AI Magazine Volume 22 Number 4 (2001) (© AAAI)

Interface Agents in Model World Environments

Robert St. Amant and R. Michael Young

■ Choosing an environment is an important decision for agent developers. A key issue in this decision is whether the environment will provide realistic problems for the agent to solve, in the sense that the problems are true to the issues that arise in addressing a particular research question. In addition to realism, other important issues include how tractable problems are that can be formulated in the environment, how easy agent performance can be measured, and whether the environment can be customized or extended for specific research questions. In the ideal environment, researchers can pose realistic but tractable problems to an agent, measure and evaluate its performance, and iteratively rework the environment to explore increasingly ambitious questions, all at a reasonable cost in time and effort. As might be expected, trade-offs dominate the suitability of an environment; however, we have found that the modern graphic user interface offers a good balance among these trade-offs. This article takes a brief tour of agent research in the user interface, showing how significant questions related to vision, planning, learning, cognition, and communication are currently being addressed.

Choosing an environment is an important decision for agent developers. A key issue in this decision is whether the environment will provide realistic problems for the agent to solve, in the sense that the problems are true to the issues that arise in addressing a particular research question. For robotics researchers, realism might derive from ambiguous sensing information in a navigation environment or from constrained manipulation in a close physical environment. In practical planning research, a realistic simulation might pose problems of fluctuating resources and dynamic changes in the environment (Cohen et al. 1989; Hanks, Pollack, and Cohen 1993). In other simulation work, realism can be interpreted literally: How closely does the simulation approach a natural target environment, which is presumably too complex, too expensive, or inaccessible for use during development? Often, we evaluate the realism of an environment by whether the problems that arise under controlled conditions are those that an intelligent agent (human or artificial) might be expected to encounter and solve in the real world.

Beyond the question of realism, a number of other, related questions are important in the selection of a problem-solving environment for an agent.

Are the Problems Tractable?

The opposite side of the realism coin is tractability. Problems should be realistic but not too difficult for an agent; progress would otherwise be hindered. Thus, some RoboCup competitions use an enclosed field and colored goals (Asada et al. 2000); in some office-robot environments, the robots can rely on artificial navigation cues and target object markings.

Tractability is an issue for software agents as well. In one well-known example, researchers found the TILEWORLD environment too complex for a planned set of experiments, forcing a simplification of the environment to a grid world without tiles (Kinny and Georgeff 1991).

Can the Environment Be Tailored to a Given Research Question?

Even if an environment can, in principle, pose realistic, tractable problems to an agent, these problems can inextricably be tied up with other problems not under consideration. For example, a classical planning researcher might prefer a static domain description to a dynamic simulation if the simulation also requires that the agent process sensor data, communicate with other agents, and so forth.

Can Performance Be Evaluated?

Another important issue for environments is what Hanks, Pollack, and Cohen (1993, p. 18) describe as "the tension between realism and the possibility of experimental control." Rigorous evaluation of an agent's performance requires experimentation, usually with the need for instrumentation and experimental control of the environment.

Is the Environment Extensible? Can New Problems Be Easily Defined?

Few environments are so general that they exercise all agent capabilities of interest. Extensibility to new classes of problems, as well as to new classes of agents, can influence the acceptance of an environment by developers.

In the ideal environment, researchers can pose realistic but tractable problems to an agent, measure and evaluate its performance, and iteratively rework the environment to explore more ambitious questions, all at a reasonable cost in time and effort.

Unfortunately, trade-offs dominate the suitability of an environment. A simulation might be tailored to, say, a class of learning algorithms, able to generate a broad set of difficult but not impossible problems and appropriately instrumented for performance evaluation. If these problems are all arbitrarily generated abstractions, however, with no relationship to real-world problems, then the simulation will not reflect the environments in which we would eventually expect to see an agent perform. Analogously, a real environment, either a natural one such as an office space or an artificial one such as the World Wide Web, can be available at low cost but might not lend itself to experimental control, or it might demand too much of an agent's sensing or representation abilities.

In recent work, we have found that the modern graphic user interface offers a good balance among these trade-offs. A *graphic user interface* is an interactive visual environment that supports direct manipulation of the tools and objects associated with a problem domain (Lewis 1998; Shneiderman 1998). Work in human-computer interaction (HCI) over the past two decades has identified many factors that contribute to the usability of interactive software, including the continuous visibility of objects in the environment, the directness of the mapping between interface objects and real-world objects, the simple command syntax, and the richness of the representational forms that can be conveyed through a visual medium (Norman 1991; Shneiderman 1998). In addition to benefiting human users, these factors make the user interface attractive as a general-purpose research environment.

This article is concerned with systems that accommodate and exploit the properties of the graphic user interface as an environment. Many of these systems are interface agents, in the sense defined by Maes (Shneiderman and Maes 1997, p. 53): "An (interface) agent basically interacts with the application just like you interact with the application." Although nowadays the term *interface agent* encompasses all types of agents that interact with users, even those that don't depend at all on the characteristics of the interface medium, we focus on a more specialized interpretation of the term. We mean agents that interact with the objects in a direct-manipulation graphic user interface, or in other words, softbots whose sensors and effectors are the input and output capabilities of the interface (Lieberman 1998). Other efforts we describe, not necessarily related to agents, operate within the same constraints, treating the user interface as an environment rather than only a communication medium.

The graphic user interface provides a problem-solving environment with attractive features for AI researchers.

Realism: The problems that users solve using interactive software are by definition of interest in the real world. Interface agents can be applied to the same tasks, using the same tools.

Tractability: In the user interface, the "physics" of interaction is far simpler, and tasks are far more structured than in the natural world. Contrast drawing concentric circles in an illustration package with the visual coordination required of pencil and paper manipulation. The degree to which the user interface facilitates appropriate action can lead to sophisticated agent-environment interactions at a relatively low computational cost.

Measurement and control: The user interface is designed for stability and user control; it is usually a small matter to tailor it to the needs of an experiment. Applications are usually also designed for efficient presentation of relevant information (Woods 1991), making instrumentation straightforward.

Extensibility: Because of the breadth and richness of problems solved in interactive software, it is relatively easy to identify new problems for an agent to solve. Furthermore, user interface design relies heavily on transfer of existing knowledge and procedures from one application to the next, structure that can also be exploited by an agent.

Research in the user interface is a microcosm of research in AI, encompassing vision, planning, learning, cognition, and communication, among other areas. This article takes a brief tour of these areas. We begin by introducing the properties of the user interface as a problem-solving environment, then continue with a discussion of representative systems that capitalize on these properties. Intelligent user interface research in this vein draws on AI, cognitive science, and human-computer interaction; the projects we discuss have made contributions in all these areas.

