Virtual Environments for Training

Quarterly Status Report
(11)

April 1 through June 30, 1998

Submitted by R. Stiles
Lockheed Martin
Advanced Technology Center

Prepared for
Office of Naval Research
Contract N00014-95-C-0179
September 1998
(CLIN 0004, CDRL A001 Progress Report)

Abstract: This report describes the Lockheed-Martin VET team efforts and accomplishments during the eleventh quarter of the contract. Activity is reported for each of the software components of the Training Studio: VIVIDS, Steve, and Vista, as well as domain development and evaluation study. This report contains material submitted for subcontracts by Dr. Allen Munro at USC/BTL, Dr. Lewis Johnson at USC/ISI.

Progress on productization of the VET Training Studio software includes increased robustness for Vista virtual environment display and interaction services, a new capability to use the STEVE visual representation within VIVIDS, and improved visual and spoken dialog capabilities for STEVE.

1   Summary

This report describes efforts for productization of the Training Studio software for the Virtual Environments for Training contract during the period from April 1 - June 30, 1998.

The Lockheed Martin Advanced Technology Center oversaw productization for itself and subcontractors, releasing several improvements to Vista for speeding up 3D text display and modification, and selective loading of inlined 3D models. As part of productization, standalone Performer VRML libraries were updated for eventual release to our subcontractors and select government agencies.

At Behavioral Technology, research and development during this quarter has included efforts in the following areas:

- Student interface improvements.

- Additional improvements in the integration of a VET-featured VIVIDS with the other VET components, including Vista, autonomous agents, TrishTalk, and the VET sound server.

- Improved *participant* handling in VIVIDS instruction.

- Allowing the instruction author to use a 'directable' Steve for certain types of instructional remediation within VIVIDS structured lessons.

During this quarter, ISI continued research and development on their pedagogical agent, Steve, greatly improving visual and spokent dialog with the student. In addition, Ben Moore continued improving the speech recognition component that allows people to communicate with agents in the virtual environment, making a release of this Java-based interface, RecApple, just prior to our July 9 ONR demo in Arlington, VA.

2   Introduction

This report describes the efforts of Lockheed Martin, USC/ISI, and USC/BTL for the Virtual Environments for Training contract during the period from April 1 - June 30, 1998. The purpose of our work is to explore, develop, and evaluate novel techniques for incorporating automated individual and team instruction in virtual environments.

The ATC team extended the Vista Viewer capabilities for human-computer interaction in a networked, real-time immersive training environment, continued work to optimize the Vista software, and supported the development requirements of USC colleagues at ISI and BTL. Efforts toward productization of the Training Studio were increased during this period.

The Lockheed Martin team members for the VET project are: Randy Stiles (Program Manager), Sandeep Tewari, Mihir Mehta, and Laurie McCarthy.

At ISI, the STEVE pedagogical agent visual and spoken dialog capabilities have been improved. Steve is now much more responsive to the student. To achieve this, the building blocks for Steve's dialogue have been decomposed into smaller pieces. This decomposition allows Steve to respond to interruptions -- including changes in the virtual world as well as interruptions from the student -- more frequently.

During the second quarter of 1998 the USC/ISI team consisted of the following individuals: Dr. Lewis Johnson (principal investigator), Dr. Jeff Rickel (research scientist), Mr. Marcus Thiebaux (programmer). The project was also assisted by Richard Angros (a graduate student), Ben Moore, and Anna Romero (undergraduate students), all working on the AASERT grant associated with the VET project. Romero stopped working on the project in May, and Moore returned to the project in late May

The authoring system for building simulation behaviors and structured tutorials for virtual environments is called VIVIDS (Virtual

Interactive Intelligent Tutoring System Development Shell). During this quarter of the contract productization efforts have been undertaken to ensure the robustness, completeness, and the openness of the authoring and delivery system and of the Gas Turbine Engine simulation constructed using VIVIDS.

During the second year of this project, the USC/BTL team for the VET project consisted of the following individuals: Dr. Allen Munro (Principal Investigator), Dr. Quentin Pizzini, and David Feldon.

3 Methods, Assumptions & Procedures

We have been conducting a number of research investigations, each of which is directed at one or more of the objectives mentioned in the introduction. For each of the three system components, one or more members of the primary research team, in collaboration conduct these investigations with the other VET project participants.

3.1 Advanced Technology Center

The research effort at the ATC has operated on several hypothesis; 1) a component-based virtual environment architecture can support the integration of pedagogical agents and simulation-based training 2) immersed, networked interaction with 3D (VRML) models can be isolated to the virtual environment interaction component (Vista) 3) instructional interaction in a virtual environment can be specific to each participant, supporting team training and still accomplishing individual remediation.

The ATC approach focuses on providing those capabilities that accomplish communications and scene display and manipulation for Steve and VIVIDS, as well as optimizing human interactions within the virtual environment. New capabilities are developed, tested, and released in a fast cycle to collaborating VET organizations for further evaluation and critique. Other capabilities are developed in response to a direct request by one of the other team members; or provided as a solution to a

problem encountered by one of the collaborators.

3.2 Information Sciences Institute

USC/ISI's focus has been on incorporating pedagogical capabilities in an intelligent agent architecture called Steve. We are investigating the following hypotheses: 1) that an agent architecture and knowledge representation can be developed that permits autonomous agents to act as guides, mentors, and team members, 2) that machine learning and high level languages can be employed to assist instruction developers in creating agent-based instruction, and 3) virtual environment technology enables new types of interactions between trainees and instructional systems, which improve the quality of instruction provided by the instructional systems.

USC/ISI research methodology is as follows. We identify a new capability that, if incorporated into Steve, would contribute to validating one of our research hypotheses. We then design a set of extensions to the Steve system that implements the capability. We develop a prototype implementation of the capability, and conduct a series of demonstrations and in-house tests. We then make arrangements for further evaluation of the capabilities by our partner organizations or ourselves.

3.3 Behavioral Technology Laboratory

The VET research effort at Behavioral Technology Laboratory (USC), previously demonstrated the correctness of the hypothesis that the 2D behavior authoring interface of RIDES can be adapted and extended to provide an effective and natural way to specify simulations for virtual environment training. The VIVIDS authoring system constitutes the first system for *authoring* (as opposed to *programming*) robust complex interactive simulations for virtual environments. Furthermore, these authored simulations have features that support the near-automatic construction of certain types of structured tutorials. The combination of productive simulation authoring with efficient tutorial

development is designed to make feasible the application of virtual environment technologies to a very wide range of technical training requirements at reasonable cost. Extensions to the original VIVIDS system permit several levels of collaboration with agents and support team training in virtual environments.

USC/BTL methodology has been to progressively adapt VIVIDS functionalities to provide appropriate simulation and instruction services for a virtual environment delivered by Vista, to provide services to the Steve autonomous agent, and to exploit appropriately the speech (TrishTalk) and sound capabilities of the VET environment. Developing large simulations and instruction materials using the revised authoring tools tests these new capabilities. Two levels of formative evaluation are pursued: both the usability of the revised authoring system and the functionality of the tutorials it produces must be examined. Based on, first, in-house evaluations, and, after initial revisions, the evaluations of our research partners at Lockheed Martin, at USC's Information Sciences Institute, and at the U.S. Air Force Laboratory, further modifications are made, and the tool-development, authoring and testing cycle resumes.

4   Results and Discussion

This section covers the results accomplished during this reporting period and discusses the significance of this work in terms of the VET project goals as well as contributions to respective research communities at large.

4.1   Software Development

Lockheed Martin, USC/BTL, and USC/ISI each accomplished major milestones regarding development of their respective components: Vista Viewer, VIVIDS, and Steve.

4.1.1   Simulation-based Training

This section describes the research and development efforts with respect to the VIVIDS component. Vivids changes were focused on: improvements in the integration of a VET-featured VIVIDS with the other VET components, enhancing the Gas Turbine Engine (GTE) control system simulation, improving the immersed student interface, and 'productizing' the prototype VIVIDS for improved operation in VET systems.

4.1.1.1   Student Interface Improvements

The graphical user interface that supports student commands in VET during VIVIDS instruction was significantly improved. Contributing developments include the addition of clear text labels rather than obscure icons, and new instructional commands: Jump to viewpoint and Repeat last utterance

Previously, the instructional command interface was composed of an opaque palette with three icons, the meanings of which were not always clear to students. The same icon had different meanings depending on the instructional mode being used. When a structured lesson was being presented, for example, a question-mark icon meant, "I don't know the answer. Show me." When the student was engaged in the free-play/browsing mode, however, the question-mark icon meant, "I'm about to touch an object for which I'd like to see available textual information." Using text labels on the palette for the available commands eliminated these sources of confusion.

New instructional commands were added to the palette, including "Change Viewpoint" (jump to the next viewpoint in a previously authored list of viewpoints) and "Repeat Text" (repeat the last thing that was said using the VET speech output system).

4.1.1.2   Improved Component Integration

VIVIDS now collaborates more effectively with Vista, Steve, TrishTalk, and the sound server, by actively tracking their presence in the instructional environment. VIVIDS is aware of the *Activity State* of these components and maintains an internal representation of the active components in the VET instructional environment. If, for example, TrishTalk were not present, the old VIVIDS could hang indefinitely while waiting for a signal that Trish had finished

speaking an instructional utterance that VIVIDS had sent to it. This kind of problem can now be avoided for any case in which a collaborating component exits normally. When a particular component (such as the speech server) is absent, VIVIDS does not attempt to access that component. (Of course, if a collaborating component crashes, VIVIDS is not aware of its absence.)

4.1.1.3 *Improved* Participant *Handling.*

VIVIDS now automatically retrieves participant names from system information. In previous releases, it was necessary to enter the names of the participants in a team training exercise into each computer.

4.1.1.4 *Using Steve to Enhance VIVIDS Instruction*

VIVIDS can now exploit the Soar-independent Steve provided by our colleagues at ISI. This makes possible a new mode of instruction that can take advantage of the visual cues provided by using a graphically embodied agent in the rapidly authorable structured exercises that can be created with VIVIDS. Authors can specify that certain instructional items can make use of Steve for presentations or for remediations. The VIVIDS instructional item types that support the optional use of Steve are:

- *Highlight Item.* When Steve is used to highlight an item, he moves to the item and points at it with his right hand.
- *Control Item - Remediation.* When a student fails to perform a required control manipulation, VIVIDS can now demonstrate the correct action by having Steve move to the control and carry out the action, using his left hand.
- *Indicator Item - Remediation.* If a student fails to make an indicator observation correctly, authors can specify a remediation in which Steve moves to the indicator, points to it, and tells the student what value it displays.
- *Goal Item - Remediation.* When goals are not achieved, authors can

use Steve to carry out a set of actions that achieves the goal, and they can have Steve point out those aspects of the new situation that indicate that the goal has been achieved.

Steve can also be made to follow a path from one object to another without passing through walls. Sample training materials (lessons) were developed in the content of the Gas Turbine Engine simulation to test the automatic use of the 'directable' Steve in VIVIDS structured lessons.

4.1.2 Pedagogical Agent Development

This section relates improvements to capabilities of the Steve Pedagogical Agent in the use of dialog-centered speech, motor control in a complex graphical setting, Steve's graphical representation, and task authoring. It also relates ISI efforts in developing a sound server.

4.1.2.1 *Dialogue*

During this quarter, Rickel made significant improvements in Steve's dialogue capabilities, enhancing the timing and completeness of visual and spoken dialog with the student.

Steve is now much more responsive to the student. To achieve this, Rickel decomposed the building blocks for Steve's dialogue into smaller pieces. This decomposition allows Steve to respond to interruptions -- including changes in the virtual world as well as interruptions from the student -- more frequently.

Steve's ability to take turns during a dialogue has been improved: 1) after telling or showing the student something, Steve now looks at the student and pauses (to give the student an opening) before continuing, and 2) Steve now recognizes when a student begins speaking and looks at the student while listening until the speech is complete. Previously, Steve could only detect when the student finished saying something.

Steve's dialogue capabilities are more robust than before: he now tries to respond to all

requests and questions, even if they come at unusual times.

Steve's text generation has been improved: his answers to questions are less terse than before.

Moore improved Steve's use of intonation: after some experimentation, we were able to make his intonation and spoken emphasis sound much more natural by raising the upper limit on his pitch range. Moore is also studying other ways of improving Steve's prosody.

Rickel is supervising a Master's student this summer, Sung-Oh Jung, who is investigating the addition of plan recognition capabilities to Steve. If Steve had a better ability to assess the student's current intentions and plans, he could better tailor his suggestions and feedback. (Sung-Oh is not currently receiving any financial support for his work.)

We view Steve's dialogue capabilities as a crucial area for future research, and will be continuing work in this area under the VET AASERT grant.

4.1.2.2 Motor Control

During this quarter, we also improved Steve's motor control capabilities.

Thiebaux added arms to Steve's body and designed and implemented the animation primitives for the arms. These animation primitives are simpler and more efficient than approaches based on inverse kinematics, yet they are very robust and effective, providing very natural-looking motion for our purposes. He also added hands with more degrees of freedom and smoother animation than our old approach. Rickel integrated these new body parts into Steve's motor control module. The result is a much more natural-looking virtual human. Response from everyone who has seen the new body has been very positive.

Rickel and Thiebaux extended Steve so that he now attaches himself to his student's display when he first comes up and when he is monitoring the student. This allows him to

follow the student around. This technique was previously used with the old Inventor body, but hadn't been possible with the new VRML body until now. Also, unlike the old Inventor body, Steve can now look around when attached to the student's display.

As her senior project under the supervision of Rickel, Anna Romero completed a web-based interface for experimenting with Steve's facial expressions. The interface allows people to create new facial expressions by using sliders to manipulate Steve's eyes, eyelids, eyebrows, and lips, and she also created a set of expressions covering a range of different emotions that could be useful in a tutorial context. Although these expressions have not yet been integrated into Steve's tutorial behavior, Anna's work lays the foundation for such extensions.

Marcus Thiebaux has packaged up and documented his code for providing human figure animation in Vista. This code provides low-level animation primitives for controlling Steve's body. Currently, the code is driven by Steve's motor control module. However, by making this code available to our other VET partners, they can make use of Steve-like agents for their own purposes. In particular, Pizzini of BTL is adding code to allow VIVIDS to control a Steve-like agent to guide the student around during familiarization lessons. Although their agent will not be as intelligent or reactive as Steve, they feel that this approach will be more effective than their previous methods for guiding the student and highlighting objects.

4.1.2.3 Preparing for Field Use

As the VET project draws to a close, our focus has shifted towards making Steve robust enough for eventual field use at the end of the project. We continued our progress in that direction this quarter. Rickel has focused on testing Steve under a wide range of conditions, and has made some performance optimizations. Thiebaux has also been making performance optimizations on his code to control the animation of Steve's body. We have run some informal evaluations to assess usability issues, and we expect to

continue such experiments. To our documentation on how to use Steve, Rickel added documentation on authoring Steve. Angros completed a tutorial on how to author Steve by demonstration, and he is designing an evaluation of his dissertation work. Finally, Rickel made releases of Steve, the graphical models for Steve, and TrishTalk (the text-to-speech component) to all our VET partners. We expect to make one more release of Steve during the next quarter; that will serve as the final release under VET funding, and any subsequent releases will be based on the work of students under VET AASERT funding.

4.1.2.4 Domain Development

Our work on domain development is winding down, but we did perform some work in that area during this quarter. We spent significant time testing Steve in both the HPAC and GTE environments, including working with students on individual tasks and with students and agents on team tasks. We made some improvements to Steve based on this testing, and we also extended Steve's domain knowledge in some places, notably his knowledge of where to stand relative to particular objects and some of his text strings for describing task steps.

4.1.3 Virtual Environment Interaction

Virtual environment interaction, where 3D scenes and objects are displayed and used in real-time, is the ATC's primary technical area. The interaction capability is realized in the Vista software component. During this quarter, Vista development centered on fleshing out text display and library support for agent optimization.

4.1.3.1 External Library Functions

Two new external library functions were developed for USC/ISI colleagues to use/test for controlling VRML models and to speed up agent rendering. The first of these is a selective inline capability that can be used to add to a VRML scene. The second is a new node name lookup that more efficiently finds node pointers used during figure animation.

4.1.3.2 Dynamic 3D Text

Vista development included improved dynamic 3D text capabilities, including new support for the VRML text node, resulting from an example from colleagues at Brooks Air Force Base as well as Tewari's work on elimination of multiple pfFont initializations. The improved speed for changing text allowed significantly faster rendering of text menus.

4.1.3.3 GTE Model

During this quarter, a new version of the GTE models was released. The updated model provided invisible waypoint objects which to allow Steve to travel while near the pipes in the engine room. Also included were proximity sensors that control viewpoint snapping turned off, so that you can fly outside or above of the engine room and central control station without getting popped back to a reference location. This is useful to see from above, or pulling a little further out, switching off a lot of geometry to test the update rate for multiple Steve agent visual representations.

4.1.3.4 Profiler

As part of the productization task, modification to the Profiler was begun to allow all Training Studio processes to be controlled from one central interface. As part of this effort, Vista's response to TScript vrStop was modified. Now, vrStop must be qualified by participant to stop Vista; i.e., vrStop all will stop everything, but vrStop vista <participant> will only stop the Vista that user/student <participant> is using, while vrStop vista all will stop all Vistas.

4.1.4 Productization Efforts

As discussed in the last report, productization is a major task during Option 2 of the VET contract and underlies the design and development of the system extensions during this last phase. This quarter saw continued development and refinement of interfaces to each of Vista, VIVIDS, and Steve components for authoring. These efforts are described in the separate descriptions of each component in sections 4.1.1 - 4.1.3 of this report. Component

documentation has been collected and organized into a draft reference manual. This documentation will be reviewed and revised by all team members at a planning meeting scheduled for July/early August. The final version of the manual will provide both the descriptions and instructions of Training Studio use and operations.

4.2 Meetings

In preparation for our July 9 demo to ONR in Washington DC, Randy Stiles of the LM ATC met with Ben Moore of USC/ISI to prepare the Maximum Impact for use standalone at ONR, test a new revision of the Java-based speech recognition interface RecAppl, and test the standalone network configuration of 3 workstations for the demo.

4.3 Presentations and Publications

McCarthy presented Enabling Team Training in Virtual Environments at the Collaborative Virtual Environments (CVE) 98 conference in Manchester, UK. The paper was the result of collaboration with Stiles, Johnson, and Rickel. The presentation included an updated video of our most recent developments. The authors have been invited to extend the paper for journal publication in a special issue of the Virtual Reality: Research, Developments and Applications.

Steve was featured in several papers and presentations during this quarter. Rickel and Johnson revised and submitted the final version of their paper for the journal Applied Artificial Intelligence (included with this report). Rickel and Johnson revised and submitted the final version of their paper for the AAAI workshop on Multi-Modal Human-Computer Interaction.

