Instructible agents: Software that just keeps getting better

by H. Lieberman
D. Maulsby

Agent software is a topic of growing interest to users and developers in the computer industry. Already, agents and wizards help users automate tasks such as editing and searching for information. But just as we expect human assistants to learn as we work with them, we will also come to expect our computer agents to learn from us. This paper explores the idea of an instructible agent that can learn both from examples and from advice. To understand design issues and languages for human-agent communication, we first describe an experiment that simulates the behavior of such an agent. Then we describe some implemented and ongoing instructible agent projects in text and graphic editing, World Wide Web browsing, and virtual reality. Finally, we analyze the trade-offs involved in agent software and argue that instructible agents represent a "sweet spot" in the trade-off between convenience and control.

The idea of intelligent agent software has recently captured the popular imagination. From Apple Computer's Knowledge Navigator** video to Microsoft Corporation's Bob**, to the recent explosion of personalized search services on the World Wide Web, the idea of intelligent software that performs the role of a human assistant is being explored in a wide range of applications.

Indeed, there is a growing realization that such software is not only desirable, but necessary. The complexity of computer interfaces has accelerated recently. The prevalent view has been that a computer interface is a set of tools, each tool performing one function represented by an icon or menu selection. If we continue to add to the set of tools, we will simply run out of screen space, not to mention user tolerance. We must move from a view of the computer as a crowded toolbox to a view of the computer as an agency—one made up of a group of agents, each with its own capabilities and expertise, yet capable of working together.

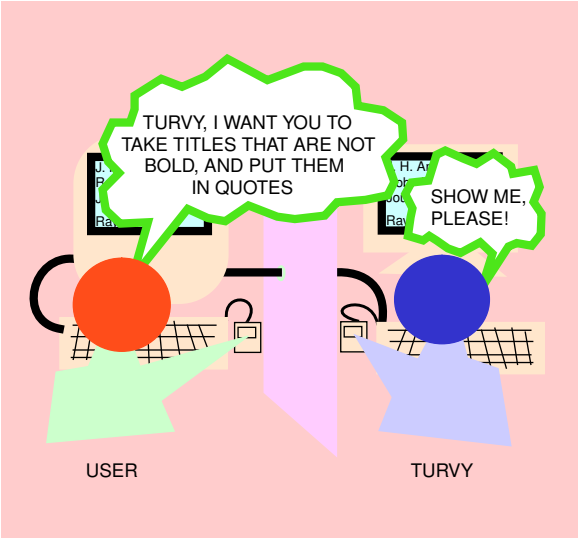
Conspicuously rare in today's agent software is a capability considered essential in human agents: they *learn*. To be truly helpful, an assistant must learn over time, through interacting with the user and its environment—otherwise, it will only repeat its mistakes. Learning is essential for a software agent; no software developer can anticipate the needs of all users. Moreover, each person's needs change from day to day, and knowledge learned from experience can lead to a better understanding of how to solve problems in the future.

Most current agent software and most of the proposed scenarios for future agent software have an agent acting to meet a user's goals, but ignore the problem of how the agent comes to learn those goals or actions. We are interested not only in how to build a smart agent from scratch, but also in how an agent can become just a little bit smarter every time it interacts with the user. The question is: "What is the best way for an agent to learn?"

We have an answer: We think agents should learn *by example*. The reason for this is simple. If we are striving to make a software agent act like a human assis-

©Copyright 1996 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

Figure 1 The Turvy experimental setup



tant, we should make the communication between user and agent as human-like as we can—not necessarily personal, but intuitive and flexible. So the question of how an agent might learn becomes: "What are the best ways for people to teach and agents to learn?"

Agents might learn by being given explicit instructions, but giving instructions is much like conventional computer programming, and we know that programming is difficult for many people. However, people are extraordinarily good at learning from observing and understanding examples, especially examples in the context of real work. So we can take advantage of that ability and design user-agent interaction around the activity of demonstrating, recording, and generalizing examples. Do you want your agent to solve a new kind of problem? Show it how.

If an agent learns by watching the user work on concrete examples, by listening to the user's comments, and by asking questions, we call it an *instructible* agent. Work on instructible agents is a synthesis of many disparate fields of artificial intelligence and human-computer interaction: machine learning, knowledge acquisition, program synthesis, user modeling, direct-manipulation interfaces, and visual interface design. Specific techniques such as constraints, natural language processing, or statistical methods might come into play in different kinds of instructible interfaces.

Much of the work on instructible agents has appeared under an interdisciplinary topic called *programming by example* or *programming by demonstration*.[1] The use of the word "programming" here may be misleading; while most of these systems do learn procedures, the process by which they learn and how it appears to the user is typically very different from what we currently think of as computer programming. It is more like the interaction between a teacher and a student, or a coach and a trainee—where the user is the coach and the computer is the trainee.

Instructible *interface agents* are embedded in direct-manipulation interfaces like those in common use today: text editors, graphical editors, spreadsheets, and the like. In the near term, instructible agents will work within the framework of familiar applications. A user can choose to ignore the agent and operate the application as usual. However, the agent can, either on its own or at the user's behest, make suggestions or automate actions.

The central problem for such agents is to infer what the user wants from observing the user's actions in the interface. There are two approaches, both explored in this paper.

The first approach is to design an interface in which actions express their intent. For example, the Emacs text editor provides many navigation commands, such as "forward word" and "forward line," that express movement and selection relative to text structures. But interfaces like Emacs based on typed commands are not for everyone. In several of the learning systems we describe, the user can point to an object and say, "This is an example." The user thereby directs the system to generalize operations involving the example object, so that an analogous procedure can be performed on another similar object in the future. The user may annotate the example, specifying the features to be adopted in the generalization.

The other, more ambitious approach is to let the system decide for itself which aspects of the demonstration are relevant. This reduces the tedium of explaining examples. As with a human agent, the computer is given the opportunity to make mistakes. The agent communicates with the user and learns by correcting its mistakes.

These two approaches—direct instruction and inductive inference—are complementary and have been used together in some systems, such as Cima.[2]

Instruction has also been combined with deductive inference (i.e., plan recognition) in the Instructo-Soar system,[3] which enables the user to supply missing knowledge about planning operators and objects whenever Soar fails to explain an action.

Instructible agents: Issues, experiments, and systems

In this paper, we explore a variety of scenarios and techniques for building instructible agents. We show examples from learning agents we have built for interactive applications such as graphical editors and text editors. We show a variety of interaction styles, from instructing through direct-manipulation menu selection and mouse clicks to natural language input, graphical annotation, and visual observation. We also discuss scenarios for future agents in virtual-reality worlds.

We begin by describing an experiment, Turvy, which simulates user-agent interaction in a text editor, with a human playing the role of the agent. The goal is to understand what kind of vocabulary could serve as a means of discourse between a user and an instructible agent. We use that experience to develop a set of general criteria for agent-user interaction. A system that implements some of this agent's learning ability, Cima, is then described.

We move on to Mondrian, an agent that learns new graphical procedures, both by demonstration and by graphical annotation of video. The agent describes what it learns through feedback in pictorial and natural language. It accepts the user's advice about generalization via speech input. Mondrian illustrates how a media-rich, multimodal interface can enhance the experience of teaching instructible agents. Mondrian represents an approach where the user understands that the purpose of the interface actions is explicitly to teach the agent.

Another approach, which relies less on explicit instruction and more on observing the user's behavior, is represented by Letizia,[4] an agent that assists a user in browsing the Web. Letizia also illustrates continuous, incremental learning interleaved with performance.

Finally, we discuss a scenario of work in progress for an instructible agent, ACT, that learns new behavior for animated characters in a virtual reality environment. Here both the user and the learner are represented by characters in the virtual reality world ("avatars") and observation, advice, and explicit instruction play an integrated role.