User Interface Design

Modern graphic user interfaces rely on a pervasive metaphor described by Hutchins (1989) as the model world metaphor. In contrast to the conversation metaphor, in which human-computer interaction is likened to a dialog about a task, the model-world metaphor involves the creation of an artificial environment, a model of the real world in which a user or agent directly manipulates objects to accomplish the task. Instead of talking about activities, one simply acts.

It turns out that many of the features developed to make model-world interfaces effective for human users also make them tractable for AI systems. We find a striking correspondence between user interface design guidelines, on the one hand, and assumptions that AI systems commonly make about their environments, on the other. For each guideline that follows, we discuss its properties and its correspondence to assumptions common in AI systems. These guidelines do not apply universally but describe the most common applications in use today.

Accessible State Information

One of the basic design guidelines for direct-manipulation systems is that relevant objects and actions should be continuously visible (Shneiderman 1998). In other words, a good interface makes available all information relevant to the decision-making process. The implication for an AI system using a well-designed user interface is that information relevant to a given decision will almost always be accessible in the current state in interface. If it is not immediately accessible, it will be available through simple observation strategies (Dix 1993; St. Amant and Zettlemoyer 2000), limiting the amount of inference required of an agent in interacting with its environment.

Environmental Stability

An important assumption for many AI systems is that the agent's environment does not change except because of the actions of the agent. In the user interface, two design guidelines promote a steady-state environment, commonly described as (1) *perceived stability* and (2) *user control*. The perceived stability guideline entails that the user should be able to depend on stability (for example, in visual layout) as time passes (Apple 1992). Further, many designers hold that users should exercise nearly complete control over the user interface, which is one of the defining characteristics of direct manipulation (Shneiderman 1998).

In practice, this degree of control and stability is not always present. Exogenous events might occur in some circumstances (for example, e-mail arrives, online news pages are updated, the status of a network connection changes, the system clock marches forward). In applications for collaborative work, as with games, exogenous events are even integral to their functions. Nevertheless, in most single-user productivity applications, event occurrences are of well-defined types and can be dealt with by simple, fixed strategies, as with accessing state information; often such events can safely be ignored.

Discrete States and Actions

Newell and Simon's (1972) early work on human problem solving has been as influential in human-computer interaction research as in AI. One important result of this influence is a correspondence between the objects and actions of a model world and the symbol structures and elementary information processes that can be used to compose a problem space. Specifically, objects, their properties, and relationships to other objects in the user interface tend to be discrete and easily described in symbolic terms. Buttons and other controls have clear visual boundaries; they are distinct and differentiated from one another. Actions are

Research in the user interface is a microcosm of research in AI, encompassing vision, planning, learning, cognition, and communication, among other areas.

also discrete, and this property extends to their preconditions as well as their effects.

When the mouse pointer moves over or selects some object, its properties change to reflect the action: A button simulates being depressed on selection; a scroll arrow becomes highlighted for the duration of its activation. One straightforward interpretation is that the objects behave in such a way as to provide visual information about the preconditions of relevant actions. Objects similarly change state visually to reflect the effects of actions.

Most applications follow common visual guidelines for designing controls in the user interface (for example, Mullet and Sano [1995].) Again, these guidelines are not universal, as an examination of realistic games, virtual reality environments, and gesture-based personal digital assistant interfaces shows, but hold for a wide variety of applications nonetheless.

Deterministic Actions

Actions are furthermore deterministic. From a user interface design perspective, this point is reflected in a variety of guidelines, most clearly in the property of consistency, especially procedural consistency (Apple 1992). Consistency can be described in terms of matching users' expectations: The same action taken under a well-defined set of conditions should always produce the same effect. The implication for an AI system is that contingency planning, reasoning about uncertainty, and comparable activities are not needed in large classes of applications. In combination with the accessibility and stability of an interface, this property allows that an AI system can expect to have full state information at the execution of an action and that its effect will be predictable.

Decomposable Problems

User interfaces rely heavily on a divide-and-conquer approach to problem solving. As the complexity of an application grows, it soon becomes impractical to have all its functions available at a single mouse click. In a word processor, for example, changing the properties of a text selection can cause a new dialog window to pop up with a new set of controls to manipulate. The user can make the desired changes, usually relying on local information and actions and, for the moment, ignoring the global context.

Software applications are deliberately structured to support such problem solving. A common guideline is that tasks should be cleanly separated to allow their independent execution (Hix and Hartson 1993). This separation facilitates an incremental style of problem solving in which one task is brought to completion before the next is begun.

Bottom-Up Solutions

In a discussion of guidelines for interactive systems that support hierarchical task execution, Hix and Hartson (1993) advocate that goals be satisfiable in bottom-up fashion, with intermediate solutions constructed and combined opportunistically at higher levels. This rule, that is, what Hix and Hartson advocate, can be seen in the prescription of nonmodal behavior for user interfaces. For example, in a drawing application, graphic objects can be selected in any order for a group operation, and in a word processor, the typeface and size properties of text can be modified independently, in either order. In most applications that allow wide user latitude in selecting operations (wizards and some drawing applications being notable exceptions), the system does not enter into states that require a strict sequential ordering of solution components.

This discussion might continue at length. Solutions for problems in the user interface tend to be short, with regular feedback about progress. Actions are generally reversible (providing what user interface developers call *forgiveness* in an environment), so that simple mistakes can easily be undone. Forcing functions are often implemented to guard against potentially catastrophic effects (for example, "If you really want to reformat your hard drive, press OK, otherwise Cancel"). These and related prescriptions fill the HCI literature on informal design guidelines (for example, Hix and Hartson 1993; Preece et al. 1994; Shneiderman 1998) and user models for HCI (Dix 1993; Dix et al. 1998; Runciman and Hammond 1986; Young, Green, and Simon 1989).

Problem Solving

A number of cognitive modeling systems have been developed to solve problems in the user interface. These models support accurate simulation of human performance in a software environment at the cognitive and often perceptual-motor levels. To some extent, these models also exploit the properties of the user interface that facilitate interaction. It turns out that the conditions under which a cognitive model can effectively reproduce human behavior are well matched by certain properties of the user interface, as described earlier.

Systems in this area range from detailed, general-purpose cognitive models to task-analysis models specialized for tasks in the user interface. Prominent examples include the following:

SOAR is a general architecture designed to model human cognition (Newell 1990). It has been applied to a broad range of activities, including the modeling of complex interaction tasks (John and Vera 1992).