Steve was featured in a survey paper by Elliott and Brzezinski in AI Magazine 19(2). Rickel and Johnson submitted a paper to the ITS '98 Workshop on Pedagogical Agents. (Both Rickel and Johnson are on the program committee.) Rickel and Johnson submitted a paper to the First Workshop on Embodied Conversational Characters.

Johnson gave a talk at the Second International Conference on Autonomous Agents. Johnson gave an invited talk at the International Workshop on Interaction Agents in L'Aquila, Italy. Rickel gave an invited talk at the Virtual Humans 3 conference. Rickel gave a talk at USC's Center for Scholarly Technology. Both Johnson and Angros gave talks at the Soar Workshop.

5 Conclusions

Progress continued over the 11th quarter with special emphasis on productization of the system. Development of authoring tools is important during this phase and efforts in this area are underway for each Training Studio component. Final documentation and manuals are being developed and the video produced July 1997 has been updated with research results through March 1998. A final updated video is being planned.

The next quarter will complete the development effort on the Training Studio. Demonstration of the system to date will be presented at ONR in July. The ATC focus is on ensuring smooth closure and completion of contract deliverables. Plans for closure will be made during a development team meeting to be held in the upcoming quarter.

During the next quarter, ISI plans to complete work under the VET grant and continue research via students under AASERT funding. ISIÃs main focus under the VET funding will be to support a demo at ONR on July 9, make a final release of all VET software, and complete documentation and final report. In addition, Rickel will give an invited talk in London at the Virtual Reality for Education and Training conference in July, and will demo Steve at AAAI later that month. Under AASERT funds, USC/ISI will continue research and development on pedagogical agents for training in virtual reality through their students Ben Moore, Richard Angros, and our new Ph.D. student, Taylor Raines.

In the coming quarter, USC/BTL efforts on the VET project will be completed. Ongoing VIVIDS instruction enhancements will be tested in the context of the VET Gas Turbine

Engine control systems trainer.   In collaboration with colleagues at ISI and at Lockheed Martin, we will contribute to a Final Report on the VET project.

During the next quarter, the ATC will oversee the packaging of the Training Studio components, drafting of a user's guide, final report, and summary video, as well as exploration of transition opportunities.

Symbols, Abbreviations & Acronyms

AAAI   American Association for Artificial Intelligence

AASERT   ONR Grant for graduate student development at USC/BTL and USC/ISI associated with VET contract

AFHRL   U.S. Air Force Human Resources Laboratory

ATC   Advanced Technology Center, located In Palo Alto, CA, part of Lockheed martin Missiles & Space

BTL   Behavioral Technologies Laboratories, located in Redondo Beach, CA, a performing organization in the Lockheed Martin VET effort, a laboratory of the University of Southern California.

COTR   Contracting Office Technical Representative. The Program Manager or Program Officer from the funding agency who provides technical direction for the program.

DIS   Distributed Interactive Simulation, a real-time distributed message protocol used in training and operational simulations developed by ARPA and now an International Standards Organization standard.

GTE   Gas Turbine Engine - similar to jet engine, which drives propulsion of a Navy Ship. In our case we are usually referring to the LM2500 Gas Turbine Engine on USS Arleigh Burke (DDG-51) ships.

HPAC   High Pressure Air Compressor, an oil-free air compressing system prevalent on many navy vessels, which prepares compressed air for gas turbine engines.

ICAI   Intelligent Computer Aided Instruction, a method of instruction whereby an intelligent model of a student's understanding is used to guide a student during instruction using a computer.

IPEM   Integrated Planning, Execution and Monitoring architecture for coordinating different planning strategies as required for SOAR activities.

ISI   Information Sciences Institute in Marina del Rey, CA, a performing organization in the Lockheed Martin VET effort, affiliated with the University of Southern California in Los Angeles, CA.

ITS   Intelligent Tutoring Systems Ò an AI approach where the student, domain, and instructional techniques are modeled and used to actively instruct the student

MCO   Multi-Channel Option for Silicon Graphics Onyx Workstations, a necessary option to provide separate video channels used in immersive virtual environment displays.

ONR   Office of Naval Research, the funding agency for the VET effort.

RIDES   Rapid Instructional Development for Educational Simulation

SIGART   Special Interest Group on Artificial Intelligence

SGI   Silicon Graphics Incorporated, a workstation company whose whole culture centers around fast 3D graphics.

SOAR   A platform independent, cognitive architecture based on a production system which seeks to address those capabilities required of a general intelligent agent.

STEVE   SOAR Training Expert for Virtual Environments

Tcl/Tk   A windowing interface toolkit assembled around a UNIX-shell like interpreter originally developed at UC Berkeley.

TScript   Training Script message protocol for virtual environments

URL   Uniform resource locator, a tag that indicates a media format and location on the Internet as part of the World Wide Web.

USC   The University of Southern California.

VE   Virtual Environment, a 3D visual display and accompanying simulation which represent some aspect of an environment. Expanded forms of VE also address other senses such as audio, touch, etc.

VET   Virtual Environments for Training, a Defense Department focused research initiative concerned with applying virtual environment technology to training

VR   Virtual Reality *see Virtual Environment*

VRIDES   Virtual Rapid Instructional Development for Educational Simulation. A special version of the RIDES program for use in developing simulations and tutorials that collaborate with Vista Viewer and Soar to deliver training in virtual environments.

VIVIDS   *See VRIDES above*

VRML   Virtual Reality Modeling Language, an analog to HTML used for documents, but focused on 3D objects and scenes for the World Wide Web.

WWW   World-Wide Web, a system incorporating the HTTP message protocol and the HTML document description language that allows global hypertext over the Internet.

Appendix A

Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control

To appear in
Applied Artificial Intelligence

Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control

Jeff Rickel and W. Lewis Johnson
Information Sciences Institute & Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292-6695
rickel@isi.edu, johnson@isi.edu
http://www.isi.edu/isd/VET/vet.html

May 4, 1998

Abstract

This paper describes Steve, an animated agent that helps students learn to perform physical, procedural tasks. The student and Steve cohabit a three-dimensional, simulated mock-up of the student's work environment. Steve can demonstrate how to perform tasks and can also monitor students while they practice tasks, providing assistance when needed. This paper describes Steve's architecture in detail, including perception, cognition, and motor control. The perception module monitors the state of the virtual world, maintains a coherent representation of it, and provides this information to the cognition and motor control modules. The cognition module interprets its perceptual input, chooses appropriate goals, constructs and executes plans to achieve those goals, and sends out motor commands. The motor control module implements these motor commands, controlling Steve's voice, locomotion, gaze, and gestures, and allowing Steve to manipulate objects in the virtual world.

1 Introduction

To master complex tasks, such as operating complicated machinery, people need hands-on experience facing a wide range of situations. They also need a mentor that can demonstrate procedures, answer questions, and monitor their performance, and they may need teammates if their task requires multiple people. Since it is often impractical to provide such training on real equipment, we are exploring the use of virtual reality instead; training takes place in a three-dimensional, interactive, simulated mock-up of the student's work environment. Since mentors and teammates are often unavailable when the student needs them, we are developing an autonomous, animated agent that can play these roles. The agent's name is Steve (Soar Training Expert for Virtual Environments).

Steve integrates methods from three primary research areas: intelligent tutoring systems, computer graphics, and agent architectures. This novel combination results in a unique set of capabilities. Steve has many pedagogical capabilities one would expect of an intelligent tutoring system. For example, he can answer questions such as "What should I do next?" and "Why?". However, because he has an animated body, and cohabits the virtual world with students, he can provide more human-like assistance than previous disembodied tutors. For example, he can demonstrate actions, use gaze and gestures to direct

the student's attention, and guide the student around the virtual world. Virtual reality is an important application area for artificial intelligence because it allows more human-like interactions among synthetic agents and humans than desktop interfaces can. Finally, Steve's agent architecture allows him to robustly handle a dynamic virtual world, potentially populated with people and other agents; he continually monitors the state of the virtual world, always maintaining a plan for completing his current task, and revising the plan to handle unexpected events.

Steve consists of a set of domain-independent capabilities that utilize a declarative representation of domain knowledge. To teach students about the tasks in a new domain, someone must provide the appropriate domain knowledge. We assume that this person will be a course author, a person with enough domain knowledge to create a course for teaching others. Importantly, we do not assume that this person has any programming skills. Ensuring that Steve only relies on types of knowledge that a course author can provide imposes strong constraints on Steve's design.

Steve is designed to coexist with other people and agents in a virtual world. Our goal is to support team training, where teams of people, possibly at different locations, can inhabit the same virtual world and learn to perform tasks as a team. Agents like Steve can play two roles in such training: they can serve as tutors for individual team members, and they can play the role of missing team members. We have recently extended Steve to understand team tasks and function as a team member. We will not address those issues in this paper; here, we focus primarily on Steve's ability to work with a single student on a one-person task. However, as will become clear, ensuring that Steve can function in an environment with other people and agents has placed important constraints on Steve's design.

This paper describes Steve's architecture in detail, including perception, cognition, and motor control. First, Section 2 illustrates Steve's capabilities via an example of Steve and a student working together on a task. Next, as background, Section 3 briefly describes the larger software architecture for virtual worlds of which Steve is a part; more detail is available in an earlier paper (Johnson *et al.* 1998). Finally, Section 4 gives an overview of Steve's architecture, and the remainder of the paper provides the details.

2 Steve's Capabilities

To illustrate Steve's capabilities, suppose Steve is demonstrating how to inspect a high-pressure air compressor aboard a ship. The student's head-mounted display gives her a three-dimensional view of her shipboard surroundings, which include the compressor in front of her and Steve at her side. As she moves or turns her head, her view changes accordingly. Her head-mounted display is equipped with a microphone to allow her to speak to Steve.

After introducing the task, Steve begins the demonstration. "I will now check the oil level," Steve says, and he moves over to the dipstick. Steve looks down at the dipstick, points at it, looks back at the student, and says "First, pull out the dipstick." Steve pulls it out (see Figure 1). Pointing at the level indicator, Steve says "Now we can check the oil level on the dipstick. As you can see, the oil level is normal." To finish the subtask, Steve says "Next, insert the dipstick" and he pushes it back in.

Continuing the demonstration, Steve says "Make sure all the cut-out valves are open." Looking at the cut-out valves, Steve sees that all of them are already open except one. Pointing to it, he says "Open cut-out valve three," and he opens it.
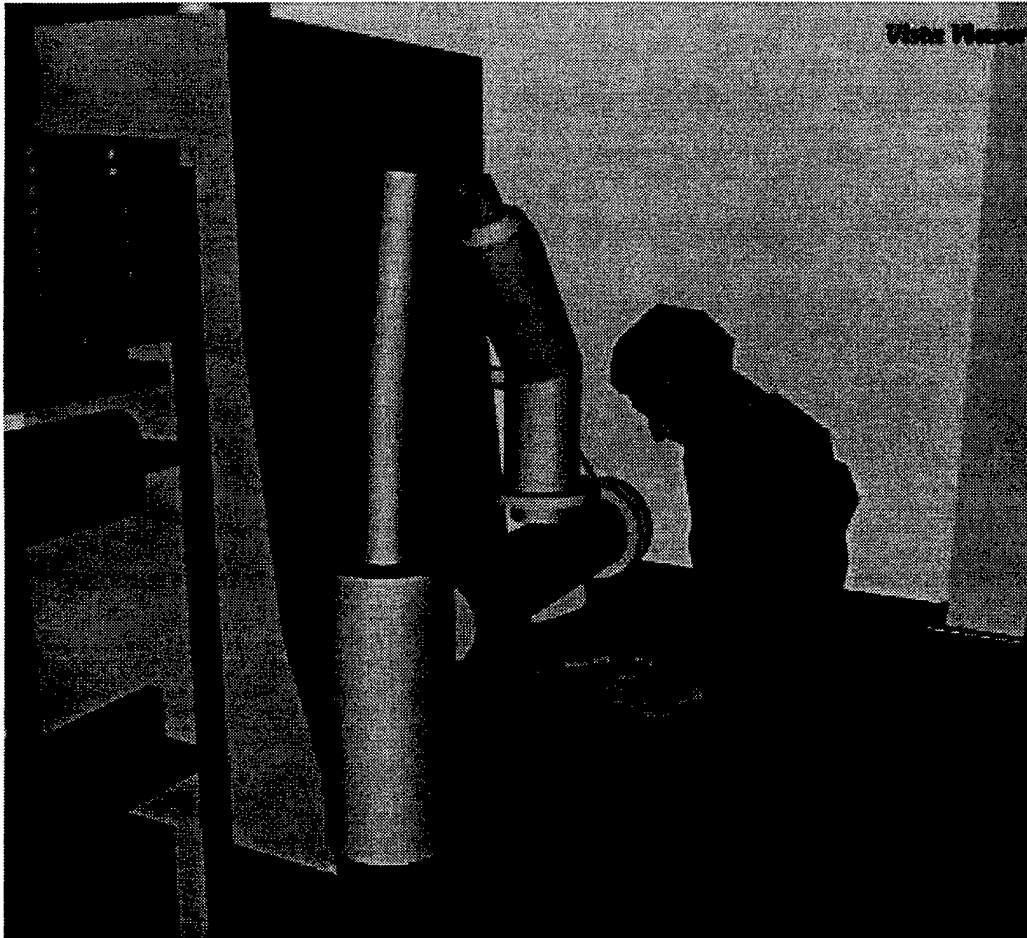
2

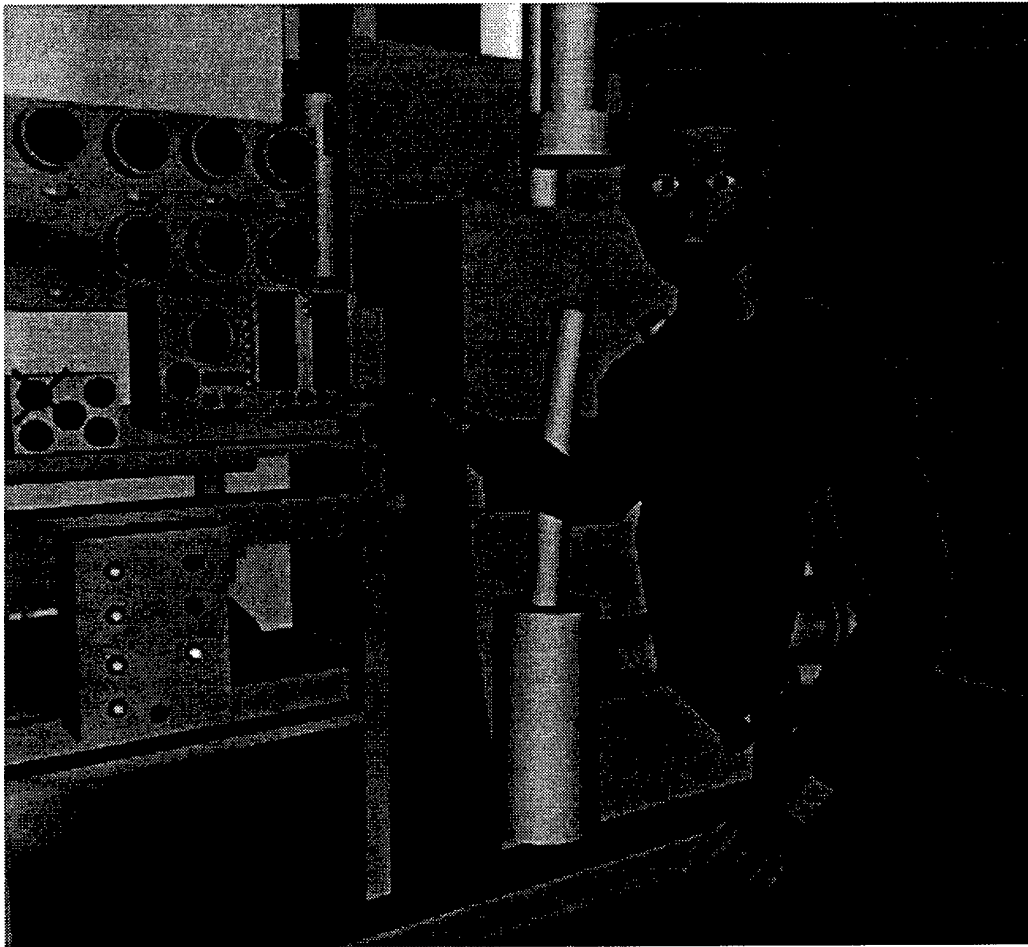Figure 1: Steve pulling out a dipstick

Figure 2: Steve describing a power light

Next, Steve says "I will now perform a functional test of the drain alarm light. First, check that the drain monitor is on. As you can see, the power light is illuminated, so the monitor is on" (see Figure 2). The student, realizing that she has seen this procedure before, says "Let me finish." Steve acknowledges that she can finish the task, and he shifts to monitoring her performance.