Figure 2 Sample data from the "citation headings" task

BEFORE EDITING

Philip E. Agre: *The dynamic structure of everyday life*: PhD thesis: MIT: 1988.

J. H. Andreae, B. A. MacDonald: *Robot programming for non-experts*: Journal IEEE SMC: July 1990.

Kurt van Lehn: *Mind bugs: the origins of procedural misconceptions*: MIT Press: 1990.

AFTER PUTTING AUTHOR AND DATE INTO HEADING

[Agre 88]
Philip E. Agre: *The dynamic structure of everyday life*: PhD thesis: MIT: 1988.

[Andreae 90]
J. H. Andreae, B. A. MacDonald: *Robot programming for non-experts*: Journal IEEE SMC: July 1990.

[van Lehn 90]
Kurt van Lehn: *Mind bugs: the origins of procedural misconceptions*: MIT Press: 1990.

How should the user and the agent communicate?

To investigate possibilities for how a user might communicate with an intelligent computer agent, a user study[5] was conducted, in which a researcher, hidden behind a screen, pretended to be an agent that learns word-processing tasks. Figure 1 illustrates the setup: The researcher playing the role of the agent, seated at right, is known to the user as "Turvy."

Example: Formatting a bibliography. Figure 2 illustrates the example task: making citation headings for bibliography entries, using the primary author's surname and the last two digits of the year of publica-

Figure 3 Dialog between Turvy and the user while formatting the bibliography

Pam: Take the last name, before the colon. [Copies "Agre," pastes it to a new paragraph, types brackets, sets style.] This is a list of publications. Now we'll add the last two digits of the date. [Types "88."] Do you want another demo?

Turvy interprets Pam's question to mean the example is complete. Turvy has formed an initial generalization and now offers to edit the second entry (author "Andreae").

Turvy: I can try if you want.
Pam: OK.
Turvy: [Picks "Macdonald."] Looking for word before first colon in paragraph, OK?
Pam: No—stop! You want the first author.
Turvy: Show me, please.
Pam: [Picks "Andreae."]
Turvy: You picked word before comma. You want word before first comma or colon?
Pam: Yes! The first comma or colon.
Turvy: [While doing actions] Copy, go to start of paragraph, OK?
Pam: OK.
Turvy: Type open bracket, paste, close bracket, change style to "citation," OK?
Pam: OK.

Now Turvy makes a mistake with the date. When Pam typed "88," Turvy assumed the number was a constant.

Turvy: Type in blank space "88," OK?
Pam: No, the last two digits at the end.
Turvy: Show me, please.
Pam: [Picks "75."]
Turvy: Two digits before period at end of paragraph?
Pam: Yes.

… Later, Turvy encounters author "van Lehn," mistakenly selecting "Lehn." When the user corrects it, Turvy asks for an explanation.

Turvy: What's different about this case?
Pam: Hmm.
Turvy: You selected these two words, why?
Pam: Because these two words make up his last name.
Turvy: Is there some feature of this word [highlights "van"] I should look for? Lowercase word?
Pam: [Hesitant] In this case, yes.

Turvy does the rest of the task without error.

tion. When first introduced to Turvy, users are told only that its goal is to help them perform repetitive tasks, that it watches what they do and understands "a bit of English." Users soon learn that Turvy knows nothing about bibliographies or proper names or dates. Instead, it must learn how to recognize them. They come in many varieties: a primary author's surname is the capitalized word before the first comma, unless there is only one author, in which case it precedes a colon. Sometimes it precedes "(ed)" and sometimes it has a baronial prefix like "van."

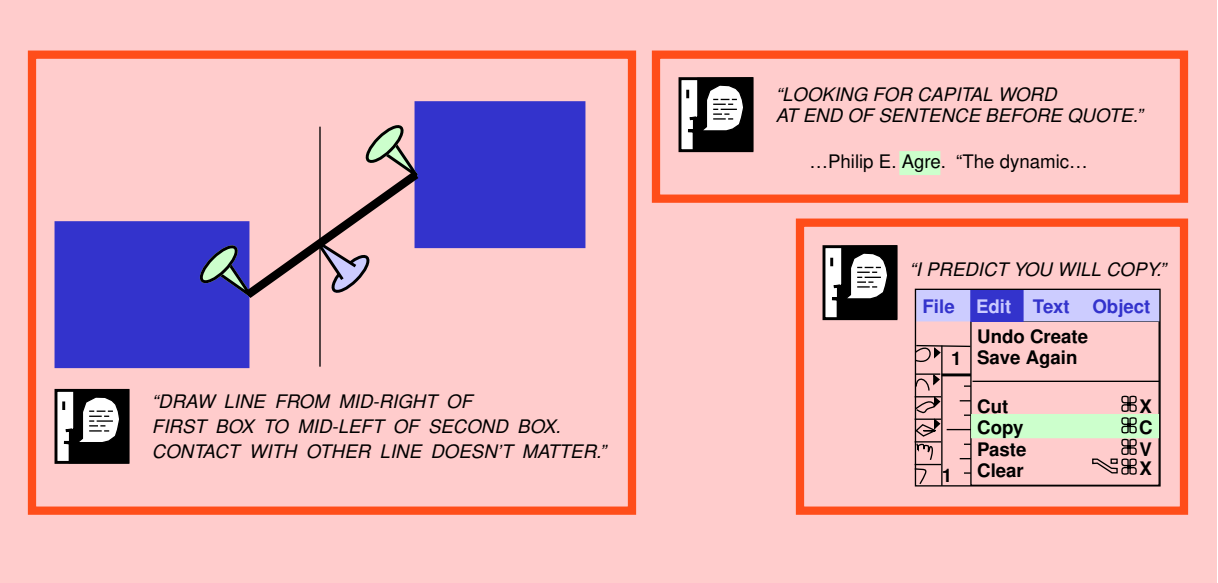
To perform this task manually on the 18 bibliography items, the user would have to execute more than 110 mouse clicks, 70 key presses, and 90 menu commands. Given the variety of special cases, creating a keyboard macro for this task would be impractical. Writing a C or BASIC program for this task would take even longer. In the user tests, Turvy performed nearly all the edits on 14 of the 18 entries, the other 4 serving as examples. When predicting actions, Turvy averaged 4 mistakes, which users had to correct.

To really understand Turvy, it is helpful to see it in action. Figure 3 presents "Pam," a composite of two users from the study, working on the task described above.

Design issues

When tested by more than a dozen users on word-processing, file management, and drawing tasks, Turvy revealed important design issues:

Figure 4 Examples of visual and spoken feedback



- The agent should learn from examples: Verbal specifications alone are too difficult for users to compose, and they are riddled with ambiguity and error. Though progress is being made in instructible natural language interfaces,[6,7] we believe examples should play a primary role.
- The agent should accept verbal and gestural hints in conjunction with examples; this speeds learning by focusing attention on relevant features of the examples.
- The agent should teach users its language and its background knowledge through the feedback it gives; verbal feedback should include only terms the agent can recognize. Figure 4 illustrates some forms of feedback.
- When the agent needs a hint because it cannot explain some action, it should not ask "Why did you do this?" Instead, the agent should make a guess, to which the user can respond with a hint.
- Learning about special cases (e.g., "van Lehn") as they arise makes the agent much easier to teach; the agent should discourage users from giving long verbal instructions that try to anticipate special cases.
- The agent should watch what the user does *not* do; implicit negative examples (e.g., the user selected surnames but not initials) speed learning and reduce predicted negative examples (i.e., mistakes).

Although Turvy represents just one kind of instructible agent, it exemplifies characteristics we advocate for instructible agents in general. To summarize them:

- The agent should be able to be personalized; it should learn what the individual user is trying to do.
- The user should be able to control the agent by teaching it.
- The user should be able to develop a mental model of what the agent knows about the task.
- The user should not be required to interact; the agent can learn by watching the user work.
- The agent should learn from its mistakes.
- The agent should learn continually, integrating new cases with prior knowledge.

