# Agents that Learn to Instruct

**Richard Angros, Jr.**

Hughes Aircraft Company &
Information Sciences Institute & Computer Sciences Department
University of Southern California

**W. Lewis Johnson and Jeff Rickel**

Information Sciences Institute & Computer Sciences Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292-6695
angros, johnson, rickel@isi.edu

**Abstract**

This paper describes a software agent that learns procedural knowledge from a human instructor well enough to teach human students. In order to teach, the agent needs more than the ability to perform a procedure. It must also be able to monitor human students performing the procedure and be able to articulate the reasons why actions are necessary. Our research concentrates on helping an instructor instruct the agent in a natural manner, on reducing the burden on the instructor, and on focusing learning on the procedure being taught. Initially the agent has little domain knowledge. The instructor demonstrates a procedure by directly manipulating a simulated environment. However, one demonstration is not sufficient for understanding the causal relationships between a demonstration's actions. Unfortunately, the more demonstrations a procedure requires, the greater the instructor's burden. However, fewer demonstrations can be required if the agent autonomously experiments. Our experiments attempt to understand the causal dependencies between a demonstration's actions by perturbing the order of the demonstration's actions.

## Introduction

One way for human students to practice tasks is in a virtual reality version of their work environment. Not only could students practice on many training problems, but they could also have exposure to many unusual situations. This approach for teaching procedural tasks could be especially valuable in domains where training is expensive or failure is hazardous, such as surgery and machine maintenance.

However, training could be more effective if a helpful instructor were available. Some forms of help include demonstrating a task to students, monitoring students performing a task, and explaining the reason for the actions in the task. Unfortunately, a human instructor's availability may be limited. One alternative is to provide a software program (or *agent*) to serve as an instructor.

Before instructing a student, the agent needs to acquire the knowledge of domain tasks necessary for teaching. One method used by the AI community is tutorial instruction aided by machine learning techniques (Tecuci and Hieb 1996, Huffman and Laird 1995, Redmond 1992). The most basic type of knowledge is being able to demonstrate (or perform) a task. For our purposes, a domain task (or *procedure*) is a sequence of steps, where each step represents an action to perform. In fact, almost all systems that learn procedural knowledge only attempt to achieve this goal (Pearson 1995, Huffman and Laird 1995, Shen 1994, Benson 1995). In contrast, our goal is more challenging because teaching requires using knowledge in multiple ways. An agent that teaches also needs to be able to explain a task to students and to monitor human students performing a task.

Part of teaching a task is explaining it to students. One type of explanation is stating what the system is doing and why. Many systems have this capability. However, another type of explanation is explaining the causal relationships between steps. While many systems contain enough knowledge to produce some form of explanation, their explanations would seem inadequate to humans. Many explanations would be either too complicated (Shen 1994), contain incorrect knowledge (Pearson 1995) or the have unacceptable content (Benson 1995). One class of system that has problems explaining causal relationships are systems that only learn how to react in specific situations (Huffman and Laird 1995, Pearson 95). In short, the system should not only be able to explain what it is doing but also be able to describe the causal relationships between its actions.

Another part of teaching is monitoring students as they perform a task. Human students may legitimately perform a task's steps in a different order than demonstrated by the instructor. An agent that merely performs a task only requires knowing enough about the causal relationships between the steps for the task to succeed. However, changing the order of a procedure's steps can expose errors and incompleteness in the agent's knowledge of causal relationships. Thus, a system that teaches procedural tasks should know the necessary causal relationships rather than those that are just sufficient to perform a task.

Acquiring knowledge of domain tasks for a software agent is often very difficult. Because it is so difficult, it has been called the *knowledge acquisition bottleneck.* One reason for the difficulty is that human domain experts, who may not be programmers, might have to enter tasks in an awkward and unnatural manner. Some reasons for awkward and unnatural data entry are the use of arcane specification languages or the requirement for many training examples. Because of awkward and unnatural data entry, instruction may not be focused on the relevant problem features. The lack of focus and the agent's limited knowledge may also place a greater burden on the domain expert. Even after data are entered, the domain expert may be unable to get a reasonable bound on the uncertainty in the agent's knowledge. Thus, some factors that contribute to the knowledge acquisition bottleneck include unnatural and unfocused instruction as well as the inability of an agent to communicate with a human about problems in its knowledge.