The student steps forward to the relevant part of the compressor, but is unsure of what to do first. "What should I do next?" she asks. Steve replies "I suggest that you press the function test button." The student asks "Why?" Steve replies "That action is relevant because we want the drain monitor in test mode." The student, wondering why the drain monitor should be in test mode, asks "Why?" again. Steve replies "That goal is relevant because it will allow us to check the alarm light." Finally, the student understands, but she is unsure which button is the function test button. "Show me how to do it" she requests. Steve moves to the function test button and pushes it (see Figure 3). The alarm light comes on, indicating to Steve and the student that it is functioning properly. Now the student recalls that she must extinguish the alarm light, but she pushes the wrong button, causing
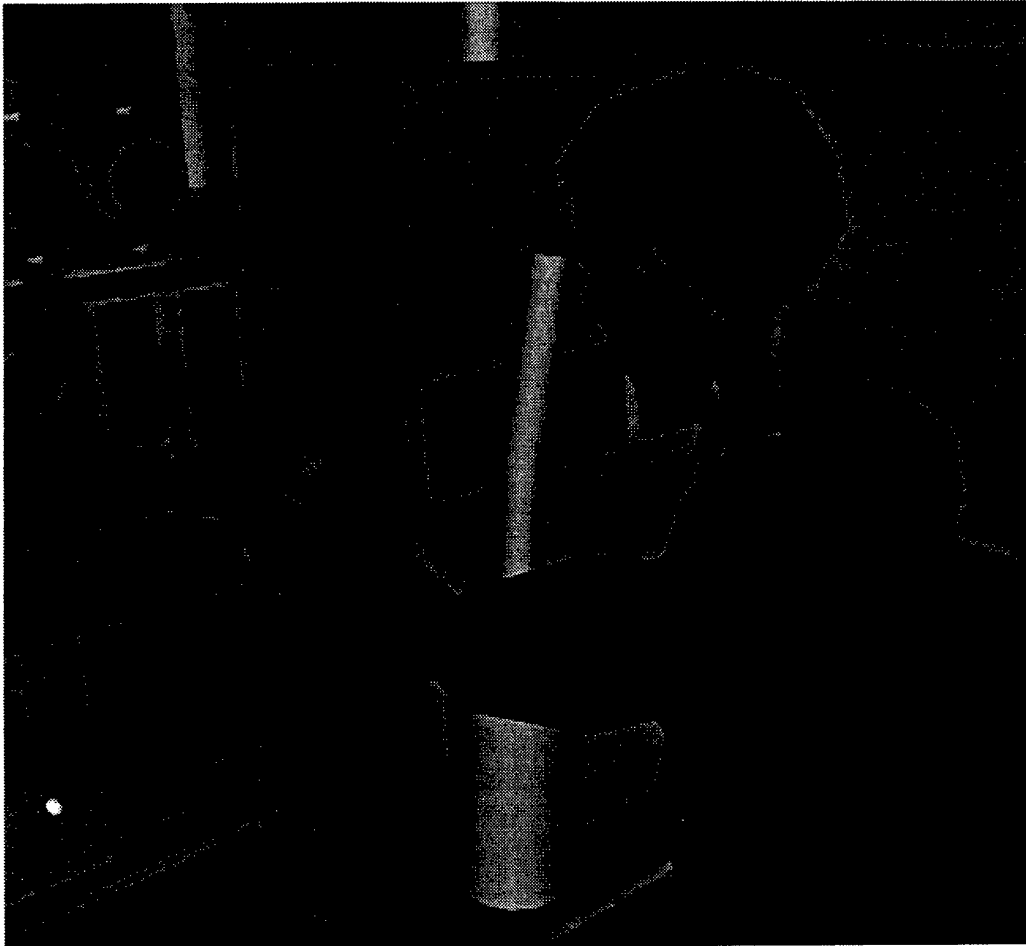
4

Figure 3: Steve pressing a button

a different alarm light to illuminate. Flustered, she asks Steve "What should I do next?" Steve responds "I suggest that you press the reset button on the temperature monitor." She presses the reset button to extinguish the second alarm light, then presses the correct button to extinguish the first alarm light. Steve looks at her and says "That completes the task. Any questions?"

The student only has one question. She asks Steve why he opened the cut-out valve.[1] "That action was relevant because I wanted to dampen oscillation of the stage three gauge" he replies.

This example illustrates a number of Steve's capabilities. He can generate and recognize speech, demonstrate actions, use gaze and gestures, answer questions, adapt domain procedures to unexpected events, and remember past actions. The remainder of the paper describes the technical details behind these capabilities.

---

[1] Such after-action review questions are posed via a desktop menu, not speech. Steve generates menu items for all the actions he performed, and the student simply selects one. A speech interface for after-action review would require more sophisticated speech understanding.
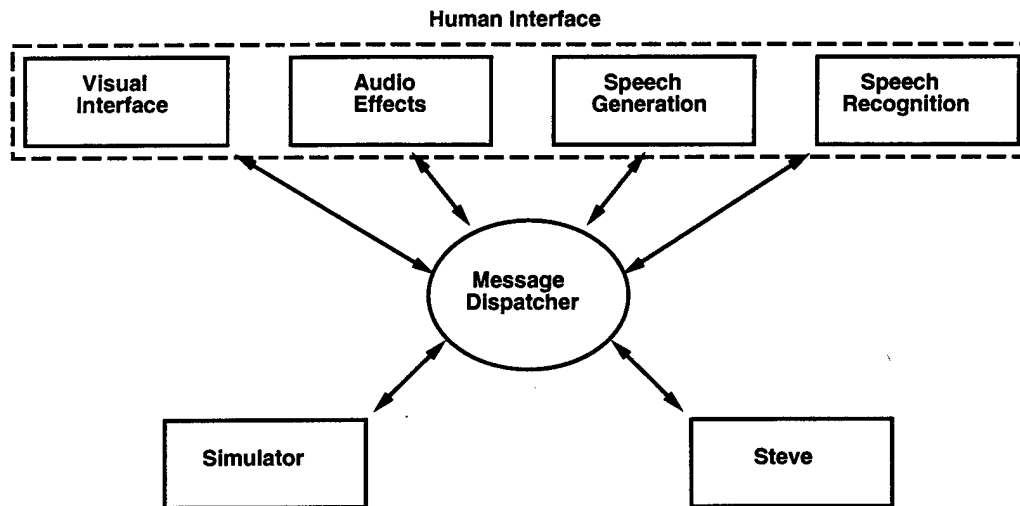
Human Interface

Figure 4: An architecture for virtual worlds. Although the figure only shows components for one agent and one human, other agents and humans can be added by simply connecting them to the message dispatcher in the same way.

3 Creating Virtual Worlds for People and Agents

Before we can discuss Steve's architecture, we must introduce a software architecture for creating virtual worlds that people and agents can cohabit. With our colleagues from Lockheed Martin Corporation and the USC Behavioral Technologies Laboratory, we have designed and implemented such an architecture (Johnson *et al.* 1998). For purposes of modularity and efficiency, the architecture consists of separate components running in parallel as separate processes, possibly on different machines. The components communicate by exchanging messages. Our current architecture includes the following types of components:

Simulator The behavior of the virtual world is controlled by a simulator. Our current implementation uses the VIVIDS simulation engine (Munro & Surmon 1997), developed at the USC Behavioral Technologies Laboratory.[2]

Visual Interface Each human participant has a visual interface component that allows them to view and manipulate the virtual world. The person is connected to this component via several hardware devices: their view into the world is provided by a head-mounted display, their movements are tracked by position sensors on their head and hands, and they interact with the world by "touching" virtual objects using a data glove. (They can also pinch objects using a pinch glove or click on objects using a 3D mouse; these actions are all treated the same by the visual interface component, which supports all these alternative devices.) The visual interface component plays two primary roles:

- It receives messages from the other components (primarily the simulator) describing changes in the appearance of the world, and it outputs a three-dimensional graphical representation through the person's head-mounted display.

---

[2]VIVIDS is a descendant of the RIDES and VRIDES systems mentioned in our earlier papers.

- It informs the other components when the person interacts with objects.

Our current implementation uses Lockheed Martin's Vista Viewer (Stiles, McCarthy, & Pontecorvo 1995) as the visual interface component.

Audio Each human participant has an audio component. This component receives messages from the simulator describing the location and audible radius of various sounds, and it broadcasts appropriate sounds to the headphones on the person's head-mounted display.

Speech Generation Each human participant has a speech generation component that receives text messages from other components (primarily agents), converts the text to speech, and broadcasts the speech to the person's headphones. Our current implementation uses Entropic's TrueTalk^{TM} text-to-speech product.

Speech Recognition Each human participant has a speech recognition component that receives speech signals via the person's microphone, recognizes the speech as a path through its grammar, and outputs a semantic token representing the speech to the other components. (Steve agents do not have any natural language understanding capabilities, so they have no need for the recognized sentence.) Our current implementation uses Entropic's GrapHVite^{TM} product.

Agent Each Steve agent runs as a separate component. The remainder of this paper focuses on the architecture of these agents and how they communicate with the other components.

The various components do not communicate directly. Instead, all messages are sent to a central message dispatcher. Each component tells the dispatcher the types of messages in which it is interested. Then, when a message arrives, the dispatcher forwards it to all interested components. For example, each visual interface component registers interest in messages that specify changes in the appearance of the virtual world (e.g., a change in the color or location of an object). When the simulator sends such a message, the dispatcher broadcasts it to every visual interface component. This approach increases modularity, since one component need not know the interface to other components. It also increases extensibility, since new components can be added without affecting existing ones. Our current implementation uses Sun's ToolTalk^{TM} as the message dispatcher.

4 Overview of Steve's Architecture

4.1 Perception, Cognition, and Motor Control

Steve consists of three main modules: perception, cognition, and motor control. The perception module monitors messages from the message dispatcher and identifies events that are relevant to Steve, such as actions taken in the virtual world by people and agents and changes in the state of the virtual world. The cognition module interprets the input it receives from the perception module, chooses appropriate goals, constructs and executes plans to achieve those goals, and sends out motor commands to control the agent's body. The motor control module decomposes these motor commands into a sequence of lower-level commands that are sent to other components via the message dispatcher. For example, upon receiving a motor command to push a button in the virtual world, the motor control
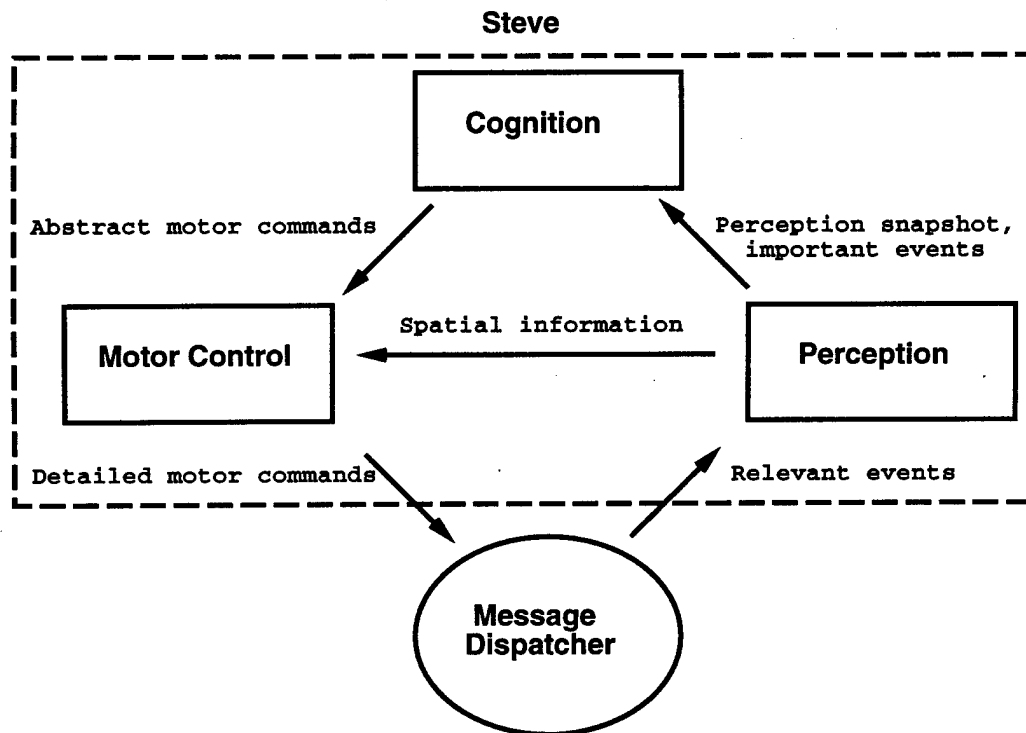
Figure 5: The three main modules in Steve and the types of information they send and receive.

module would send animation primitives to cause Steve's graphical finger to move to the button and would then send a message to the simulator to simulate the effects of the button being pressed.

In our current implementation, cognition runs as one process, and perception and motor control run in a separate process. This split has two advantages. First, it allows each module to be implemented in a suitable language. The cognition module is built on top of Soar (Laird, Newell, & Rosenbloom 1987; Newell 1990), which is intended as a general architecture for cognition; most of Steve's cognitive capabilities are implemented in Soar production rules. In contrast, the perception and motor control modules are implemented in procedural languages, namely Tcl/Tk and C. The second advantage of the split is that cognition can run in parallel with perception and motor control. This is especially important when there is a high volume of message traffic arriving at the perception module, as would be the case for a highly dynamic world; we do not want the perceptual processing to slow down cognition. If the motor control module were computationally expensive, it might pay to run perception and motor control as separate, parallel processes as well, but this has not been the case so far.

The perception, cognition, and motor control modules communicate directly, not via the message dispatcher. The cognition module communicates with the other two by message passing. It sends a message to the perception module when it is ready for an update on the state of the virtual world; the perception module responds with a snapshot of the state of the world and a set of important events that occurred since the last snapshot it sent (e.g., actions taken by people and agents). The cognition module also sends motor command messages to the motor control module. The motor control module resides in the same process as the perception module, so it accesses perceptual information freely via procedure calls and shared variables.

4.2 Domain Knowledge

To allow Steve to operate in a variety of domains, his architecture has a clean separation between domain-independent capabilities and domain-specific knowledge. The code in the perception, cognition, and motor control modules provides a set of general capabilities that are independent of any particular domain. To allow Steve to operate in a new domain, a course author simply specifies the appropriate domain knowledge in a declarative language. This declarative language was designed to be used by people with domain expertise but not necessarily any programming skills. Steve's general capabilities draw on the knowledge to teach it to students. The domain knowledge that Steve requires falls in two categories:

Perceptual Knowledge This knowledge tells Steve about the objects in the virtual world, their relevant simulator attributes, and their spatial properties. It resides in the perception module, and will be discussed in Section 5.

Task Knowledge This knowledge tells Steve about the procedures for accomplishing domain tasks and provides text fragments so that he can talk about them. It resides in the cognition module, and will be discussed in Section 6.

5 Perception

The role of the perception module is to receive messages from other components via the message dispatcher, use these messages to maintain a coherent representation of the state

of the virtual world, and to provide this information to the cognition and motor control modules. This section describes the representation that the perception module maintains and how it obtains the information, thus illustrating the types of information available to an agent in virtual reality.

5.1 Representing the State of the Virtual World

5.1.1 Representing the Simulator State

Most information about the state of the virtual world is maintained by the simulator. The perception module represents the simulator state as a set of attribute-value pairs. Each attribute represents a state variable in the simulator, and the attribute's value represents the value of the variable. For example, the state of an indicator light, say `light1`, might be represented with the attribute `light1_state` with possible values `on` and `off`. This simple representation was chosen to allow Steve to operate with a variety of simulators; while some simulators allow more sophisticated object-oriented representations, nearly all of them support this simple attribute-value representation.

The perception module tracks the simulator state by listening for messages from the simulator (via the message dispatcher). The perceptual knowledge provided to Steve by the course author includes a list of all relevant attributes. When Steve starts up, the perception module asks the simulator for the current value of each one. It also informs the message dispatcher that it is interested in messages describing changes in these attributes. The simulator broadcasts messages whenever the simulation state changes. Each message specifies the name of an attribute that changed and its new value.

The perception module uses these messages to maintain a snapshot of the simulation state. The cognition module periodically asks for this snapshot, so the perception module must always have one ready to be sent. After the perception module initializes its snapshot, it can simply update it whenever it receives a message from the simulator, except for one complication: some groups of messages from the simulator represent simultaneous changes. For example, suppose that a light should be illuminated whenever a button is depressed. When the button is pressed, the simulator will send two messages: one specifying that the button is depressed, and another specifying that the light is on. If the perception module were to update the simulation snapshot after each message, the cognition module might ask for a snapshot before both messages have been received and processed, and hence it could receive an inconsistent state of the world. This situation is analogous to a database transaction (Korth & Silberschatz 1986); either the cognition module should see the effects of all the simultaneous changes, or it should not see the effects of any of them.

To avoid this possibility, the simulator must use start and end messages to delimit messages representing simultaneous changes. After receiving a start message, the perception module stores subsequent simulator messages on a queue. When the end message arrives, the perception module updates the simulation snapshot by processing all the queued messages. This update is atomic; the cognition module cannot ask for a snapshot during the update. Thus, if the cognition module asks for a snapshot before the end message arrives, it sees none of the changes; if it asks for a snapshot after the end message arrives, it sees all of them.

5.1.2 Representing Spatial Properties of Objects and Agents

In order to control Steve's body in the virtual world, Steve needs to know the spatial properties of objects, such as their position, orientation, and spatial extent. In principle, the simulator could maintain such properties and provide them to the perception module as described in the previous section. In practice, however, this is often inconvenient. The simulator controls the appearance of the virtual world by instructing the visual interface components to load graphical models for objects and by sending messages to change properties of the objects, such as location and color. Therefore, the simulator itself may have no representation for the geometric properties of the objects; these details are in the graphical models themselves, which are typically created by a course author using a 3D modeling tool and stored in files. Moreover, the simulator may not even have simple information such as the location of the objects. This is because graphics objects are typically organized into a hierarchy, where each object has its own coordinate system that is relative to its parent. For example, the simulator might know how to move a button in and out relative to its graphical parent, a console, but may not know the global (world) coordinates of the button, which is what Steve needs.

Fortunately, the visual interface components can provide such information. Currently, the perception module queries a visual interface component for such information when it is needed. When the motor control module needs to interact with an object (e.g., point to it), it asks the perception module for its location and bounding sphere. The location specifies the origin of the object in Cartesian coordinates, as an (x, y, z) point. The bounding sphere is specified by the smallest radius around that origin that encompasses the object.

The perception module can get these properties of other agents as well. Each agent has a graphical body in the virtual world. To the visual interface components, these bodies are no different than any other graphical object, so the perception module can query for the location and bounding spheres of any agent.

In addition to keeping track of the location of agents in Cartesian coordinates, the perception module also keeps track of Steve's location in terms of objects. To move to an object, the cognition component sends a motor command to that effect. The motor control module converts this request into a location in Cartesian coordinates and sends a message to move Steve there. When Steve arrives, the perception module receives a message from the visual interface component, and it records his location as being at the desired object. The cognition module works at this level of abstraction, ignoring the actual Cartesian coordinates.

To interact with objects, Steve needs other spatial information that is not provided by the visual interface components. Therefore, we require the course author to provide the following perceptual knowledge for each object:

front vector To interact with an object, Steve must know where its front side is. When interacting with an object, Steve will use this knowledge to position himself in front of the object. The course author specifies the front of an object by a vector in the x-y plane that points to the front of the object from its origin. (We currently assume that this vector does not change dynamically.)

grasp vector If Steve may need to grasp the object, he needs to know the appropriate orientation for his hand. The course author specifies this as a vector in three-space pointing from the object's origin in the direction in which Steve would pull the object.

(Even if Steve has no reason to pull the object, this provides an orientation with which to grasp it.)

press vector If Steve may need to press the object (e.g., a button), he also needs an appropriate orientation for his hand when doing so. The course author specifies this as a vector in three-space pointing from the object's origin in the direction in which Steve should press the object.

agent location When interacting with an object, Steve stands in front of it and slightly to the right (to avoid blocking the student's view). Using the object's location, bounding sphere, and front vector, Steve can choose his location. Typically, this approach works well, because it ensures that Steve is out of the student's way when the student is standing in front of the object. However, if the object has an irregular shape, the bounding sphere might lead Steve to stand unnecessarily far from it. Or, if there are other objects surrounding the desired object, Steve might need to adjust his position to avoid colliding with them. If Steve's default location is not appropriate, the course author can specify a more appropriate location, by specifying how far in front, above, and to the right of the object Steve should stand. (Negative numbers can be used to force Steve to stand behind, below, or to its left when necessary.)

