# Agents for the User Interface


## Henry Lieberman and Ted Selker
### Media Laboratory
### Massachusetts Institute of Technology


## Introduction

We often use the word “agent” to describe people who have a helping or assistive relationship to us – travel agents, secretaries, butlers. The job of such an agent is to act autonomously to satisfy goals that we may have, give us a greater sense of productivity and reduce our workload. In this article, we’ll use the word agent to describe software that plays a similar role – providing help, advice, and “running errands” for the user.

Just as with a human agent, the nature of the relationship between the agent and the client is paramount to success. So with computer agents, it is often the user interface which holds the key. The image of an agent or agent-oriented system as acting in a helping role like a butler, secretary, or service organization is most vivid when that helping role is reflected directly in interaction with the user.

Oliver Selfridge, whose 1959 Pandemonium paper [] introduced the term “agent”, referred both to this sense and also to a sense of agent internal to a system, where multiple goal-seeking entities both compete and cooperate to produce intelligent behavior. Many of the other chapters of this book describe these “back-end” agents, where most of the agentry is taking place behind the scenes, and interaction with the user happens in a relatively conventional manner. But in this article, we’ll focus on agents that display the characteristics of intelligence in interacting directly with the user.

### What do you mean by “intelligent interface agent”?

Terminology in the field is a problem, since researchers have varying definitions, especially of the term “agent”. But for the purposes of this article, we’ll break down the term “intelligent interface agent” as follows:

- *Intelligent*. One way to define computer intelligence is that it is when the computer exhibits behavior that is like what we call intelligence in people. Human intelligence is composed of a wide variety of thought, affect and behavioral mechanisms, from something as simple as holding a pen, to the complexity of interpersonal relationships. Some of these mechanisms are conscious and others are opaque to our introspection. When computers appear to solve problems in ways we would not have expected to be able to do mechanically, it is reasonable to ascribe intelligence to the machine.

While it is unrealistic to expect that an interface agent have human-level intelligence, many agents do exhibit some characteristics of human intelligence. Some of them do reasoning and inference, and some of them have domain-specific knowledge or procedures that enable them to perform useful tasks. Some of them learn through interaction with their users and adapt to context. Daniel Dennett [] proposed the *intentional stance* as a criteria for machine intelligence – if it makes sense to ascribe intention to the machine as the best way to explain its behavior, as in "the machine learned what I wanted" – then you can call it intelligent.

- *Interface*. For an agent to be considered an "interface" agent, we'll require that the agent communicate with the person directly through the input and output of the user interface or interface to the environment. An interface agent can observe actions taken by the user in a direct manipulation interface, can sense the objects that the user sees on the screen, and can itself take actions by invoking the commands provided by the interface. The agent can add graphics or animation to the interface, it can use speech input or output, or communicate via other sensory streams. Increasingly, agents also will use sensors and effectors that sense and act directly with the real world. An agent that senses force in an exercise machine or moves a bulldozer blade can also be an interface agent.

- *Agent*. This is the most controversial term. Among the myriad senses of agents described in this book, all seem to involve the ascription of some form of human characteristics to the agent, whether it is autonomy, mobility, intelligence, etc. Here we'll concentrate on the function of an agent in an assistive role to the user. The job of the agent, like human agents such as travel agents or stockbrokers, is to further the person's task, that is, to do things that make the person's goals succeed. It can act on the person's behalf, which we call an *assistant*, or teach the person how to perform the task, which we call an *advisor*. We believe that there is an*agential stance*, similar to Dennett's intentional stance. The machine can be considered an agent if the best way to explain its behavior is by analogy to the agential role that humans can play.

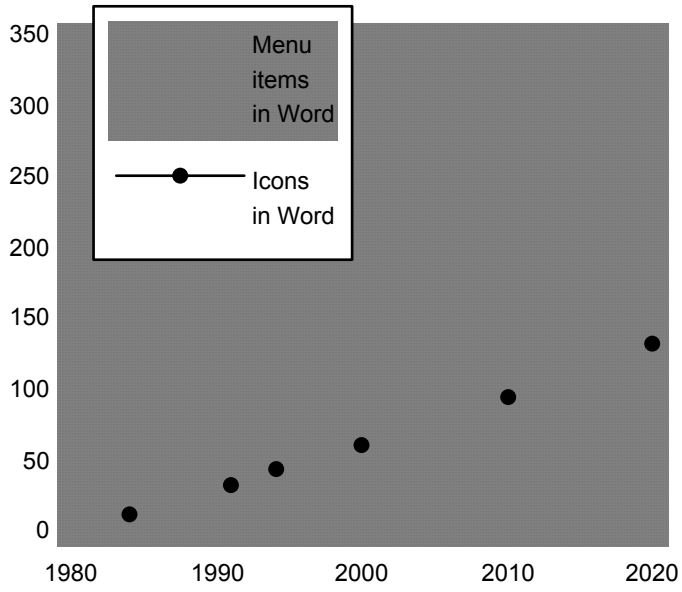
Why interface agents?

Since the 1980s, the field of human computer interface, as represented at the annual CHI and Interact conferences, has concentrated on the development of the so-called direct manipulation interface [Shneiderman] of multiple-windows, menus, icons, and pointing devices. This model has served us well for two decades, but it alone will not suffice for future applications. The reason is that the growth of functionality in these interfaces is not sustainable. Interface agents represent a way out of this dilemma.

We can think of many current interactive applications as like a tool box containing hammers, screwdrivers, and pliers. Each menu operation, typed command, or graphic icon can be thought of as the means of access to an individual tool. It is up to the user

to decide which tool to use for what purpose, and what sequence of steps of use of individual tools will accomplish a given task. The problem is that, as the scope of applications increases, each purpose may require a separate tool, and the number of tools available to the user grows too large.

