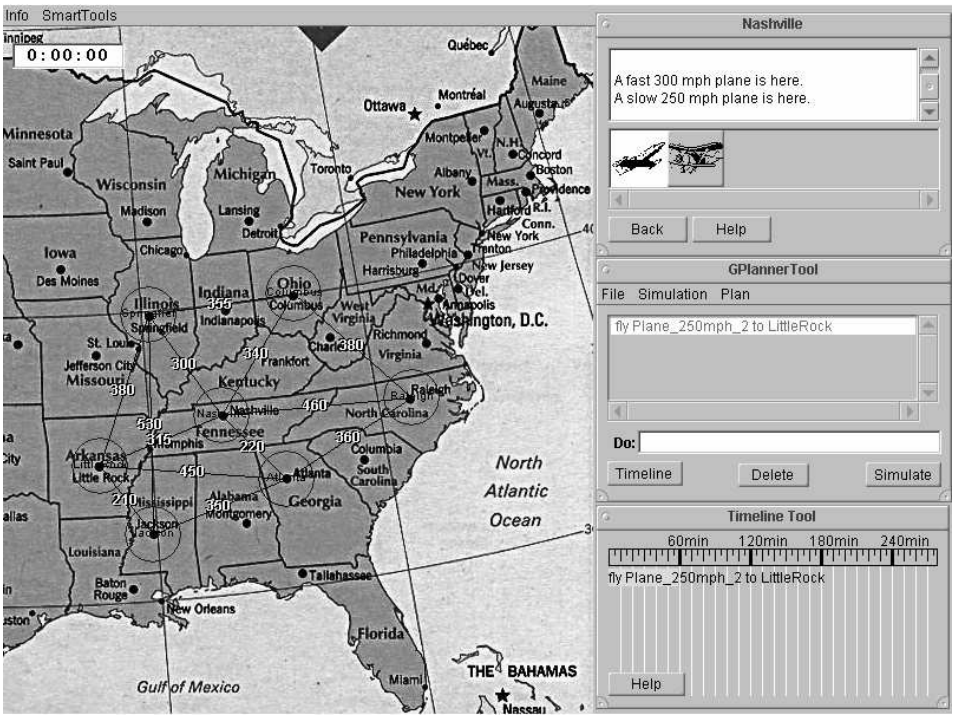

Figure 1: Simulation Environment (left), Inspector, Planner and Timeline (right, top to bottom)

Figure 1 illustrates the simulation-interface where the student can generate and test partial and final solutions to trip-planning problems. Figure 4 (top, left and right) shows the mini-assessment environment that allows the student to experiment, create, and test distance-rate-time SmartTools. The simulation displays agent-objects that model Locations, Items, Vehicles, and Persons. Vehicles animate movements and Persons can act as drivers and pilots. They also may interact socially with the student and provide coaching hints.

Double clicking on locations marked by circles on the map in Figure 1, reveals content information in the inspector-panel (Figure 1, top-right). For example, double clicking on Nashville reveals that there are two airplanes at this location: one that can fly at 250 mph, and a second that can fly at 300mph. A click on the image of an actor (e.g., airplanes and persons) will display its description in the inspector panel. For example, clicking on an airplane icon will display its fuel-consumption, payload, speed, and weight. Additional information about actors and the domain in general can be provided through external multimedia hyper-links that are resolved through the web browser. Students use this information to derive solutions to package delivery problems, starting with the aircraft available at the initial city. Since the airplanes have different speeds, and the availability of aircraft vary from one city to another, students have to make decisions on the most effective route to fly and which aircraft to use for delivery.

Students enter their solution as plan into a planner (see Figure 2, right). In its simplest form, a plan is a linear sequence of steps, where the next action immediately follows the completion of the previous action. Plan steps are expressed in simple language: *"at time do action with actor using resource for duration"*, e.g., "fly the 250 mph plane to Atlanta with a payload of 2000 lb for 2 hours". Each action is parsed at execution time and translated into messages that are sent to actor-agents that drive the simulation. At this point, the actors verify that

that preconditions are met, and if they are violated generate simulation-errors, such as *"Plane does not have a pilot"* or *"Plane out of fuel"* at the appropriate time. We believe that providing errors at the time at which they occur in the simulation provides students with more informed feedback that they can use to reason about their errors, and easily correct their problems. Parse and syntax-errors (e.g. reference to a non-existent actor) are detected and reported at plan-creation time, but they do not tend to occur often with our drag and drop interface. A simulation run animates the aircraft's flight, and when necessary displays diagnostic feedback. A plan starts at time zero and runs in discrete time-steps until it either finishes successfully, or terminates with an error.