ACT-R (Anderson and Lebiere 1998) is a production system theory of human knowledge and behavior. ACT-R has modeled restricted interface tasks such as menu selection (Byrne and Anderson 2001) but has also produced models for much more complex domains, such as experiment design (Schunn and Anderson 1998).

EPIC (executive-process interactive control) is a detailed cognitive model of low-level interaction with a computer interface, including visual, aural, speech, and motor behavior (Kieras and Meyer 1997).

LICAI, a linked model of comprehension-based action planning and instruction taking (Kitajima and Polson 1997, 1995) is based on Kintsch's (1998) construction integration theory of human cognition. LICAI simulates the execution of exploratory tasks in applications with graphic user interfaces.

GLEAN (GOMS language evaluation and analysis) is an engineering tool for simulation of task- analysis models based on the GOMS (goals-operations-methods-selection) rule paradigm (Kieras 1999; Newell and Simon 1972). GLEAN is part of a long history of research toward the goal of building engineering models that reflect user behavior to improve interface development and evaluation (John and Kieras, 1996a, 1996b; Kieras 1988).

Even a brief discussion of these systems and their differences is beyond the scope of this article. It is instructive, however, to examine representative examples of the development and evaluation of these models in the interface. Our discussion is not intended to show that cognitive modeling results are limited in scope by their application to problems in the user interface but, rather, that the user interface provides a congenial environment for exploring and evaluating the behavior of a cognitive model.

In our first example, in LICAI, the user is shown a two-dimensional line plot showing predicted versus observed response times for data from an experiment. The user must generate the line plot by generating a sequence of actions within the CRICKET GRAPH package on the Apple MACINTOSH (Kitajima and Polson 1995). This process involves breaking down the task into subtasks, such as creating a default graph and modifying it to match the target display. Primitive actions include reading labels, selecting data columns, and pressing buttons. The goal of the model is to simulate a skilled user in carrying out this task, even to the level of reproducing action slips.

This modeling task, although complex, depends on a number of simplifying features of the user interface, beyond those discussed earlier. Actions are drawn from a large but fixed set. Interface objects are represented internally as objects with small sets of named attributes. Attributes represent static properties of an object, such as its type, location, color, and shape, or dynamic properties that describe its appearance and state, such as whether a check box is selected, a button depressed, or a text box activated. Simple spatial relationships between objects are represented as well in symbolic form. All this information can easily and unambiguously be extracted from the interface. The result is a rich but tractable environment for modeling and analyzing human problem solving.

The second example, in SOAR, departs from many approaches to modeling in the interface by taking on interaction within highly dynamic, complex military training simulations. To serve as effective team members and opponents within the simulation environment, the SOAR agents are designed to match human performance in a challenging context. In these systems, a SOAR agent must control complex military vehicles (for example, fighter aircraft) and move through the environment, building internal maps, interacting with environmental features and objects, and dealing with opponents; the agents do not have any special advantage in information or action with respect to human users. Simulations such as these can be much more difficult to interact with than conventional applications. Common guidelines for building word processors and drawing packages are here reversed: Important information can be hidden from the user, the environment can change unpredictably, actions are not always successful, and catastrophic actions are rarely reversible.

Nevertheless, the agents still gain some advantage, even in this challenging environment, from properties of the interface that facilitate interaction. Objects are explicitly defined by their type; actions are similarly explicit. Problems to be solved remain largely decomposable, with short, bottom-up sequences providing effective behavior.

In both these examples, as in many other approaches to modeling in the user interface, we find rich opportunities to explore and better understand intelligent problem-solving behavior. This research is driven by questions about human cognition but is also made feasible, often in significant ways, by the properties of the user interface, as discussed in the previous section.

Perception

Autonomous systems that interact with applications, such as the previous modeling systems, generally do not interact with off-the-shelf applications directly through the user interface in the way that ordinary users do. Instead, they interact with an application program interface, programmatically (Laird 2000), sometimes with a simulation of the application (Kieras 1999), or even with a specialized user interface management system extended to support cognitive modeling functions (Ritter et al. 2000). Of these, the last approach is the most flexible and most easily lends itself to model evaluation in ecologically valid scenarios. Ritter et al.'s (2000) work is some of the most mature, having resulted in simulated perception and motor abilities for SOAR and ACT-R as well as interfaces to systems for air traffic control, puzzle solving, and graphics.

VisMap supports the control of an application through its graphic user interface, through the same medium as ordinary users.

Recently, attention has turned to the possibility that an interface agent (or a cognitive model) might process the visual information presented by an interface. After all, the user interface is tailored to human visual processing; giving a system such capabilities might improve its performance as either an agent or a cognitive model. Building a vision system for the user interface might seem to be an impractical proposition, but interfaces are built on a wide range of regularities that make the problem tractable.

Shape: Manipulable objects such as buttons and text boxes are usually rectangular. Even oddly shaped icons are often selectable by a click within their rectangular bounding box.

Orientation: Objects are generally two dimensional and appear in only one orientation, making template-based pattern matching straightforward.

Color: Objects tend to be uniformly colored, with distinct rectilinear borders separating them from one another and the background.

Layout: Except for icons in specific situations and domain-specific graphics, objects are generally arranged in columns, rows, grids, or other simple symmetric structures. Specific types of objects, such as OK and Cancel buttons in a dialog, are even placed in standard positions, making them easy to find.

Annotation: Buttons and menu items are labeled with their functions; simple inspection is often enough to establish the correct interpretation of objects in the environment.

Further, the dynamic behavior of the user interface provides visual cues for appropriate action. Visual highlighting indicates that a type-in field has the current focus or that a specific button is the default action. The selection of a window changes the appearance of its title bar. In general, when implicit static structure is insufficient for correct interpretation of information, explicit dynamic cues are supplied.

The VisMap (visual manipulation) system exploits such regularities to generate input for an external controller using visual processing (St. Amant and Zettlemoyer 2000; Zettlemoyer and St. Amant 1999; Zettlemoyer, St. Amant, and Dulberg 1999). VisMap supports the control of an application through its graphic user interface, through the same medium as ordinary users. Input to VisMap is a pixel-level representation of the display; output is a description of the high-level user interface components. Processing is application and domain independent and follows the general functional constraints of biological visual processing, although it departs significantly at a detailed level.