This is in fact what is happening in computer interfaces today. The graph below shows the growth of the tool set for a typical application, Microsoft Word. Under the [perhaps charitable] assumption that the growth is linear, our grandchildren will have to deal with an absurdly large number of menu operations and icons. The application of computers to more and more ambitious tasks shows no sign of slowing. Inevitably, we will have to at some point abandon the simple one-to-one correspondence of one user interface element per tool.




*Growth of the tool set in a typical application*

A single user interface element will need to have the potential of accessing any number of underlying tools, making independent decisions depending on the current interactive context, without unnecessary user interaction at each step. User interface elements will become user agents.

Shared context cuts down on the amount of communication necessary between the computer and the user. When we talk to other people, do we need to enumerate all the possible things we could say or do? Of course not. It is assumed that intelligent use of the communication channel means to choose only those possibilities that are meaningful and appropriate.

The only plausible alternative to thinking about an interface as being a set of tools is thinking about an interface as acting as one or more agents. Users delegate their goals to agents, rather than directly manipulate screen representations of objects. Negroponte

[] uses the metaphor of a highly trained English butler, who uses personal knowledge of their employer and proactive anticipation of the employer's needs and goals to provide service. Other kinds of agents, such as travel agents, stockbrokers, and car mechanics might have different areas of expertise and service. On-line travel planners and electronic traders are now beginning to serve some of the functions that were formerly performed by human agents in these areas, and we expect that trend to continue. More and more, personal history, anticipation of needs and goals, adaptivity to context, and long-term learning will play important roles in the user interface.

## A story: Evolution of the washing machine

If I described an automated washing machine of today to a person in 1920, they would call it an agent—something that does something a person or they would have had to do—because washing clothes was a procedure that required intelligence: washing the clothes, thinking about them, etc. Today we don't think of the washing machine that way, we think of a washing machine as a washing machine—a simple, inexpensive tool that everyone can have.

Early in this century washing was thought to be something where you put soap on fabric, rubbed it, rinsed it out, looked at it, put soap on the hard places, rubbed it again, rinsed it out, wrung it out, and hung it up to dry. An automated washing machine came along which could do some parts of that. A person would turn on the motor to make the fabric and the soap and water rub against each other, then turn it off, turn a valve which would let the water out. More water would be put in buckets and the fabric agitated again to get rid of the soap. Finally, the person would wring out the clothes. At some point a very exciting idea arose -- to have a timer that would time when different parts of this process occurred, deciding to wash all the clothing, rinse it, and *spin* it dry.

The procedures that had once been totally manual, and then mechanical, were suddenly automated. The list of steps was the equivalent of a program, and activation of the timer was the input from the user. Sensors monitor the process so that an alert signal occurs if the machine is off balance, or users may choose options that allow notification during the process, such as when to add fabric softener.

The automation of all these processes, putting them together and allowing a person to simply put in the clothes and the soap, became the washing machine that we know today. At the time, even when the mechanical washing machine was created, it would have fit our definition of agent. That is, the timer was the intelligence that made the mechanical aspects of the washing machine replace what the person thought they had to do before. Instead of just replacing the mechanical aspects of moving the fabric, the timer replaced the *procedure* that the person had to do. In that way, the washing machine's timer makes the washing machine into an agent. Certainly today we don't think of the washing machine as an agent—a washing machine is a washing machine. In that way, many things that we think of as agents will over time become thought of simply as tools, units that are

inseparable from the way they are used and integral to our lifestyle.

Visions of future interface agents

In 1990, Apple [CHI Video Review] released the controversial film, Knowledge Navigator, which presented their vision of what a future computer interface would be like. It presented a scenario of a professor preparing material for a class, managing his schedule, retrieving articles, communicating with a colleague.

Technological breakthroughs such as continuous speech and natural language understanding were posited, but the most significant departure from present-day interfaces was the presence of Phil. Phil was a bow-tied character who appeared on the screen in animated video, responding to commands, offering advice, providing reminders, keeping track of goals, fetching and integrating data from diverse sources when needed. Phil performed the function of a secretary or assistant at human-level competence.

The *Knowledge Navigator* video was the most widely circulated, but was only one of many "vision videos" that included the earlier Atari "Intelligent Encyclopaedia", and later HP's "1995", NTT's "Seamless Media" Sun's "Starfire", AT&T's "Connections" and videos by Ford, Philips, IBM, Atari and others. Some of these provide scenarios that differ in important respects from Apple's vision, but all potray the computer as playing a helpful and assistive role. Science fiction had long portrayed intelligent robots and computers, such as 2001's HAL, but these videos were statements that many of the capabilities were indeed close to being within our grasp.

The film was controversial in many respects. First, it was clear that this vision, if it could be achieved, would represent a radical improvement in the helpfulness of computers in daily life over present ways of working. But some people also felt wary of the scenario; not everyone wants a butler. Some found the agent annoying or intrusive: "Why should I have to negotiate with a little dip in a bow tie whenever I want to do something?"

There was doubt about its feasibility. However, [Miller and Norman] performed an analysis showing that most of the ingredients were indeed technically feasible. Voice recognition is now on the verge of practicality, while full natural language understanding remains elusive. Videoconferencing and the ease of data access we now enjoy on the Web is commonplace. Striking in the scenario but still not achieved today was its seamlessness – there were no "applications", no file dialog boxes, no cut and paste. We could certainly eliminate these stumbling blocks today if we wanted.

Nevertheless, Knowledge Navigator presented a provocative vision of how agents might transform the interface. Seeing these

vision videos still is one of the best ways to get a feel for what agents in the interface might mean.