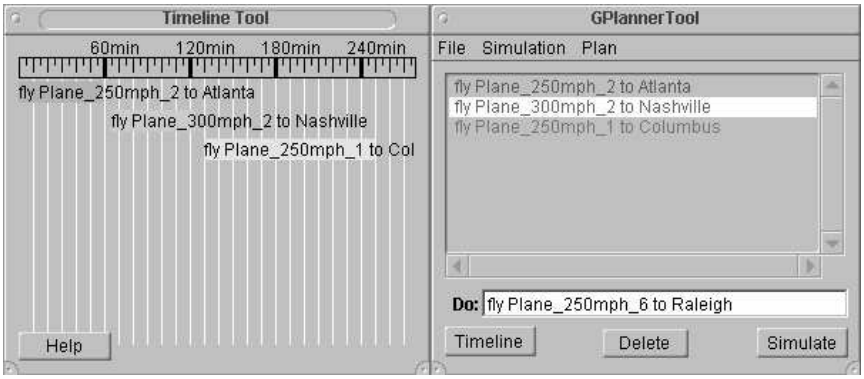

Figure 2: Timeline and Planner Tools

To completely specify the trip-planning solution, the student has to enter a start-time and duration for each plan-step in the timeline-tool (see Figure 2, left). The simulation initiates each step at the scheduled start-time. The duration that has been supplied by the student is compared to the actual duration computed by the simulation tool. A mismatch causes the simulation environment to generate error messages like *"Specified time too long"* or *"Specified time too short"*. When simultaneous actions are allowed, the system may generate a more sophisticated error, such as *"The precondition action for the current action has not been completed."* For problems in which time is not important, start-time and/or duration may be omitted and plan-steps execute in sequence one after the other.

2.2. Smart Tools

Smart Tools provide the basis for students to generalize the knowledge they gain from one or two problem solving episodes to classes of recurring problems. Students are motivated to generate a set of working tools that can be applied across multiple problem-solving episodes in the learning environment, enabling them to work "smart." The basic idea for Smart Tool implementations comes from *cognitive tools* [18] that provide well-designed representational structures and interfaces to help students develop problem solving concepts, which they would find difficult to understand and apply by themselves. When the student chooses to use a specific tool, the system instantiates a blank tool. Different representational tools for the same task allow students to study the effectiveness of distinct representations (e.g., the table vs. the graph representation in rate problems).

A SmartTool consists of a graphical user-interface, a controller, and an optional artificial-intelligence component, called the *ToolExpert*. The controller is responsible for the load, save, and display operations. Implemented within the object-oriented framework, this allows for easy reusability of tools across multiple problem solving episodes, and even across multiple macrocontexts. The artificial intelligence component embedded in the tool makes it an agent. When used in a simulation environment with other tools, the agent has the capability to analyze its own state and evaluate its correctness. For example, the ToolExpert can detect a mismatch in time-units between its distance-rate-time graph and the simulation environment.

Using the coach, the ToolExpert provides appropriate feedback to students when they ask for help, or when it appears that they are hopelessly stuck and unable to proceed further. The idea is not to intervene every time an error occurs (this would make the student depend on the tool to correct his or her errors), but to avoid frustrations when students get stuck, and to help them improve their performance when they ask for help [6].

The GraphTool is a complete implementation of a two-dimensional linear graph representation (see Figure 3, left-bottom). The student can specify axis labels and units for each quantity, and the range for each axis. After that, lines and points are added to the graph, to create the student's own representation of the concept. Some special functions, such as *translate* and *rotate* line are provided to permit operations that can be done with a ruler on graph paper. Students can create their graphs by simulating and collecting data points in a mini-assessment environment (see Figure 4). They may also draw a graph and then verify its correctness by running simulations in the mini-assessment environment. The GraphTool is implemented using a *Model View Controller* pattern. This allows the underlying data to be presented through multiple interfaces, such as tables and graphs of various forms (e.g., linear graphs and radial representations). In the future, we plan to introduce students to simple regression techniques so that they may derive analytic relations from a set of data points or a graphical relation.