Visual processing follows a three-stage process of (1) segmentation, (2) feature computation, and (3) interpretation. The segmentation algorithm begins at an arbitrarily selected pixel and incrementally generates like-colored regions with eight-neighbor connectivity. The process can be limited to a specific region and can be reactivated if you want to complete the process or if parts of the screen buffer change. The result is a set of internally homogeneous, often rectilinear pixel groups. These pixel groups pass through a bottom-up feature computation stage. Within-group features include width, height, color, bounding box points, area, and perimeter. Between-group features capture simple spatial relationships between pixel groups, such as containment. Finally, the top-down application of rules imposes an interpretation of pixel groups and their combination as objects. Each rule corresponds to a different type of interface object; each can be considered a specialized implementation of a template for an object. For an object on the screen to be recognized, a template must exist that describes how specific components must be present and in the correct relative positions. Templates recognize right angles; rectangles; circles; check marks; and up, down, left, and right triangles. These simple templates are components of more complex templates that identify buttons, windows, check boxes, radio buttons, list boxes, vertical and horizontal scroll bars, plus more specialized objects.

The process recognizes all familiar user interface controls in Microsoft WINDOWS, such as buttons, scroll bars (including the scroll box, scroll arrows, and background regions), list boxes, menu items, check boxes, radio buttons, and application windows. It can even parse text, doing a simple form of optical character recognition, for the standard system typeface.

An agent based on VISMAP or a comparable perception substrate gains access to information through the interface that is not always available programmatically. Augmented by an effector module that generates mouse gestures and keyboard input, such an agent can perform a broad range of interface tasks with only limited reliance on application-specific mechanisms. Agents based on VISMAP can read and edit text in a word processor (St. Amant and Riedl 2001), draw and manipulate pictures in an illustration package (St. Amant and Zettlemoyer 2000), and perform intelligent interpretation of user gestures (Dulberg, St. Amant, and Zettlemoyer 1999). Figure 1 shows the user interface in an intermediate stage of a VISMAP agent's activities. The agent has strong limitations; for example, it cannot recognize many of the objects it can draw, lacking the appropriate visual templates. It can nevertheless manipulate such objects in standard ways by recognizing and interacting with their control points and selection boxes. In general, systems such as VISMAP allow agent developers to explore issues of perception and problem solving in a simple but challenging environment (figure 1).

Planning, Acting, and Narrative

Some interfaces are explicitly designed and accessed as (virtual) worlds in which the user forms goals and performs actions in their pursuit. These interfaces contrast with conventional productivity applications by relaxing some—but not all—of the interface constraints that facilitate problem solving, as discussed earlier. In most three-dimensional (3D) interactive computer games, the primary interface for the system is a first-person point of view through which the user peers into a world with which he/she interacts by controlling his/her own virtual embodiment, or character. There are a number of reasons why interactive gaming environments of this type provide interesting interface models for AI researchers. First, the worlds that games portray are often quite similar to the rich dynamic environments modeled by conventional AI applications. Second, unlike many real-world domains, gaming environments are entirely accessible; that is, agents operating in these worlds typically have broad and accurate access to state information that characterizes their environments. Finally, it is now commonplace for these gaming systems to be highly extensible. Commercial gaming environments typically include editing tools, scripting language development environments, and other components that allow the flexible and straightforward extension of the system for access by intelligent agents. Through their scripting languages, the internal run-time state of these environments is also accessible. Further, use of these environments typically bears a direct relation to real-world tasks (that is, either the worlds they model are direct simulations of real-world environments, such as in educational or training simulations, or the virtual worlds are their own models, as is often the case for entertainment applications).

One particularly exciting aspect of the 3D interactive gaming interface is the central role that the presentation and comprehension of action sequences plays in its use. Users of gaming systems simultaneously observe actions taken by other characters in the world and perform actions within the world themselves, and the synergy between these actions is at the core of the user's experience. In this sense, the 3D gaming environment presents an interface that is inherently narrative in nature. Stories are played out by the collective actions of the user(s) and the system, and the system's effective creation of action sequences determines, to a great degree, the level of engagement in the story world felt by each user.

For the designer of a narrative-oriented system that allows substantive user interaction, the greatest design challenge revolves around the balance between coherence and control. For a user to understand an action sequence as an unfolding story line, he/she must be able to view the actions as a coherent plot where one action is related to another in a temporal sequence based on some rules of narrative (for example, Balkanski [1993]), much like adjacent utterances in a natural language discourse must coherently be related to one another to be comprehensible (Hobbs 1985; Mann and Thompson 1992).

For a user to feel immersed within a virtual environment, he/she must feel that he/she has substantive control of his/her character as the environment's story line unfolds around him/her. However, control and coherence are in natural opposition to one another in interactive systems with a strong narrative orientation. If an interface's design removes all control from the user, the resulting system is reduced to conventional narrative forms such as litera-