## Characteristics of interface agents

All present-day direct-manipulation interfaces are essentially editors for sets of objects represented graphically on the screen, be they text, pictures, spreadsheet cells, e-mail messages, etc. What is striking about the vision contained in these videos, is that very little of the interaction centers on object-editing. Rather it centers on the user's goals [writing a paper, making a business presentation, arranging a schedule], and what object editing does occur is incidental. Most of the interaction is in the form of the computer supplying information to teach a user or delegating tasks to an interface agent.

Delegation to agents, just like delegation to humans, always involves some risk and followup; the risk that intentions could be misunderstood, and the risk that the task might not be performed correctly. But we don't let that stop us from delegating many tasks to people that we work with, and the result is that we are able to do more by collaborating with others.

Collaborations often seem slower because it is hard to communicate and delegate accurately. Often the most important part of the collaboration is learning from the other person. Many of the most successful user interface agents have been systems that carefully decide what kind of information and how much information to tell the user, and know how to stay out of the way when users prefer to act for themselves.

### Initiative

Most conventional interfaces simply sit still unless the user is actively commanding them. Interface agents, on the other hand, can be proactive, actively working while the user is thinking, or performing other actions. This saves the user time, since the system is making use of time that would otherwise be wasted. It allows the agent to anticipate the needs of the user. The system may even take initiative in interrupting the user [for example, if a higher-priority event occurs]. The agent can alert the user to opportunities that would otherwise be missed. This capability must be used wisely, or users will object to being interrupted. This leads to a mixed-initiative dialog, where either the user or the agent may take the lead.

### Assistants and Advisors

We can distinguish between two roles that an interface agent might play, assistants and advisors.

- *Assistant*. An assistant agent is one that does things that you could have done. By being an assistant, as a butler or a servant, the agent is doing things for you. Instead of you having to type in something, it is typing in something; instead of you having to arrange something, it is arranging something; instead of you having to find something, it is finding something for you. To the extent that a new relationship between a person and a computer is enhanced and built by such an assistant, the person is protected from having to do this work. The danger of the computer learning what you need and learning it in your language

is that a private relationship is created between you and the computer, where you have a dependency relationship on the agent.

- *Advisor*. An advisory agent is one in which the person does all of the work—the computer only teaches the person or suggests to the person. The kind of agent that teaches, tutors, suggests, or documents can be even more productive than assistant agents, and avoids many of the problems of loss of responsibility feared by the critics of agents. Advisory agents were the first type of agents for which experimental evidence verified that they improved user performance [].

Personalization

Users of conventional applications expect to see exactly the same interface. But “one size fits all” is not appropriate for computer interfaces. Interface agents can learn the individual characteristics, idiosyncracies, unique needs and preferences of a user, and adapt, giving each user a personalized interface.

Until recently, the idea of the computer changing for the individual use to seemed like science fiction. We have changed our intuitions over the last few decades. We've gotten very used to having our mailing list, our vocabulary list, our most commonly used function list, our aliases, all of these personalized, It is becoming simpler to using systems where our actions are remembered. Now many of the popular Internet facilities personalize themselves so that they do not show news that they have shown you before. They show you products that are like products that you have purchased before. They allow you to identify other similarities that you believe will be useful to the computer in extracting information for you. But now, we moving to a time when, models of what a person wants are explicitly and even adaptively created by the computer, or through a mixed initiative.

User modeling

Every computer program has a user model, even if only implicitly. The user model is the program’s expectation of what a user will do, what they need to be told and how they will respond. Historically, it is hard-wired into the program. Command-line interfaces are predicated on the unrealistic assumption that the user knows all commands and can type them in without spelling or punctuation errors.

This implicit model of the user interface coexists with a model of the task and domain. In more sophisticated user interface agents, it is often useful to be make the model more explicit so it can be manipulated and reasoned about. Interface agents may also need to move from the static, implicit models found in conventional programs, to more explicit and dynamic models. In a dynamic user model, the system could notice that the user is using a new feature that he or she never has before, and offer tutorial information.

The most sophisticated way to acquire user models are adaptive, which means they can create dynamic behavior without being pre-programmed. The user interface could learn common

misspellings and typos and remind a user of how to correct them. The interface could offer syntactic or presentation alternatives. The interface could watch user performance and suggest ways in which it might be improved.

The simplest way for an agent to construct a user model is to have explicit interaction with the user. If the user accepts suggestions from the computer such as corrections to misspelled words, those could be added to a user model. More interesting is when the user model is constructed implicitly, by the computer making observations about user behavior that can be used to improve subsequent performance. For example, the user habitually drops an item close to a window instead of in the window and then makes a correction, the computer could at some point surmise that the person's goal is to put things into the window. The computer should either suggest it or do it for them.

Even in something as seemingly simple as a pointing device, models of the user's behavior lead to better performance. The IBM TrackPoint [] software incorporates behavioral, perceptual and motor models that improve accuracy and user satisfaction.

Somehow, overshoot was always a problem when force was applied to position control Sometime in 1987 it was discovered that overshoot could be eliminated by a model that reduced the speed that a person was moving the cursor to a speed that a an eye could track. Where it had appeared that a mechanical control was at fault and a person could not control the stopping of a cursor with a joy stick, in fact, the cause was ignoring of cognitive, perceptual limitations of eye-hand coordination. The TrackPoint incorporated this model of people needing to see what they are looking at and a model of the shakiness of the finger.