Figure 5 shows the flow of information between the user and the agent, while Figure 6 illustrates the flow of information between application and agent. In particular, the lower half of the figure shows interactions required for the agent to operate the application and to augment its output with feedback (as in Figure 4). Agents often add functionality to applications; for example, graphical aids such as Mondrian[8] deal with interobject constraints that an ordinary drawing program might not represent. In this case, the application

Figure 5 Interactions between user and agent

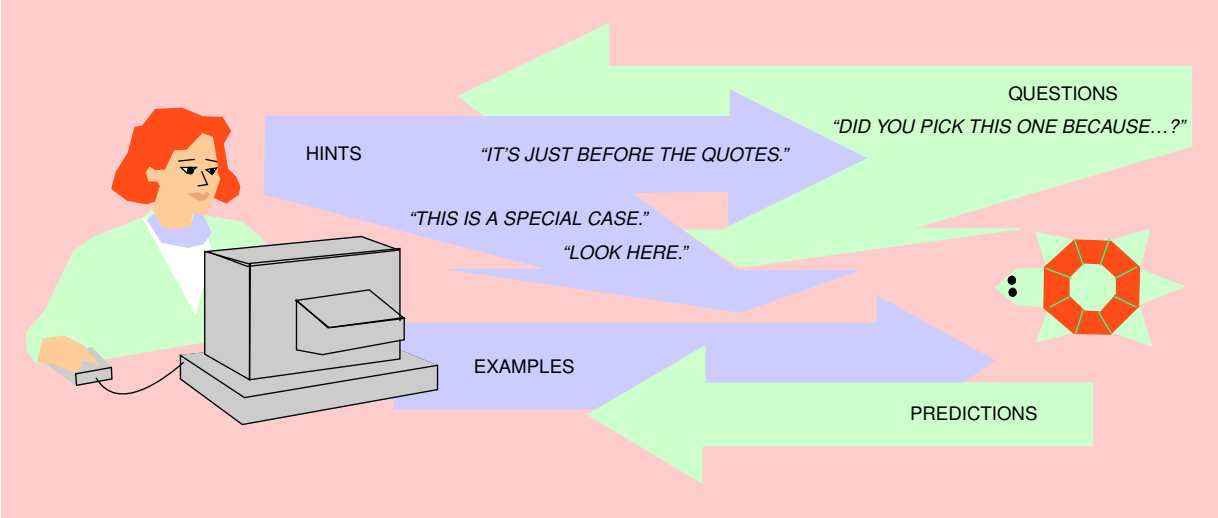
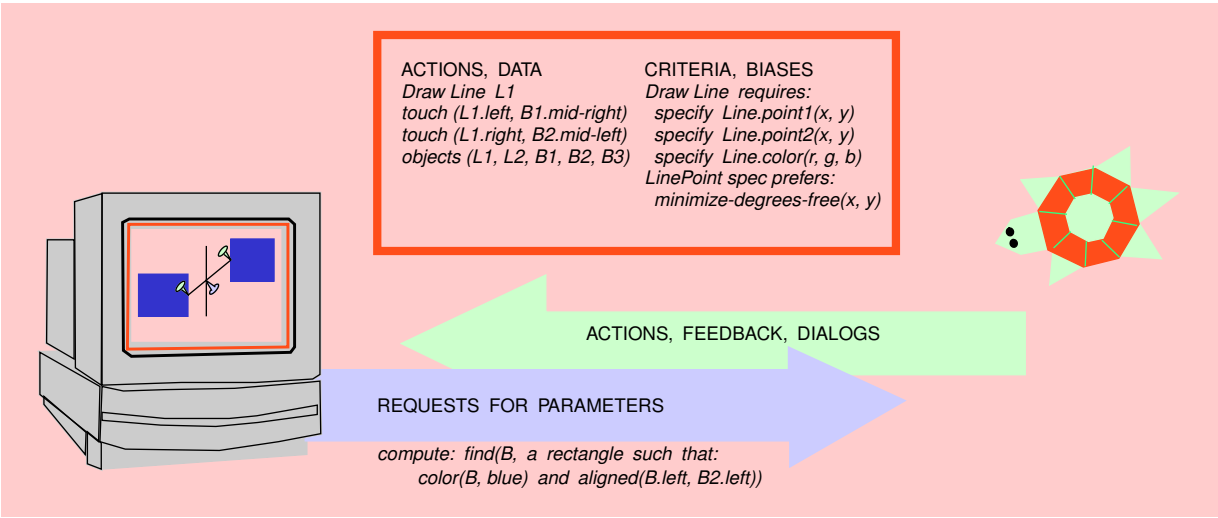


Figure 6 Interactions between application and agent



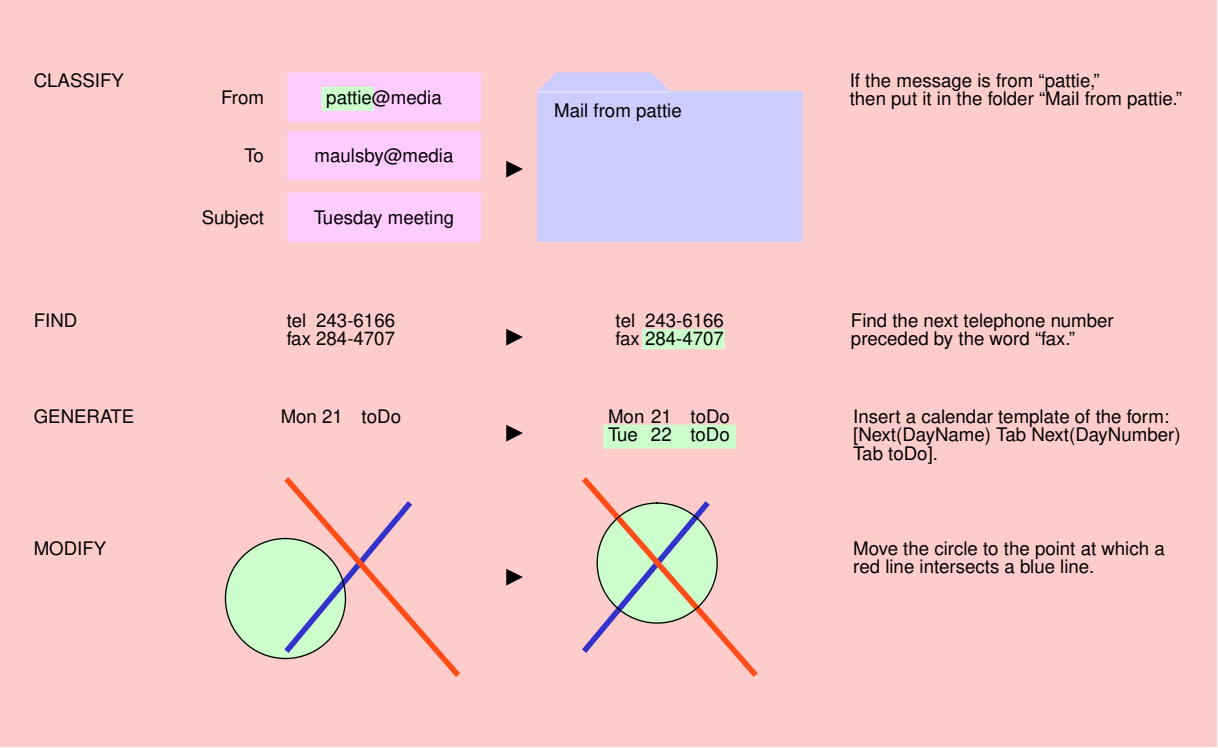
may be unable to execute all of the script directly, and must call back to the agent to compute some parameters.

