Demonstrational Techniques for Instructible User Interface Agents

Henry Lieberman

Media Laboratory
Massachusetts Institute of Technology
Cambridge, Mass. USA
lieber@media.mit.edu

The AAAI Sprint Symposium on Software Agents defines the term "*software agent*" as

> *"A sensor/effector system that operates within a real-world software environment... observes features of this external environment, and can both alter the state of the environment directly and communicate with other agents."*

But what is this "environment"? For the majority of applications used by most mainstream computer users, such as spreadsheets, text editors, mail readers, graphical editors, CAD programs, etc., a direct-manipulation graphical interface constitutes the "environment" with which a software agent must interact in order to be ultimately useful to the user. The program might have internal representations of objects of interest to the user [as frames, propositions, neural nets, or specialized data structures], and perform inference on these representations, but the sensors must detect actions taken by the user in the interface, and the effectors must deliver effects that are ultimately visible or audible in the user interface.

Too much past work in AI has made the implicit assumption that decisions about what learning algorithms to choose, how to generalize from specific situations, and what actions to take, can be completely abstracted away from user interface issues. In fact, our experience indicates that it is very important to choose generalization and learning strategies that are clear in the user interface, and for the agent to take interface issues into account when deciding what actions to take.

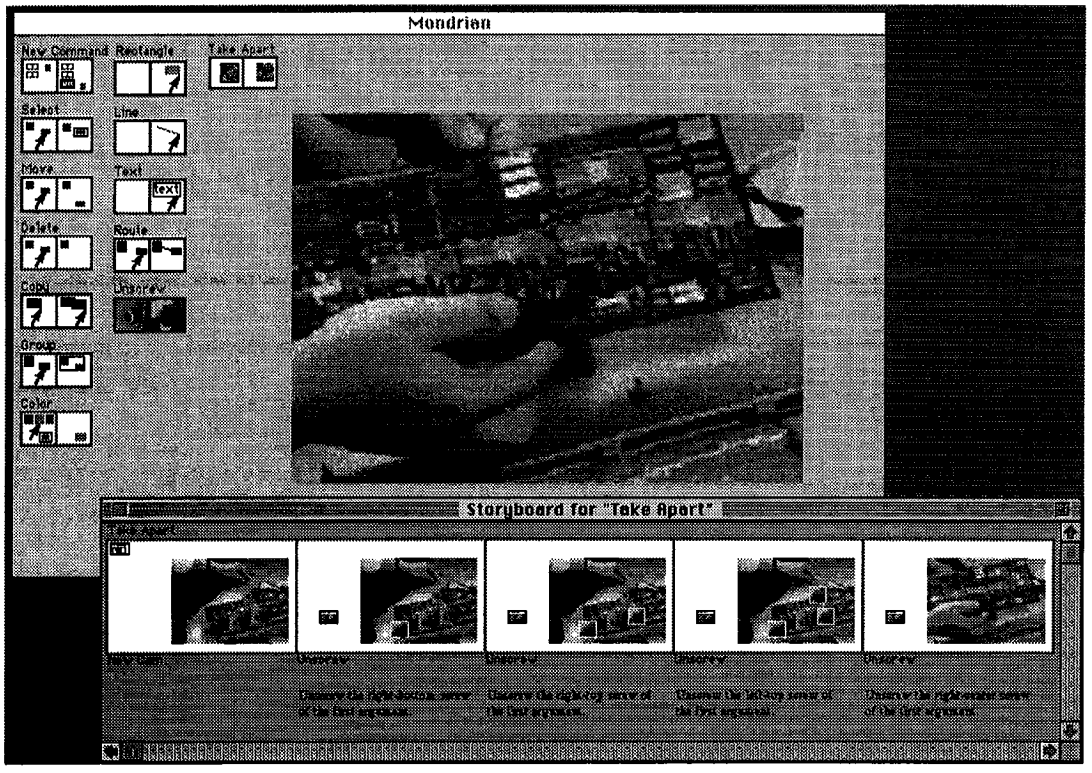
In particular, a prime criterion for agents that operate in the user interface is that they be *instructible*, that the user be able to interact with the agent in order to teach it new behavior or influence some aspects of its present behavior. We conjecture that a lot of the apprehension that future agent scenarios generate among some people is largely due to their fear of losing "control" to an agent whose operation is invisible and cannot be influenced. Convincing people that agents are always willing to learn new behavior and communicate what they know will go a long way towards allaying those fears.

An effective technique for instructing agents is to have the user *demonstrate examples* in the user interface, and for the agent to record the user's actions and generalize them to influence future behavior. This approach is called *programming by demonstration*, or *programming by example*. The metaphor for communicating with an instructible agent should be a teacher-student, or master-apprentice relationship. The user plays the role of a teacher, and the challenge is to make the computer effective in its role as a student or junior apprentice. Demonstrating examples is effective because the user is presumed to be already familiar with the language of the application's graphical interface. Presentation of examples typically makes more sense to the user than languages of abstract assertions, weightings and thresholds of neural networks, and [particularly where the problem domain is graphical], sometimes even natural language instruction.