3. Learning with Teachable Agents

In previous work, we have created teachable agents in the domain of river pollution studies [9, 19]. An initial version of this system used an animated form of a high school student, Billy Bashinall, who is working with a classmate on a project for monitoring the water quality of a local river. When the partner expresses concern about analysis of the data, Billy assumes a nonchalant and indifferent approach to the project. At this point, Billy is transported to the headquarters of the Dare Force, a group of cartoon characters whose mission is to ensure that other students do not make the mistakes they made in school, and to help students to improve their understanding of material taught in the classroom. A discussion with this group makes Billy realize that he needs help. Students in the classroom view this scene on videotape. Then they are asked to teach Billy so he can do better on the water quality monitoring project.

Though the agent's interactions with the students were pre-scripted, our studies have shown that this approach was highly motivating to fifth graders. The students performed detailed research in order to teach Billy, and observed the effects of their teaching on his behavior. Students made efforts to self-assess their knowledge, and showed improved performance on the assessment. This led to the development of the computer based agent Betty (Betty's Brain in [9]). Students teach this agent by explicitly creating an executable domain knowledge base as a *concept map* [20]. The agent uses the concept map to answer queries generated by the students or a teacher, and the students use the correctness of the answers and an explanation of how they were generated to reason about their own knowledge of the domain material.

We extend the teachable agent architecture to the domain of distance-rate-time problems. Students teach Billy to construct SmartTools that they can later use for solving problems in scheduling flights. We adopt a learning strategy first developed as a software program called **S**oftware **T**echnology for **A**ssessment and **R**eflection, in short STAR-Legacy [21]. Legacy is a multimedia shell that helps teachers, instructional designers, and students manage complex inquiry activities with the goal to provide a dynamic self-assessment environment for learning and problem solving.

Students are presented with a challenge and asked to think of a solution. They start by generating their own ideas, get useful problem solving tips from experts, and use this information to research and improve their understanding of relevant concepts by accessing resources provided by the system. Students can assess their understanding of the material, and

when they are satisfied with the solutions they generate they publish them for others to use and evaluate. This often motivates them to research and study new material in depth, which improves their understanding and problem solving skills. Mini-assessment environments (simple simulations) allow students to assess their own understanding. This prompts the student to go back and perform more research. After going public, students move on to more difficult challenges.

Figure 3 illustrates our adapted implementation of the STAR.Legacy cycle. The cycle is set up for students to help Billy Bashinall solve a sequence of increasingly complex problems involving the delivery of packages between cities on the map in Figure 2. In general, the problems require Billy to choose among airplanes that travel at different speeds (100, 200 and 300 mph), and decide on alternate routes to perform the deliveries in a way that minimizes the time for delivery (sometimes multiple hops using planes that travel at different speeds may be better than a direct route). Once students have successfully taught Billy to solve these problems, they have completed a learning cycle and can move on to the next set of problems.

The learning cycle has four primary components: (i) *Introduction of the problem*, (ii) *Teach Billy*, (iii) *See Billy's Behavior*, and (iv) *Correct Billy's Behavior*. The cycle covers two important aspects of the teaching process:

(a) *Preparing to teach* – this involves going through the material, and understanding and organizing it in a manner that can be taught to others, and

(b) *Teaching process* – this involves delivering the material and answering questions. The dialogue that ensues forces the teacher to gain a deeper understanding of the domain materials.

To situate the student at the start of the interaction process, a video clip shows Billy providing background information about his class work and asking for the student's help in solving the package delivery problems. In the dialogue, Billy states: *"My class is involved in a project that requires us to predict the time it will take to fly from one city to another. Our teacher has suggested that we create SmartTools …".*

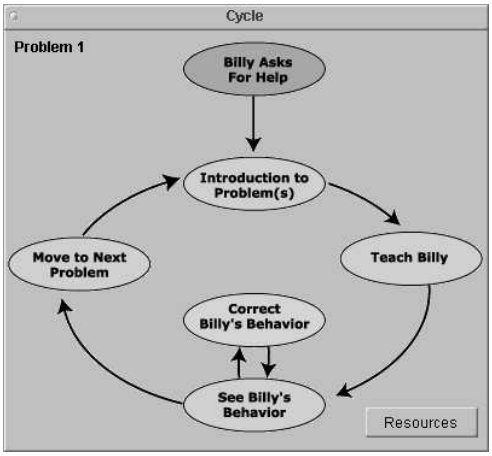

Figure 3: The Learning/Instruction Cycle

As discussed, the Teachable Agent environment allows students to access resources at any time to help develop and refine their understanding of concepts. The *Resources* button opens an HTML-page to provide the student with example problems, demonstrations of solution strategies, and other tutorial material that the teacher may have covered in class.