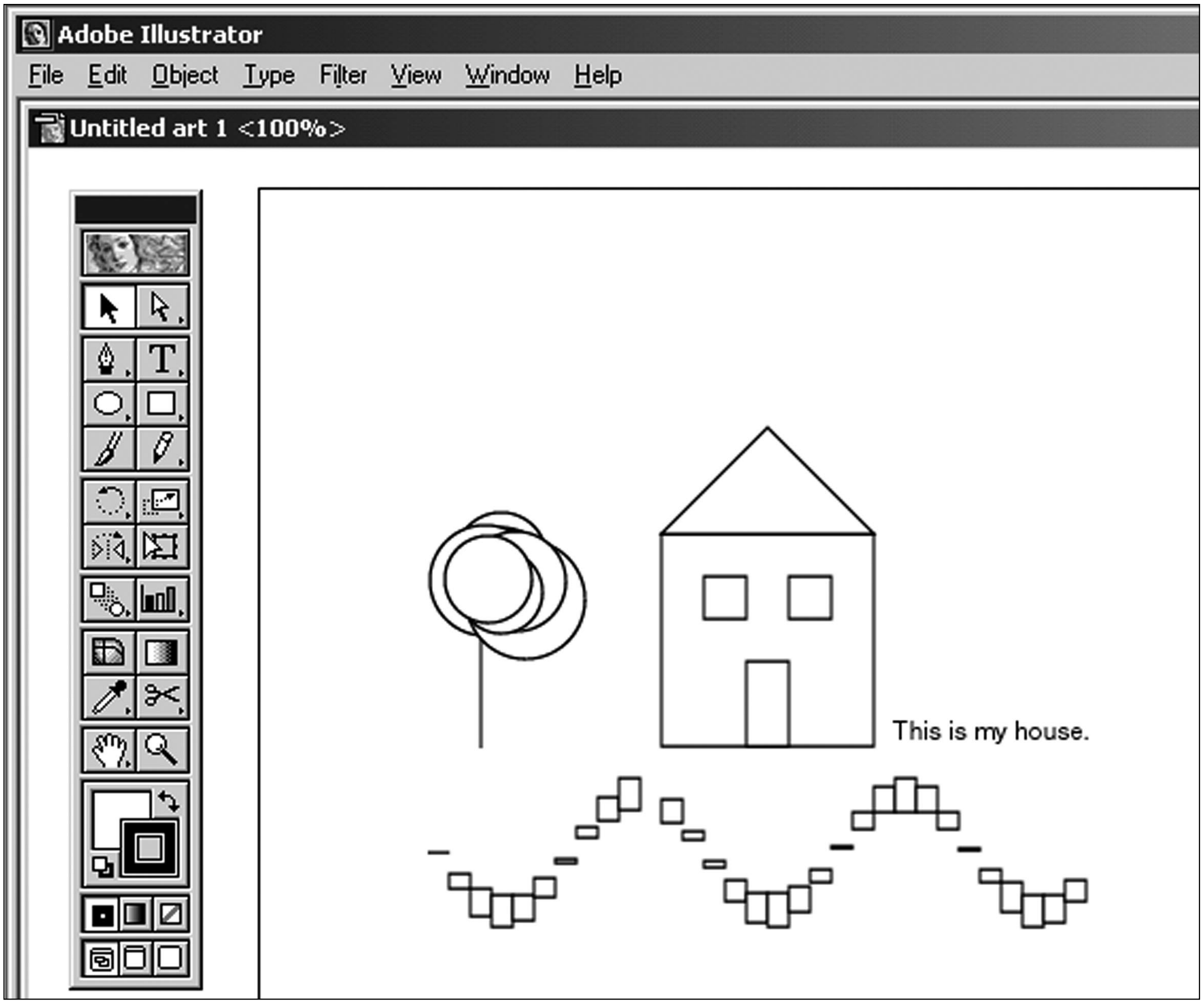


Figure 1. A VisMap Agent's Results in Adobe ILLUSTRATOR.

ture or film. If a design provides the user with complete control, the narrative coherence of a user's interaction is limited by his/her own knowledge and abilities because without an understanding of the unfolding story world around him/her, a user's actions can unintentionally interfere with the story's structure.

Because the problem of determining the proper balance between control and coherence is an open research issue, most interactive narrative-based systems have taken a middle ground, specifying at design time sets of actions from which the user can choose at a fixed set of points throughout a narrative. The resulting collection of narrative paths is carefully designed so that each path provides the user with an interesting narrative experience. This approach, of course, limits the number and type of stories that can be told.[1]

Several researchers have integrated AI techniques into multiuser gaming environments in both academic (Cavazza and Palmer 2000; Laird 1999) and commercial (Davis 2000; Stout 1996) settings. The MIMESIS Project takes the notion of narrative as its central research issue. The goal of the MIMESIS system is to provide a dynamically created, interactive, narrative-based virtual environment that produces in its users the same cognitive and emotional responses as conventional film and other narrative media. In contrast to the design of most computer gaming systems and many agent-based architectures, MIMESIS makes use of a centralized controller to coordinate all the activi-

Figure 2. A Scene in MIMESIS*.*

ties in the virtual world not initiated by the user. Actions available in a gaming environment are represented declaratively and accessed by a planning system to create, monitor, and maintain compelling story lines in the face of user activity. Although many virtual reality systems strive to create systems that are indistinguishable from the reality that they model, the MIMESIS system intentionally seeks to provide the user with an illusion of reality (Thomas and Johnston 1981), one which, when interacted with, provides more engaging and comprehensible interaction than the unstructured activity often found in exploratory virtual reality systems. Figure 2 shows a scene from a MIMESIS story line.

To create story lines that are at once coherent and engaging, the MIMESIS system utilizes recent models of plans and planning to represent the hierarchical and causal nature of narratives identified by narrative theorists. Specifically, MIMESIS uses the DPOCL planner (Young, Pollack, and Moore 1994), a partial-order, causal link planner that directly integrates hierarchical reasoning into the causal link framework. Two aspects of DPOCL's action representation lend themselves to the representation of interactive narrative structure. First, the plan representations that are used contain a rich, formal representation of the causal structure of a plan. Each causal dependency between goal, precondition, and effect in the plan is carefully delineated during plan construction. Second, the plan construction process used is one of search through the space of all possible plans rather than the incremental construction of a single plan. Consequently, the system can characterize a plan relative to the broader context in which it occurs. Both of these features are central to the capacity to create and maintain narrative structure in an interactive system.

At its core, the MIMESIS controller is broken down into two distinct components:

First, the DPOCL planner is used to form plans for user and system interaction that contain such interesting and compelling narrative structure as rising action, balanced conflict between protagonist and antagonist, suspense, and foreshadowing.

Second, an action mediator runs as an embedded process within the game server itself, detecting user activities that deviate from the planned narrative (called *exceptions*) and deciding how to respond.

Exceptions are dealt with by MIMESIS in one of two ways. The most straightforward way is by intervention. Because all a user's actions in the environment are checked by MIMESIS before they are executed by the gaming system, it is MIMESIS itself that determines whether an action succeeds or fails. Typically, the success or failure of an action within a virtual environment is determined by software that approximates the rules of the underlying story world (for example, firing a pistol directly at an attacking monster results in the creature's timely death). However, when a user's action would violate one of the narrative plan's constraints, the system can intervene, causing the action to fail in some subtle or unanticipated manner.

The second response to an exception is to adjust the narrative structure of the plan to accommodate the new activity of the user. The resolution of the conflict caused by the exception can involve only minor restructuring of the plan, for example, selecting a different but compatible location for an event when the user takes an unexpected turn down a new path. Alternatively, this conflict resolution might involve more substantive changes to the plan, for example, should a user stumble on the key to a mystery early in a narrative or unintentionally destroy a device required to rescue a narrative's central character.

MIMESIS differs from work discussed in previous sections by leveraging only a few of the facilitating properties of user interfaces, such as the discreteness of states and actions and the accessibility of state information. By carefully managing other such properties, such as the predictability of actions and interleaving of plans, the system maintains an effective balance between realism and tractability, to the extent of being able to rely on many classical planning assumptions about the environment.