I will illustrate the role of demonstrational techniques with my work on Mondrian, an instructible agent for a MacDraw-style graphical editor. The learning method is an incremental form of explanation-based generalization. This work will be demonstrated live [equipment permitting]. More generally, I will also discuss issues in building instructible agents, and the role of user interface issues in machine learning.

• *How are software agents different from standard computer programs?*

Demonstrational interfaces try to present to the user an interface that retains all the "look and feel" of an application's direct manipulation interface. Importantly, the user can use the application in the normal manner, without taking account of the agent. However, if the user



The illustration shows the interface to Mondrian, an object-oriented graphical editor augmented with an instructible user interface agent that learns procedures from examples. Here, the system is being shown how to disassemble a circuit board. The agent records user actions, and produces a program capable of performing an analogous procedure on a different circuit board, using an incremental form of explanation-based generalization.

The system represents operations with *domino* icons that present before-and-after pictures of each command. The lower portion of the picture shows a *storyboard* that displays recorded images of the history of operations and automatically-generated natural language descriptions of each step. The system also uses speech output for feedback, and accepts speech input to advise the generalization process.

can direct the system to look "over the user's shoulder" and learn new behavior from watching examples. In some agents, the "record" mode is on all the time and the agent proposes suggestions automatically. There are a few operations that allow the user to explicitly communicate with the agent and the agent provides visual feedback about what it has learned. To the user, the application operates normally, but the interface can be extended with new behavior or repeated patterns of behavior can be automated.

• *What are appropriate software agent architectures?*

There is a range of choices that can be made for architectures for demonstrational agents. One important choice is whether the agent records and generalizes actions, or whether it records successive states and looks for similarities and differences between states. Mondrian adopts the former approach. A variety of machine learning strategies described in the literature can be appropriate, but attention needs to be paid to how the learning is perceived in the interface. There is a tradeoff between making the learning algorithm "smarter" and maintaining its understandability to the user. Mondrian uses a form of incremental explanation-based generalization which notices a set of relations deemed "significant" in the input, and tracks depedencies of actions. Voice input is used to disambiguate potentially ambiguous generalizations. An architecture for introducing domain-dependent generalizations is included.

• *What are appropriate languages for inter-agent communication?*
• *Is explicit representation of capabilities and states of other agents necessary for the success of software agents?*
• *What are appropriate "social laws" for societies of agents?*

Unfortunately, there are not yet any multiple-agent demonstrational interfaces. I can at most speculate on how multiple agent societies might be integrated into a demonstrational model. However, even the demonstrational interfaces that exist so far are "multi-agent" in the sense that the user acts as an agent as well as the software. Thus many aspects of inter-agent communicate are reflected in the user interface. A language for communicating between two software agents can then take on some of the characteristics of user-agent communication. This suggests that inter-agent communication should be done by example as well.

• *What are fundamental learning problems for software agents?*

Generalizing from user actions is the major problem for a demonstrational software agent. What relations are noticed by the agent? How does a user communicate his or her desires to an agent? How does the agent give feedback to the user about what it has learned, so that the user can verify that his or her instructions have been correctly understood? How does the user make changes to the behavior of an already existing agent? How can the user learn what existing agents are capable of, or choose between alternative agents?

• *How do the notions of planning and execution in software domains compare with the classical notions?*

"Planning" corresponds to what the agent does when it initially records a demonstration from recording the user's actions. The recorded procedure constitutes a plan for dealing with future examples. "Execution" occurs when applying this plan to a new example. This amounts to making an analogy between the example initially presented and the new example. There exists a need for "execution monitoring", at least to be able to debug the execution if the result is not as desired.

• *What mechanisms are necessary to ensure long-term agent survival?*

Probably the main criterion for long-term agent survival is if the human user is happy with how the agent behaves in the interface! Some criteria that will begin to emerge after agents have had substantial lifetimes are how the agent can manage large sets of examples as they grow over time; how the agent can control level of detail in communication with the user; and how long-term global changes in behavior can be managed.

Acknowledgments

Major support for this work comes from research grants from Alenia Corp., Apple Computer, ARPA/JNIDS and the National Science Foundation. The research was also sponsored in part by grants from Digital Equipment Corp., HP, and NYNEX.

References

[Cypher, ed. 93] Cypher, Allen, ed. Watch What I Do: Programming by Demonstration, MIT Press, 1993.

[Lieberman 92] Lieberman, Henry, Capturing Design Expertise by Example, in East-West Conference on Human-Computer Interaction, St. Petersburg, Russia, August 1992.

[Lieberman 93a] Lieberman, Henry, Mondrian: A Teachable Graphical Editor, in [2].