This work addresses factors that contribute to the knowledge acquisition bottleneck by investigating how a pedagogical software agent can acquire knowledge of domain tasks with the aid of a simulation and the guidance of a human instructor, who may not be a programmer. The instructor demonstrates a procedure in a natural manner by directly manipulating a simulated environment. Next, to better understand the demonstration, the agent uses the simulation that controls the simulated environment to perform experiments. The experiments help the agent understand the causal relationships between the procedure's steps and help reduce the uncertainty in the agent's knowledge. After the agent has finished experimenting, it interacts with the instructor to further refine its knowledge.

This paper describes the *Diligent* learning component of *Steve* (Soar[1] Training Expert for Virtual Environments) (Rickel and Johnson 1997a, Johnson 1995, Johnson et al. 1997, Rickel and Johnson 1997b). The learning component is diligent because a few examples (demonstrations) are examined carefully in order to maximize its use of limited knowledge.

## Sources of Knowledge

The sources of knowledge available to the agent include a helpful, human instructor, a simulated environment, and some limited, initial domain knowledge.

[1] Steve is implemented using Soar (Laird, Newell and Rosenbloom 1987).

We want the human instructor to tutor the agent in a natural manner. The instruction is natural because instruction is as close to as possible to performing the task and because instruction resembles the interaction with a human student. The instruction needs to be natural because, even though the instructor is a domain expert, he may not be an experienced programmer. As a domain expert, the instructor cannot only demonstrate tasks but can also provide focus and answer questions. However, the instructor will have limited patience so his time must be utilized effectively.

The simulated environment supports instruction by providing a model of the domain. The environment allows the instructor to demonstrate procedures for the agent. The environment also allows the agent to test its understanding and reduce its uncertainty by performing experiments. Additionally, the environment allows the agent to reset the environment's state to a given configuration. The ability to reset the environment is appropriate in pedagogical systems where students start problems in specified initial states. The ability to reset the environment also allows the agent to perform multiple experiments from the same initial state. Although the environment can be reset, the environment does not allow the agent or the instructor to make arbitrary changes to its state.

If the agent has a limited ability to change the state, then why not make it part of the simulation controlling the environment? The agent's manipulation of the simulation state should always result in well-formed and feasible states. If the agent is free to make arbitrary state changes, inconsistent or impossible states might result. To avoid this problem, we restrict the agent to either setting the state to some previous simulation state or performing actions that correspond to actions that the user can perform via the user interface. A major drawback of this approach is that a portable agent has little domain dependent initial knowledge.

The agent's initial domain knowledge allows it to understand and manipulate the simulated environment. The agent's initial domain knowledge contains the ability for the agent to recognize actions performed in the environment and the ability to perform actions in the environment. These actions are the building blocks from which procedures are built.

It is very unusual for a system to utilize a helpful instructor, a simulated environment that supports experiments along with limited domain knowledge. Tecuci and Hieb (Tecuci and Hieb 1996) describe such a system, but it learns a very different type of knowledge than our agent and requires much more interaction with the instructor.

## How the Agent Learns

The agent gets its initial knowledge about a procedure from an instructor's demonstration. The instructor demonstrates a procedure by directly manipulating the

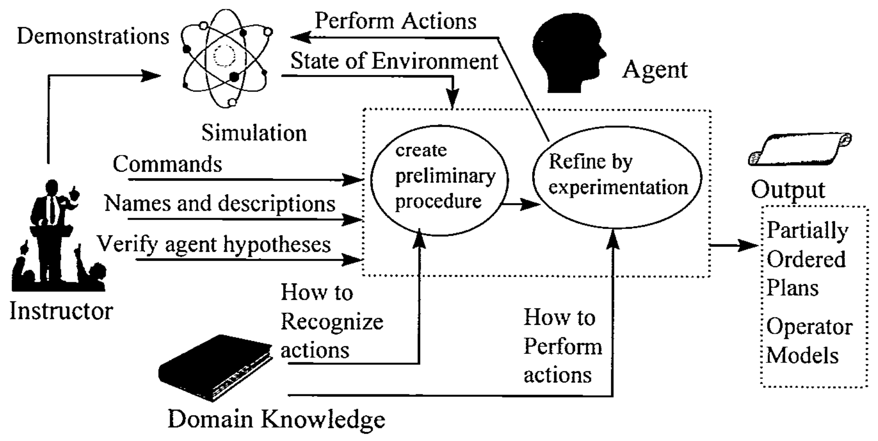


Figure 1: Agent Input/Output

simulated environment. In our current system, direct manipulation is done by using a mouse in a 3-D virtual world. Acquiring procedures in this manner is called *programming by demonstration* (Cypher et al. 1993). Because demonstrations are interactive, it would be too burdensome on the instructor to require more than a few demonstrations. The limited number of demonstrations requires that the agent get maximum use out of each.