Once students have reviewed this material, they can click on the *Teach Billy* icon to open up the teach environment. As shown in Figure 4, this environment contains a mini-simulation (top-left), simulation control (top-right), a GraphTool (bottom-left) and the teaching interface (bottom-right). The student teaches Billy by answering multiple-choice questions that pertain to different parameters in creating a graph. The student's response leads to an immediate change in the displayed GraphTool, indicating that Billy is responding to the instruction. If

the student is not sure about a concept, he or she may run simulation experiments in the mini-assessment environment, and modify the answer to questions.

When students are satisfied with their teaching they click on the '*Done Teaching*' button. In response, Billy (for an example see Figure 4, white text box at bottom-right) starts a dialogue with the student by asking a few questions that are related to the created graph. This represents Billy making sure he has understood the material that the student has taught. At the same time it forces the student to reflect on his or her choices. Some of Billy's questions are generic. Others are directly related to the student's answers, and are intended to make students reflect on and revise wrong choices that they may have made (They are not told that their answer is wrong).

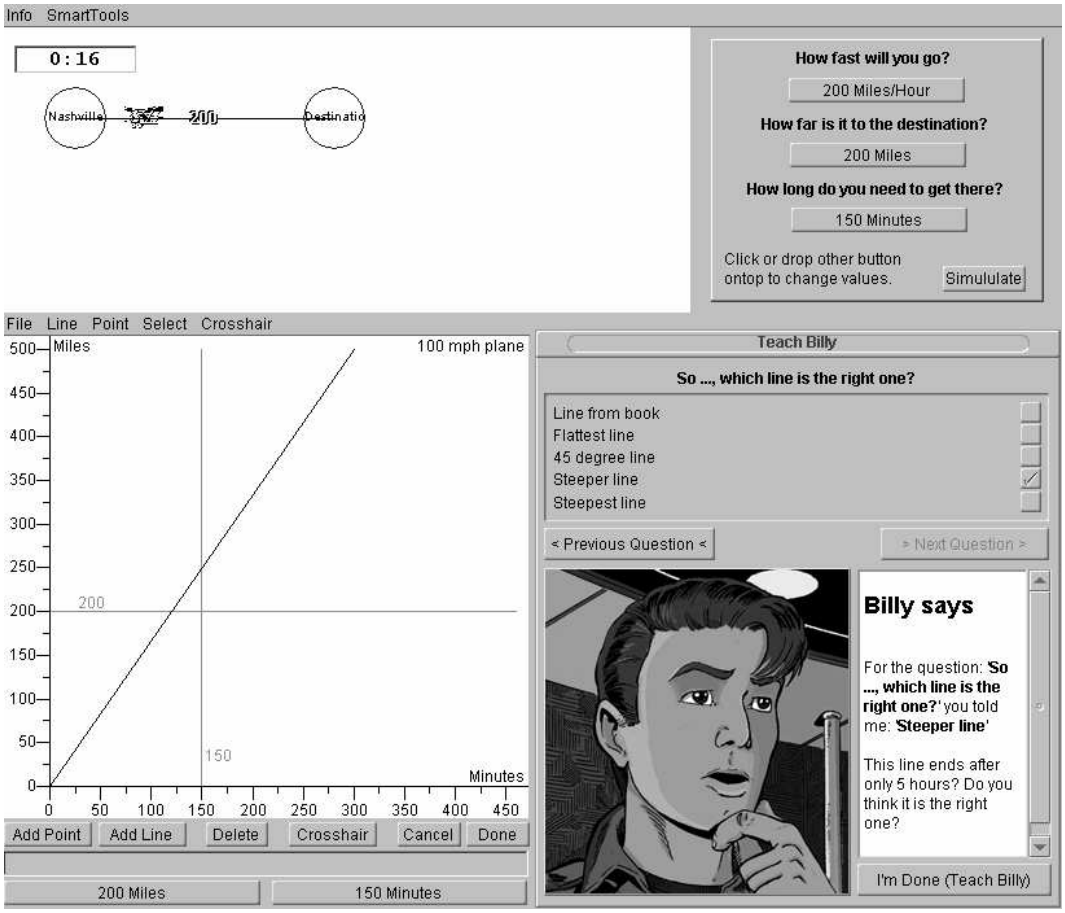

Figure 4: The Teach Billy Interface

When students are satisfied with their instruction, they continue the cycle with *See Billy's Behavior*. For this step, a new window appears with a list of questions (not shown). Billy's problem solving behavior is directly related to what he has been taught. The student can click on a question, and Billy's solution to the problem is displayed in a text-box under the list. We have designed and implemented a rule-set that can evaluate the current GraphTool and generate relevant answers. The student can also use the same graph that Billy used to run simulations and check results. Along with the solutions Billy also summarizes the number of right and wrong answers, and comments made by the teacher and other students about his solutions. Student uses this information as feedback. They can reflect on what they have taught, and attempt to make corrections and improvements if needed. This cycle continues until the student has satisfactorily taught Billy. Then the student *Moves to the Next Problem*, i.e., the next challenge.

4. Studies

Predecessors of the current version of our Teachable Agent system were tested in two pilot studies to assess potential benefits and implications for the design of teachable agents. The first study was conducted on undergraduate students in Cognitive Science at Vanderbilt University to verify ease of use and identify weaknesses in the system. Students solved the Rescue at Boone's Meadow adventure [1]. The optimal solution required a plan, where an ultralight-airplane and truck operate concurrently to rescue a wounded eagle found in the woods. After a brief tutorial, students worked in pairs, and had an hour to derive the problem solution using AdventurePlayer. To our surprise, most of the students failed to derive the optimal solution. Only one of the six groups derived a plan where the vehicles moved concurrently. In a previous, more extensive study with sixth-grade students in a Kentucky school [3], students provided with more detailed instruction and practice, did derive the optimal solution. This was mainly because of the presence of a coach agent, which provided students who had a correct initial solution, with hints on achieving a better solution, and eventually an optimal solution.