It has an approach to allow a person to make a pixel size selections and character size selections that is based on study of how the human hand can control something. It appears that the human hand has too little ability for direct force control to make the accurate selections directly. In Track Point, a consistent, predictable speed is used for very fine motions. Whereas people are not as good at controlling the force they are good at controlling the time at which they do something. So the action of fine motor control is transferred from force to speed. Such a match where the persons physiological and perceptual capabilities are taken into account in an explicit or an internal model in the computer can make a gigantic difference in the performance.

In TrackPoint, these and other user modeling approaches improve the speed in which the person could make a selections by 25%. A large number considering in many cases, in that people can tell the difference when even 5% improvements are made.

This is always a concern that the computer will adapt and the person will adapt simultaneously with expectations of each other. Certainly we even see this in people when we anticipate that a person cannot do something and after some practice they

achieve it and surprise us. In some of the early experiments with Track Point the computer would change and the person would change and it was very easy for the system to get out of phase with the persons expectations of what the computer could do. The adaptive procedures had to be abandoned in some cases because it was too difficult to make the user anticipate the adaptation that the computer would perform. This is a typical control theory problem, termed damping convergence.

While adaptation is not appropriate for every situation, it can be helpful in reducing the brittleness of many systems. Even when adaptive user models are used, it is important to keep in mind that control problems may result, if only temporarily, as the user and the computer adjust their behavior to each other.

Trust

For an agent to be useful, the user must have trust in the agent. Now, when computers routinely lose files or issue cryptic error messages, some may be skeptical about whether computers can really be trusted, but a relationship of trust could be built up out of successful interactions with the agent over a long term. The authority given to the agent could be slowly increased as the user becomes confident in the agent's abilities. Trust in the agent is encouraged by those agents which provide for feedback in their operation and allow the user to influence their operation.

In both industrial design and user interface design, the idea of *affordances* is important [Norman]. Affordance is the idea that the appearance of a tool or agent has connotations that allow a user to infer what the likely function of that component is. A door handle that looks like it should be pulled rather than pushed leads people to expect to perform that operation. The design of the appearance of an agent and the actions that the agent takes should be appropriate to what that agent's function is in the interface. Agents generate expectations about what the interface is supposed to do, and if these expectations are fulfilled or exceeded, the user's trust and confidence in the agent grows.

Feedback

Important to developing confidence in an agent is feedback. The agent should always be able to explain its actions, or have the results of its actions examined and critiqued by the user. If the agent is performing actions in a direct-manipulation interface, the user can use the same interface to examine or edit the objects directly.

Instructibility

When we interact with others, they learn from interactions and improve their interaction with us over time. The same should be true of interface agents. Learning is therefore a critical capability, one we shall return to in detail. Agents should record user actions and try to use them to improve future agent actions. Rather than simply issue commands, as in a conventional user interface, the user may instruct the agent by issuing advice. Advice [McCarthy] is more flexible than commands, since it needn't be as precise, can be given in non-sequential order, and influences the ongoing processes of the agent.

Anthropomorphization

Should an agent be represented by a humanoid character on the screen? The argument for doing so is that it may facilitate the formation of a perceived relationship between the user and the computer. Once there is a relationship a person acts differently towards the computer -- they engage more, they might try harder, they might be able to focus their attention on something the computer is presenting to them. This is most easily seen in how kids react to on-screen characters used as avatars or guides in games and educational software. Research by Nass and Reeves at Stanford [] has shown that even adults cannot help but treat the computer as a social actor and relate to it in human-like ways even though they know it is not human.

The central issue regarding anthropomorphism is whether we are attracting a person's attention to the things they are actually trying to do or distracting them. If the character is entertaining or supportive, then anthropomorphism can be helpful. If on the other hand it takes a person's attention away from the things that they are focusing on, then it's not.

Of course, there is always the danger that people will treat the on-screen character as a real person, or their expectations for the human-like behavior of the system will inevitably be overextended. This is a real danger, and one that we must keep in mind and address as we are designing systems.

However, we believe that this danger should not be regarded as a "show-stopper" for anthropomorphic agents. People are generally pretty good at keeping in mind the differences between computer characters and real humans, and will get better with this as their experience with on-screen agents grows. Our everyday experience with household pets, cartoon characters, video games, etc. shows that we can deal with talking about non-humans in anthropomorphic terms without dangerously overextending our ideas of what they can do.

That leads us the views of the critics.

Critical views

Some critics have argued that since computers are not likely to achieve full human-level intelligence, replacing human decision-making with computer decision making in an interface is always a bad idea. Because the system will likely not be able to predict or understand the user's purposes, it will always be better to provide tools rather than assistance.

This view, which Ben Shneiderman has become famous for arguing in public forums, promotes that direct manipulation is the ideal form of communication with a computer. That view says that the more direct the communication with a computer is, the better the user will be able to use their own judgment, and any indirection or assistance will only screw things up. Unfortunately for Ben, growing success in the interface agent field erodes this argument, as we shall argue in the conclusion of this paper. Some agents can be experimentally shown to improve performance []. And, as interface agents take their places alongside traditional direct manipulation, we increasingly see that we can have both kinds of interfaces.

Jaron Lanier [Lanier] provides a different argument. He argues that as computers get smarter, humans will get lazier, and we will come to rely on computer agents to an extent that is no longer healthy for us. This, too, is indeed a danger, but that charge has been leveled against every labor-saving technology since the wheel. The important question is, does it make us better able to do the things we want to do? Many people feel that their time is overcommitted, and being able to delegate tasks to agents improves the proportion of time they spend on activities of most interest to them. And it is human nature to stretch the boundaries of what is possible by attempting more and more ambitious tasks as soon as simple tasks can be done without thinking about them. Again, striking a balance between convenience and control is the key.