The agent gets maximum use out of a demonstration by experimenting with it, biasing learning towards changes in the environment that occur during the demonstration, and asking the instructor questions. The agent's questions ask for a specific type of knowledge (e.g. name) or ask the instructor to verify an hypothesis. The agent's experiments, learning bias and questions are good sources of knowledge because they reduce the instructor's burden and focus learning.

We believe that the agent should focus its experiments on understanding demonstrations rather than solving practice problems. A *demonstration* has an initial state and a sequence of actions that lead to its final state. In contrast, a *practice problem* has an initial state and a final, goal state but does not specify the sequence of actions necessary to change the environment to the goal state. Practice problems are useful for learning general knowledge in systems that do not use demonstrations (Gil 1992). Practice problems are also useful in system's have a lot of domain knowledge. However, practice problems are hard to solve with little domain knowledge and may not be focused on understanding the dependencies between the actions in a demonstration. This is seen in OBSERVER, which requires many demonstrations and practice problems (Wang 96). Requiring many demonstrations and practice problems also places a much greater burden on the instructor. The burden on the instructor is greater because demonstrations are tedious and time consuming and because selecting practice problems requires care if the agent is to solve them. An alternative method would be to reduce the number of required demonstrations by focusing experiments on better understanding each demonstration. In fact, studies of human learning seem to support this approach. Human students who are good problem solvers tend to study a few examples in detail rather than take a shallow look at many examples (Chi et al. 1989).

Experiments can be focused on a demonstration by perturbing a trace of the demonstration, executing it and observing what happens. Some earlier systems perturb a demonstration by changing the state of the environment (Porter and Kubler 1986). However, the simulation that controls the environment is an external program that does not allow the agent to make arbitrary state changes. The agent overcomes this by perturbing the demonstration's sequence of actions rather the environment's state. This has the additional benefit of focusing on the actions inside the demonstration. After all, what the student learns are the procedure's actions.

Experimentation is complemented by biasing learning towards environment state changes during a task. What experiments reveal are the preconditions of actions which, in turn, result in causal dependencies between steps. The agent assumes that the order of actions in a demonstration has significance and that the effects of earlier actions are likely to be preconditions for later actions. The agent uses

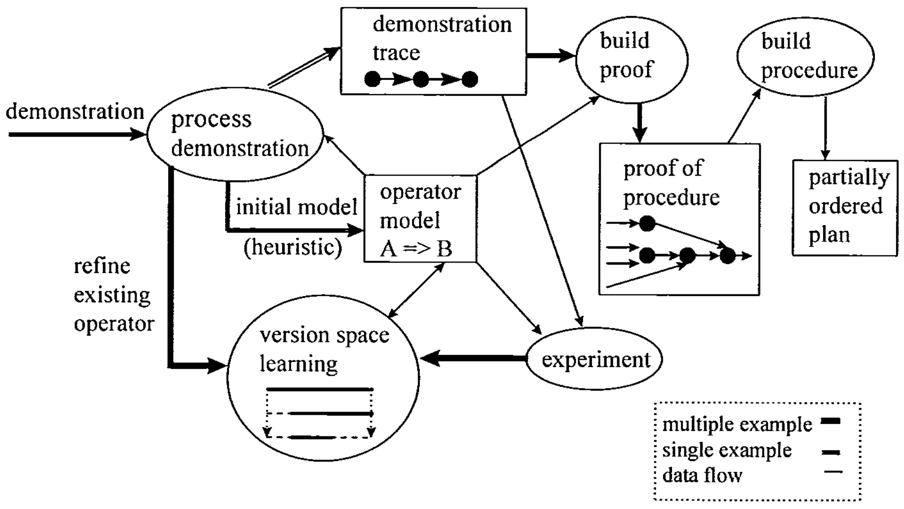


Figure 2: How the Agent Learns Procedures

this assumption to create heuristic preconditions for the actions it observes. The heuristic preconditions provide the agent with some initial task competence, while maximizing the use of the limited knowledge in a single demonstration.