5.1.3 Representing Properties of Human Participants

The perception module also keeps track of human participants. The visual interface component for a person uses the position sensor on their head-mounted display to track their location in Cartesian coordinates (specifically, the point between their eyes) and their line of sight, and the perception module can request this information when it is needed by the motor control module (e.g., to look at a person).

If Steve is working with a student on a task, the perception module also keeps track of the student's field of view. More specifically, it keeps track of which objects in the virtual world lie within the student's field of view. For each object, the perception module asks the student's visual interface component whether that object is in the student's field of view. Subsequently, the visual interface component broadcasts a message when an object enters or leaves the student's field of view, so the perception module can maintain a snapshot of which objects the student can see.

5.1.4 Representing Perceptual Knowledge for Path Planning

Steve must navigate through the virtual world from object to object, avoiding collisions. There are several approaches to collision-free navigation, most of them originally developed by robotics researchers and later adapted for graphical worlds. Steve follows one standard approach; he carves the virtual world into a graph, where the nodes of the graph are places, and there is an edge between two nodes if Steve can move directly between the places without colliding into anything. As his set of places, Steve uses the objects in the virtual world, or, more specifically, the places he stands when interacting with each object. Currently, our work focuses on relatively static environments, so we assume the graph does not change over time. By default, there is an edge between any two nodes (places). However, if there is something blocking the path between two objects (e.g., a wall), the course author can specify that there is no direct path between the objects, effectively removing that edge from the graph. (For sparse graphs or subgraphs, the author can alternatively just specify the

permissible edges.) The resulting adjacency graph serves as Steve's perceptual knowledge for navigation; using it, the motor control module can plan a path between any two nodes, as described in Section 7.2.

5.2   Representing and Handling Events

Whenever the perception module passes a snapshot of the state of the world to the cognition module, it also passes a list of important events that occurred since the last snapshot. If the cognition module could only see periodic snapshots of the state of the world, it might miss some events. For example, if a button were pressed and released in between snapshots, the cognition module would never know it had been pressed. By receiving both a snapshot of the world and a list of important events that occurred since the last snapshot, the cognition module gets a complete view of the world and its changes.

The perception module receives and forwards to the cognition module several types of events:

state changes As described earlier, the simulator sends messages whenever the state of the virtual world changes. The cognition module does not need most of these; they are summarized by the snapshot it receives. However, the perception module passes a select few to the cognition module, specifically those that provide feedback on Steve's object manipulations. These "important state changes" are specified in Steve's perceptual knowledge.

actions on objects When a human participant interacts with an object (e.g., touches it with a data glove), that participant's visual interface component broadcasts a message specifying the name of the participant and the object they touched. The meaning of this interaction depends on the object. For example, touching a button causes the button to be pressed, while touching a valve allows the human participant to turn it. The result of the action is determined by the simulator; the message from the visual interface component only specifies the participant and object. The visual interface component also sends a message when the person stops touching the object. Agents interact with objects by sending these same messages, listing themselves as the participant.

human's speech Steve receives messages from a speech recognition component when a human participant begins talking and when they finish. The former message simply specifies which person is speaking, while the latter additionally includes a semantic token that represents the sentence that was recognized. (If the speech recognizer did not understand the sentence, it returns an unknown token.)

agent's speech Steve agents can also tell when other agents are talking. An agent sends out a message to the speech generation components to generate speech. Therefore, an agent can listen for such messages to detect when other agents begin speaking. When a speech generation component finishes producing the speech for its human participant, it sends a message to this effect. Therefore an agent can also tell when other agents have finished speaking. Moreover, an agent can use such messages to detect when its own utterance is complete. Currently, these messages do not include a semantic token, like their corresponding messages representing human speech. Instead, agents send separate messages representing the semantic content of their speech; these messages are loosely based on speech acts, much like KQML (Labrou & Finin 1994).

13

6 Cognition

6.1 The Layers of Steve's Cognition

The cognition module is organized into three main layers:

- Domain-specific task knowledge

- Domain-independent pedagogical capabilities

- Soar

Steve is built on top of the Soar architecture (Laird, Newell, & Rosenbloom 1987; Newell 1990). Soar was designed as a general model of human cognition, so it provides a number of features that support the construction of intelligent agents. This paper will not focus on Soar, since an understanding of Steve does not require an understanding of Soar. However, much of Steve's design was influenced by features of the Soar architecture.

Soar is a general cognitive architecture, but it does not provide built-in support for particular cognitive skills such as demonstration, explanation, and question answering. Our main task in building Steve was to design a set of domain-independent pedagogical capabilities such as these and layer them on top of the Soar architecture. These capabilities are implemented as Soar production rules, and they will be discussed later in this section.

To teach students how to perform procedural tasks in a particular domain, Steve needs a representation of the tasks. A course author must provide such knowledge to Steve. Given appropriate task knowledge for a particular domain, Steve uses his general pedagogical capabilities to teach that knowledge to students. Thus, our layered approach to Steve's cognition module allows Steve to be used in a variety of domains; each new domain requires only new task knowledge, without any modification of Steve's abilities as a teacher.

6.2 Domain Task Knowledge

Intelligent tutoring systems typically represent procedural knowledge in one of two ways. Some, notably those of Anderson and his colleagues (Anderson *et al.* 1995), use detailed cognitive models built from production rules. Such systems perform domain tasks by directly executing the rules. Other systems use a declarative representation of the knowledge, usually some variant of a procedural network representation (Sacerdoti 1977) specifying the steps in the procedure and their ordering. Such systems perform tasks by using a domain-independent interpreter to "execute" the procedural network (i.e., walk through the steps). Production rule models provide a more flexible ontology at a price: they are laborious to build. The labor may be justified in domains like algebra and geometry, where a tutor, once built, can be used for many years by many people. In contrast, procedural network representations are more practical for domains like operation and maintenance of equipment; procedures may change frequently in such domains, so it must be easy for domain experts or course authors to represent procedures, examine them, and change them when necessary. For these reasons, Steve uses a procedural network (plan) representation of domain tasks.

Steve represents domain tasks as hierarchical plans, using a relatively standard representation (Russell & Norvig 1995). First, each task consists of a set of steps, each of which is either a primitive action (e.g., press a button) or a composite action (i.e., itself a task). Composite actions give tasks a hierarchical structure. Second, there may be ordering constraints among the steps, each specifying that one step must precede another. These

```
Task: functional-test

  Steps: press-function-test, check-alarm-light, extinguish-alarm

  Causal Links:
      press-function-test achieves test-mode for check-alarm-light
      check-alarm-light achieves know-whether-alarm-functional for end-task
      extinguish-alarm achieves alarm-off for end-task

  Ordering constraints:
      press-function-test before check-alarm-light
      check-alarm-light before extinguish-alarm
```

Figure 6: An example task definition

constraints define a partial order over the steps. Finally, the role of the steps in the task
is represented by a set of causal links (McAllester & Rosenblitt 1991). Each causal link
specifies that one step achieves a goal that is a precondition for another step (or for termi-
nation of the task). For example, pulling out a dipstick achieves the goal of exposing the
level indicator, which is a precondition for checking the oil level.

Figure 6 shows an example of a task definition: the task of performing a functional test
of one of the subsystems of a high-pressure air compressor aboard a ship. It consists of three
steps: press-function-test, in which the compressor operator presses the test button on the
control panel, check-alarm-light, in which the operator examines the light to make sure it
is functional (i.e., not burned out), and extinguish-alarm, in which the operator presses the
reset button to reset the light. In addition, every task has two dummy steps: a begin-task
that precedes all other steps, and an end-task that follows all other steps. Several causal
links exist among the steps. For example, press-function-test puts the device in test-mode
(i.e., illuminates the alarm light), which is a precondition for check-alarm-light. In order for
the task to be complete, the operator must know whether the alarm light is functional, and
the alarm light must be off; thus, these end goals are shown as preconditions for end-task.
Similarly, if the task depended on conditions that must be established prior to starting the
task, these conditions would be represented as effects of begin-task.

The plan representation only defines the structure of a task, in terms of its goals and
steps. To complete the description, the course author must define the goals and primitive
actions it references. Each goal is defined by an attribute-value pair. Steve can represent two
types of goals: attributes of the virtual world, and attributes of his own mental state. For
the former, the attribute is one that will appear in Steve's perception (e.g., light1_state),
and the value is its desired value (e.g., on). The goal is satisfied when that attribute-
value pair is part of Steve's current perceptual snapshot. For the latter, the attribute is
one that will appear in Steve's mental state. Such attributes are stored as the result of
certain primitive actions that Steve executes, namely *sensing actions* (Russell & Norvig
1995). Sensing actions are used to record the state of some attribute of the virtual world
at a particular point during a task. For instance, in the functional-test example above,
check-alarm-light is a sensing action that causes Steve to record the resulting state of the
light as the value of a check_alarm_light_result attribute in mental state. (A mental

state goal can optionally specify an attribute without any specific value; for example, a goal specified only as `check_alarm_light_result` is satisfied if Steve knows the result of the test, regardless of the particular result.) Thus, Steve can represent two types of goals: goals that require putting the virtual world in some desired state, which Steve can evaluate using perception, and goals of acquiring information, which Steve can evaluate by checking his mental state.

Primitive actions require Steve to interact with the virtual world, typically via motor commands. To simplify the course author's job of describing the primitive actions in a task, we are developing a library of primitive actions that are appropriate for a virtual world; the course author defines each primitive action in a task as an instance of one in the library. The library is organized as a hierarchy of very general actions and their specializations. For example, one general action in the library is `ManipulateObject`. To define a task step as an instance of `ManipulateObject`, the course author must specify the name of the object in the virtual world to be manipulated (e.g., `button1`), the name of the motor command that will perform the manipulation (e.g., `press`), and the perceptual attribute-value pair that will indicate that the manipulation has finished (e.g., `button1_state depressed`). Other actions in the library are defined as specializations of such general actions, to provide a shorthand for course authors. For example, the library includes `PressButton` as a specialization of `ManipulateObject`; a course author could define the previous example as an instance of `PressButton` by merely specifying the name of the button. It is relatively easy to extend the action library, but it does require writing some simple Soar productions, so we would not expect course authors to extend it themselves.

To complete the task knowledge, the course author must provide text fragments that Steve can use for natural language generation. Steve does not currently include any sophisticated capabilities for natural language generation; speech utterances are constructed by plugging domain-specific text fragments into text templates. Steve currently requires three types of text fragments:

- He requires one fragment for each goal, in a form that would complete the sentence "I want ...".

- He requires two fragments for each task step. The first is a a simple imperative description of the step (e.g., "press the power button"). The second has the same form and purpose, but may include more elaboration. Steve uses the second fragment when a verbose description of the step is appropriate.

- For sensing actions, he requires a fragment for each possible result (e.g., "the oil level is low" and "the oil level is normal"). Steve uses these fragments when describing the results of sensing actions to a student.

Our representation for domain task knowledge provides the information that Steve needs while only requiring declarative knowledge that a course author can provide. In contrast to simple partial-order plans, our hierarchical plan representation provides several benefits: it allows the course author to chunk complex procedures into subtasks, which may be reused in multiple tasks, and it provides more structure to Steve's demonstrations, allowing him to chunk complex procedures into subtasks to aid students' comprehension. Our inclusion of causal links in the task representation differs from previous tutoring systems; previous systems that used a declarative representation of procedural knowledge, such as those of Burton (1982), Munro et al. (1993), and Rickel (1988)), only included steps and ordering

constraints. As we will discuss shortly, Steve's knowledge of causal links allows him to automatically generate explanations and to adapt procedures to unexpected circumstances, making him more robust than these previous systems.

The central purpose of Steve's task knowledge is to allow him to create a task model when he is required to demonstrate a task or monitor the student performing the task. He creates the task model by simple top-down task decomposition (Sacerdoti 1977). First, he initializes the task model to contain the name of the task. Next, he adds the task representation (steps, ordering constraints, and causal links) for that task. Steve recursively repeats this process for any composite step in the task representation, until the task has been fully decomposed into primitive actions. The result is the full hierarchical representation of the given task. This task model includes all the steps that might be required to complete the task, even if some are not necessary given the current state of the world. As described shortly, this task model is an important resource for Steve's plan construction.

6.3 Steve's Decision Cycle

The cognition module operates by continually looping through a decision cycle. In our current implementation, Steve executes about ten decision cycles per second. Once Steve is given a task and has created the task model, as described in the last section, each decision cycle goes through five phases:[3]

1. Input phase: Get the latest perceptual information from the perception module.

2. Goal assessment: Use the perceptual information to determine which goals of the current task are satisfied. This includes the end goals of the task as well as any intermediate goals (i.e., preconditions of task steps).

3. Plan Construction: Based on the results of goal assessment, construct a plan to complete the task.

4. Operator Selection: Select the next *operator*. Each operator is represented by a set of production rules that implement one of Steve's capabilities, such as answering a question or demonstrating an action. Steve's operators serve as the building blocks for his behavior.

5. Operator Execution: Execute the selected operator. In most cases, this will cause the cognition module to output one or more motor commands.

The general notions of decision cycle, input phase, and operator selection and execution are provided by Soar. The particulars of Steve's decision cycle are unique to Steve.

During the input phase, the cognition module asks the perception module for the state of the virtual world. As discussed in Section 5, the cognition module receives three pieces of information:

- the state of the simulator, represented as a set of attribute-value pairs (as described in Section 5.1.1)

- a set of important events that occurred since the last snapshot (as described in Section 5.2)

---

[3]Actually, Soar executes phase 5 concurrently with phases 1-3 of the next decision cycle.

- the student's field of view, represented as the set of objects that lie within it (as described in Section 5.1.3)

The remainder of this section discusses the rest of the decision cycle. First, we discuss goal assessment (Section 6.4) and plan construction (Section 6.5). Then, we discuss Steve's operators (i.e., his individual capabilities). The discussion of operators is organized around three primary modes: demonstrating a task to a student (Section 6.6), monitoring a student's performance and providing help (Section 6.7), and answering questions about past actions (Section 6.8).

6.4 Goal Assessment

In order to construct a plan to complete the current task, Steve must know which of the task goals are already satisfied. As described in Section 6.2, each goal in the task model is associated with an attribute-value pair. Therefore, Steve can assess each goal by simply determining whether its associated attribute-value pair is satisfied given his current perceptual input and mental state.

Our implementation of this process exploits Soar's truth maintenance system. When the course author defines a goal, an associated Soar production rule is created. This rule simply checks the current perceptual input or mental state, whichever is appropriate. When the goal becomes satisfied, the rule fires, marking the goal satisfied. As long as the goal is satisfied, this result will remain, without any further processing required. If the goal becomes unsatisfied, Soar retracts the rule, along with its result. Thus, Steve need not evaluate every goal on every decision cycle; each rule automatically fires or retracts when the status of its goal changes.

6.5 Plan Construction

Whether demonstrating a task to a student or monitoring the student's performance of the task, Steve must maintain a plan for completing the task. The plan allows Steve to identify the next appropriate action and, if asked, to explain the role of that action in completing the task. As a teacher, Steve's ability to rationalize the action is just as important as his ability to choose it.

We faced conflicting design criteria when designing Steve's planner. To handle dynamic environments containing people and other agents, Steve must be able to adapt procedures to unexpected events. This argues against a rote execution of domain procedures, in favor of a general planning and replanning capability. Thus, we might encode domain actions as STRIPS operators (Russell & Norvig 1995) and use a standard partial-order planner (Weld 1994) to construct plans. However, we also want Steve to follow standard procedures whenever possible. Thus, we would have to augment the partial-order planner with substantial control knowledge to discourage unusual plans. Moreover, Steve must be able to construct and revise plans quickly, since he and the student are collaborating on tasks in real time. This can be a problem for general partial-order planners, which often require exponential search. Finally, we must only require task knowledge that course authors can easily provide, yet formulating STRIPS operators and control knowledge for a partial-order planner is difficult even for AI researchers.

To satisfy these criteria, Steve uses the task model, as described in Section 6.2, to guide his plan construction and revision. Recall that, when given a task to demonstrate or monitor, Steve uses top-down task decomposition to construct a task model. The task

18

model includes all the steps that might be required to complete the task, even if some are not necessary given the current state of the world. Every decision cycle, after Steve gets a new perceptual snapshot and assesses the goals in the task model, he constructs a plan for completing the task. He does so by marking those elements of the task model that are still relevant to completing the task, as follows:

- Every end goal of the task is relevant.

- A primitive step in the task model is relevant if it achieves a relevant, unsatisfied goal.

- Every precondition of a relevant step is a relevant goal.

Thus, Steve starts by marking all the end goals as relevant (i.e., in the plan). For each one that is not already satisfied, he finds the step in the task model that achieves it and adds that step to the plan. Each step that is added may have unsatisfied preconditions, and each such precondition becomes a new goal that must likewise be achieved. This is exactly how a general partial-order planner operates. However, Steve's use of the task model eliminates much of the complexity that a partial-order planner must handle:

- A partial-order planner may have multiple actions that could achieve each goal, so it must search through alternative plans. In contrast, Steve uses the task model as an oracle for choosing the appropriate action to achieve each relevant, unsatisfied goal, so there is no search. Thus, Steve's plan construction is predictably fast.

- A partial-order planner must identify threats (i.e., two unordered steps that could interact undesirably if executed in the wrong order) and add appropriate ordering constraints. In contrast, Steve simply uses the ordering constraints in the task model; if two steps in the plan have an ordering constraint in the task model, that ordering constraint is added to the plan. As long as there are no unresolved threats in the task model, there will be no unresolved threats in the plan.

- A partial-order planner must create steps in the plan by instantiating STRIPS operators. Therefore, it must maintain a set of binding constraints, and it may have to search when there are alternatives. In contrast, the steps in the task model are instances of actions in the action library, so they have no variables. Hence, Steve need not reason about binding constraints.

This approach satisfies our design criteria. It is efficient, and it forces Steve to follow standard procedures as much as possible, yet it still allows him to adapt to unexpected events: Steve re-executes parts of his plan that get unexpectedly undone, and he skips over parts of the task that are unnecessary because their goals were serendipitously achieved. Thus, unlike videos or scripted demonstrations, Steve can adapt domain procedures to the state of the virtual world, and he does so efficiently.

To execute the plan (or evaluate the student's actions), Steve must also determine which steps to do next. A plan step is ready for execution if it is "applicable" (i.e., all its preconditions are satisfied) and not "precluded" (i.e., no other plan step necessarily comes before it). Note that there may be a single next step, there may be multiple next steps (since this is a partially-ordered plan), and there may be no next steps (if no subset of the task model will get Steve from the current state to task completion).

Steve's plan construction exploits Soar's truth maintenance system, making it even more efficient. Each of the three rules for determining relevance listed above is implemented as

a production rule. Depending on which goals in the task model are satisfied, instances of these production rules fire, marking appropriate parts of the task model as relevant (i.e., in the current plan). As goals become satisfied or unsatisfied, only affected instances of the production rules fire or retract, so only those parts of the plan that are affected by changes in the current state are revised.