Cognitive style

An often overlooked, if intangible, variable that determines the appropriateness of agent interfaces is the user's cognitive style. Just as some might appreciate the attentiveness of an English butler and others find it intrusive, people vary in the amount of external assistance they wish to receive. This is fine. We don't want to force unwanted "help" on users when they prefer to act for themselves. Thus an essential characteristic for successful agent interfaces is to give the user the user the ability to adjust the degree of initiative that the agent can take. Furthermore, this decision should be dynamically adjustable, since people may want lots of help in one situation and to be left alone in another. Agent interfaces should also always be on the lookout for clues that the user wants more or less interaction.

AI and HCI perspectives on agents

Traditional work in Artificial Intelligence is oriented toward the goal of creating intelligent programs that can solve difficult cognitive problems. Since the focus is on the intelligence of the machine, simple interfaces such as sequential conversational interfaces are assumed. As a strategy for developing intelligent systems, this approach is brittle, because any failure of the machine intelligence to deal with the problem leads to a failure of the system as a whole.

The traditional CHI approach suffers from the opposite problem. Problem-solving intelligence is assumed to reside almost completely with the user's initiative. The job of the interface is to provide the best possible coupling of the computer's data-manipulation ability to the user's intelligence. This approach is also brittle, as the complexity of the problem can easily overwhelm the user's short-term memory, searching and reasoning capabilities.

The intelligent agent approach is to consider the combination of the user and the machine as a single system, the intelligence of the user and that of the system collaborating to solve a problem. The approach is more robust, as the agent can compensate for shortcomings of the user's memory and attention, and the user can compensate for any incompleteness in the agent's ability to solve the task or decide what information is relevant.

Relationships between agent and user

Just like people can have different relationships with each other, there can be many different kinds of relationships between an agent and its user[s]. The most prominent in many envisionments of future agent systems is that the agent acts as a secretary or English butler. Thus the agent anticipates the needs and desires of its client, and acts independently as the client's representative.

While this is certainly a goal, an intermediate step and useful capability in its own right is to imagine the agent-user relationship to be more like a teacher-student relationship. After all, even if we have agents capable of being butlers or secretaries, at first they'll be new on the job and have to learn what it is we want. And before we are willing to trust them completely we might be willing to accept their advice, then act on our own. In the following sections we'll focus on agents that have that kind of relationship to the user.

## Agents that teach you

The application area that has the longest history of use of interface agents is the subfield of Artificial Intelligence called Intelligent Tutoring Systems. Because tutoring systems tend to deal with inexperienced computer users, ease of use in the interface is a prime consideration. Tutoring systems are generally implemented as conversational advisors that decide when and how to present tutoring material to the student, and help the student with problems or answer questions. The agent can be helpful to the user because it can have more knowledge of the material than the beginning student, and it can also encode the experience of teachers in recognizing common problems or errors.

Intelligent Tutoring System examples

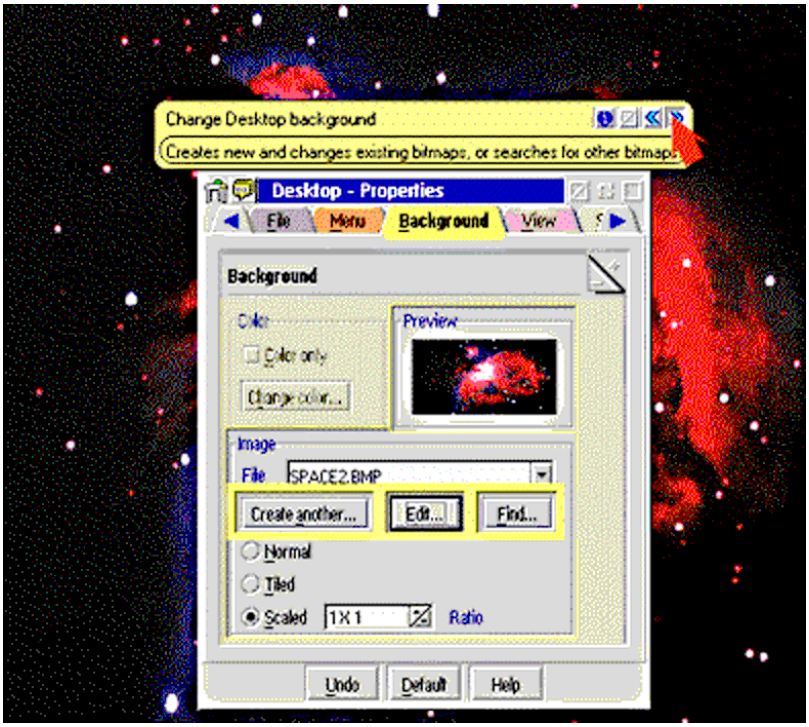
A good example of an early ITS system was Brown's Debuggy []. Debuggy worked in the domain of electronic circuits and monitored the user's questions asking for the values of current, voltage, etc. at each point in the circuit. It tried to infer from that information a user model concerning what the student did or didn't know, the hypotheses that the student was likely to have made, etc. in order to provide intelligent advice. In another early example, Eliot Soloway [] implemented an ITS for introductory programming that analyzed student programs written in Pascal. The tutor had knowledge of typical classes of errors that beginners made, and attempted to analyze a given erroneous program as belonging to one of the categories, which enabled it to make appropriate suggestions to the student. CMU Interbook [] provides a more recent example, where student interaction with a tutorial hypertext is monitored, and inferences regarding what the student has or hasn't seen are used to adjust the presentation to an appropriate level of detail and subject.

In all these projects, modeling the user was crucial for a successful interaction. Conventional software doesn't keep any

significant information about individual users beyond perhaps a simple preference table. Being able to adapt its behavior to the needs and context of different users, or a single user over time, is essential for the user to perceive the interface as being intelligent.