Because the preconditions are heuristic, the agent is uncertain about them. The uncertainty in the heuristic preconditions is represented by using a version space to bound the set of candidate preconditions. A *version space* identifies the most specific and most general candidate preconditions (Mitchell 1978). By bounding its uncertainty, the agent can focus learning towards reducing the uncertainty. Additionally, the explicit representation of the agent's uncertainty supports communication with the instructor.

After the agent has finished experimenting, it can ask the instructor questions to verify its task knowledge. Mistakes in the preconditions for actions can result from the agent's lack of domain knowledge and limited amount of input. These cause the agent to rely on induction and heuristics. However, the burden placed on the instructor can be reduced by experiments which correct mistakes in the task knowledge. The instructor's burden has been reduced because fewer mistakes allow the instructor to verify rather than explicitly generate task knowledge.

In summary, maximizing the use of limited knowledge is supported by biasing experiments and learning towards demonstrations and maintaining a precondition representation that supports communication with the instructor.

## System Overview

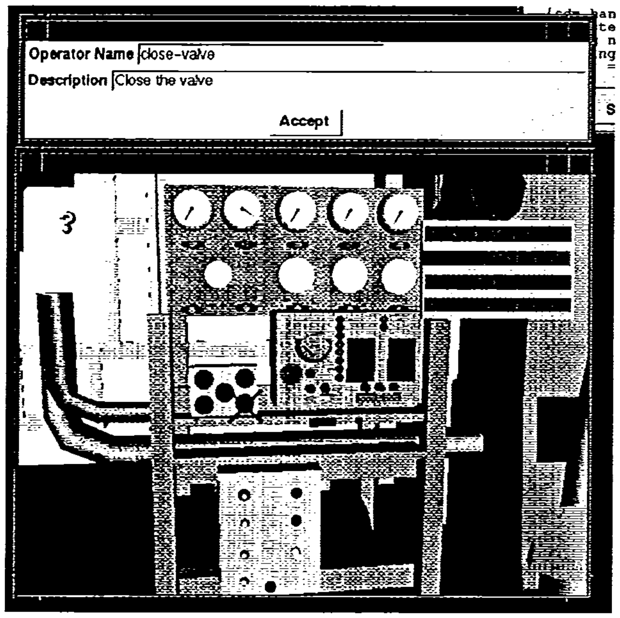
### Input and Output

Figure 1 shows the knowledge used by Diligent to learn procedures. Initially, the agent can recognize and perform actions in the environment. The agent uses the ability to recognize actions for understanding demonstrations and the ability to perform actions for experimenting. Initially, the agent knows nothing about the preconditions and effects of actions

The instructor supplies the framework for the knowledge which the agent is initially lacking. The instructor can control the interaction with the agent by giving it commands (e.g. learn a new procedure). The instructor can also demonstrate procedures in the simulated environment. The instructor augments a demonstration by supplying names and English descriptions. In our current system, descriptions include the purpose of procedures and actions as well as messages associated with causal relationships. The English descriptions are used to communicate with human students. The instructor also provides the agent with feedback by verifying or modifying hypotheses proposed by the agent.

The environment simulation[2] provides a detailed model

[2] Our current simulation environment is authored with Rides (Munro et al. 1993)


Operator Name close-valve

Description Close the valve

figure 3: Learning a new operator

that can be manipulated and observed. The instructor demonstrates procedures by manipulating the environment. The simulation then notifies the agent about the actions performed and changes to the environment's state. The simulation also provides the agent with the ability to perform actions and observe their result.

When a procedure has been learned, it is output in the form of a partially ordered plan. A *partially ordered plan* contains a set of steps and sets of causal links and ordering constraints. Each step corresponds to an action performed in the environment. Causal links record how the effects of one step are preconditions for subsequent steps (McAllester and Rosenblitt 1991). Ordering constraints indicate the required order of execution between two steps. Ordering constraints prevent latter steps from clobbering the preconditions of earlier steps.