An improved version of the system with better SmartTool interfaces was used in a follow-up study on a group of middle-school students. The introduction of the mini-assessment environment and the GraphTool, proved to be much more successful. Students worked in pairs to solve a set of problems that dealt with the delivery of packages using airplanes (250 mph, 300 mph) from a source to a destination city in the quickest possible time. The focus was on using the SmartTools to solve multiple problems efficiently. For this, students were shown how they could use the mini-assessment-environment (Figure 4, top) to perform experiments, collect data, and use that data to generate graphs. After this phase, students worked on their own using the SmartTools they had created to solve the package delivery problems. We ran this study on 6 groups, and all of them successfully derived the answers with little help from the experimenters. The feedback on the system was very positive. It was heartening to see that students wanted this system installed on computers in their classroom so they could use it for their everyday assignments.

We played the role of human tutors, and this provided us with valuable information on the design of computer-based teachable agents. We found a significant variation in the procedures students used to create graphs. Some differences were switched axes, different minimum and maximum values on the axes, varying verbosity in specifying axis-labels and auxiliary graph-content, like additional points. Another problem was rounding errors introduced by drawing lines, which resulted in variation in the slopes of the graphs. We began to realize that this would complicate the design of computer-based agents in assessing the correctness and usefulness of the graphs that students created. To overcome these variabilities, and to make it easier to design the software system, we adopted a multiple-choice format to simplify the interaction between the computer agent, Billy and the student. Preliminary pilot studies indicate that students have much less trouble in using the tools and understanding the interfaces, therefore, they can concentrate more on the teaching and learning processes.

5. Conclusions

This paper has presented the evolution of our work from simulation-based Intelligent Learning Environments to Teachable Agents. Intelligent Learning Environments with a simulation tool and coaching component, successfully assisted students in constructive and generative learning of complex mathematics problems. The introduction of SmartTools into this environment provided mechanisms for generalization and transfer of knowledge across multiple problem solving episodes and different domains. Introducing Teachable Agents into these exploratory environments enhances our instructional paradigm to the concept of learning *by*

teaching. Our studies [9] provide supporting evidence that creating situations where students think they are teaching others and helping them learn, increases their own motivation to learn and gain deeper understanding of the subject material. In future work, we hope to conduct detailed experimental studies on our system to demonstrate the beneficial learning characteristics in our student teachers.