Incorporating feedback and user-agent dialogs can be problematic, since most current systems provide no means for other applications to modify their displays. To skirt these problems, many existing agent systems such as Eager[1] are tightly coupled to applications.

Machine learning requirements

Cima ("Chee-mah") is a symbolic learning algorithm that has been implemented to provide the kind of learning necessary to realize the Turvy scenario. Cima learns about data, actions, and the conditions under which actions are performed. Although a generic learning system, it is designed to be readily customized for particular applications, such as text editing.

Figure 7 General types of action on data



At present, it has been partially integrated with a text editor so that it can record and learn how to search for syntactic patterns in the text. Its design illustrates two important differences between learning in an instructible agent and traditional machine learning: reliance on task-oriented background knowledge, and the interpretation of ambiguous instructions.

Action microtheory. An agent must learn generalizations that map to executable operations within an application. We have identified some categories of actions and their operational criteria. This task knowledge, primitive though it is, biases the learner so that it forms plausible, operational hypotheses from very few examples.

Most user-interface actions apply some primitive operator (like copy or move) to selected data. Thus, the first key to learning about actions is to learn about data. *Data descriptions*[1] specify criteria for selecting and modifying objects. For instance, suppose the user wants an agent to store electronic mail messages from Pattie Maes in the folder "Mail from pattie," as shown at the top of Figure 7. The data description *sender's id begins "pattie"* tells it which messages to select—those from Pattie, regardless of her current workstation. The data description *folder named "Mail from <first word of sender's id>"* tells it where to put them.

Conventional machine-learning algorithms learn to classify examples. But agents *do* things with data, and to be useful, data descriptions may require features in addition to those needed for classification. Figure 7 illustrates four types of actions: classify data, find data, generate new data, and modify properties. Cima encodes utility criteria that specify the characteristics of a well-formed data description for each type of action.

Classify actions have a single utility criterion: to discriminate between positive and negative examples. Features with the most discriminating power are therefore strongly preferred.

Find adds a second criterion: The description must delimit objects, and, in some domains, state the direc-

tion of search. Thus a text search pattern specifies where the string begins and ends, and whether to scan forward or backward. Features that describe more delimiters or constraints are preferred.

Generate adds a third criterion: The description should specify all features of a new instance. If generating a graphic, the description must specify size, shape, color, etc.; for text, it must specify the actual string. Though *user input* is a valid feature value, the system strongly prefers value "generators"—constants, such as *"toDo,"* or functions, such as *Next(DayName)*.

Mondrian is a prototype graphical editor that can learn graphical procedures demonstrated by example.

Modify stipulates two criteria: The description should discriminate between positive and negative examples, and it should generate the property's new value. Features that determine the new value are preferred to those that merely constrain it. The graphics example in Figure 7 shows a conjunction of two features, *touch(Circle.center, Line1)* and *touch(Circle.center, Line2),* that together determine a property value, the circle's new (*x*,*y*) location. By itself, each intersection leaves one degree of freedom on the circle's location. The utility criteria for setting an object's location assume that the goal is to remove all degrees of freedom if possible. Hence, features that remove both degrees of freedom, e.g., *touch(Circle.center, Line1.midpoint)*, are preferred over features that remove one degree of freedom. Cima continues adding *touch(Circle.center, Line)* features until zero degrees of freedom remain. If the user rejects an example in which the circle touches two blue lines, Cima adds a third feature—that one of the lines be red—to satisfy the classification criterion.

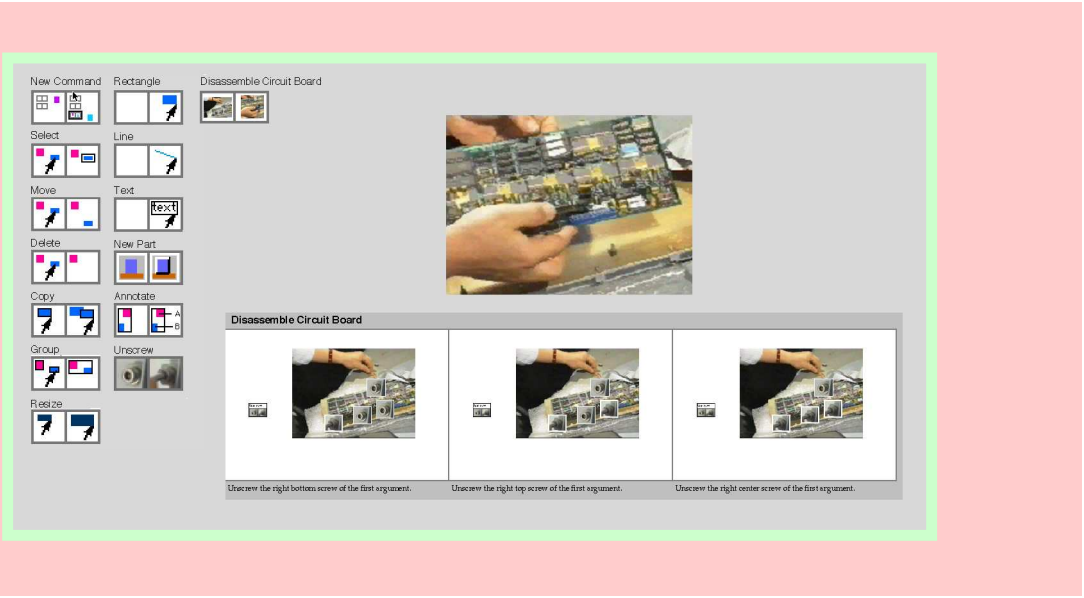
Interpreting user instructions. Most machine learning systems generalize from positive and negative examples, using biases encoded in their algorithms and perhaps in domain knowledge. Cima learns from examples, but also from hints and partial specifications. Hints, which may be verbal or gestural, indicate whether features of data (such as the color of a line) are relevant. The salient characteristic of hints is ambiguity: Cima must choose among multiple interpretations. Partial specifications, on the other hand, are unambiguous, though incomplete, descriptions. They specify relevant features, and even rules, and may be input as textual program fragments or by editing property sheets. Cima's learning algorithm composes rules (which are conjunctions of features) that meet the utility criteria for describing an action. It combines the evidence of examples, hints, specifications, and background knowledge to select the most justifiable features for each rule. By combining knowledge sources, Cima is able to maximize information gained from a small set of examples and to choose the most justifiable interpretation of ambiguous hints.

Evaluating instructible agents. It is difficult to perform evaluation studies on instructible agents, because users tend not to use an instructible system "in the same way" as they do an ordinary noninstructible system. However, one can specify particular tasks with potential repetitive or semirepetitive subtasks, and test whether users are able to successfully teach the system how to do such tasks. Such a study was conducted at Apple Computer,[9] which is a good example of how instructible agents can be evaluated. Potter introduces the notion of "just in time" programming,[10] a criterion for evaluating instructibility. He argues that the benefit of automating the task should outweigh the cost of performing the instruction and the possible risk of the instruction being performed incorrectly.

Mondrian: User interface requirements for learning from explicit instruction

As another example of an instructible agent, we consider Mondrian, a prototype object-oriented graphical editor that can learn new graphical procedures demonstrated by example. Like Cima, Mondrian observes user actions and automates procedures in the context of a familiar interactive application—in Mondrian's case, a graphical editor. Mondrian implements a much simpler learning algorithm than Cima, because it relies more on explicit instruction, and it provides an interface for teaching that is carefully integrated into the underlying interactive application. Procedures are demonstrated by user interface actions, and the results

Figure 8 Teaching a technical procedure through a video example



are put back into the graphical interface in the form of icons generated by the system, so they can take part in defining future operations.