Coach

Coach [Selker] is a teaching agent that formed a user model by recording examples that the user tried in the course of interaction with a programming system, or with an operating system shell.




*Coach*

When the user asks for help, Coach used the user's own examples and the immediate interactive context to make its points, rather than "generic" examples. Like a good teacher, making the material relevant to the user's particular time, interests and situation enhances comprehension and retention.

Critics

Another role that the system can play is that of a critic. The user proceeds by using the direct-manipulation interface as usual and produces a design, and then the system can critique the design. The critic criticizes according to additional knowledge it had that the user might not possess or might not find convenient to apply. A good example of this sort of system is Fischer et al.'s system that critiqued architectural design of kitchens []. For example, an architect usually applies a "rule of thumb" in designing kitchens that the distance of the frequently-traveled triangle between the sink, stove, and countertop be minimized. The agent can inform the user if this informal rule is violated and perhaps suggest

alternative designs. Critics are also sometimes applied in intelligent tutoring systems to critique student work. As in situations where humans criticize each other, particular care must be taken to ensure that the criticism comes at a time and in a manner that is welcome to users rather than offensive.

Wizards

Wizards [] are conversational interfaces oriented towards doing very particular tasks, that present the user with a sequence of questions, the answers to which enable the wizard to perform the task. They are intended to provide a shortcut through the maze of endless menus and dialog boxes that would otherwise be required to perform even the simplest cases of that task. While wizards do certainly perform an assistive function, most present-day wizards lack the user modeling capability, adaptivity, and flexiblility that characterize the intelligent tutoring systems projects. They merely lead the user through a predefined sequence of interactions and that sequence is not substantially affected either by the user's answers, or by the user's past behavior. Nevertheless, if users do like the kind of interaction offered by wizards, there is the potential for adding more context sensitivity and developing them into truly intelligent guides.

## Agents that you teach

Equally important is the relationship in which the user teaches the agent. Agents that can be taught by the user are what we will term *instructible agents* [Lieberman and Maulsby]. The simplest kind of interface agent might already have its goals and methods built-in, and be able to perform its services for the user without any other explicit input from the user. The next step up would be if the user tells the agent explicitly what he or she wants, by communicating commands to the agent or "programming" the agent in some sort of procedural language. This is the kind of interaction that we are most accustomed to with today's computers. We are also already familiar with the problems with this kind of communication – the languages are inflexible and unforgiving, unable to deal with incomplete information, and the programming process is tedious.

Programming by Example

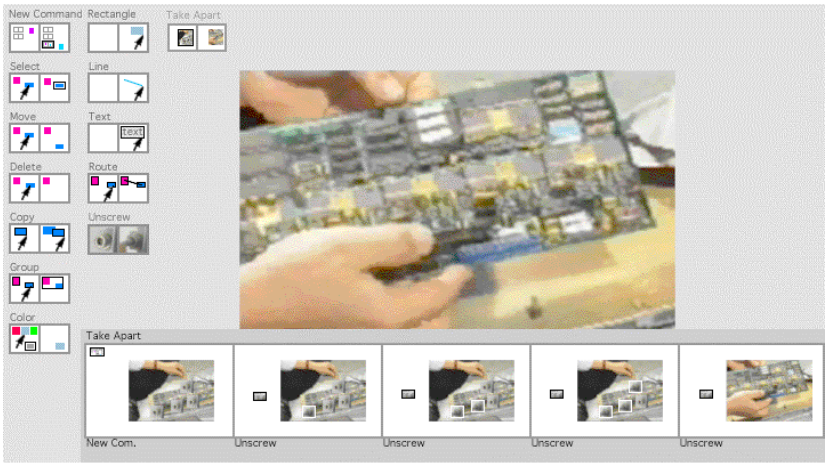
A way out of this dilemma is to give the agent the ability to learn dynamically from interaction with the user. An exciting and underappreciated technique for doing this is called *programming by example* [sometimes also called *programming by demonstration*]. With programming by example, the user uses a conventional direct-manipulation interface such as a text editor, graphic editor, spreadsheet, Web browser, etc. The interface is equipped with a recording mechanism that keeps a history of the user interactions and their results. Then, machine learning techniques are used to *generalize* the program so that it can be used in situations that are analogous to, but not exactly the same as the situation on which the agent was originally taught. The book Watch What I Do, edited by Allen Cyper [Cypher 93], is the definitive reference on the topic, and collects descriptions of more than 15 such systems in a wide variety of applications.

The beauty of programming by example is that it corresponds to a natural "show and tell" kind of interaction common when people try to teach each other procedures. It requires less explicit

instruction by the teacher because the student can observe the actions of the teacher directly. Users can do their work in realistic exmpales and program their agents as a side-effect of the ineraction.

Many users are familiar with “macro recorders” which simply record a sequence of operations and play them back exactly as they were recorded. Users also recognize that such macros are brittle – if anything changes in the data environment between the time the macro is recorded to the time it is played back, the macro is unlikely to work. Programming by example is like “macros on steroids”. By generalizing the procedure [replacing constants with variables, inferring general descriptions of concrete objects], the agent can learn a procedure that is less sensitive to accidental details and more effective in a wide variety of situations.

Mondrian




Mondrian [Lieberman 95] is a graphical editor with an agent that records graphical procedures using programming by example. The user demonstrates a procedure using the graphical editing operations on concrete graphical objects, indicating which ones are to be taken as examples. The system then records a procedure that can be used later with new objects in place of the examples. Mondrian’s learning procedure is similar to what is called in the literature *explanation-based generalization* [], where a tree of dependencies among operations is constructed and the generalizations are propagated through this dependency tree.