6.6 Demonstration

To demonstrate a task to a student, Steve must perform the task himself, explaining what he is doing along the way. First, he creates the task model. Then, in each decision cycle, he updates his plan for completing the task and determines the next appropriate steps, as discussed in the previous section. After determining the next appropriate steps, he must choose one and demonstrate it. First, we discuss how he chooses, and then we discuss how he demonstrates.

6.6.1 Choosing the Next Task Step to Demonstrate

At any point during a task, there may be multiple steps that could be executed next. That is, each of these steps may be applicable (i.e., all their preconditions are satisfied) and not precluded (i.e., no other step in the plan must necessarily come first). From the standpoint of completing the task, any of these steps could be chosen. However, from the standpoint of communicating with the student, they may not be equally appropriate.

Students will more easily follow the demonstration if Steve follows certain human conventions. For example, it is easier to follow a demonstration that focuses on one subtask at a time. If two subtasks could be interleaved arbitrarily, Steve could alternately execute one step from each subtask until they are both complete, but this would be unnecessarily confusing. As another example, suppose that Steve were demonstrating a subtask (e.g., configuring a console) when an unrelated, higher-priority task step suddenly became relevant (e.g., acknowledging an alarm). After acknowledging the alarm, Steve could move on to an unrelated subtask, but the student will expect him to resume the interrupted subtask (e.g., configuring the console). Researchers in computational linguistics have studied this problem of *discourse focus* for many years, and they have identified common conventions in types of discourse as different as rhetorical persuasion and dialogues regarding tasks. To ensure coherent demonstrations, Steve must obey these conventions.

Following Grosz and Sidner (1986), we represent the discourse focus as a stack. When Steve begins executing a step in the plan (either primitive or composite), he pushes it onto the stack. Therefore, the bottom element of the stack is the main task on which the student and Steve are collaborating, and the topmost element is the one on which the demonstration is currently focused. When the step at the top of the focus stack is "complete," Steve pops it off the stack. A primitive action is complete when it is no longer in the current plan, while a composite step is complete when all its end goals are satisfied.

Steve uses the focus stack to help choose the next step to demonstrate. When there are multiple plan steps ready for execution, he prefers those that maintain the current focus or shift to a subtask of the current focus. To operationalize this intuition, Steve first fleshes out the list of candidates for demonstration:

- Any step in the current plan that is ready for execution is a candidate. Each of these is a primitive action, since the plan never includes any composite steps.

- If a step (primitive or composite) is a candidate, and its parent (composite step) in the task model is not somewhere on the focus stack, that parent step is a candidate.

- The previous rule is applied recursively. That is, if a composite step is added as a candidate, and its parent in the task model is not somewhere on the focus stack, that parent is added as a candidate.

Having enumerated the candidates, Steve chooses among them as follows:

- Executing a parent step next is preferable to executing any of its children. Intuitively, this means that Steve should shift focus to the (sub)task and introduce it before he begins demonstrating its steps.

- A task step whose parent is the current focus (i.e., the topmost element of the focus stack) is preferable to one whose parent is not.

- If there are remaining candidates that are unordered by these preferences, Steve chooses one randomly.

Let's illustrate these rules with a few examples:

- Suppose Steve is beginning a new demonstration. Therefore, the focus stack is empty. Suppose the task is "start the compressor," the first subtask is "check the oil," and the first step of that subtask is "pull out the dipstick." Therefore, the first step of the plan will be "pull out the dipstick." Since that step's parent ("check the oil") is not on the focus stack, it is a candidate for demonstration, and is preferable to "pull out the dipstick." Since the parent of "check the oil," namely "start the compressor," is not on the focus stack, it is a candidate for demonstration, and is preferable to "check the oil." Thus, "start the compressor" is added to the focus stack first, and Steve executes it by introducing the task to the student. Next, Steve will push "check the oil" onto the stack and execute it by introducing this first subtask. Finally, Steve can push "pull out the dipstick" onto the stack and demonstrate it to the student; at this point, Steve has introduced the appropriate hierarchical context for performing this action.

- Suppose Steve could perform two subtasks in any order, such as "check the oil" and "check the coolant," and he randomly chooses to check the oil first. Next, since "check the oil" is the current focus, he will prefer "pull out the dipstick" to "check the coolant level" or any of its steps, so he will push it onto the focus stack and demonstrate it. When the dipstick is out, it will be removed from the plan and popped off the focus stack, making "check the oil" the current focus again. This process will repeat for each step of "check the oil," until that subtask is completed and popped off the focus stack.

- Suppose that Steve is performing one subtask (e.g., "configure console") when an unrelated, higher-priority (based on ordering constraints) task step suddenly becomes relevant (e.g., "acknowledge alarm"). Steve will add "acknowledge alarm" to the plan, and it will be the only step ready for execution (since it precludes the remaining steps of "configure console," so Steve will push it onto the focus stack and demonstrate it. When the alarm is acknowledged, it will be removed from the plan and popped off the focus stack, and Steve will resume "configure console."

6.6.2 Demonstrating a Task Step

Once Steve chooses the next task step and pushes it onto the focus stack, he demonstrates it to the student. If the step is a composite step, Steve simply introduces the (sub)task, using its associated text fragment. If it is a primitive action, Steve demonstrates it as follows:

1. First, Steve moves to the location of the object he needs to manipulate by sending a locomotion motor command, along with the object to which he wants to move. Then, he waits for perceptual information to indicate that he has arrived. (This typically takes multiple decision cycles; during this period, Steve repeatedly executes a simple "wait" operator.)

2. Once Steve arrives at the desired object, he explains what he is going to do. This involves describing the step while pointing to the object to be manipulated. To describe the step, Steve outputs a speech specification with three pieces of information:

   - the name of the step – this will be used to retrieve the associated text fragment
   - whether Steve has already demonstrated this step – this allows him to acknowledge the repetition
   - a rhetorical relation indicating the relation in the task model between this step and the last one Steve demonstrated – this is used to generate an appropriate cue phrase

   Research has shown that human speakers often use cue phrases to indicate the rhetorical relation between one utterance and another (Grosz & Sidner 1986; Moore 1993). Steve currently uses cue phrases to mark several types of rhetorical relations:

   - If the last step was to introduce a composite step, and the current step is a child of that step, Steve says "First, ...".
   - If the previous step achieved a precondition of the current step, Steve says "Now we can ...".
   - If there is an ordering constraint in the task model specifying that the last step must precede the current step, Steve says "Next, ...". (This is used only when the previous cue phrase does not apply.)
   - If the current step precedes the last step in the task model, it represents an interruption, so Steve says "Oh, I need to ...".

   These cue phrases help to structure the demonstration, hopefully aiding the student's comprehension. Once Steve sends the motor command to generate the speech, he waits for an event from the perception module indicating that the speech is complete.

3. When his speech is complete, he performs the task step. This is done by sending an appropriate motor command and waiting for evidence in his perception that the command was executed. For example, if he sends a motor command to press button1, he waits for his perception snapshot to include `button1_state depressed`.

4. If appropriate, he explains the results of the action, using the appropriate text fragments.

Actually, this sequence of events in demonstrating a primitive action is not hardwired into Steve. Rather, each item in the sequence is an independent capability, and each action type in the action library is associated with an appropriate suite of such items. Each suite is essentially a finite state machine represented as Soar productions. By representing a suite as a finite state machine rather than a fixed sequence, Steve's demonstration of an action can be more reactive and adaptive. Most of the actions in our current action library use the sequence above, but our approach gives Steve the flexibility to demonstrate different types of primitive actions differently.

Steve is sensitive to the student while demonstrating. For example, when Steve references an object and points to it, he checks whether the object is in the student's field of view. If not, he says "Look over here!" and waits until the student is looking before proceeding with the demonstration.

6.6.3 Let me finish

Steve's demonstrations can end in one of two ways. Typically, he completes the task and announces his completion. However, we also allow the student to request "Let me finish." In this case, Steve acknowledges that the student can finish the task, and he shifts to monitoring the student.

6.7 Monitoring a Student

Often, Steve's role is to monitor a student performing a task, providing assistance when needed. For example, Steve might first demonstrate a task and then suggest that the student try it. Or, as described in the previous section, the student might interrupt Steve's demonstration and ask to finish the task. In either case, Steve's role in monitoring a student is to maintain his own plan for completing the task and to use it to assess the student's actions and to answer questions.

Steve's ability to adapt to unexpected events is especially useful when monitoring a student. Most tutoring systems require the student to follow the tutor's plan, because the tutor would be unable to adapt to unexpected deviations. In contrast, we want to give the student the flexibility to deviate from the standard procedure, make mistakes, and learn to recover from them. Such flexibility is a prime advantage of simulation-based training; it allows students to gain exposure to a wide variety of situations, and it encourages them to learn from their own mistakes. Steve's approach of repeatedly re-evaluating and possibly revising his plan supports such flexibility; he can typically provide assistance to the student even when the student took unexpected actions and landed in an unusual state of the world.

Steve's approach to goal assessment and plan construction is the same for monitoring as it is for demonstration. The main difference between monitoring and demonstration is that, when monitoring, Steve allows the student to take the actions. There is one exception: Steve must still perform any sensing actions in the plan (e.g., checking whether a light comes on). Sensing actions do not cause observable changes in the virtual world; they only change the mental state of the student. In order to update his own plan, Steve must recognize when the goals of a sensing action are achieved. Therefore, whenever a sensing action is appropriate (i.e., the next step in Steve's plan), if the student is looking at the appropriate object (i.e., it is in the student's field of view), Steve performs the sensing action, records the result, and assumes that the student did the same.

In the remainder of this section, we outline Steve's capabilities relevant to monitoring a

student. The details of these capabilities are not important; additional sophistication could, and will, be added to each. The important point is to show how Steve's domain knowledge, and his abilities to use the knowledge, allow him to assist the student in a variety of ways.

6.7.1 Evaluating the student's actions

Using his own assessment of the task goals, and his plan for completing the task, Steve can evaluate the student's actions. When the student performs an action, Steve must identify the steps in the task model that match the action. If none of the matching steps is an appropriate next step, the student's action is incorrect. In this case, Steve could provide feedback to the student, ranging anywhere from a simple shake of his head or look of disapproval to an explanation of why the action is incorrect (e.g., a precondition is not satisfied or the step is precluded by another step). Currently, Steve simply says "no" and shakes his head, but we will be experimenting with different forms of feedback soon. When the student's action is correct, Steve nods his head in agreement.

6.7.2 What should I do next

The student can always ask Steve "What should I do next?" To answer this question, Steve simply suggests the next step in his own plan. Unlike most tutoring systems, Steve can suggest appropriate steps even when the student deviates from the standard procedure, as mentioned earlier. This is a direct consequence of Steve's ability to adapt the procedure to unexpected events, in this case the student's unexpected actions.

If there are multiple possible next steps, Steve currently enumerates them. In some cases, this is appropriate. However, in other cases, Steve could provide a more focused answer if he knew more about the student's current focus (e.g., the subtask on which the student is currently working). Plan recognition algorithms infer such information, so they could be used to maintain the discourse focus stack during monitoring. Steve's plan representation provides the information that most plan recognition algorithms require, but we have not yet added this capability.

It is also possible that Steve does not know what to do next. This could happen if no subset of the task model is sufficient for completing the task. For example, the student may have permanently damaged the virtual equipment. In the domains where we have tested Steve, the simulator has not supported such irreversible actions. Nonetheless, Steve currently handles such situations by simply explaining that he does not know what to do next. In the future, we could extend Steve to explain the flaws in his plan that he does not know how to resolve (i.e., the preconditions he does not know how to achieve).

6.7.3 Show me what to do

The student may understand *what* to do but not *how*. In this case, the student can tell Steve "Show me what to do." Steve responds to such questions by demonstrating the next step, as described in the previous section. Clearly, this is a capability that traditional disembodied tutors cannot provide.

If there are multiple possible next steps, Steve currently chooses one of them randomly. As mentioned before, plan recognition could provide information about the student's current focus, leading to a more informed choice.

```
Steve: I suggest that you press the function test button.
Student: Why?

Steve: That action is relevant because we want the drain monitor in test mode.
Student: Why?

Steve: That goal is relevant because it will allow us to check the alarm light.
Student: Why?

Steve: That action is relevant because we want to know whether the alarm light
       is functional.
```

Figure 7: Example explanations generated by Steve

6.7.4 Explaining the relevance of a step or goal

When Steve suggests that the student perform an action, we want to allow the student to ask what the role of that action is in the task. Without an understanding of the rationale for each step in a procedure, students are forced to simply memorize the steps. In contrast, an understanding of the causal structure of a task should help students remember the procedure, adapt it when necessary, and apply their knowledge to related tasks.

Figure 7 shows Steve's ability to rationalize suggestions. In this example, Steve is monitoring the student and suggests that the student press the function test button. When the student asks why, Steve explains the goal of that action: it will put the drain monitor in test mode. The example also illustrates Steve's ability to answer follow-up questions; when the student asks why that goal is relevant, Steve explains that it will enable another relevant action. The student can continue asking such follow-up questions until, ultimately, the initial suggestion has been related to an end goal of the task that the student was given.

Steve generates such explanations from the causal links in the plan. Recall from Section 6.5 that if a step or goal is relevant (i.e., in the current plan), it is for one of three reasons:

1. It is an end goal of the top-level task.

2. It is a precondition of a relevant primitive plan step.

3. It is a primitive plan step that achieves a relevant, unsatisfied goal.

These connections between steps and goals are specified by the causal links in the plan. Thus, one advantage to having Steve maintain a plan is that he can use it to rationalize his suggestions.

Although our current approach to explanation simply follows causal links one by one (driven by follow-up questions), our plan representation supports many other explanation strategies as well. For example, using a model of the student's knowledge, Steve could skip over causal links that the student is presumed to understand. Similarly, Steve could purposely skip over some causal links in order to motivate an action in terms of a more distant goal, forcing the student to relate the action to that goal. Also, since plans are represented hierarchically, Steve could provide suggestions and explanations at various levels

of detail based on the student's knowledge and Steve's pedagogical style. Providing a rich foundation for explanation was a prime motivation for choosing hierarchical plans as the representation for tasks.

6.8 Episodic Memory and After-Action Review

The previous section described Steve's ability to rationalize his suggestions. In that case, Steve can explain the relevance of a step or goal to completing the task by inspecting his current plan. In addition, we wanted Steve to be able to rationalize his own actions during an after-action review. When Steve completes a demonstration, he asks the student whether they have any questions. At this point, they can ask him to rationalize any one of his actions during the demonstration, and they can ask follow-up "Why?" questions as described in the previous section. To answer such questions, Steve cannot rely on his current plan, since the task is already complete and the step in question is no longer relevant.

To support such questions, Steve employs the episodic memory capability of the Debrief system (Johnson 1994). Debrief includes a set of production rules that enable Soar agents to remember their actions and the situations in which they occurred. It uses Soar's chunking capability (Laird, Newell, & Rosenbloom 1987) to represent and recall situations efficiently. When the student asks why Steve performed an action, Steve triggers the Debrief productions to recall the situation in which the action was performed (i.e., Steve's perception snapshot and mental state). Given the recalled situation, Steve uses his standard methods for goal assessment and plan construction to reconstruct his plan. Using this past plan, Steve rationalizes his action and answers follow-up questions as described in the previous section.

7 Motor Control

7.1 Overview

The motor control module receives motor commands from the cognition module and decomposes them into a sequence of lower-level commands that are sent to other components via the message dispatcher. Therefore, this module controls Steve's appearance and voice, and it allows Steve to cause changes in the virtual world.

The motor control module accepts a variety of motor commands:

- Speak a text string to a person, agent, or everyone.

- Send a speech act to an agent (this allows the agent to understand associated spoken text).

- Move to an object.

- Look at an object, agent, or person.

- Nod the head in agreement or shake it in disagreement.

- Point at an object.

- Move the hand to a neutral position (i.e., not manipulating or pointing at anything).

- Manipulate an object. For each primitive action in the cognition module's action library, there is a corresponding motor command that the motor control module accepts. These are easy to add, since they are built on top of Steve's lower-level body control capabilities, which are discussed below. Currently, Steve can press objects (e.g., buttons), flip switches, turn valves, move objects short distances (i.e., distances that do not require Steve to move also), and pull and push objects (e.g., a dipstick).

The motor control module maps these commands into messages that it sends to the message dispatcher to cause changes in the virtual world. The messages it sends fall into three categories:

actions Some messages inform the simulator of Steve's actions. Steve takes actions by sending the same messages that would be sent by a visual interface component if a person took the action: he can "touch" and "release" objects. In addition, to manipulate objects that a person would touch and drag (e.g., a throttle), Steve sends a message specifying the desired endpoint of the manipulation (e.g., set the throttle at 3000 rpm); the simulator responds to such messages by moving the object gradually to the specified endpoint.

speech When the cognition module sends a motor command to generate speech, the motor control module sends a corresponding message to the message dispatcher, which will cause the appropriate speech generation components to generate the speech. When starting Steve, a user can configure his voice (gender, speaking rate, vocal tract size, and pitch), and this voice will be used whenever he speaks.

body animation Steve supports a set of primitive body control commands. The motor control module converts motor commands from the cognition module into some combination of these primitive commands. Each primitive command causes Steve to broadcast low-level messages to the visual interface components to move or rotate Steve's body parts. To create a new body for Steve, one only has to redefine these primitive commands, which include the following:

- Move to an object.
- Look at an object, agent, or person (turn the head only).
- Look at an object, agent, or person (focus the whole body).
- Nod the head in agreement or shake the head in disagreement.
- Point at an object.
- Press an object.
- Grasp an object.
- Move the hand to a neutral position.
- Switch to a "speaking" facial expression.
- Switch to a neutral (non-speaking) facial expression.

(We are currently extending this set to include a wider variety of facial expressions.)

The ability to completely replace Steve's body by reimplementing a small set of body primitives allows us to experiment with different bodies. Since Steve teaches physical tasks, some variant of a human form seems most appropriate. The question is how much detail is

27

needed. For simply demonstrating actions, a hand is often sufficient. Adding a head opens up additional channels of communication; for example, it allows the student to track Steve's gaze. Simple representations, such as a head and hand, are actually better than a full human figure in some respects. For example, a full human figure is more visually obtrusive, which can be a disadvantage since current head-mounted displays offer a relatively narrow field of view. Nonetheless, a full human figure representation offers exciting possibilities; it allows more realistic demonstrations of physical tasks and a richer use of gestures and other types of nonverbal communication. Because our architecture makes it easy to plug in different bodies, we can evaluate the tradeoffs among them.