Mondrian addresses the problem of teaching not only procedural, but also declarative knowledge. A teacher might need to teach not only a sequence of actions, but also new concepts in the course of the demonstration. Mondrian's approach to this is *graphical annotation.* The user can draw annotations on an image to introduce new objects and to specify relations between objects. These objects and relations become hints to the learning process, as expressed in the discussion of Turvy and Cima. Whereas Instructo-Soar[3] lets the user describe the relevant features of data in a restricted form of natural language, in a graphical domain, drawn annotations can be more natural than verbal descriptions.

In the text domain in which Cima operates, the display is relatively straightforward. In Mondrian's graphic domain a rich set of media interfaces is used to give the user the feeling of interacting with virtual objects. Graphical objects can be drawn, moved, copied, etc. Video images can be used as sources for the demonstration and new objects and relations introduced from the video frames. Voice input can be used to give advice to the agent and voice output is used to communicate the agent's interpretation of the user's actions. This lets the user adopt a natural "show-and-tell" method for teaching the agent.

To make the discussion more concrete, we introduce one of Mondrian's application domains, automating and documenting operational and maintenance procedures for electrical devices. The MIT Media Laboratory is collaborating with the Italian electronics company Alenia Spazio S.p.A., which makes many specialized electronic devices for the aerospace industry. These devices are characterized by the following attributes: they are complex, have relatively small user communities, must be operated by relatively untrained personnel under time constraints, and the consequences of operational mistakes can be expensive. All these factors conspire to make the programming of automated equipment, or documentation for teaching the procedures to people, expensive, time-consuming, and error-prone.

A relatively untrained technician who is familiar with the operation of the device might "teach" the computer the steps of the procedure by interacting with an on-screen simulation of the device. Figure 8 shows the interface for teaching Mondrian how to disassem-

Figure 9 Parts annotated with text labels

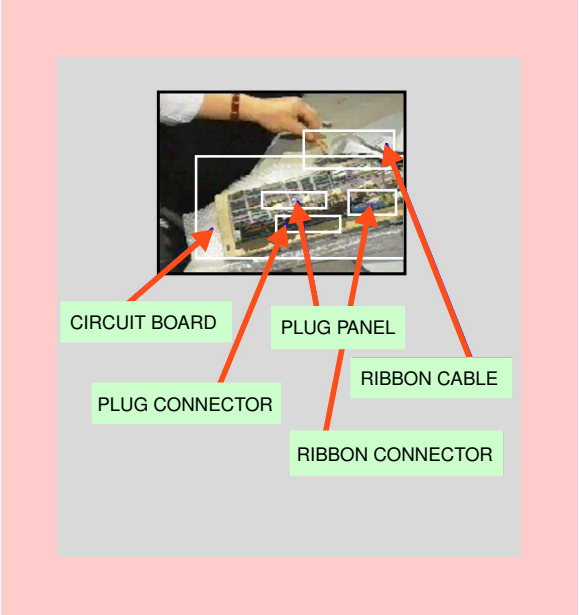
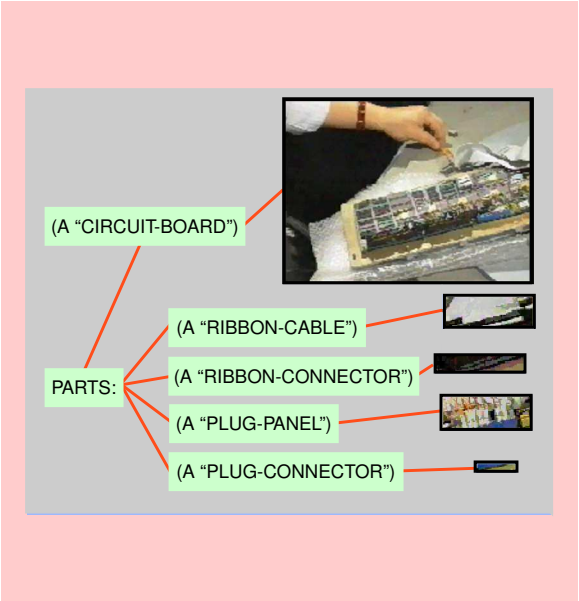


Figure 10 Part/whole graph



ble a circuit board. The simulation is created by teaching the computer (1) about the objects involved, by annotating images captured from videotaped demonstrations of the procedure, and (2) the actions involved in the procedure, by demonstrating actions in the user interface. Mondrian records and generalizes the user-interface actions, and synthesizes a procedure that can be applied to other examples. Such a procedure could also be used to drive robotic assembly or test equipment on an assembly line.

Identifying objects in video frames. We would like to give the user the illusion of interacting with a simulation of a device. If we were to program such a simulation "from scratch," the programmer would produce renderings of each object, and operating on these objects would produce a corresponding change in the renderings to reflect the consequences of the action. But to do this, the underlying model of the device and its structure must already exist. Here the purpose of the interaction is to capture the object model of the device and to teach the system about it.

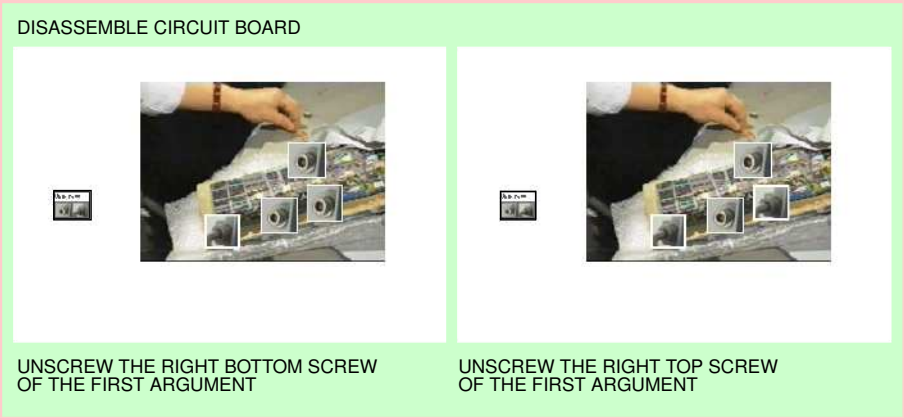
For each step that the user wishes to describe, he or she must select salient frames that represent the states before and after the relevant action. Subsets of the video image serve as visual representations of the objects. The next step is to label the images with a description of the objects and their structure. This is shown in Figure 9. The group structure of the graphical editor is used to represent the part/whole structure of the device.

A simple visual parser relates the text and lines to the images of the parts they annotate. A part/whole graph gives the user visual feedback, shown in Figure 10. The tree of labels shows the conceptual structure, and the leaves show specific images that represent each concept in the example.

As computer vision systems become more practical, the agent could perhaps automatically recognize some objects and relations. Some partial vision algorithms such as object tracking could streamline the annotation process. An alternative to choosing and annotating video frames might be to have the user sketch the parts and relations, then relate them to images through a technique like Query by Image Content.[11]

Descriptions of actions. After annotating the objects in the frame representing the initial state, the user selects a frame to represent the outcome of the operation. To construct a description of the procedure step, the user must use the operations of the graphical editor to transform the initial state to (an approximation of) the final state.

Figure 11 Storyboard for Disassemble Circuit Board



The editor provides both generic operations (Cut, Copy, Paste, Move, …) and operations specific to the domain (for the repair domain: Assemble, Disassemble, Turn On and Turn Off switches and lights, Open and Close fasteners, etc.).

Once a procedure has been recorded, it appears in Mondrian's palette as a domino, with miniaturized frames of the initial and final states of the procedure. The newly defined operation can then be applied to a new example. Assume that we have recorded a two-step procedure that disassembles the ribbon cable assembly and the plug panel assembly. First, we choose a video frame, then we outline the parts and add annotation labels that represent the input to the procedure. Clicking on the domino for the Disassemble-Circuit-Board procedure results in performing the disassembly of both parts in the new example.