Graphical annotation

Mondrian also possessed the ability to learn declaratively as well as procedurally. The user could put *graphical annotations* on images or video frames that created a visual representation of objects that can be manipulated by the graphical editor. These graphical annotations named objects and established a graphical part-whole hierarchy. The user could then operate on the graphical objects and the procedures would be recorded in terms of the relations described by the graphical annotations. In [], the user can teach the agent how to take apart a motor by annotating a video of a human performing the same procedure in the real world. By asking the user to graphically annotate objects, there

was no need to have computer vision procedures for recognizing objects in the scene. Graphical annotation represents a method for the user to use the interactive interface to communicate intent to the agent.

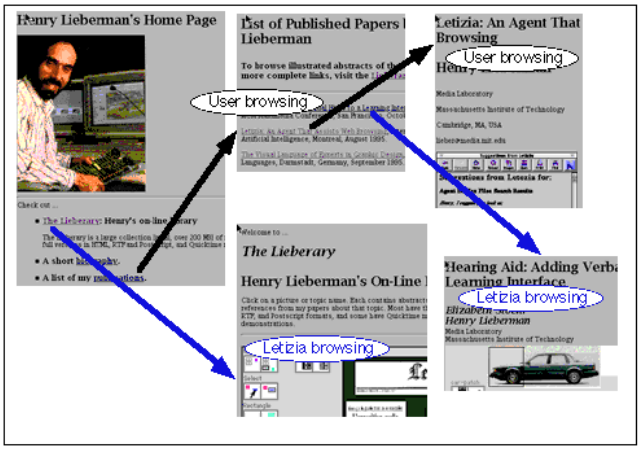
Reconnaissance agents

While agents like Mondrian learn by explicit instruction, another kind of agent learns simply by observation. While programming by example demands active participation from the user, it may also in many situations be desirable for the agent to simply make what inferences it can from observing the user's actions without any additional input. A class of user-interface agents called *reconnaissance agents* learns by watching actions in the user interface and compiling a user profile. The user profile serves to help the agent anticipate what the user might want or need. These agents perform reconnaissance, a look-ahead process that anticipates the user's needs and interests and saves the user time.

A simple example familiar to many is in the Microsoft Office assistant, where user interface actions are fed to a Bayesian network which is used to help the assistant guess which documentation topics are relevant when the user poses a question []. Although many users are turned-off by the too-aggressive pop-up animation of the current incarnation of this agent, the user tracking and heuristic inference of topics are important features that will hopefully be packaged in a more user-friendly manner in the future.

Letizia

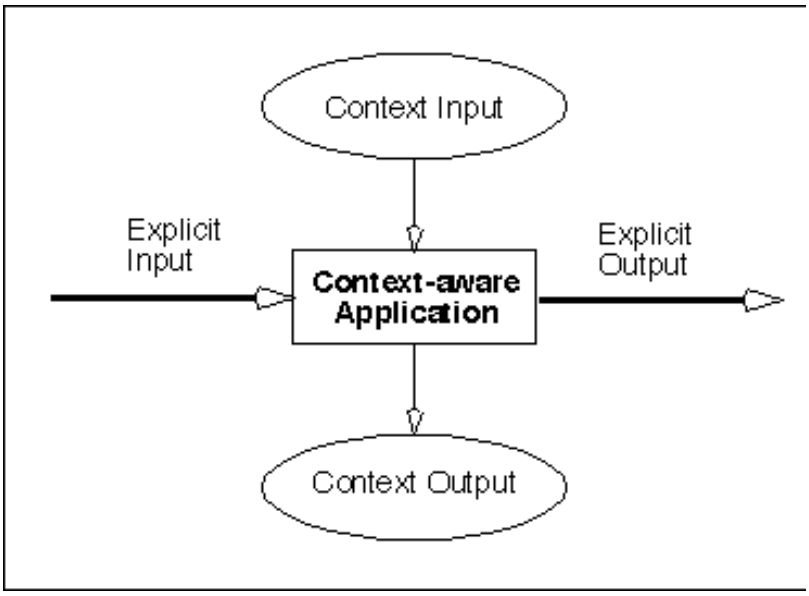
Letizia is a user interface agent to assist a user browsing the Web. It observes the user's Web browsing and reads the pages chosen by the user, analyzing them with a keyword-frequency algorithm similar to that used by search engines to extract topics. As the user is reading a page, Letizia actively searches a breadth-first neighborhood of the page, spiraling out from the link the user is currently looking at. A continuous series of recommendations is generated by filtering the pages searched through the profile created by the user's browsing patterns. The user need give Letizia no additional input beyond clicking links in the browser to view each page. Thus the agent learns in a completely autonomous manner.



## Agents and context

We forsee that a growing role for user interface agents will be in the management of the user's *context*. A lot of what we call intelligence in people is really just being sensitive to context, and taking action appropriate to context. A secretary realizes that his or her boss is busy, and decides whether or not to interrupt the boss for any given event on the basis of that context. A travel agent knows what kinds of tickets are appropriate for vacation travelers versus business travelers. Neither the secretary's boss nor the travel agent need be told the context explicitly. They can infer it from the situation. Part of what makes human agents so useful is that they can figure out what parts of the context are relevant and apply them when needed.

Context is *everything but* what the user tells the system explicitly. A context aware application is one that makes its decisions based on information other than what it gets explicitly from the user.




*Context aware applications go beyond explicit I/O*

Context-aware agents may sense their additional input from the environment using sensory input such as computer vision, speech recognition, electric or magnetic field sensors, global positioning systems, etc. Context can be the history of interaction, or the user model maintained by the agent. A context agent may also affect the environment in ways that go beyond explicitly returning a value or printing a reply as conventional applications do.