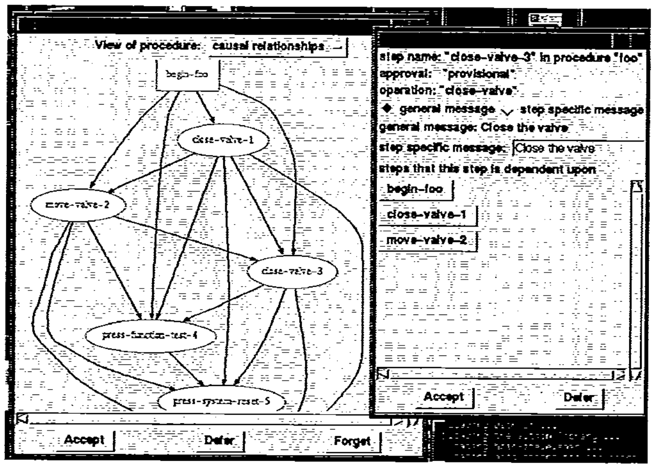
The preconditions and effects of steps are derived using operator models. *Operator models* map actions performed in the environment to their preconditions and effects. The agent also outputs a set of operator models, which can be reused in other procedures.

## Learning Procedures

Figure 2 shows how the agent converts a demonstration into a partially ordered plan.

Initially, the instructor provides the agent with a demonstration. The agent creates a trace of the demonstration, which is the demonstration's sequence of steps. The trace does not indicate the causal relationships between the steps.

During a demonstration, the agent creates and refines operator models for each of the demonstration's steps. Each step corresponds to some action in the environment. If the agent has not seen the action before, it creates a new


step name: "close-valve-3" in procedure "foo"

approval: "provisional"

operation: "close-valve"

◆ general message ✓ step specific message

general message: Close the valve

step specific message: Close the valve

steps that this step is dependent upon

begin-foo

close-valve-1

move-valve-2

figure 4: Plan represented as a graph[3]

operator. Otherwise, an existing operator is refined. Each operator maps a set of preconditions to a set of effects. Preconditions are stored in a version space representation (Mitchell 1978). The version space bounds the "real" precondition between a most specific and a most general candidate precondition. Because version space learning may converge slowly, the agent uses a heuristic precondition in between the most general and specific candidate preconditions. The heuristics favor state changes during the procedure. It is the heuristic precondition that is used to generate plans.

Figure 3 shows the agent learning how to close a valve. The valve is closed by selecting the circular object in the middle of the scene that appears to be sitting on an X shaped object. The first time a valve is closed the agent does not recognize the operation. The agent asks the instructor for an operator name and a description that can be given to students. As shown in figure 2, the agent also creates an initial model for the close valve operator.

After a procedure has demonstrated, the agent needs a representation of the procedure that can be understood by the instructor. Figure 4 shows a procedure represented as a graph. Each of the procedure's steps is represented by node. By selecting a node, the instructor can access menus that allow examination and modification of the agent's understanding of a step and its relation to other steps.

Once a procedure has been demonstrated and operator models are defined, figure 2 shows that the instructor can tell the agent to generate a proof for the procedure. The *proof* records the causal relationships between the preconditions and effects of the procedure's steps. A step's preconditions are the heuristic preconditions in the operator models. Given a proof, it is trivial to transform the proof into a partially ordered plan.

[3] The graph was produced with TCL/TK (Ousterhout 1994) and the tkdot portion of the Graph Visualization tools from AT&T Laboratories and Bell Laboratories (Lucent Technology) (Krishnamurthy 1995).

However, a plan may have mistakes in its causal relationships if the operator models are not mature enough. The agent can reduce these mistakes by experimenting. The agent performs multiple experiments where the demonstration's steps are executed in a slightly perturbed order. The change in order exposes the interdependencies between the steps. Once the operator models have been sufficiently refined, the instructor can ask the agent to generate a new version of the plan.

## Status

A prototype learning component has been incorporated into the Steve agent and tested on some simple, non-hierarchical procedures.

One area for improvement is expanding the types of interaction between the agent and the instructor. This can done by allowing an instructor to demonstrate a procedure multiple times, build hierarchical procedures and test how well the agent has learned a procedure. Allowing multiple demonstrations does not contradict our goal of reducing the number of required demonstrations. Our focus has been on learning as much as possible from each demonstration so that many demonstrations become redundant. This, however, does not mean that an instructor should not be able to provide extra demonstrations if he desires to do so.