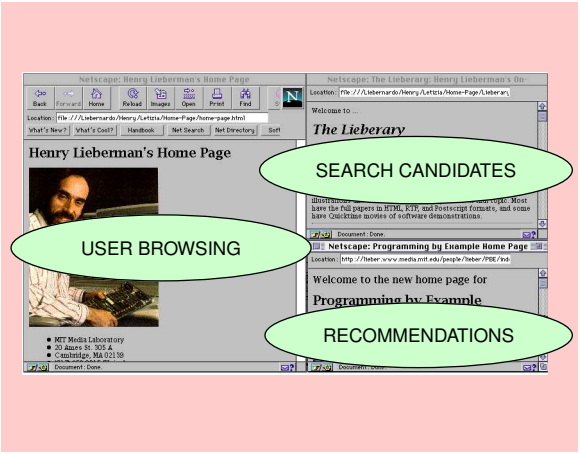
Documenting the procedure. We use a graphical storyboard to represent the procedure in the interface. The storyboard, shown in Figure 11, consists of a set of saved pictures of each step of the procedure, together with additional information that describes the step; the name of the operation chosen, a miniature icon for the operation, and the selected arguments are highlighted. The system also captions each frame of the storyboard with a text description of the step. A simple template-based natural language generator "reads the code" to the user by constructing sentences from templates associated with each function and each kind of object. The storyboard provides a form of automatically generated documentation, which, although it does not match the quality of documentation produced by a technical documentation staff, is inexpensive to produce and guaranteed not to omit crucial steps.

Mondrian and design criteria for instructible agents

Mondrian operates broadly within the design criteria for instructible agents outlined in the Turvy experiment, but takes a somewhat different approach than Cima in responding to those criteria. The basic collaboration between the user and the agent is the same. The user demonstrates the task on a concrete example in the interactive application, and the system records the user's actions. A learning algorithm generalizes the sequence of actions so that they can be used in examples similar to, but not exactly the same as, the examples used to teach the agent. The user can also supply hints that control the generalization process interactively. Mondrian's graphical and verbal feedback communicates to the user what its interpretations of the user's actions are, and through this, the user becomes familiar with its biases and capabilities. Thus it directly meets the first three design criteria outlined for Turvy.

In Mondrian, the user is assumed to be presenting the examples for the explicit purpose of teaching the agent, so the instruction tends to be more explicit than in other systems, such as Eager, that rely more on

Figure 12 Letizia helps the user browse the World Wide Web



unaided observation. The user explicitly indicates which objects are the examples at the start of the demonstration, and all subsequent operations are generalized relative to those examples. Mondrian does not interrupt the user during the demonstration the way Turvy does, although it optionally can provide its graphical and verbal feedback at each step. Mondrian cannot dynamically select which features of the examples to pay attention to, as does Cima, but there is no reason why Cima's feature-selection algorithm could not be incorporated.

Turvy's last three design criteria—hints in response to explanation failures, special cases, and negative examples—are not covered in Mondrian, but represent possibilities for future work. Lieberman's earlier instructible agent for general-purpose programming, Tinker,[12] demonstrated conditional and recursive procedures using multiple examples. This approach allows special cases and exceptional situations to be presented as supplementary examples after the most common, typical case is demonstrated. This technique might also work for Mondrian.

Mondrian, like most pure action learning systems, in terms of the action categories discussed for Cima, concentrates on the Generate and Modify actions. The system does not do pure Classification. It relies on the user to explicitly perform much of the Find actions. However, graphical annotations do allow the system to perform a kind of Find, since actions taken on a part described by an annotation in a teaching example will be performed on an analogous part in a new example that is selected by the relations specified in the annotation.

Letizia: An agent that learns from passive observation

Another kind of instructible agent is one that learns not so much from explicit instruction as from relatively passive observation. We illustrate this kind of agent with *Letizia*, an agent that operates in tandem with a conventional browser such as Netscape Communication Corp.'s Navigator** to assist a user in browsing the World Wide Web. The agent tracks the user's browsing behavior—following links, initiating searches, selecting specific pages—and tries to anticipate what items may be of interest to the user. It continuously displays a page containing its current recommendations, which the user can choose either to follow or to ignore and return to conventional browsing activity. It may also display a page summarizing recommendations to the user, as shown in Figure 12.

Here, there is no pretense that the user is explicitly teaching the system. The user is simply performing everyday activities, and it is up to the agent to make whatever inferences it can without relying on the user's explicit help or hints. The advantage of such agents is that they operate more autonomously and require less intervention on the part of the user. Of course, this limits the extent to which they can be controlled or adapted by the user.

Letizia is in the tradition of *behavior-based* interface agents,[13] so called to distinguish them from *knowledge-based* agents. These have become popular lately because of the knowledge bottleneck that has plagued pure knowledge-based systems. It is so difficult to build up an effective knowledge base for nontrivial domains—as in conventional expert systems—that systems that simply track and perform simple processing on user actions can be relatively attractive. The availability of increased computing power that can be used to analyze user actions in real time is also a factor in making such systems practical. Rather than rely on a preprogrammed knowledge representation structure to make decisions, the knowledge about the domain in behavior-based systems is incrementally acquired as a result of inferences from the user's concrete actions.

In the model adopted by Letizia, the search for information is a cooperative venture between the human user and an intelligent software agent. Letizia and the

user both browse the same search space of linked Web documents, looking for "interesting" ones. No goals are predefined in advance. The user's search has a reliable static evaluation function, but Letizia can explore search alternatives faster than the user can. In parallel with the user's browsing, Letizia conducts a resource-limited search to anticipate what pages a user might like to see next. Letizia adopts a mostly breadth-first search strategy that complements the mostly depth-first search strategy encouraged by the control structure of Web browsers such as Netscape's Navigator. Letizia produces a recommendation on the current state of the user's browsing and Letizia's own search. This recommendation is continuously displayed in an independent window, providing a kind of "channel surfing" interface.

User interest is inferred from two kinds of heuristics: those based on the content of the document and those based on user actions in the browser. Letizia reads each page and applies a simple keyword-based measure of similarity between the current document and the ones the user has explicitly chosen so far. This measure simply counts keywords and tries to weight them so as to find keywords that occur relatively frequently in the document but are rare otherwise. This is a standard content measure in information retrieval.

The second kind of heuristic is perhaps more interesting. It uses the sequence of user actions in the browser to infer interest. Each action that the user takes—selecting one link rather than another, saving links, or performing searches—can be taken as a "training example" for the agent trying to learn the user's browsing strategy. For example, if the user picks a page, then quickly returns to the containing page without choosing any of the links on the page, it might be an indication that the user found the page uninteresting. Choosing a high percentage of links off a particular page indicates interest in that page.

One characteristic of instructible agents is the method used to temporally relate the actions of the user and the actions of the agent. In Turvy, Cima, and Mondrian, the interaction is conversational: the user and the system "take turns" in performing actions or giving feedback. In Letizia, the agent is continuously active and accepting actions in parallel with the user's activity. This increases the sense of autonomy of the agent.

Because of this autonomy, Letizia does not satisfy many of the advisability criteria developed in the Turvy study. The user's influence on the agent is limited to browsing choices—but these choices are made not to influence the agent, but to directly accomplish the user's browsing goals.

However, each individual decision that Letizia makes is not so critical as it is in Cima or Mondrian. Because Letizia is only making suggestive recommendations to the user, it can be valuable even if it makes helpful suggestions only once in a while. Because it is continuously active, if it misses a possible suggestion at some time, it will likely get another chance to find that same choice in the future because of consistencies in the user's browsing behavior. There is a real-time trade-off between requiring more interaction from the user to make a better decision and exploring more possibilities. However, in the future we will probably back off from the pure unsupervised learning approach of Letizia and provide facilities for accepting hints and explicit instruction.

In Letizia, the agent is continuously active and accepting actions in parallel with the user's activity.