We have experimented with several bodies for Steve. At the simple end of the spectrum, we tried a hand alone and then a hand and head. At the complex end, we tried a full human figure, using the Jack software (Badler, Phillips, & Webber 1993) developed at the University of Pennsylvania. In the long run, Jack is an exciting prospect. However, our use of Jack was limited, since Jack comes with its own visual interface, and cannot run in others. Since his visual interface does not support our architecture for creating virtual worlds, our use of Jack was awkward: we had to send him movement commands, then query him for the resulting position and orientation of his body parts, then update our own graphical representation of Jack's body. Our most recent body for Steve was shown in Section 2. It includes the upper half of a full human figure, and the head includes movable eyes, eyelids, eyebrows, and lips.

Regardless of which body we use, our approach to animation is the same: the motor control module sends out messages to move and rotate graphical models of Steve's body parts. In contrast, some other researchers, such as Stone and Lester (1996) and Andre and Rist (1998), create a library of animation sequences, and they dynamically string these together to control their agent's behavior. Our approach provides a finer granularity for behavior and allows Steve to interact with new virtual worlds without requiring the course author to build a domain-specific library of animation clips.

The remainder of this section will discuss control of Steve's body in more detail, specifically locomotion, gaze, and hand control.

7.2 Locomotion

To control Steve's locomotion, the cognition module sends a motor command to move Steve to a specified object. To implement this command, the motor control module performs several steps. First, it plans a collision-free path from Steve's current location to the specified object. Recall from Section 5 that the perception module maintains an adjacency graph for the objects in the virtual world. An edge between two objects in the graph indicates that Steve can move from one to the other without colliding with other objects (e.g., a wall). Given Steve's current location (one object) and his specified destination (another object), the motor control module uses Dijkstra's shortest path algorithm (Cormen, Leiserson, & Rivest 1989) to compute a path.

Next, the motor control module moves Steve along this path, one leg at a time. For each leg of the path (i.e., movement from one object to the next), it does the following:

1. It determines the location, in Cartesian coordinates, where Steve should end up. To do this, it asks for a bounding sphere for the destination object from the perception module. Starting with the object's origin, it uses the object's radius and front vector to determine a point at the front, right corner of the object. Finally, it uses a default

offset to move slightly farther in front of the object and to its right. (If the course author specified an agent location offset for the object, this is used instead of the default.)

2. Next, it sends a message to the visual interface components to cause Steve's body and gaze to focus on the destination object.

3. After waiting half a second for Steve's shift of gaze to complete, it sends another message to move Steve along a path from his current location to the specified location.

When Steve arrives at the desired location, the visual interface components send a message. At this point, the perception module updates Steve's location and the motor control module sends him on the next leg of the path.

7.3 Gaze

Steve shifts his gaze in many different situations. Some of these shifts are triggered explicitly by the cognition module. Others are triggered by the motor control module in performing another motor command. In rare cases, gaze shifts can be triggered directly by the perception module (a sort of knee-jerk reaction). Gaze shifts occur in the following situations:

- When moving from location to location, he looks where he is going (triggered by motor control module).

- He looks at an object when manipulating it (triggered by motor control module).

- He looks at an object before pointing at it (triggered by motor control module).

- He looks at a person or agent when talking to them (triggered by motor control module).

- If someone other than he interacts with an object, he looks at the object (triggered by perception module).

- If he is waiting for someone, he looks at them (triggered by cognition module).

- When he is monitoring a student performing a task, he looks at the them (triggered by cognition module).

- When executing a sensing action, he looks at the object being sensed (triggered by cognition module).

- When someone informs him of something, he looks at them and nods (triggered by cognition module).

The code to control Steve's gaze has recently become more autonomous. Previously, each movement of the head required the perception module to query the visual interface components for the position of the gaze's target. After receiving this information, the motor control module sent a command to the visual interface components to rotate the head towards the target. More recently, the visual interface components accept a command to have Steve's gaze track an object, person, or agent; after animating the shift, the head is rotated automatically every frame to remain looking at the target. Moreover, the visual

interface components will recognize Steve's limits of motion; for example, if an object is moving around Steve, he will track it over his left shoulder until it moves directly behind him, at which point he will track it over his right shoulder.

7.4 Hand Control

To animate Steve's hands, we defined four possible poses for each one: resting, pointing, pressing, and grasping. When Steve is not doing something with his hands, they are resting at his sides. To manipulate or point at an object, the motor control module first gets the bounding sphere for the object. Next, it sends commands to animate the movement of the hand to the object. The pressing and grasping hands are placed at the front side of the object (as specified by the object's front vector), and their orientation is determined by the press and grasp vectors for the object, whichever is appropriate. The pointing hand is placed at the point on the object's bounding sphere closest to Steve's corresponding shoulder, oriented so that it points to the object's origin. The visual interface components animate the movement of the hand from its initial position to its target position, controlling the corresponding movements of the arms as needed.

When Steve's hand is in the proper position, the motor control module sends a command to tether it to the object (i.e., sustain a constant position and orientation relative to the object). This serves two purposes. First, it allows Steve to turn his body (e.g., to speak to the student) without causing an undesired change in the hand's position relative to the object. Second, it supports the hand animation for Steve's object manipulations. For example, after tethering Steve's finger to a button, the motor control module sends a command to the simulator to simulate the button being pressed. The simulator animates the movement of the button, and Steve's finger (and hence hand and arm) track the movement of the button, providing the illusion that he is pushing it. This approach works well when the object's movement is within the flexibility of Steve's arms and hands, which has been the case so far.

8 Status and Evaluation

Steve has been tested on a variety of Naval operating procedures. He can perform tasks on several of the consoles that are used to control the gas turbine engines that propel Naval ships, he can check and manipulate some of the valves that surround these engines, and he can perform a handful of procedures on the high-pressure air compressors that are part of these engines. We are continuing to extend his capabilities in these areas.

We are planning a set of evaluations, both within USC and in collaboration with the Air Force Armstrong Laboratory. We plan to investigate experimentally which factors contribute to the effectiveness of agent-based instruction. In particular, we are interested in determining which of the following factors are critical: a) whether or not the agents can cohabit the virtual world with students, b) the type of embodiment (graphical realization) of the agent, c) whether or not the agents have pedagogical capabilities, and d) the degree of fidelity and believability of the agent's behavior.

While this paper focuses on training a single student to perform a a one-person task, we have extended Steve to support team training. This required extensions to Steve's task knowledge to represent the various team members and the task steps for which they are responsible, extensions that allow Steve to make use of such knowledge, and extensions to allow Steve to generate and understand task-specific communication with teammates. A

short paper by Johnson et al. (1998) provides a brief overview, and the details will appear in a future paper. In our most complicated team scenario to date, five team members must work together to handle a loss of fuel oil pressure in one of the ship's gas turbine engines. This task involves a number of subtasks, some of which are individual tasks while others involve sub-teams. All together, the task consists of about three dozen actions by the various team members. We have tested this scenario with two students and five agents; three of the agents serve as the students' team members, and two of the agents serve as their tutors.

9 Related Work

The most closely related pedagogical agent for virtual reality was developed by Billinghurst and his colleagues (Billinghurst & Savage 1996; Billinghurst *et al.* 1996). Their agent inhabits a three-dimensional, simulated nasal cavity, providing assistance in sinus surgery to medical students. The agent can demonstrate surgical steps, monitor students performing surgery, intervene when a student skips a step, and tell a student what to do next when asked. However, their agent does not have an animated form; it communicates with students via a disembodied voice, and it demonstrates surgical steps by moving virtual instruments around and controlling the student's viewpoint. Unlike Steve, their agent is also capable of natural language understanding and gesture recognition. Their agent represents domain tasks as hierarchical scripts (Schank & Abelson 1977), which are similar to Steve's hierarchical plans. However, whereas Steve continually re-evaluates his plans against the current state of the virtual world, their agent merely keeps track of which steps have been executed, so it cannot adapt to unexpected events or allow the student flexibility in performing tasks as Steve can.

Lester and his colleagues are developing two animated pedagogical agents, Herman the Bug (Stone & Lester 1996) and Cosmo (Lester *et al.* 1998). These agents do not inhabit three-dimensional virtual worlds; they appear as two-dimensional characters floating on top of a two-dimensional image of a simulated world. The agents are notable for their approach to behavior control; they control their behavior by dynamically selecting audio and visual segments from a large, domain-specific library. This approach is quite labor-intensive, requiring considerable effort by artists and animators in building up the library, but it results in high quality animation. Unlike Steve, Herman and Cosmo do not interact with a simulator, nor do they have any abilities to plan or replan procedural tasks.

Several people have developed animated agents that can generate presentations. The PPP Persona (Andre & Rist 1996; Andre, Rist, & Mueller 1998) is an animated agent that combines speech and gestures to describe procedures for operating physical devices. The agent's body is controlled by flipping between different bitmap images of the agent in different poses. The agent cannot interact with a simulation, and it has no pedagogical capabilities except the ability to describe a procedure. However, it is notable for its ability to plan and schedule a sequence of presentation acts (e.g., speech and gestures). Another agent, Presenter Jack (Noma & Badler 1997), is a full human figure that uses speech, gestures, and short-range locomotion to give presentations. The human figure animation is provided by the Jack software (Badler, Phillips, & Webber 1993). Unlike Steve, the presentations are not interactive; they are scripted by a human. The work is notable for its use of a full human figure and its analysis of how gestures and gaze are used in presentations.

A variety of researchers have studied control of animated human figures. Several projects

at the University of Pennsylvania are most relevant to our work. Although none of these projects has focused on pedagogical or presentation capabilities, they are notable for their sophisticated control of animated humans. Trias et al. (1996) developed an agent that can play hide-and-seek with other virtual agents. The agent uses a hierarchical planner for some complex actions, incorporates a separate search planner for finding objects in the environment, and can move around in the virtual environment. Geib et al. (1994) developed an agent that integrates a high-level planner with a search planner for finding objects and another planner for manipulating objects. The ability to realistically grasp objects in a task-dependent manner, as described by Douville et al. (1996), would be an especially valuable extension to Steve. Cassell et al. (1994) developed an agent that integrates speech, gestures, and facial expressions in the context of a dialogue. Their agent uses a greater variety of nonverbal communicative acts than Steve, and these acts are also more tightly integrated with spoken utterances; such close coupling of verbal and nonverbal communication is crucial to achieving human-like conversational abilities in Steve.

In addition to improving Steve's conversational abilities, we must improve the student's ability to communicate with Steve. The most critical problem is that Steve is not capable of understanding natural language, so the student is limited to prespecified speech utterances. The TRAINS system (Allen *et al.* 1996; Ferguson, Allen, & Miller 1996) supports a robust spoken dialogue between a computer agent and a person working together on a task. However, their agent has no animated form, and does not cohabit a virtual world with users. Because TRAINS and Steve carry on similar types of dialogues with users, yet focus on different aspects of such conversations, a combination of the two systems seems promising. Ultimately, we must allow students to use the full range of nonverbal communicative acts that people employ in face-to-face communication. For example, the Gandalf agent (Thorisson 1996; Cassell & Thorisson 1998) supports full multi-modal conversation between human and computer. Like other systems, Gandalf combines speech, gesture, intonation and facial expression. Unlike most other systems, Gandalf also perceives these communicative signals in humans; people talking with Gandalf wear a suit that tracks their upper body movement, an eye tracker that tracks their gaze, and a microphone that allows Gandalf to hear their words and intonation. Although it may be some time before technology like Gandalf is practical, the system points the way towards an exciting future for human-computer interaction.

10 Conclusion

Steve illustrates the enormous potential in combining work in agent architectures, intelligent tutoring, and graphics. Steve draws on work in agent architectures by sensing the state of the world, assessing task goals, constructing and revising plans, and sending out motor commands to control the virtual world, all in a decision cycle that is executed multiple times per second. He draws on work in intelligent tutoring by explaining tasks, monitoring students, and answering questions. He draws on work in computer graphics to control his animated body, including locomotion, gaze, gestures, and demonstrations of actions. When combined, these technologies result in a new breed of computer tutor: a human-like agent that can interact with students in a virtual world to help them learn.

11 Acknowledgments

References

Allen, J. F.; Miller, B. W.; Ringger, E. K.; and Sikorski, T. 1996. Robust understanding in a dialogue system. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 62–70.

Anderson, J. R.; Corbett, A. T.; Koedinger, K. R.; and Pelletier, R. 1995. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences* 4(2):167–207.

Andre, E., and Rist, T. 1996. Coping with temporal constraints in multimedia presentation planning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 142–147. Menlo Park, CA: AAAI Press/MIT Press.

Andre, E.; Rist, T.; and Mueller, J. 1998. Employing AI methods to control the behavior of animated interface agents. *Applied Artificial Intelligence*. This issue.

Badler, N. I.; Phillips, C. B.; and Webber, B. L. 1993. *Simulating Humans*. New York: Oxford University Press.

Billinghurst, M., and Savage, J. 1996. Adding intelligence to the interface. In *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS '96)*, 168–175. Los Alamitos, CA: IEEE Computer Society Press.

Billinghurst, M.; Savage, J.; Oppenheimer, P.; and Edmond, C. 1996. The expert surgical assistant: An intelligent virtual environment with multimodal input. In *Proceedings of Medicine Meets Virtual Reality IV*.

Burton, R. R. 1982. Diagnosing bugs in a simple procedural skill. In Sleeman, D., and Brown, J., eds., *Intelligent Tutoring Systems*. Cambridge, MA: Academic Press. 157–183.

Cassell, J., and Thorisson, K. R. 1998. The power of a nod and a glance: Envelope vs. emotion in animated conversational agents. *Applied Artificial Intelligence*. This issue.

Cassell, J.; Pelachaud, C.; Badler, N.; Steedman, M.; Achorn, B.; Becket, T.; Douville, B.; Prevost, S.; and Stone, M. 1994. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *Proceedings of ACM SIGGRAPH '94*.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1989. *Introduction to Algorithms*. New York: McGraw-Hill.

Douville, B.; Levison, L.; and Badler, N. I. 1996. Task-level object grasping for simulated agents. *Presence* 5(4):416–430.

Ferguson, G.; Allen, J.; and Miller, B. 1996. Trains-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third Conference on AI Planning Systems.*

Geib, C.; Levison, L.; and Moore, M. B. 1994. Sodajack: An architecture for agents that search and manipulate objects. Technical Report MS-CIS-94-16/LINC LAB 265, Department of Computer and Information Science, University of Pennsylvania.

Grosz, B. J., and Sidner, C. L. 1986. Attention, intentions, and the structure of discourse. *Computational Liguistics* 12(3):175–204.

Johnson, W. L.; Rickel, J.; Stiles, R.; and Munro, A. 1998. Integrating pedagogical agents into virtual environments. *Presence.* Forthcoming.

Johnson, W. L.; Marsella, S.; and Rickel, J. 1998. Pedagogical agents in virtual team training. In *Proceedings of the Virtual Worlds and Simulation Conference,* volume 30 of *Simulation Series.* San Diego, CA: Society for Computer Simulation International.

Johnson, W. L. 1994. Agents that learn to explain themselves. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* 1257–1263. Menlo Park, CA: AAAI Press.

Korth, H. F., and Silberschatz, A. 1986. *Database System Concepts.* New York: McGraw-Hill.

Labrou, Y., and Finin, T. 1994. A semantics approach for KQML – a general purpose communication language for software agents. In *Proceedings of the Third International Conference on Information and Knowledge Management.* ACM Press.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1):1–64.

Lester, J. C.; Voerman, J. L.; Towns, S. G.; and Callaway, C. B. 1998. Deictic believability: Coordinating gesture, locomotion, and speech in lifelike pedagogical agents. *Applied Artificial Intelligence.* This issue.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91),* 634–639. Menlo Park, CA: AAAI Press.

Moore, J. D. 1993. What makes human explanations effective? In *Proceedings of the 15th Annual Conference of the Cognitive Science Society,* 131–136.

Munro, A., and Surmon, D. 1997. Primitive simulation-centered tutor services. In *Proceedings of the AI-ED Workshop on Architectures for Intelligent Simulation-Based Learning Environments.*

Munro, A.; Johnson, M.; Surmon, D.; and Wogulis, J. 1993. Attribute-centered simulation authoring for instruction. In *Proceedings of the World Conference on Artificial Intelligence in Education (AI-ED '93),* 82–89. Association for the Advancement of Computing in Education.

Newell, A. 1990. *Unified Theories of Cognition.* Cambridge, MA: Harvard University Press.

Noma, T., and Badler, N. I. 1997. A virtual human presenter. In *Proceedings of the IJCAI Workshop on Animated Interface Agents: Making Them Intelligent,* 45–51.

Rickel, J. 1988. An intelligent tutoring framework for task-oriented domains. In *Proceedings of the International Conference on Intelligent Tutoring Systems.*

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, NJ: Prentice Hall.

Sacerdoti, E. 1977. *A Structure for Plans and Behavior.* New York: Elsevier North-Holland.

Schank, R., and Abelson, R. 1977. *Scripts, Plans, Goals and Understanding.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Stiles, R.; McCarthy, L.; and Pontecorvo, M. 1995. Training studio: A virtual environment for training. In *Workshop on Simulation and Interaction in Virtual Environments (SIVE-95).* Iowa City, IW: ACM Press.

Stone, B. A., and Lester, J. C. 1996. Dynamically sequencing an animated pedagogical agent. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 424–431. Menlo Park, CA: AAAI Press/MIT Press.

Thorisson, K. R. 1996. *Communicative Humanoids: A Computational Model of Psychosocial Dialogue Skills.* Ph.D. Dissertation, Massachusetts Institute of Technology.

Trias, T. S.; Chopra, S.; Reich, B. D.; Moore, M. B.; Badler, N. I.; Webber, B. L.; and Geib, C. W. 1996. Decision networks for integrating the behaviors of virtual agents and avatars. In *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS '96)*, 156–162. Los Alamitos, CA: IEEE Computer Society Press.

Weld, D. S. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Appendix B

Pedagogical Agents

Invited Paper at the International Conference on Computers in Education

Pedagogical Agents

W. Lewis Johnson
Center for Advanced Research in Technology for Education (CARTE)
USC / Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292 USA
johnson@isi.edu; http://www.isi.edu/isd/carte/

Abstract

Pedagogical agents are autonomous agents that support human learning, by interacting with students in the context of interactive learning environments. They extend and improve upon previous work on intelligent tutoring systems in a number of ways. They adapt their behavior to the dynamic state of the learning environment, taking advantage of learning opportunities as they arise. They can support collaborative learning as well as individualized learning, because multiple students and agents can interact in a shared environment. Given a suitably rich user interface, pedagogical agents are capable of a wide spectrum of instructionally effective interactions with students, including multimodal dialog. Animated pedagogical agents can promote student motivation and engagement, and engender affective as well as cognitive responses. This paper surveys current research in pedagogical agents, and describes some current methods for investing agents with pedagogical capabilities.