Traditional computer science has trouble with the idea of context, since it is oriented towards describing computer programs as functions computing relations between inputs and outputs. The

input and the output must be explicitly given in the formal models. We think it is time to move beyond that and move towards systems that can take account of context as well.

## Integrating agents with conventional user interfaces

Especially for the next few years, agents will be appear increasingly as add-ons to more conventional direct-manipulation applications. Agents will need to use these applications as if they were tools, using the conventional application to access, edit, and display objects in the domain of interest. It is not just a problem of transfer of data from the application to the agent; users wish to retain the ability to use a conventional interface to edit the data interactively themselves, as well as delegate handling the data to an agent.

However, conventional applications, such as text editors, graphic editors, mail systems, CAD systems, etc. have typically been designed to be operated in real time by a live human user, not by an external program, agent or otherwise. What's a poor agent implementor to do?

Currently, the agent implementor is faced with the following choices.

- *Don't rely on an external application*. Implement the interactive interface that edits the data from scratch in order to be able to work with the agent. This is the most reliable option, but obviously the most work for the implementor. If the users already have an application that they like for editing the data, they may be unwilling to give it up.

- *Modify an existing application to work with the agent.* This is next best. However, the agent implementor may not have access to the source code for the application, or it may be difficult to change it and debug changes. New versions of the application will necessitate new modified versions.

- *Use an API* [application programmer's interface]. When the application provides such as interface, this can be a good choice. Unfortunately, many APIs only provide partial access to the capabilities of the application.

- *Slip between two layers of programs.* We call these *intermediaries.* If there is a "natural boundary" between two layers of the internal organization of a program, such that all input or output passes through this layer, that gives an opportunity for an agent to "insert itself" into the workflow. For example, when a computer program collects all characters and mouse input, as does the Coach system, or they collect all HTTP information in a Web application [WBI]. An intermediary can completely take over the look, the feel, the input, the output of any program. In this way we are in a very powerful position, even with unchanged, unintegrated computer systems.

The state of the art of interface agents

As we have noted, the idea of an interface agent as an intelligent assistant dates back to the pioneering visions of early AI researchers such as Selfridge, Minsky and McCarthy in the 50s. The modern “agent movement” picked up steam in the early 90s when people started to realize that some of the visions were now achievable [Kay, Maes], and frustration with conventional direct-manipulation interfaces was growing. But where are we now [as of this writing, June 1999]?

We are seeing agent capabilities slowly make their way into commercial applications. One of the most prominent examples [if perhaps not the best] is the “dancing paper clip” help agent in the Microsoft Office applications. Though many find it annoying because of its distracting and meaningless fidgeting while the computer is idle, Microsoft should at least be credited with the attempt to provide context-sensitive help in a mainstream application based on tracking user interface actions with a Bayesian network [].  More sophisticated help systems such as IBM OS/2’s Coach kept track of your actions, and then used your own examples to explain new procedures and concepts to you in their natural context.

Spelling and grammar checkers have evolved from simple dictionary-lookup to using sophisticated linguistic analysis, predictive and heuristic matching. They employ real-time and in-context delivery of their suggestions, such as the wavy-underlining in Microsoft Word. Predictive typing interfaces such as the Reactive Keyboard [] show quantitative improvements in input productivity and are gaining in popularity.

Personalization is becoming more widespread on Web sites. Book companies like Amazon.com and Barnes and Noble track user buying and browsing patterns, and provide recommendation agents based on collaborative filtering techniques. News and information sites are beginning to tailor their content to personal preferences, geographic location, and past browsing activities so that they don’t show you the same things you’ve already seen. Shopping agents such as Bargain Finder [] and Tête-à-Tête [], automate comparison shopping by accessing merchant Web sites and comparing according to algorithms based on the user’s expressed priorities.

Games are an area where anthropomorphism for agents is so natural that it is rarely even commented upon. Some computer games use computer-controlled players [sometimes called “AIs” in gaming magazines] that play alongside avatars for human players. Tamagotchi-like toys are becoming more sophisticated in their artificial-life simulations. So-called “bots”, automated players on MUDs, even while simple, can be effective in holding players’ attention. The Eliza-like Julia [] ,had extensive knowledge of soccer, and played a flirting female that captivated potential suitors for considerable lengths of time before they discovered its true nature.

Large, tough problems like optical character recognition and speech recognition, which used to be considered substantial

subfields of AI, are rapidly reaching the point where they will be taken for granted in the interface. OCR programs that separate text recognition from graphics, recognize page formatting, scan business cards regardless of font or layout, are uses of intelligence in software that have become almost invisible. These are instances of the well known “disappearing AI” problem, where when AI achieves a goal it gets redefined as being part of its application area, so that AI as a field is not credited with the success.

All of these examples show that the idea of interface agents is not so crazy as it may have seemed in the 70s and the 80s. But for all these examples, we still have not reached the point where agent interfaces become a matter of course. Industry is sticking its toes into the waters of agent applications, but has yet to dive into the pool. Even newly-introduced programs still mire the user in a thicket of dialog boxes, endless options, file selections and icon manipulation. We keep filling out the same Web site forms over and over and the programs don’t remember what we did with them and improve their behavior. We are still at the mercy of one-size-fits-all user interfaces. The behavior of programs improves, if at all, only slowly with new releases that come months or years apart. But as we have shown, user interface agents point a way out of this mess, and it will be the task of the early years of the next millennium to make a new generation of truly agent-enabled applications. We expect that, probably within a decade, it will be hard to look back and remember a time we didn’t view having knowledge and learning in user interfaces as crucial to making computers usable.