In Cima's discussion of categories of actions for instructible agents, Letizia's task is one of almost pure Classification—classifying a possible link as either interesting or not. Find, Generate, and Modify are not relevant.

ACT: Teaching virtual playmates and helpers

In the ALIVE project (Artificial Life Video Environment), developed in the Autonomous Agents Group at the MIT Media Laboratory,[14] people experience an imaginative encounter with computer agents, as their own video images are mixed with three-dimensional computer graphics. ALIVE agents are designed with "believability" in mind: the goal is to have the user perceive the agent as having distinctive personal qualities. Following ethological principles, their behavior

arises from complex interactions among instinctual motivations, environmental releasing mechanisms, and the dampening effects of frustration and satiation. At present, all behaviors must be preprogrammed. While some unanticipated behavior emerges from the interaction of these programs, the agents are nonetheless noticeably limited, especially in their ability to play with users.

Perhaps the most interesting class of real agents includes those who interact with us adaptively, changing their behavior so that we cannot quite predict them. They learn to help us or frustrate us, becoming a more useful assistant or challenging opponent. The goal of the ALIVE Critter Training project (ACT) is to augment agents with the ability to learn from the user. ACT is an attempt to bring the kind of learning represented by Cima to a virtual reality world. By training agents, users can customize and enrich their ALIVE experiences. And dealing with the unpredictable results of training is in itself entertaining and instructive.

The goal is to have the user perceive the ALIVE agent as having distinctive personal qualities.

The initial version of the ACT system learns to associate predefined actions with new releasing mechanisms. In particular, actions may be chained together as a task. Conceptually, ACT is an agent's learning "shadow," watching the agent interact with the user and other ALIVE agents and learning to predict their actions. ACT has an implicit motivation—to please the user by learning new behaviors.

One scenario consists of an ALIVE world in which the user and a synthetic character named Jasper play with virtual LEGO** blocks. Figure 13 shows Jasper helping the user to build a slide; since he does not already know what the user wants to build or what a slide is, he watches the user's actions and pitches in to perform any action he can predict, whether it is to gather up the right sort of blocks or to do the actual building. The assembly task here resembles the disassembly task that we taught to Mondrian earlier by graphical annotation of example video frames, and the learning strategy of recording and generalizing user actions is like that explored in Turvy and implemented in Cima. As in Cima, the agent looks for patterns in sequences of user actions and uses utility criteria to choose generalizations.

The spectrum of instructibility

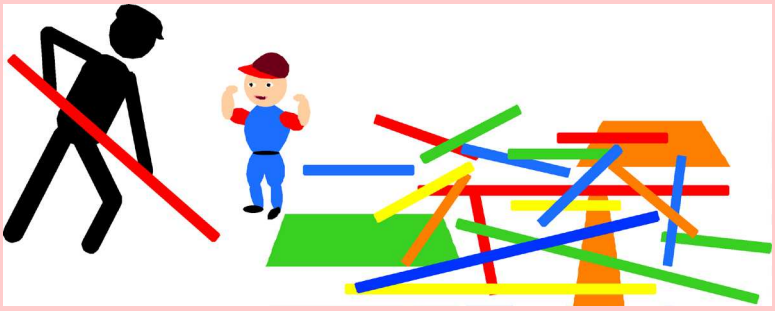
Just as "direct manipulation" and "software tool" are metaphors describing the user's perception of operating the computer, so too is "instructible agent." Physical tools translate a person's actions in order to manipulate objects, and software tools translate physical actions (captured by a mouse, keyboard, etc.) into commands for manipulating data. In contrast, an agent *interprets* the user's actions, treating them as instructions whose meaning depends on both the context in which they are given and the context in which they are carried out. Whether its task is to help format text or search the World Wide Web, the agent, unlike a tool, must deal with novelty, uncertainty, and ambiguity. The relationship between user and agent is based on delegation and trust, rather than control.

Because of this, some researchers have objected to the notion of agents on the presumption that they rob users of control over their computers.[15] At the heart of the antiagent argument lie two beliefs: first, that direct manipulation is good enough for the vast majority of tasks; and second, that "real" programming must be done in a formal language.

Figure 14 illustrates the spectrum of approaches to accomplishing tasks with a computer, comparing them in terms of their effectiveness and ease of use. The "English Butler" at the far right requires an introduction: this is a pre-educated, intelligent agent of the sort portrayed in Apple's Knowledge Navigator video. It understands a good deal of natural language and already knows how to carry out a variety of tasks; the user need only give a vague description and the Butler reasons its way to an executable program.

On the graph shown in Figure 14, brighter hue means more or better. The top line compares the approaches by the degree of artificial intelligence they require. Other than its skill at program optimization, a C++ compiler evinces little intelligence, as do direct manipulation interfaces, though some have "smart" modes: for instance, drawing programs that snap

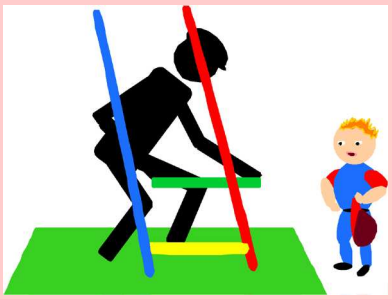
Figure 13 Building a slide together



The user (shown in silhouette) wants to build a slide for the ALIVE character, Jasper, using virtual LEGO blocks.

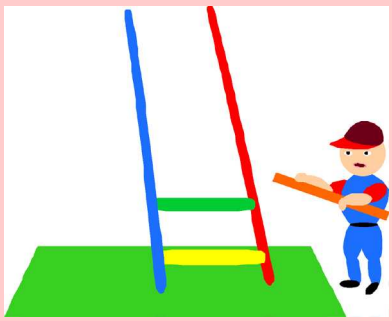
Jasper is naturally helpful: he tries to learn what the user is building so he can pitch in.

The user sets up the ladder frame and attaches two rungs.

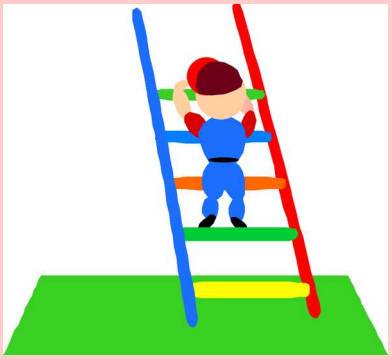


Jasper pays attention to the way the blocks are connected. He watches for repeated actions.

Jasper predicts that rods of various colors should be attached some distance apart along the frame.



When the ladder gets too tall for his arms' reach, Jasper's "behavior system" adopts another method—climbing or jumping—to extend his reach.



Soon he has placed rungs all along the frame. Meanwhile, the user has attached the slide.

All done!

Figure 14 Comparing a range of instructible systems

objects to one another (Intellidraw**) and text editors that predict input (Quicken**) or correct it automatically (Microsoft Word**). In contrast, the English Butler must be highly intelligent if it is to translate vague instructions into viable procedures; it must interpret natural language, maintain a model of discourse with the user, formulate plans, and change plans "on the fly" when they fail or when unexpected opportunities arise.

Instructible agents cover a range of intelligence. Some, like Mondrian, use only very simple rules of inference. This makes them highly efficient and effective for learning certain tasks. They rely on a well-designed user interface to harness the user's intelligence. Other systems, like Cima, use more sophisticated learning methods and background knowledge, trying to maximize the agent's contribution to the learning/teaching collaboration. What all instructible agents have in common is that they avoid relying on the agent's intelligence to plan the task: for this, they rely on the user.

The second scale on Figure 14, run-time adaptability, refers to a system's ability to change its program in response to changing conditions. Once a program is compiled, it is set in stone—unless written in a dynamic language such as LISP. Direct-manipulation user interfaces generally do not adapt their behavior, except to store default values and preferences. The English Butler must be highly adaptable, otherwise its plans would fail. Instructible agents range from macro recorders that fix on a procedure after a single example (Mondrian) or a few examples (Eager), to systems that learn continuously and try to integrate new concepts with old ones (Cima).