1. Introduction and Background

Over the last several years there has been significant progress in techniques for creating autonomous agents, i.e., systems that are capable of performing tasks and achieving goals in complex, dynamic environments. Architectures such as RAP (Firby 1994) and Soar (Laird et al 1987) have been used to create agents that can seamlessly integrate planning and execution, adapting to changes in their environments. They are able to interact with other agents, and collaborate with them to achieve common goals (Müller 1996, Tambe et al 1995). Robust autonomous agents have been built in a variety of application areas, including mobile robots (Murphy and Hershberger 1996), softbots (Doorenbos et al 1997), and entertainment (Foner 1997).

A promising application area for autonomous agents is education and training. The term *pedagogical agent* is used to refer to agents that are designed to support human learning, interacting with students in order to facilitate their learning. Although pedagogical agents build upon previous research on intelligent tutoring systems (Wenger 1987), they bring a fresh perspective to the problem of facilitating on-line learning, and address issues that previous intelligent tutoring work largely ignored. Peda-gogical agents can adapt their instructional interactions to the needs of the student and the current state of the learning environment, helping students to overcome their difficulties and taking advantage of learning oppor-tunities. They can collaborate with students and with other agents, integrating action with instruction; this contrasts with typical intelligent tutoring systems that only comment from the side and are only able to interact with one student at a time. They are able to provide continual feedback to students during their work. Finally, they can appear to the students as lifelike characters, and induce the same kinds of affective responses that other kinds of lifelike characters engender.

The move from intelligent tutoring systems to pedagogical agents began about ten years ago, when researchers began to explore new types of interactions between computers and students. (Chan and Baskin 1990) developed a simulated *learning companion*, which acts as a peer instead of a teacher. (Dillenbourg 1996) investigated the interaction between real students and computer-simulated students as a collaborative social process. (Chan 1996) has investigated other types of interactions between students and computer systems, such as competitors or reciprocal tutors. Current pedagogical agent work further develops the notion of learning-system-as-agent by placing learners and pedagogical agents in rich interactive environments and broadening the bandwidth of interaction between learners and agents. This increases the complexity of interaction between pedagogical agents and their environment, and hence the need for agent architectures that can manage this complexity; it also affords new possibilities for interacting with students in order to foster learning.

Because pedagogical agents are autonomous agents, they inherit many of the same concerns that autonomous agents in general must address. They must exhibit robust behavior in rich, unpredictable environments; they must coordinate their behavior with that of other agents, and must manage their own behavior in a coherent fashion, arbitrating between alternative actions and responding to a multitude of environmental stimuli. Their environment includes both students and the learning environment in which the agents are situated. Student behavior is by nature unpredictable, since students may exhibit a variety of aptitudes, levels of proficiency, and learning styles.

The need to support instruction imposes a combi-nation of requirements on pedagogical agents that other

types of agents do not always satisfy. They need to have knowledge of the tasks and skills that the students are learning to perform, so that they can participate in the students' activities as needed. However, a pedagogical agent requires different types and representations of domain knowledge than do agents whose job is simply to perform the task. A pedagogical agent usually needs to be capable of offering helpful hints when needed, giving clarifying explanations, and answering student questions. In order to support such instructional interactions, a pedagogical agent requires a deeper understanding of the rationales and relationships between actions than would be needed simply to perform a task (Clancey 1983).

Particularly interesting issues arise when pedagogical agents appear to the student as animated characters. An animated pedagogical agent can engage in a continuous dialog with the student, and emulate aspects of multimodal dialog between humans in instructional settings. Such animated agents share aspects in common with synthetic agents developed for entertainment applications (Elliott and Brzezinski 1998): they need to give the user an impression of being lifelike and believable, producing behavior that appears to the user as natural and appropriate. In the case of pedagogical agents, they must produce behavior that seems natural and appropriate for the role that the agent is playing, i.e., a virtual instructor or guide. As (Bates et al. 1992) have argued, it is not always necessary for an agent to have deep knowledge of a domain in order for it to generate behavior that is believable. To some extent the same is true for pedagogical agents. We frequently find it useful to give our agents behaviors that make them appear knowledgeable, attentive, helpful, concerned, etc. These behaviors may or may not reflect actual knowledge representations and mental states and attitudes in the agents. However, the need to support pedagogical interactions generally imposes a closer correspondence between appearance and internal state than is typical in agents for entertainment applications. We can create animations that give the impression that the agent is knowledgeable, but if the agent is unable to answer student questions and give explanations, the impression of knowledge will be quickly destroyed.

This article is intended to introduce the reader to some of key capabilities of pedagogical agents, and techniques for implementing these capabilities. A full technical account is beyond the scope of this brief article; the reader is encouraged to consult the publications cited in the reference section for further information.

2. Example Pedagogical Agents

The following discussion will make frequent reference to the specific instances of pedagogical agents that have been built in research laboratories around the

world. These systems will be used to illustrate the range of behaviors that these agents are capable of producing, and the design requirements that such agents must satisfy. Some of these behaviors are similar to those found in intelligent tutoring systems, others are quite different and unique.

USC / Information Sciences Institute's Center for Advanced Research in Technology for Education (CARTE) has developed two pedagogical agents: Steve (Soar Training Expert for Virtual Environments) and Adele (Agent for Distance Learning – Light Edition). Steve is an advanced prototype designed to interact with students in networked immersive virtual environments, and has been applied to naval training tasks such as operating the engines aboard US Navy surface ships (Johnson et al 1998, Johnson and Rickel 1998, Rickel and Johnson 1998, and Rickel and Johnson 1997). Immersive virtual environments permit rich interactions between humans and agents; the students can see the agents in stereoscopic 3D and hear them speak, and the agents rely on the virtual environment's tracking hardware to monitor the student's position and orientation in the environment. Steve software is combined with 3D display and interaction software by Lockheed Martin, simulation authoring software by USC Behavioral Technologies Laboratory, and speech recognition and generation software by Entropic Research to produce a rich virtual environment in which students and agents can interact in instructional settings. Adele, in contrast, was designed to run desktop platforms with conventional interfaces, in order to broaden the applicability of pedagogical agent technology. Adele runs in a student's Web browser, and is designed to integrate into Web-based electronic learning materials (Johnson and Shaw 1997). Adele-based courses are currently being developed for continuing medical education in family medicine and graduate level geriatric dentistry, and further courses are planned for development both at USC and at the University of Oregon.

North Carolina State University's Multimedia Laboratory has developed two pedagogical agents: Herman the Bug (Lester and Stone 1997) and Cosmo (Towns et al 1998). Herman was developed as part of the Design-A-Plant learning environment, a learning environment that helps middle school students to understand botanical anatomy and physiology by designing plants for various hypothetical environments. Cosmo operates in the realm of computer networks, and helps students to solve problems such as how to route packets between network hosts so as to avoid high-traffic routes. These projects have investigated a number of research issues such as how to combine various agent behaviors in order to enhance the impression of believability on the part of the student, and how to manage mixed-initiative dialog. Herman the Bug has been used in large-scale empirical

evaluations that have demonstrated the effectiveness of pedagogical agents in facilitating learning (Lester et al 1997).

André, Rist, and Müller at DFKI at the University of Saarbrücken have developed an animated persona for giving on-line presentations, called PPP Persona (André et al 1998). PPP Persona guides the learner through Web-based materials, using pointing gestures to draw the student's attention to elements of the Web pages, and providing commentary via synthesized speech. The underlying PPP system generates multimedia presentation plans for PPP Persona to present; PPP Persona then executes the plan adaptively, modifying it in real time based on user actions such as repositioning the persona on the screen or asking follow-on questions.

3. Types of Interaction with Pedagogical Agents

Pedagogical agents can interact with students in a number of different ways. The following examples illustrate the various types of student-agent interactions that have been explored to date. Screen shots and text descriptions have been used to give the reader a sense of how these interactions are manifested. However, such static presentations are a poor substitute for live interactions with these agents. Live demonstrations and downloadable software are available on the World Wide Web, both at the CARTE Web site (http://www.isi.edu/isd/carte/), and the DFKI PPP Web site (http://www.dfki.edu/~jmueller/ppp).

When students are first introduced to a topic, it is often necessary to *demonstrate* to them how to solve problems and perform tasks. Pedagogical agents are well suited to performing such demonstrations. Figures 1 and 2 show Steve performing such a demonstration, showing how to operate a High Pressure Air Compressor (HPAC) aboard a US Navy ship.

Demonstrations by themselves are not very instructive unless the student watching the demonstration understands what is being done. Steve therefore integrates his demonstrations with explanatory commentary. Text descriptions of objectives and actions are generated, and are uttered using a commercial text-to-speech generator. Figure 1 shows Steve in the context of explaining what to do, where he says the following.

> I will now perform a functional check of the temperature monitor to make sure that all of the alarm lights are functional. First press the function test button. This will trip all of the alarm switches, so all of the alarm lights should illuminate.

Steve then proceeds with the demonstration, as shown in Figure 2. As the demonstration proceeds Steve points out important features of the objects in the environment that relate to the task. For example, when the alarm lights illuminate, the Steve points to the lights, and says: "All of the alarm lights are illuminated, so they are all working properly."
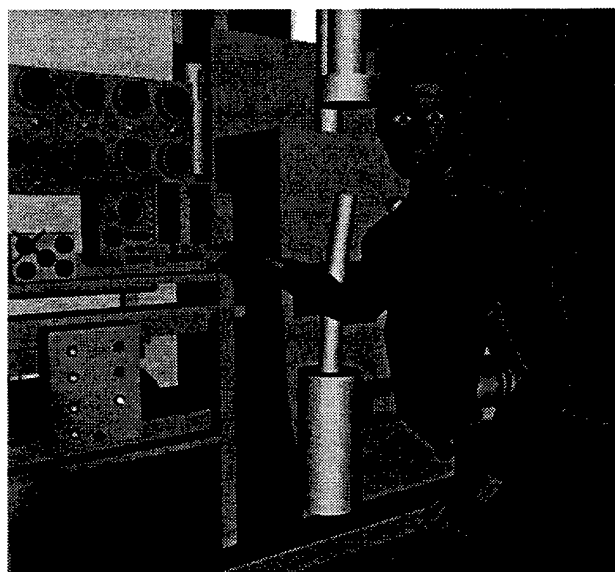

Figure 1. Steve pointing to a button on the HPAC console
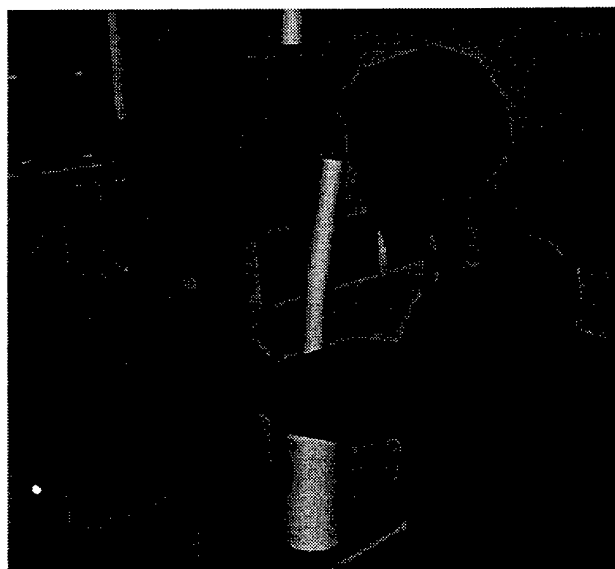

Figure 2. Steve pressing a button on the console

Having an agent demonstrate tasks, instead of simply showing a student a video of the procedure, offers a number of advantages. The student is free to move around in the environment, and view the demonstration from various perspectives. If the demonstration is being performed in a dynamic environment, as in Steve's case, the demonstration dynamically adapts to the current state of the environment. This allows Steve to demonstrate the

operation of the HPAC in different initial states and failure modes. Steve also adapts his demonstrations according to the actions of the user. Steve is gazing toward the user in Figure 1: this illustrates how Steve dynamically directs his gaze toward the student during the demonstration whenever he wants to attract the student's attention or speak to the student. Demonstrations also adapt to shifts in control between Steve and the student. At any time the student can say, "Let me finish" to Steve, at which point Steve lets the student complete the task himself while Steve monitors the student's actions. Then if the student encounters difficulties he can ask Steve to "Show me what to do," at which point Steve demonstrates the appropriate next action to take. Thus *student monitoring*, the ability to track and interpret the intent behind the student's actions, is essential in order to permit mixed initiative demonstrations.
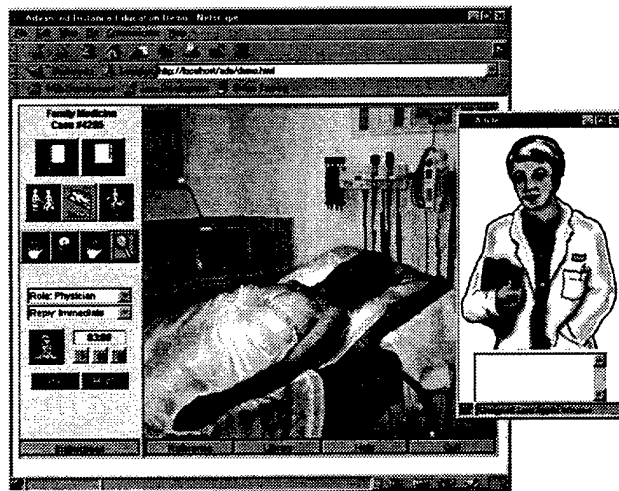

Figure 3. Adele observing a case

Steve is alone among current pedagogical agents in having a well-developed demonstration capability, integrating demonstrations with explanations. However, other agents have the ability to *guide* a student through a task, much as intelligent tutoring systems do, and guiding is similar to demonstration in that it helps students unfamiliar with the task to work their way through it. For example, Adele has several capabilities that help to guide the student, which she invokes if she is operating in Advisor mode.[1] If the student performs an action that is inconsistent with standard practice, she will interrupt the student and suggest an action to perform instead. Figures

_____

[1] Other interaction modes include Practice mode, where she is available to give advice if asked but does not interrupt if the student, and Examination mode, where she observes and evaluates the student but does not offer assistance.

3 and 4 illustrate this. Figure 3 shows an application of Adele to clinical decision making. Adele observes as the student performs a clinical evaluation of a patient. In this example, a patient has arrived complaining of a lump on her chest. If the student starts ordering laboratory tests such as chest X-rays without first completing a physical examination of the patient, Adele will interrupt saying that "Before ordering a chest X-ray it would be useful to listen to the condition of the lungs."
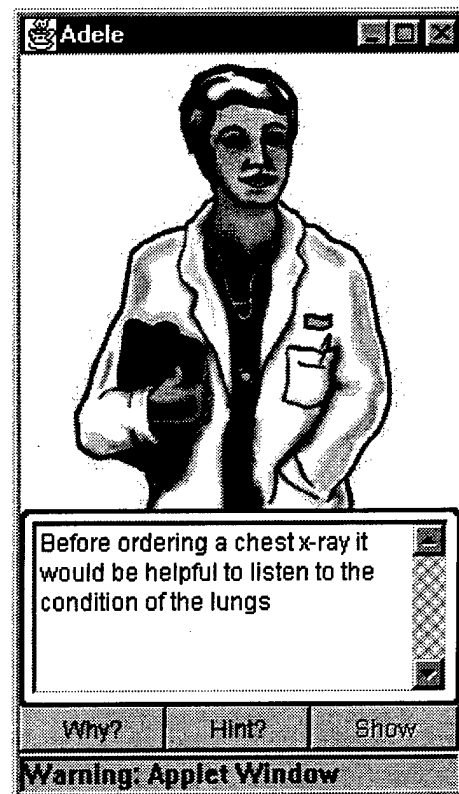

Figure 4. Adele critiquing a student's actions

Steve and Adele can both assist the student by means of *hints*. These help to guide the student if he or she is unclear about what to do. Hinting is usually available at any time in courses assisted by Steve or Adele, unless the student is being tested on their proficiency with the skill being taught.

Expert instructors frequently use *leading questions* to make sure that students properly understand the current situation as they are solving a problem. Pedagogical agents can also employ leading questions to probe students' understanding. For example, in one of the clinical decision making courses using Adele, the students are presented with the case of a patient who has a lesion that has been slowing growing over a period of months. The student finds this out by asking the simulated patient a series of questions about her disease history. As soon as the student finds out that the patient's lesion has been

growing slowly, Adele jumps in and asks the student identify the type of disease suggested by such a disease process, i.e., fibroma.

Such use of leading questions is a special case of *opportunistic instruction*, i.e., providing instruction when situations arise where it is appropriate. Opportunistic instruction is a valuable capability for pedagogical agents, because it allows instruction to be delivered to students in the context of solving problems, so that the students can immediately put it to use. Herman the Bug, for example, makes extensive use of problem solving contexts as opportunities for instruction. When the student is working on a selecting a leaf to include in a plant, Herman uses this as an opportunity to provide instruction about leaf morphology. Another type of opportunistic instruction commonly provided by Adele is providing students with pointers to on-line medical resources that are relevant to the current stage of the case work-up. For example, when the student selects a diagnostic procedure to perform on the simulated patient, Adele may point the student to video clips showing how the procedure is performed.

All of the agents mentioned in the previous section are capable of generating *explanations* as needed. Whenever Steve or Adele gives a hint, the student can ask "Why" to find out the rationale for the hint. Steve takes this further by allowing a series of "Why" questions, each of which causes Steve to present higher-level rationales, until Steve runs out of rationales to give. Herman and Cosmo will generate unsolicited explanations if the student makes a mistake or seems to be having difficulty deciding how to proceed with the problem. Suppose, for example, that a student is selecting a type of leaf to use in a cold climate. If the student rolls over the textual descriptions on the screen for thirty seconds, and does not choose a leaf type, Herman will jump in and provide an explanation the relationship between cold temperature and leaf size, leaf thickness, and leaf skin thickness. If the explanation does not enable the student to make a choice, Herman will then provide direct advice of what action to perform.

Animated pedagogical agents are increasingly invested with the capability of generating *emotive responses* to student actions. Emotive behaviors such as facial expressions and body language can help engage and motivate the learner, and alleviate student frustration by appearing to empathize with the learner. A wide repertoire of emotive behaviors have been built into Cosmo, which are combined with speech utterances and other types of nonverbal gestures when generating explanations (Towns, FitzGerald, and Lester 1998). Behaviors such as applause are used in conjunction with congratulatory speech acts; head scratching or shrugging are used when Cosmo poses a rhetorical expression.

Adele employs emotive facial expressions, showing satisfaction when a student answers a question correctly, agitation if the a situation has arisen in the learning environment that requires the student's immediate attention (e.g., the patient is experiencing difficulty breathing), and displeasure if the student makes an error that she should have known how to avoid.