Situated Action

The work described to this point has taken a traditional approach to AI, emphasizing the role of symbolic knowledge representation, planning, learning, and so on. An alternative perspective that has sparked some interest in planning and robotics focuses on the situated nature of human activity (Agre 1997; Agre and Chapman 1987; Agre and Rosenschein 1996; Arkin 1998; Murphy 1999). Agre succinctly characterizes the situated-action perspective by describing three ways in which it contrasts with the traditional view of intelligent behavior as planning. First, people engage in activity as an extended form of improvisation rather than build detailed plans in advance. Second, the reason that everyday activity appears to be organized is not because of plans that people impose on the world but rather because activity tends to follow routines, which lends order and coherence to interactions with the world. Third, this world is fundamentally benign; much of the world is structured by design to support the activities people engage in. Under the situated view, plans are one resource people draw on in interacting with the world but not at all the only one (Agre and Chapman 1990; Suchman 1987).

This view has a natural appeal in human-computer–interaction circles (Flach et al. 1995; Gaver 1991; Kirlik, Miller, and Jagacinski 1993; Norman 1999, 1991; St. Amant 1999). User interface designers can build interaction features into the environment that facilitate specific types of behavior by matching and constraining known properties of human perception, action, and cognition. Taking this observation a step further, agent developers can build autonomous systems that rely on these same interaction features. VISMAP and the other systems discussed here do this to some extent. One of the earliest and best examples, though, is the series of TRIGGERS systems (Potter 1999, 1993).

TRIGGERS explores the idea of interprocess communication using device-level operations, in particular, through pixel data access. Using pixel data access, a system reads and processes the contents of the display buffer to retrieve information about the state of another application. This information is processed using search for pixel patterns and simulated mouse and keyboard actions combined in simple reactive rules. This kind of low-level information can sometimes support surprisingly flexible operations; TRIGGERS can perform such activities as graphic search and replace in drawing programs, circling of negative numbers in a spreadsheet, and converting of file types in the Macintosh user interface.

A TRIGGERS program is composed of an ordered set of rules, or triggers. The controller evaluates the precondition list of each trigger in turn. When a trigger fires, its action sequence is executed, and the process returns to the first trigger in the set. The objects referenced in the set of triggers are markers, rectangles, and flags. *Markers* represent specific points on the screen and can be specialized by type, to give, for example, the cursor marker or the screen origin marker. *Rectangles* (which are just pairs or markers) bound the area within which a trigger's preconditions can be tested. Flags contain the results of Boolean tests of pixel patterns, computed by matching specified patterns at a given location or within a rectangular region. As an example of the capabilities of a TRIGGERS program, consider the problem of inscribing the letter Z with given proportions inside an ellipse already drawn on the screen. Instead of using trigonometry to determine the four points at which the letter touches the ellipse, a TRIGGERS program can instead scan the screen for on pixels within appropriate regions to determine those points by observation.

Some of the correspondences between TRIGGERS and other systems for situated action (for example, PENGI [Agre 1997; Agre and Chapman 1987]) are suggestive. TRIGGERS does not maintain a complex internal representation of its environment or its activities, yet it carries out goal-directed behavior that adapts to different conditions. Although its markers and rectangles are maintained internally, TRIGGERS also displays them visually in the interface; the system could plausibly dispense with an internal representation and refer to the display instead. Control over the set of triggers is straightforward and could in principle be implemented in simple hardware. Processing of arbitrarily complex flags poses a difficulty from a cognitive modeling point of view but was not an issue for TRIGGERS, which was designed with other research goals in mind. The imposition of such constraints could be addressed by further work, as is common in the evolution of cognitive modeling systems (Anderson and Lebiere 1998).

The user interface provides an attractive arena for research on situated action. Observations about the inference (or lack of inference) necessary to carry out complex tasks have implications for agent design but also for users who operate in the same environment.

Conclusions

IUI encompasses a number of other research directions in which the model world metaphor plays a significant role:

Knowledge representation: Model-based interface specification is concerned with automating the interface development process, or significant parts of it (Castells, Szekely, and Salcher 1997; Puerta and Maulsby 1997; Szekely 1996; Szekely, Luo, and Neches 1993). Researchers have developed sophisticated AI representations of interface structure and behavior, making the close relationship between AI agents and the interface seem only natural.

Plan recognition: The ability to infer users' goals from their actions in the interface would allow an agent to simplify or even eliminate tasks for the user, anticipate requests for information, and in general act as an intelligent assistant. Plan-recognition techniques are in several ways well suited to this purpose (Lesh, Rich, and Sidner 1998; Waern and Stenborg 1995).

Machine learning: Other techniques are also applicable to the problem of learning and reproducing common sequences of actions users carry out, a problem considered in the area of programming by demonstration (PBD) (Cypher 1993; Lieberman 2001). Machine learning techniques play an increasingly important role in PBD research (Lau and Weld 1999; Maulsby and Witten 1997); user activity can often be abstracted to a form amenable to symbolic learning techniques.

These research directions rely on properties of the user interface that facilitate interaction, such as simple, discrete actions with visible effects, repeated sequences of such actions, simple control structures, and guided focus of attention, along with all the interface properties identified earlier. Much can be gained by understanding the ways in which designers have constrained a problem for human users and building interface agents and other systems that can exploit these constraints.

Not all problems, of course, can be solved by agents interacting with the user interface as a model world. Often the problems addressed by IUI researchers appear within a specific domain in which there is far less structure than suggested by the examples in this article. In tasks such as reading and responding to e-mail or browsing the World Wide Web, for example, it is much more important for an intelligent assistant to understand the information at hand than to be able to manipulate the interface supporting the task. Many other everyday tasks, however, do lend themselves to agent interaction at this level, providing rich opportunities for research and development.

Acknowledgments

This effort was supported by the National Science Foundation under award 0083281. The information in this article does not necessarily reflect the position or policies of the U.S. government, and no official endorsement should be inferred.

Note

1. Amy Bruckman. The Combinatorics of Storytelling: Mystery Train Revisited. Unpublished manuscript.

References

Agre, P. E. 1997. Computation and Human Experience. Cambridge, U.K.: Cambridge University Press.

Agre, P. E., and Chapman, D. 1990. What Are Plans For? *Robotics and Autonomous Systems* 6(1–2): 17–34.

Agre, P. E., and Chapman, D. 1987. pengi: An Implementation of a Theory of Activity. In Proceedings of the Sixth National Conference on Artificial Intelligence, 268–272. Menlo Park, Calif.: American Association for Artificial Intelligence.

Agre, P., and Rosenschein, S., eds. 1996. *Computational Theories of Interaction and Agency.* Cambridge, Mass.: MIT Press.

Anderson, J., and Lebiere, C. 1998. *The Atomic Components of Thought.* Hillsdale, N.J.: Lawrence Erlbaum.

Apple. 1992. Macintosh Human Interface Guidelines. Apple Computer, Inc., Cupertino, California.