The ability to demonstrate a procedure multiple times supports the acquisition of more complicated procedures. Because procedures are more complicated, instructors may demonstrate small pieces of a procedure in different demonstrations. For example, the instructor could demonstrate how to handle different variations in the procedure's initial state. Multiple demonstrations also allows procedures to handle different classes of initial states where each class results in very different sequences of actions being performed. Multiple demonstrations would also allow an instructor to demonstrate the independence of groups actions by changing the order in which they are demonstrated.

Providing the instructor the ability to compose hierarchical procedures out of lower level procedures also supports creating more complex procedures. Hierarchical procedures reduce the number of steps that an instructor must demonstrate by reusing existing procedures. Using a hierarchy also allows complicated procedures to be split into logical sets of lower level procedures.

Providing the ability to compose hierarchical procedures has implications on the types of experiments performed. The agent needs to limit the number of experiments performed on hierarchical procedures, which can be large and incorporate lower level procedures. Some lower level procedures may have already been tested. One approach is to perform experiments in a bottom up manner starting with the lowest level procedures and working up the hierarchy. Each experiment would only look at the current level and ignore lower levels unless an error is observed.

Providing the instructor with the ability to test how well the agent has learned a procedure will provide more confidence in the agent's knowledge. Validating the agent's knowledge through testing augments rather than replaces an instructor's ability to explicitly verify the agent's knowledge. This would allow the instructor to catch mistakes in the agent's knowledge and to watch the agent perform the actual problems given to human students. If a mistake is detected, learning should not depend as heavily on heuristics as during initial demonstrations because the agent's knowledge should have been verified by the instructor and be fairly stable.

In summary, a goal of this work is to make the agent's interaction with an instructor model more closely a human student's interaction with an instructor.

## Related Work

Many *Programming By Demonstration* (PBD) systems learn how to perform simple procedures. Most of the basic techniques needed by Diligent have been discussed in the literature. The best reference is Cypher's book (Cypher et al. 1993). Previous PBD systems learn to correctly perform a procedure by executing it steps in some order rather than to understand the causal relationships between the steps. Any agent that teaches needs to understand causal relationships in order to monitor students performing procedures and to explain the causal dependencies between steps.

Many previous systems that learn from tutorial instruction require detailed domain theories. Two such systems are ODYSSEUS (Wilkins 1990) and CELIA (Redmond 1992). This reduces the portability between domains and requires a human who has the expertise to construct the domain theory. Diligent does not need a detailed domain model because it exploits a simulation and a helpful, human instructor. The DISCIPLE (Tecuci and Hieb 1996) systems require simpler domain models, but DISCIPLE systems ask instructors more questions than Diligent because DISCIPLE systems do not use a simulation to perform autonomous experiments.

Some systems learn a representation for preconditions that does not support articulating explanations that a human would find reasonable. TRAIL (Benson 1995) and LIVE (Shen 1994) learn preconditions that are in disjunctive normal form. The preconditions are formed so that they cover all examples of when the operator should be applied. There is no effort to learn preconditions of which a human would approve. Instructo-Soar (Huffman and Laird 1995) and IMPROV (Pearson 1996) learn rules to reactively perform procedures. Reactive systems learn to recognize a portion of the current state rather than the preconditions that cause dependencies between states.

Some systems require many practice problems or demonstrations. EXPO (Gil 1992) refines an incomplete domain theory using practice problems, but EXPO is unable to correct errors in its knowledge and does not use demonstrations. OBSERVER (Wang 96) learns operator models using many demonstrations and practice problems.

Diligent and OBSERVER use very similar precondition learning algorithms. OBSERVER, however, does not perform experiments based on demonstrations. Instead, OBSERVER tries to learn general knowledge by solving practice problems. Instead of solving general practice problems, Diligent focuses its experiments on understanding the procedures that it will teach.

Diligent performs experiments by observing how the actions in a demonstration influence each other. Even though Diligent makes no attempt to model human cognition, this approach was motivated by studies of human learning (Chi et al. 1989). Another system that learns by examining a demonstration in detail is PET (Porter and Kubler 1986). PET requires the ability to make small changes to the state of the environment. In contrast, Diligent does not assume the ability to make arbitrary changes to the environment's state.