The above examples have all described one-on-one interactions between a student and an agent. However, pedagogical agents can also be naturally applied to *collaborative and team learning*. Team instruction has been a particular focus of investigation for Steve. Steve's learning environment can be simultaneously inhabited by multiple students, each of whom plays the role of a different crew member aboard a simulated ship. Steve agents may be used to assist individual team members, or play the role of missing team members. This requires having each Steve agent understand how the roles of the various team members interact and depend upon each other.

These examples do not exhaust the range of capabilities that are useful for pedagogical agents to provide. Other capabilities that have been found to be important for intelligent tutoring systems, such as student modeling and assessment, are potentially useful for pedagogical agents as well. As pedagogical agents are deployed in instructional settings, it is expected that these further intelligent tutoring capabilities will be incorporated into them.

4. Architectures for Pedagogical Agents

Given the range of capabilities that pedagogical agents are intended to provide, it is essential that an agent architecture be used that permits robust integration and reconciliation of these capabilities, and which is capable of generating behavior in real time. Three architectural approaches are taken in the agents described in this article: the behavior sequencing approach, the layered generative approach, and the state machine compilation approach.

The behavior sequencing approach

In the behavior sequencing approach, behaviors are assembled out of a collection of prerecorded primitive animations, sounds, and speech elements. The media primitives are organized into a *behavior space*, structured along several dimensions such as degree of exaggeration of movement or types of body part involved in the movement. Animated behaviors are created by a *behavior sequencing engine* that constructs coherent paths through the behavior space at real time. Assembling

behaviors out of prerecorded segments saves time in creating the animation, and can yield high-quality animations if the segments are created by expert animators. The behavior sequencing engine is responsible for all planning decisions leading up to the creation of the animation sequence.

The following example of behavior sequencing in Herman the Bug illustrates this process. If Herman intervenes in the lesson, say because the student is unable to decide on a leaf type, the behavior sequencing engine first selects a topic to provide advice about, some component of the plant being constructed. The engine then chooses how direct a hint to provide: an indirect hint may talk about the functional constraints that a choice must satisfy, whereas a direct hint proposes a specific choice. The level of directness then helps to determine the types of media to be used in the presentation: indirect hints tend to be realized as animated depictions of the relationships between environmental factors and the plant components, while direct hints are usually rendered as speech. Finally, a suitable coherent set of media elements with the selected media characteristics are chosen and sequenced.

The behavior sequencing approach is well suited for applications employing 2D graphics or 3D graphics where the camera is fixed. The main limitation of the approach is that it does not provide for real-time adaptation of behavior. If the student performs an action in the middle of execution of the sequence, the behavior sequence may no longer be appropriate and will have to be recomputed.

The layered generative approach

The layered generative approach generates animations in real time, instead of assembling them from a library of multimedia elements. The architecture is divided into cognitive decision-making layer and a perceptual-motor layer responsible for monitoring the environment and generating the animations. Similar layered architectures are used in other animated agents such as the ALIVE system (Blumberg and Galyean 1995). The cognitive component receives information about the state of the environment from the perceptual component, which may filter and abstract it into a form that is usable by the cognitive component. The cognitive component continually evaluates the state of the environment, and makes decisions about actions that the agent should be performed. These are realized in the form of motor commands that are then sent to the perceptual-motor layer for execution. This layered approach is useful because it enables a separation of concerns, allowing agent decision making and persona control to be dealt with separately. However, it increases the amount of rendering computation required to create the behavior, and provides

less scope for graphic artists to customize the agent's behavior.

Steve's architecture is a particularly clear instance of this layered approach. Steve consists of three main modules: perception, cognition, and motor control. The perception module monitors the underlying message bus used in the overall Virtual Environment for Training system for interprocess communication. If the Steve agent is instructing a student, the perceptual module tracks the student in the virtual environment, by querying the human-computer interface manager controlling the student's display. This provides information about the student's location, orientation, field of view, and interactions between the student and objects in the virtual environment. The perceptual information also receives information from the simulator module managing the state of the information about state changes, and is notified when students or other agents speak. The perceptual module uses this information to construct and maintain a symbolic model of the state of the world. The cognitive component accesses this model as needed. When it decides to take an action, the action is transmitted as a motor command to the motor control module. The motor control module in turn moves the agent's body through the virtual world in response to the command.

The following is a list of the motor commands supported by Steve's motor control module:
- Speak a text string to a person, an agent, or everyone.
- Send a speech act to an agent (e.g., Inform the agent of something).
- Move to an object.
- Look at an object, agent, or person.
- Nod the head in agreement or shake it in disagreement.
- Point at an object.
- Move the hand to a neutral position at the side of the body.
- Manipulate an object, e.g., grasp it, turn it, flip it, push it, pull it, etc.

These are motor commands are translated into one or more primitive actions both on Steve's graphical body and on the objects that Steve's body is manipulating. A command may result in a series of actions being performed, e.g., if Steve chooses to move to a particular object it may be necessary for Steve's body to perform a series of motions along a path in order to arrive at the intended object.

The cognitive component of Steve is organized into three main layers:
- Domain-specific task knowledge
- Domain-independent pedagogical capabilities
- The Soar cognitive architecture (Laird et al 1987).

Domain-specific task knowledge is provided for each task that Steve helps instruct. This is in the form of a plan

schema for carrying out the task, as will be described in the next section. The domain-independent pedagogical capabilities include the general pedagogical functions described in Section 3, such as demonstration, student monitoring, and explanation.

The choice of Soar in this context merits some further discussion. Soar was chosen because it has been used extensively to create autonomous agents that model human decision-making and behavior, e.g., in wargaming simulations (Tambe et al 1995), and in modeling human learning (Hill and Johnson 1993). It has been applied successfully to tasks that involve interacting with dynamic simulations. It provides support for integrating and arbitrating between multiple capabilities or areas of expertise.

Cognition in Soar involves repeatedly applying operators on working memory representations. Processing involves repeatedly executing a decision cycle, consisting of the following steps.
- Input information from the external environment into working memory.
- Apply elaboration rules, which draw inferences from the information in working memory. Some of these elaboration rules may propose operators for the agent to perform.
- Select an operator to apply from among the operators that are proposed by the elaboration rules.
- Execute the operator. This may involve issuing commands to manipulate the external environment.

The following capabilities in Soar make this decision cycle mechanism effective for controlling agents. A nonmonotonic reasoning mechanism is used to maintain consistency of working memory. When an elaboration rule fires and creates new working memory elements, those working memory elements remain only as long as the triggering conditions of the rule are true; if they become false, the working memory elements are retracted. In a dynamic environment such as Steve's, where the input values to Soar are continually changing, this mechanism helps to maintain consistency between working memory and the environment. Another helpful feature is the ability to write explicit rules for deciding between proposed operators, called preference rules. These preference rules thus make explicit the basis for arbitrating between alternative actions, which is helpful in pedagogical agents that are capable of multiple pedagogical actions.

The state machine compilation approach

The state machine compilation approach, as exemplified in PPP Persona, addresses the issue of real-time adaptation of agent behavior, while limiting the amount of rendering computation required. As in the behavior sequencing approach, this approach composes behaviors out of animation primitives, consisting of individual animation frames and uninterruptible image sequences. However, unlike the behavior sequencing approach, the behaviors are executed by a state machine that can adapt at run time to student actions. This approach is based in part on the approach used in the Persona architecture developed at Microsoft Research (Ball et al 1997).

Presentation generation proceeds through the following steps. Prior to execution of the plan, the persona's behaviors are compiled into a state machine called a *behavior monitor*. The behavior monitor executes the sequences of primitive behaviors used in more complex behavior sequences, and combines these with unplanned behaviors such as idle-time actions (breathing or tapping a foot) and reactive behaviors (such has hanging suspended when the user picks up and moves the persona with the mouse). The behavior monitor defines a space of possible behaviors for the persona. Then for any given presentation, a multimedia presentation planner generates a set of presentation actions to be performed, and a schedule for performing them, with qualitative or quantitative temporal constraints. When behavior execution is initiated, the persona follows the preliminary schedule. The behavior monitor may execute additional actions. These in turn may require the schedule to be updated, subject to the constraints of the presentation plan. The result is behavior that is adaptive and interruptible, while maintaining coherence to the extent possible.

5. Knowledge Representations for Pedagogical Agents

All pedagogical agents require some sort of knowledge representation describing the subject of instruction. These representations should be flexible enough to support the wide range of pedagogical functions supported by these agents. They should also facilitate knowledge acquisition or authoring, to facilitate the integration of pedagogical agents into instructional materials. These requirements constrain the types of knowledge representations that may be used.

Steve and Adele both support a wide range of pedagogical actions, and address the needs of knowledge acquisition; their representation is perhaps the most highly constrained. The representation of task knowledge used in these agents is hierarchical plans. Such hierarchical descriptions facilitate authoring, because it is usually relatively easy for most subject matter experts to conceptualize a task as a hierarchical set of steps and substeps. Each step in Steve is implemented as a Soar

operator that can be executed by Steve's decision-making mechanism.

Steve's hierarchical plan representation is augmented relations between steps: causal links and ordering constraints. These facilitate reasoning about the relevance of task steps in the current situation, and explanation of the rationales for actions. Figure 5 shows an example task description in Steve, for performing a functional test of the console shown in Figures 1 and 2. The task has three steps: to press the function test button, to check the alarm lights for illumination, and to extinguish the alarm lights. Each causal link identifies the desired effect of a step, and a subsequent step that depends upon this effect. For example, pressing the function test button causes the console to change state to test-mode, which makes it possible to check the alarm lights. Ordering constraints define a partial ordering between steps. For example, the function test button must be pressed before the alarm lights can be checked.

Task: functional-test

Steps:
    press-functional-test
    check-alarm-light
    extinguish-alarm

Causal links:
    press-function-test achieves test-mode
        for check-alarm-light
    check-alarm-light
        achieves know-whether-alarm-functional
        for end-task
    extinguish-alarm achieves alarm-off for end-task

Ordering constraints:
    press-function-test before check-alarm-light
    check-alarm-light before extinguish-alarm

Figure 5. A plan for Steve

Adele employs a similar hierarchical plan representation, but the preconditions and effects of each step are made explicit, instead of being implied by the causal links. The reason for this is that Adele generates explanations differently from Steve, and therefore does not require the same causal link structures. When Steve explains a step, it is in terms of the results that that step achieves. Adele, in contrast, explains steps in terms of motivating facts about the domain. For example, if a student asks why a chest X-ray should be ordered the explanation should be about why an X-ray is important for this type of case. Adele's plans also differ because they can include multiple possible actions that the student might choose take, depending upon the situation.

Team tasks are represented in these frameworks by assigning roles to task steps. The task thus describes what the team as a whole will need to do to perform the task. When causal links exist between steps performed by different roles, these define the interrelationships between roles. A Steve agent tutoring a student in a team setting can use this information to explain to the student how his actions affect the actions of other team members. The interactions between roles also govern the nonverbal gestures that Steve employs. If a Steve agent is waiting for another team member to complete a step in the task, he indicates this by turning and looking at the other team member. Gaze between team members is an effective indicator of the relationships between roles.

These plan representations are simpler than the complex plan representations that sometimes appear in the planning community. This was done in part to make authoring easier. For example, the author has limited ability to define constraints on variable bindings; complex variable binding mechanisms are difficult for non-programmers to understand, and are therefore typically omitted from instructional authoring tools. Nevertheless work is required to mediate between notations familiar to subject matter experts and notations used by Steve and Adele. Multiple techniques are being employed to bridge this gap. Graphical diagramming tools have been developed that aid authors in creating hierarchical tasks descriptions and expressing the relationships between plan steps. We are also experimenting with machine learning techniques that allow Steve to generalize task descriptions automatically by experimenting on the simulation environment to see how changes to the plan affect the outcome (Angros et al 1997).

6. Summary

Pedagogical agents are an interesting application of autonomous agent technology, which is fast finding its way into practical applications. Adele is currently being readied for inclusion in several on-line courses, in a range of different departments at USC. Herman and Cosmo have been evaluated closely in large-scale empirical evaluations. PPP Persona has been applied to a variety of applications, and is available for download over the World Wide Web.

Pedagogical agents bridge the gap between so-called believable agents and other kinds of intelligent agents. Their behaviors and expressions are deliberately designed in order to appear lifelike and responsive to the student. However, it is still necessary for these agents to have a rich representation of knowledge of the task domain, in order to support a wide range of pedagogical capabilities.

Acknowledgments

References

André, E., Rist, T., and Müller, J. , 1998. Integrating reactive and scripted behaviors in a life-like presentation agents. In K.P. Sycara and M. Wooldridge (Eds)., *Proc. of the Second International Conference on Autonomous Agents*, pp. 261-268, ACM Press.

Angros, R. Jr., Johnson, W.L. and Rickel, J., 1997. Agents that Learn to Instruct, *AAAI'97 Fall Symposium Series: Intelligent Tutoring Systems Authoring Tools*, Nov. 1997.

Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stankosky, A., Thiel, D., van Dantzich, M., and Wax, T., 1997. Lifelike computer characters: Th Persona project at Microsoft, in Bradshaw, J.M. (Ed.), *Software Agents*, AAAI/MIT Press, Menlo Park, CA.

Bates, J., Loyall, A.B., and Reilly, W.S., 1992. Integrating reactivity, goals, and emotion in a broad agent, in *Proc. of the Fourteenth Annual Conference of the Cognitive Science Society*, pp. 696-701.

Blumberg, B.M. and Galyean, T.A., 1995. Multi-level direction of autonomous creatures for real-time virtual environments. *SIGGRAPH 95 Conference Proceedings*, pp. 47-54.

Chan, T.-W., 1996. Learning companion systems, social learning systems, and the global social learning club. *Journal of AI in Education* 7 (2), 125-159, 1996.

Chan, T.-W. and Baskin, A.B., 1990. Learning companion systems, in Frasson, C., and Gauthier, G. (Eds.), *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, ch. 1.

Clancey, W.J., 1983. The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence* 20(3), pp. 215-251.

Dillenbourg P., 1996. Some technical implications of distributed cognition on the design on interactive learning environments. *Journal of AI in Education* 7(2), 161-179.

Doorenbos, R.B., Etzioni, O., and Weld, D.S., 1997. A scalable comparison-shopping agent for the World-Wide Web, in Johnson, W.L. and Hayes-Roth, B., 1997 (Eds.), *Proceedings of the First International Conference on Autonomous Agents*, ACM Press.

Elliot, C. and Brzezinski, J, 1998. Autonomous agents as synthetic characters, *AI Magazine* 19(2), pp. 13-30.

Firby, R.J., 1994. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems*.

Foner, L.N., 1997. Entertaining agents: A case study, in Johnson, W.L. and Hayes-Roth, B., 1997 (Eds.), *Proceedings of the First International Conference on Autonomous Agents*, ACM Press.

Hill, R.W. and Johnson, W.L., 1993. Designing an Intelligent Tutoring System Based on a Reactive Model of Skill Acquisition, *Proceedings of the World Conference of Artificial Intelligence in Education*.

Johnson, W.L. and Rickel, J., 1998. Steve: An animated pedagogical agent for procedural training in virtual environments. *SIGART Bulletin* 8, pp. 16-21.

Johnson, W.L., Rickel, J., Stiles, R., and Munro, A., 1998. Integrating pedagogical agents into virtual environments. *Presence* 7(5), forthcoming.

Laird, J.E., Newell, A., and Rosenbloom, P.S., 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1), pp. 1-64.

Lester, J.C., Converse, S., Stone, B., Kahler, S., and Barlow, T., 1997. Animated pedagogical agents and problem-solving effectiveness: A large-scale empirical evaluation. In *Proceedings of the Eighth World Conference on Artificial Intelligence in Education*, pp. 23-30. IOS Press, Amsterdam.

Lester, J.C. and Stone, B.A., 1997. Increasing believability in animated pedagogical agents. In Johnson, W.L. and Hayes-Roth, B. (Eds.), *Proceedings of the First International Conference on Autonomous Agents*, pp. 16-21, ACM Press.

Müller, J.P., 1996. *The Design of Intelligent Agents: A Layered Approach*, Springer-Verlag, Berlin.

Murphy, R.R. and Hershberger, D., 1996. Classifying and recovering from sensing failures in autonomous mobile robots. In *Proceedings of AAAI-96*, pp. 922-929, Morgan Kaufmann, Menlo Park, CA.

Rickel, J. and Johnson, W.L., 1998. Animated agents for procedural training in virtual reality: perception, cognition, and motor control. *Applied Artificial Intelligence Journal*, to appear.

Rickel, J. and Johnson, W.L., 1997. Intelligent Tutoring in Virtual Reality: A Preliminary Report, *Proc. of the Intl. Conf. on Artificial Intelligence in Education*, pp. 294-301.

Johnson, W.L. and Shaw, E., 1997. Using Agents to Overcome Deficiencies in Web-Based Courseware, *Proceedings of the AI-ED 97 Workshop on Pedagogical Agents*, pp. 48-55.

Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S., and Schwamb, K., 1995. Intelligent agents for interactive simulation environments, *AI Magazine* (6)1, pp. 15-39.

Towns, S.G., Callaway, C.B., Voerman, J.L., and Lester, J.C., 1998. Coherent gestures, locomotion, and speech in life-like pedagogical agents. IUI '98: 1998 International Conference on Intelligent User Interfaces, pp. 13-20, ACM Press.

Towns, S.G., FitzGerald, P.J., and Lester, J.C., 1998. Visual emotive communication in lifelike pedagogical agents. In *Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, San Antonio, TX.

Wenger, E., 1987. *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge*. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

LOCKHEED MARTIN

Lockheed Martin Missiles & Space
Advanced Technology Center
3251 Hanover Street, Palo Alto, California 94304-1191

Program Officer
Office of Naval Research
Code 342 - Helen Gigley
800 North Quincy Street
Arlington, VA 22217-5660

Subject: Contract No. N00014-95-C-0179 "Virtual Environments for Training"

Reference: (a)   CLIN 0004, CDRL A001 "Progress Report"

Enclosure: (1)   Progress Report covering the period 4/1/98 - 6/30/98 (1 hard copy &
                   1 diskette)

Dear Dr. Gigley:

The report required per Reference (a) is provided as Enclosure (1).

Should you have questions or require additional information, please contact me at
650/424-2006, or by fax at 650/424-3330. I can also be reached by email at
davina.brown@lmco.com

Very truly yours,

Davina Brown
Contract Specialist

cc: Director, Naval Research Laboratory, Attn: Code 2627
        Washington, DC 20375 (w/ 1 copy of enclosure)

        Defense Technical Information Center
        8725 John Kingman Road, Suite 0944
        Fort Belvoir, VA 22060-6218   (w/ 2 copies of enclosure)

        DCMC/Lockheed Martin, B/107, Sunnyvale, CA   (w/ 1 copy of enclosure)