Arkin, R. C. 1998. *Behavior-Based Robotics.* Cambridge, Mass.: MIT Press.

Asada, M.; Veloso, M. M.; Tambe, M.; Noda, I.; Kitano, H.; and Kraetzchmar, G. K. 2000. Overview of RoboCup-98. *AI Magazine* 21(1): 9–19.

Balkanski, C. 1993. Actions, Beliefs, and Intentions in Multi-Action Utterances. Ph.D. dissertation, Department of Computer Science, Harvard University.

Byrne, M. D., and Anderson, J. R. 2001. ACT-R/PM and Menu Selection: Applying a Cognitive Architecture to HCI. *International Journal of Human-Computer Studies.* Forthcoming.

Castells, P.; Szekely, P.; and Salcher, E. 1997. Declarative Models of Presentation. In Proceedings of the Third International Conference on Intelligent User Interfaces (IUI'97), 137–144. New York: Association of Computing Machinery.

Cavazza, M. and Palmer, I. 2000. Natural Language Control and Paradigms of Interactivity. Paper presented at the AAAI Spring Symposium on AI and Interactive Entertainment, 20–22 March, Stanford, California.

Cohen, P. R.; Greenberg, M. L.; Hart, D. M.; and Howe, A. E. 1989. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *AI Magazine* 10(3): 32–48.

Cypher, A., ed. 1993. *Watch What I Do: Programming by Demonstration.* Cambridge, Mass.: MIT Press.

Davis, I. 2000. Warp Speed: Path Planning for Star Trek: Armada. Paper presented at the AAAI Spring Symposium on AI and Interactive Entertainment, 20–22 March, Stanford, California.

Dix, A. J. 1993. Formal Methods for Interactive Systems. San Diego, Calif.: Academic.

Dix, A. J.; Finlay, J. E.; Abowd, G. D.; and Beale, R. 1998. *Human-Computer Interaction.* Second ed. New York: Prentice Hall.

Dulberg, M. S.; St. Amant, R.; and Zettlemoyer, L. 1999. An Imprecise Mouse Gesture for the Fast Activation of Controls. In Proceedings of the Seventh IFIP Conference on Human-Computer Interaction (INTERACT'99), 375–382. Amsterdam: IOS.

Flach, J.; Hancock, P.; Caird, J.; and Vicente, K., eds. 1995. *Global Perspectives on the Ecology of Human-Machine Systems.* Hillsdale, N.J.: Lawrence Erlbaum.

Gaver, W. W. 1991. Technology Affordances. In Proceedings of the ACM Conference on Human Factors and Computing Systems (CHI'91), 79–84. New York: Association for Computing Machinery.

Hanks, S.; Pollack, M. E.; and Cohen, P. R. 1993. Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. *AI Magazine* 14(4): 17–42.

Hix, D., and Hartson, H. R. 1993. *Developing User Interfaces.* New York: Wiley.

Hobbs, J. R. 1985. On the Coherence and Structure of Discourse. Technical Report, CSLI-85-37, Center for the Study of Language and Information, Stanford University.

Hutchins, E. 1989. Metaphors for Interface Design. In *The Structure of Multimodal Dialogue,* eds. M. M. Taylor, F. Neel, and D. G. Bouwhuis, 11–28. Amsterdam: Elsevier Science.

John, B. E., and Kieras, D. E. 1996a. The goms Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM Transactions on Computer-Human Interaction* 3(4): 320–351.

John, B. E., and Kieras, D. E. 1996b. Using goms for User Interface Design and Evaluation: Which Technique? *ACM Transactions on Computer-Human Interaction* 3(4): 287–319.

John, B. E., and Vera, A. H. 1992. A GOMS Analysis of a Graphic, Machine-Paced, Highly Interactive Task. In Proceedings of CHI'92, 251–258. New York: Association of Computing Machinery.

Kieras, D. 1998. A Guide to GOMS Model Usability Evaluation using gomsl and glean3. Technical Report, 38, TR-98/ARPA-2, Department of Electrical Engineering and Computer Science, University of Michigan.

Kieras, D. E. 1988. Toward a Practical GOMS Model Methodology for User Interface Design. In *Handbook of Human-Computer Interaction,* ed. M. Helander, 135–157. Amsterdam: North-Holland.

Kieras, D., and Meyer, D. E. 1997. An Overview of the epic Architecture for Cognition and Performance with Application to Human-Computer Interaction. *Human-Computer Interaction* 12(4): 391–438.

Kinny, D., and Georgeff, M. 1991. Commitment and Effectiveness of Situated Agents. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 82–88. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Kintsch, W. 1998. *Comprehension: A Paradigm for Cognition.* Cambridge, U.K.: Cambridge University Press.

Kirlik, A.; Miller, R. A.; and Jagacinski, R. J. 1993. Supervisory Control in a Dynamic and Uncertain Environment II: A Process Model of Skilled Human Environment Interaction. *IEEE Transactions on Systems, Man, and Cybernetics* 23(4): 929–952.

Kitajima, M., and Polson, P. G. 1997. A Comprehension-Based Model of Exploration. *Human-Computer Interaction* 12(4): 345–389.

Kitajima, M., and Polson, P. G. 1995. A Comprehension-Based Model of Correct Performance and Errors in Skilled, Display-Based, Human-Computer Interaction. *International Journal of Human-Computer Studies* 43(1): 65–99.

Laird, J. 2000. It Knows What You're Going to Do: Adding Anticipation to a Quake-Bot. Paper presented at the AAAI Spring Symposium on AI and Interactive Entertainment, 20–22 March, Stanford, California.

Lau, T., and Weld, D. S. 1999. Programming by Demonstration: An Inductive Learning Formulation. In Proceedings of the Fifth International Conference on Intelligent User Interfaces (IUI'99), 145–152. New York: Association of Computing Machinery.

Lesh, N.; Rich, C.; and Sidner, C. L. 1998. Using Plan Recognition in Human-Computer Collaboration. Technical Report, TR98-23, Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts.

Lewis, M. 1998. Designing for Human-Agent Interaction. *AI Magazine* 19(2): 67–78.

Lieberman, H., ed. 2001. *Your Wish Is My Command: Programming by Example.* San Francisco, Calif.: Morgan Kaufmann.

Lieberman, H. 1998. Integrating User Interface Agents with Conventional Applications. In Proceedings of the Fourth International Conference on Intelligent User Interfaces (IUI'98), 39–46. New York: Association of Computing Machinery.