## Conclusion

This paper describes an approach to authoring that attempts to provide a natural interaction between a human instructor and a software agent. The interaction is natural because it is as close as possible to performing the procedure and because it resembles the interaction between an instructor and a human student. Our goal is to avoid problems that result when the interaction is awkward and unnatural. The instructor answers questions, provides English descriptions, and demonstrates procedures by directly manipulating a graphical environment. The agent uses demonstrations and interaction with the instructor to focus learning. The agent overcomes its limited initial domain knowledge by experimenting. The experiments turn a single demonstration into multiple examples and reveal the causal relationships between the steps in a procedure.

## Acknowledgments

This project is funded by the Office of Naval Research, grants N00014-95-C-0179 and N00014-97-1-0598. We gratefully acknowledge the efforts of our colleagues at Lockheed Martin under the direction of Randy Stiles that developed our virtual reality software and our colleagues at the Behavioral Technology Laboratory under the direction of Allen Munro that developed the Rides software. We also wish to thank Erin Shaw at the USC Information Sciences Institute. Richard Angros would also like to thank the generous support provided by the Hughes Aircraft Company.

## References

Benson, S. 1995. Inductive learning of reactive action models. In Machine Learning: Proceedings of the Twelfth International Conference, 47-54. San Francisco, Calif.; Morgan Kaufmann.

Cypher, A. et al., eds. 1993. *Watch What I Do: Programming by Demonstration.* Cambridge, Mass.: The MIT Press.

Chi, M. T. H.; Bassok, M.; Lewis, M. W.; Reimann, P.; and Glaser, R. 1989. Self-explanations: How students study and use examples to solve problems. *Cognitive Science*, 13:145-182.

Gil, Y. 1992. Acquiring Domain Knowledge for Planning by Experimentation. Ph.D. diss., School of Computer Science, Carnegie Mellon Univ.

Huffman, S. B.; and Laird, J. E. 1995. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271-324.

Johnson, W. L. 1995. Pedagogical agents in virtual learning environments. In Proceedings of the International Conference on Computers in Education, 41-48; AACE.

Johnson, W. L.; Rickel, J.; Stiles, R.; and Munro, A. 1997. Integrating pedagogical agents into virtual environments. *Presence*, forthcoming.

Krishnamurthy, B. ed. 1995. *Practical Reusable UNIX Software*. New York, NY.; John Wiley & Sons. http://portal.research.bell-labs.com/orgs/ssr/book/reuse

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

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, Calif.; AAAI Press.

Mitchell, T. M. 1978. Version Spaces: An Approach to Concept Learning. Ph.D. diss., Dept. of Computer Science, Stanford Univ.

Munro, A. ; Johnson, M. C.; Surmon, D. S.; and Wogulis, J. L. 1993. Attribute-centered simulation authoring for instruction. In Proceedings of the AI-ED 93 World Conference of Artificial Intelligence in Education, 82-89. Edinburgh, Scotland.

Ousterhout, J. K. 1994. *Tcl and the Tk Toolkit.* Reading, Mass.; Addison-Wesley.

Pearson, D. J. 1995. Active learning in correcting domain theories: Help or hindrance. In AAAI Symposium on Active Learning.

Porter, B. W.; and Kubler, D.F. 1986. Experimental goal regression: A method for learning problem-solving heuristics. *Machine Learning*, 1:249-286.

Redmond, M. A. 1992. Learning by observing and understanding expert problem solving. Ph.D. diss., Georgia Institute of Technology.

Rickel, J.; and Johnson ,W. L. 1997a. Integrating pedagogical capabilities in a virtual environment agent. In Proceedings of the First International Conference on Autonomous Agents.; ACM Press.

Rickel, J.; and Johnson ,W. L. 1997b. Intelligent tutoring in virtual reality: a preliminary report. In Proceedings of the Eighth World Conference on AI in Education.; IOS Press.

Shen ,W. 1994. *Autonomous Learning From The Environment*. New York, NY: W. H. Freeman.

Tecuci, G.; and Hieb, M. R. 1996. Teaching intelligent agents: The disciple approach. *International Journal of Human-Computer Interaction*. 8(3):259-285.

Wang , X. 1996. Learning Planning Operators by Observation and Practice. Ph.D. diss., School of Computer Science, Carnegie Mellon Univ.

Wilkins, D. C. 1990. Knowledge base refinement as improving an incorrect and incomplete domain theory. In *Machine Learning An Artificial Intelligence Approach, volume III*, 493-513. San Mateo, Calif.: Morgan Kaufmann.