On the third scale, task automation, traditional programming and the English Butler score highest, since they are able to execute procedures without user intervention. Of course, this maximizes the risk of doing something bad. Direct manipulation automates only certain types of tasks, such as updating all paragraphs of a given style to a new format. Instructible agents can be used to automate tasks completely, but in general they are intended for "mixed-initiative" collaboration with the user, as exemplified in all the systems we have discussed.

The next scale, programmability, concerns the degree to which programmers can determine a system's behavior. A "serious" programming system enables programmers to describe any computable task and to choose an optimal method for doing so. At the oppo-

site extreme, direct-manipulation interfaces are inherently nonprogrammable, though as noted above, some offer restricted forms of task automation that the user may parameterize. Nor is the English Butler truly programmable, since it decides for itself how to accomplish a task, and it can perform only those tasks it is capable of planning. Instructible agents are programmable, and in theory could provide a "Turing-complete" instruction set, though in practice they tend to restrict the forms of program that users can create, either because they lack some primitives and control structures or because they cannot infer them from a given set of examples. The inevitable (and desirable) limit on an agent's exploration of hypotheses is a good argument for providing other means of instructing it: in other words, learning agents can be part of a programming system, not necessarily a replacement for one.

Another issue—a controversial one, as noted above—is the degree to which the end user can control the system's task behavior. A C++ program gives the end user no control, except through its user interface. A direct-manipulation interface provides a high degree of control, *provided* it is simple to understand and offers immediate feedback. Unfortunately, direct-manipulation interfaces are becoming more complex and feature-ridden, as software vendors respond to users' demands for more powerful features. There is no control without understanding, and such complexity makes understanding the interface practically impossible. The English Butler takes commands but can easily get out of control by misinterpreting them; we presume that this is the sort of agent that Shneiderman[15] decries. Instructible agents are more easily controlled, because the user can define their behavior in detail, as exemplified in Mondrian. An agent that learns continuously from user instruction, such as Turvy, affords user control over more complex behavior. Systems such as Letizia, that learn by observation only, can avoid robbing the user of control by offering suggestions rather than executing actions; since they cannot be guaranteed to do the right thing, they learn to make it easier for the user to do the right thing.

The last two scales on Figure 14 have been paid the most attention by the designers of instructible agents. The effort of instruction involves the cognitive and physical operations needed to transmit knowledge from the user to the computer. Programming in a formal language is high-effort in both respects, whereas direct manipulation has demonstrated success in minimizing effort—until tasks become repetitive or require great precision or involve visually complex structures. Delegating tasks to the English Butler is simplicity itself—in theory—because the user does not even have to know precisely what he or she wants. In practice, describing a task in words is often more difficult than performing it. Although instructible agents vary somewhat in ease of use, the goal is generally to minimize the effort of instruction beyond that needed to demonstrate an example.

Finally, learning how to instruct the computer is the first and greatest hurdle users will encounter. Although direct manipulation interfaces removed the "language barrier" between user and machine, they are now likely to come with an instructional video to guide the new owner through the maze of special features. Learning to instruct the English Butler requires little or no effort, unless it has "quirks" (such as unpredictable limitations) about which the user must learn. Similarly, the users of instructible agents must learn about their limitations (such as Turvy's restricted vocabulary) and about the tools of instruction (such as Mondrian's graphical annotations). As noted above, feedback is considered crucial for helping users learn how to teach.

We think instructible agents represent a "sweet spot" in the trade-offs represented by the spectrum shown in Figure 14. They provide a way for the user to easily delegate tasks to a semi-intelligent system without giving up the possibility of detailed control. While they take advantage of many AI (artificial intelligence) techniques, they do not require full solutions to AI problems in order to be useful, and they can enhance the usefulness of existing applications and interfaces. The best part is that, as you use them, they just keep getting better.

**Trademark or registered trademark of Apple Computer, Inc., Microsoft Corp., Netscape Communications Corp., LEGO Systems, Inc., Aldus Corporation, or Intuit.

Cited references

1. *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor, MIT Press, Cambridge, MA (1993).
2. D. Maulsby and I. H. Witten. "Learning to Describe Data in Actions," *Proceedings of the Programming by Demonstration Workshop, Twelfth International Conference on Machine Learning*, Tahoe City, CA (July 1995), pp. 65–73.
3. S. Huffman and J. Laird, "Flexibly Instructible Agents," *Journal of Artificial Intelligence Research* **3**, 271–324 (1995).
4. H. Lieberman, "Letizia: An Agent that Assists Web Browsing," *International Joint Conference on Artificial Intelligence*, Montreal, Canada (August 1995).

5. D. Maulsby, S. Greenberg, and R. Mander, "Prototyping an Intelligent Agent Through Wizard of Oz," *Conference Proceedings on Human Factors in Computing Systems*, Amsterdam, May 1993, ACM Press, New York (1993), pp. 277–285.
6. C. Crangle and P. Suppes, *Language and Learning for Robots*, CSLI Press, Palo Alto, CA (1994).
7. J. F. Lehman, *Adaptive Parsing: Self-Extending Natural Language Interfaces*, Kluwer Academic Publishers, Norwell, MA (1992).
8. H. Lieberman, "Mondrian: A Teachable Graphical Editor," *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor, MIT Press, Cambridge, MA (1993), pp. 341–361.
9. K. Kvavik, S. Karimi, A. Cypher, and D. Mayhew, "User-Centered Processes and Evaluation in Product Development," *Interactions* **1**, No. 3, 65–71, Association for Computing Machinery, New York (July 1994).
10. R. Potter, "Just in Time Programming," *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor, MIT Press, Cambridge, MA (1993), pp. 513–526.
11. M. Flickner, H. Sawney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by Image and Video Content: The QBIC System," *IEEE Computer* **28**, No. 9, 23–48 (September 1995).
12. H. Lieberman, "Tinker: A Programming by Demonstration System for Beginning Programmers," *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor, MIT Press, Cambridge, MA (1993), pp. 48–64.
13. P. Maes and R. Kozierok, "Learning Interface Agents," *Proceedings of the National Conference on Artificial Intelligence 1993,* AAAI, Menlo Park, CA (1993), pp. 459–465.
14. P. Maes, "Artificial Life Meets Entertainment: Lifelike Autonomous Agents," *Communications of the ACM* **38**, No. 11, 108–111 (November 1995).
15. B. Shneiderman, "Beyond Intelligent Machines: Just Do It," *IEEE Software* **10**, No. 1, 100–103 (January 1993).

Accepted for publication April 10, 1996.

Henry Lieberman *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: lieber@media.mit.edu).* Dr. Lieberman has been a research scientist at MIT since 1975, first with the Artificial Intelligence Laboratory, then with the Media Laboratory since 1987. At the Media Lab he is a member of the Autonomous Agents Group, and his interests lie at the intersection of artificial intelligence, the human/computer interface, and graphics. He received a B.S. degree in mathematics from MIT in 1975. His graduate degree, called *Habilitation a Diriger des Recherches*, is the highest degree given by a French university. It was awarded in 1990 from Université Paris 6, Pierre et Marie Curie, where he was a visiting professor from 1989 to 1990. He has published over 40 research papers.

David Maulsby *CAMIS Medical Informatics Section, Stanford University, Stanford, California 94305 (electronic mail: maulsby@camis.stanford.edu).* Dr. Maulsby was awarded the B.Sc., M.Sc., and Ph.D. degrees, all in computer science, from the University of Calgary, Canada. During 1995 he was a postdoctoral Fellow in the Autonomous Agents Group at the MIT Media Lab. He is spending the second year of his fellowship in the Medical Informatics Group at Stanford University.

Reprint Order No. G321-5623.