Mann, W. C., and Thompson, S. A., eds. 1992. *Discourse Description: Diverse Linguistic Analyses of a Fund-Raising Text.* Amsterdam: John Benjamins.

Maulsby, D., and Witten, I. H. 1997. cima: An Interactive Concept Learning System for End-User Applications. *Applied Artificial Intelligence* 11(7–8): 653–671.

Mullet, K., and Sano, D. 1995. *Designing Visual Interfaces: Communication-Oriented Techniques*. Englewood Cliffs, N.J.: Prentice Hall.

Murphy, R. 1999. Case Studies of Gibson's Ecological Approach to Mobile Robots. *IEEE Transactions on Systems, Man and Cybernetics* 29(1): 105–111.

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

Newell, A., and Simon, H. 1972. *Human Problem Solving*. New York: Prentice Hall.

Norman, D. A. 1999. Affordance, Conventions, and Design. *Interactions* 6(3): 38–43.

Norman, D. A. 1991. Cognitive Artifacts. In *Designing Interaction: Psychology at the Human-Computer Interface,* ed. J. M. Carroll, 17–38. Cambridge, U.K.: Cambridge University Press.

Potter, R. 1999. Pixel Data Access: Interprocess Communication in the User Interface for End-User Programming and Graphical Macros. Ph.D. dissertation, Department of Computer Science, University of Maryland.

Potter, R. 1993. TRIGGERS: Guiding Automation with Pixels to Achieve Data Access. In *Watch What I Do: Programming by Demonstration,* ed. A. Cypher, 361–382. Cambridge, Mass.: MIT Press.

Preece, J.; Rogers, Y.; Sharp, H.; Benyon, D.; Holland, S.; and Carey, T. 1994. *Human-Computer Interaction.* Reading, Mass.: Addison-Wesley.

Puerta, A., and Maulsby, D. 1997. MOBI-D: A Model-Based Development Environment for User-Centered Design. In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'97), 4–5. New York: Association of Computing Machinery.

Ritter, F. E.; Baxter, G. D.; Jones, G.; and Young, R. M. 2000. Supporting Cognitive Models as Users. *ACM Transactions on Computer-Human Interaction* 7(2): 141–173.

Runciman, C., and Hammond, N. 1986. User Programs: A Way to Match Computer Systems and Human Cognition. In *People and Computers: Designing for Usability. Proceedings of the Second Conference of the British Computer Society, Human-Computer Interaction Specialist Group,* 464–481. Cambridge, U.K.: Cambridge University Press.

Schunn, C. D., and Anderson, J. R. 1998. Perception and Action. In *The Atomic Components of Thought, eds.* J. R. Anderson and C. Lebiere, 385–428. Hillsdale, N.J.: Lawrence Erlbaum.

Shneiderman, B. 1998. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, Mass.: Addison-Wesley.

Shneiderman, B., and Maes, P. 1997. Debate: Direct Manipulation vs. Interface Agents. *Interactions* 4(6): 42–61.

St. Amant, R. 1999. User Interface Affordances in a Planning Representation. *Human Computer Interaction* 14(3): 317–354.

St. Amant, R., and Riedl, M. O. 2001. A Perception/Action Substrate for Cognitive Modeling in HCI. *International Journal of Human-Computer Studies* 55(1): 15–39.

St. Amant, R., and Zettlemoyer, L. S. 2000. The User Interface as an Agent Environment. In Proceedings of Autonomous Agents 2000, 483–490. New York: Association of Computing Machinery.

Stout, B. 1996. Smart Moves: Intelligent Path-Finding. In *Game Developer's Magazine,* October, 13–19.

Suchman, L. 1987. *Plans and Situated Action*. Cambridge, U.K.: Cambridge University Press.

Szekely, P. 1996. Retrospective and Challenges for Model-Based Interface Development. In Proceedings of Computer-Aided Design of User Interfaces (CADUI'96), 1–27. Namur, Belgium: Namur University Press.

Szekely, P.; Luo, P.; and Neches, R. 1993. Beyond Interface Builders: Model-Based Interface Tools. In Proceedings of Human Factors in Computing Systems: INTERCHI'93, 383–390. Amsterdam: IOS.

Thomas, F., and Johnston, O. 1981. *The Illusion of Life: Disney Animation*. New York: Hyperion.

Waern, A., and Stenborg, O. 1995. A Simplistic Approach to Keyhole Plan Recognition. Technical Report, T-1995-01, Swedish Institute of Computer Science.

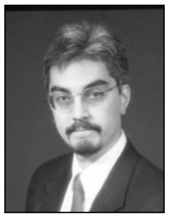
Woods, D. D. 1991. The Cognitive Engineering of Problem Representations. In *Human-Computer Interaction and Complex Systems,* eds. G. R. S. Weir and J. L. Alty, 169–188. San Diego, Calif.: Academic.

Young, R. M.; Green, T. R. G.; and Simon, T. 1989. Programmable User Models for Predictive Evaluation of Interface Designs. In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'89), 15–19. New York: Association of Computing Machinery.

Young, R. M.; Pollack, M. E.; and Moore, J. D. 1994. Decomposition and Causality in Partial Order Planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (AIPS'94). Menlo Park, Calif.: AAAI Press.

Zettlemoyer, L., and St. Amant, R. 1999. A Visual Medium for Programmatic Control of Interactive Applications. In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99), 199–206. New York: Association of Computing Machinery.

Zettlemoyer, L.; St. Amant, R.; and Dulberg, M. S. 1999. Ibots: Agent Control through the User Interface. In Proceedings of the Fifth International Conference on Intelligent User Interfaces (IUI'99), 31–37. New York: Association of Computing Machinery.



Robert St. Amant is an associate professor in the Department of Computer Science at North Carolina State University, where he codirects the Intelligent Interfaces/ Multimedia/Graphics Lab. He received a B.S. in computer science from the Johns Hopkins University in 1985 and a Ph.D. in computer science from the University of Massachusetts at Amherst in 1996. His current research deals with user interface agents, intelligent assistance for visualization, and representations for human and agent tool use. His e-mail address is stamant@cs.ncsu.edu.

R. Michael Young is an assistant professor in the Computer Science Department at North Carolina State University (NCSU). His interests center on the use of AI techniques in virtual worlds. His work involves research on planning and plan recognition, natural language generation, and computational models of narrative. He has a Master's in computer science from Stanford University (1988) and a Ph.D. in intelligent systems from the University of Pittsburgh (1997). Prior to joining the faculty at NCSU, he worked as a postdoctoral fellow in the Robotics Institute at Carnegie Mellon University. His e-mail address is young@csc.ncsu.edu.