References

[1] CTGV, Cognition and Technology Group at Vanderbilt (1997). The Jasper project: Lessons in curriculum, instruction, assessment, and professional development, Mahwah, NJ, Erlbaum

[2] CTGV, Cognition and Technology Group at Vanderbilt (1990). Anchored instruction and its relationship to situated cognition, *Educational Researche*r, 19(6), pp. 2-10

[3] Crews T., Biswas G., Goldman S., and Bransford J. D. (1997). Anchored interactive learning environments, *International Journal of AI in Education vol. 8*

[4] Owens, S., Biswas, G., Nathan, M., Zech, L., Bransford, J. D., and Goldman, S. (1995). Smart Tools, A multi-representational approach to teaching functional relationships, *International Conference on AI in Education, AIED'95, Washington D.C.*, pp. 598

[5] Balac N., Katzlberger T., and Leelawong K. (1998). Agent-enhanced intelligent learning environment, Technical Report, Dept. of Computer Science, Vanderbilt University

[6] Burton, R., & Brown, J. (1982) An investigation of computer coaching for informal learning activities. In Sleeman, D. & Brown, J. (Eds.) Intelligent Tutoring Systems, Academic Press, London.

[7] CTGV, Cognition and Technology Group at Vanderbilt. (in press). Adventures in Anchored Instruction: Lessons from Beyond the Ivory Tower.

[8] Bransford J. D., Brophy S., and Williams S., (2000). When computer technologies meet the learning sciences: Issues and opportunities, *Journal of Applied Developmental Psychology*, Volume 21, Issue 1, pp. 59-84

[9] Biswas, G., Schwartz, D., Bransford J. D. and The Teachable Agent Group at Vanderbilt (TAG-V) (2001). Technology Support for Complex Problem Solving: From SAD Environments to AI, in K. D. Forbus and P. J. Feltorich (eds.), Smart Machines in Education: The Coming Revolution in Educational Technology, AAAI/ MIT Press, Metro Park, California, USA

[10] Webb, N. M. (1983). Predicting learning from student interaction: Defining the interaction variables, *Educational Psychologist*, 18, 33-41

[11] King A. (1998). Transactive peer tutoring: Distributing cognition and metacognition, *Educational Psychology Review*, 10, 57-74

[12] Palinscar, A. S. and Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and instruction*, 1, 117-175

[13] Chan T. and Chou C. (1997). Exploring the design of computer supports for reciprocal tutoring, International Journal of Artificial Intelligence in education, Volume 8, pp. 1-29

[14] Chi, M. T. H., Siler, S. A., Jeong, H. (2000). Learning from human tutoring, to appear in *Cognitive Science.* (http://www.pitt.edu/~chi)

[15] Huffman, S. B. (1994). Instructable Autonomous Agents, Ph.D. Thesis, University of Michigan, Dept of Electrical Engineering and Computer Science.

[16] Frasson, C., Mengelle, T., Aïmeur, E., Gouardères, G. (1996) "An Actor-based Architecture for Intelligent Tutoring Systems", *ITS'96 Conference*, Lecture Notes in Computer Science, No 1086, Springer Verlag, Montréal, pp. 57-65.

[17] Bransford, J.D., Zech, L.K., Schwartz, D.L, Barron, B. Vye, N., and the Cognition and Technology Group at Vanderbilt (2000). Designs for Environments that Invite and Sustain Mathematical Thinking. On P. Cobb, E. Yackel and K. McClain (Eds.). Symbolizing and Communicating in Mathematics Classrooms: Perspectives on Discourse, Tools and Instructional Design. 275-324. Mahwah, New Jersey. Lawrence Erlbaum Associates

[18] Lajoie, S.P., and Derry, S.J. (eds.) (1993). *Computers as Cognitive Tools*, Hillsdale, NJ: Lawrence Erlbaum.

[19] Brophy, S., Biswas, G., Katzlberger, T., Bransford, J. D., and Schwartz, D. (1999). Teachable Agents: Combining insights from learning theory and computer science. In *Artificial Intelligence in Education*., S.P. Lajoie and M. Vivet, (Eds.), 21-28. Amsterdam, The Netherlands: IOS Press.

[20] Novak, J. D. (1998), Learning, Creating, and Using Knowledge: Concept Maps as Facilitative Tools in Schools and Cooperations. Hillsdale, NJ: Lawrence Erlbaum

[21] Schwartz, D.L., Lin, X., Brophy, S., and Bransford, J.D. (1999). Towards the development of flexibly adaptive instructional design, In C. M. Reigeluth (Ed.), *Instructional design theory and models: A new paradigm of instructional theory*, 2, 183-213, Marwah, NJ: Erlbaum