Affordances for acting in direct manipulation interfaces

Robert St. Amant
Department of Computer Science
North Carolina State University
EGRC-CSC Box 7534
Raleigh, NC 27695-8206
stamant@csc.ncsu.edu

Affordances are properties of the relationship between an agent and its physical environment, properties that allow and facilitate specific types of interaction: a floor affords support, a chair affords sitting, a pen affords being grasped by the hand in only a few different configurations. Software affordances exist as well, with the user acting the role of the agent, the interface in place of the environment. An ideal interface would afford every function for which it has been designed, making its use intuitive, seemingly natural, and easily learned. Unfortunately, there exists no general theory of physical affordances to give us guidance in building appropriate affordances into software. Nevertheless the tractability of the user interface, in contrast to the physical world, allows us to develop a limited theory of interface affordances. The theory is based on a well-understood representation of action used in artificial intelligence planning systems. This article describes the theory, shows how existing interface mechanisms fit into its conceptual framework, identifies novel interface affordances suggested by the framework, and outlines procedures to develop and evaluate affordances in the interface.

---

1. INTRODUCTION

The concept of affordance has gained wide currency in the literature of human-machine interaction. An affordance is a property of the relationship between an agent and the environment in which it exists, something that the environment provides the agent, for better or worse [Gibson 1979]. These properties influence the way we use objects and how we interact with features of the environment. For example, we pick up a pen in such a way that it fits comfortably in the hand, ignoring all the myriad inappropriate ways that it might be grasped. We push the buttons on a telephone, rather than trying to pull them or slide them from side to side. We walk through doorways that have been designed to afford passage of our bodies, and sit in chairs designed to afford support for our posteriors. Our natural behavior is partly born of long experience, but also arises from properties that have been designed into these artifacts to make them appropriate for specific uses.

Affordances are part of an ecological view of the world, first developed by Gibson [Gibson 1979]. The ecological approach is characterized by an emphasis on the situated nature of human behavior [Flach 1995; Vicente 1995]. Ecological researchers study the mutual interaction between humans and their environment, the system as a whole, rather than concentrating on human cognition in isolation. This shift in emphasis leads to a different flavor of research. Some researchers study simple but not well-understood problems: how is it that people can intuitively judge whether an inclined plane can be easily climbed, or a seat sat upon, or a doorway passed through [Mark et al. 1990; Warren 1995]? Others concentrate on more complex problems, including vehicle guidance and navigation, process control, even architectural design [Kirlik et al. 1993; Hancock et al. 1995]. A third important area is perception, understanding how and whether people can pick up affordances through exploratory interaction with the environment, rather than through memory and inference [Gibson 1979; Ullman 1980; Fodor and Pylyshyn 1981; Turvey et al. 1981]. In general, ecological research attempts to explain behavior in the context of an explicit environment, based on the properties that emerge from interaction of an agent with the environment.

The notions of ecology and affordances have intuitive appeal for designers. In particular, user interface designers have come to view affordances as the perceived properties of a software artifact that indicate how it can be used [Baecker et al. 1995]. To contrast this with the conventional approach, imagine the problem of designing a U-shaped desk for an office. One can easily specify its size and shape in physical measurements, its width, height, surface area, and so forth. Nevertheless, this specification misses perhaps the most important property of the desk: it surrounds its user with working space on three sides. Such observer-dependent properties can be difficult to derive or even to represent in a conventional, observer-independent language of measurement [Gibson 1979; Flach 1995]. An ecological framework, on the other hand, concentrates on exactly these properties, the activities and relationships that are afforded to the user. If we can design artifacts with good affordances for the functionality they implement, we reap obvious benefits, including short or negligible familiarization time, high user understanding, and effective use of the functionality. Given these potential benefits, how then can we integrate affordances into user interface design?

Ideally, we would specify the properties of an environment (the user interface), an agent (the user), and the kinds of actions to be supported in the interaction (the functionality of the underlying system.) We would then turn to a general theory to help us reason backward to the affordances that facilitate the actions. Unfortunately, no such general theory of affordances exists. Although some progress has been made in understanding the concept of affordances in the physical world [Flach et al. 1995; Hancock et al. 1995], there is no operational, general-purpose definition of affordance that we can apply to arbitrary problems. We suspect that because affordances are so closely tied to perception and action in the physical world, a general theoretical account of affordances must await general theories of action and perception.

Fortunately, the situation is not lost. We are interested not in the physical world, but in a relatively limited avatar, the user interface. Like softbot researchers [Etzioni and Weld 1994; Etzioni et al. 1995], we can exploit the inherent differences between a user interface and the physical world to our benefit. Our approach begins with the development of a state-based representation of an abstract user interface and the actions, or operators, that can be applied to the interface. This representation provides the basis for a limited theory of affordances, where we informally use the term "theory" to denote a conceptual framework that explains and predicts phenomena in a given domain. In this theory, an affordance is a mechanism that allows or facilitates the execution of some operator. More specifically, an affordance preserves the conditions necessary for the successful completion of the operator, by reducing the execution cost of other appropriate operators or by increasing the execution cost of inappropriate operators. The theory deals only with the

actions that can be taken in the interface, such as keyboard entry, mouse movement, and mouse gestures to activate common software widgets. It does not account for the information exchanged between the user and the system, and it does not currently extend to higher-level affordances for learning, communication, or other complex activities. Despite these limitations, the conceptual framework gives useful guidance about user interface design and evaluation.

We apply the theory to a number of mechanisms discussed in the literature of human-computer interaction (HCI), to explain how they share a common interpretation as affordances. We also examine some common sequences of operations in direct manipulation interfaces, and show how the framework directly suggests novel affordances that improve the reliability of the sequences. We finally describe a general experimental procedure to guide the calibration and evaluation of the implementation of an affordance.

The remainder of this article is structured as follows. Section 2 discusses past work on the definition of affordances and the explicit consideration of affordances in the design of software systems. Section 3 describes affordances in the physical world and how they can be applied to a simplified problem space representation of the user interface. Several different types of interface affordances, both existing and novel, are described in detail. Section 4 illustrates how the conceptual framework of affordances can be applied to a specific example.

2. RELATED WORK

Affordance is an intuitive notion, easily described and understood through examples. Like many intuitive concepts, however, it is difficult to define in precise analytical terms. For example, imagine yourself in the act of sitting down in a chair. There are at least four separate affordance-related concepts involved. First are the affordances proper: the seat of the chair is horizontal, flat, extended, rigid, and approximately knee-high off the ground, all relative to your own proportions and position [Gibson 1979, pp. 128-9]. Second is your perception of these properties, the surfaces, distances, areas, textures, relationships between parts, and so forth. Third is the mental interpretation you derive from the perceptions. Fourth and finally is the act of sitting itself. Each of these concepts has been given, at different times and by different authors, as the definition of "affordance." An examination of these positions will give a better understanding of the subtleties involved.

—*Affordances are relationships or properties of relationships.* We begin with Gibson's original conception, that affordances are what we will call ecological properties, intrinsic properties of the relationship between an agent and its surrounding environment [Gibson 1979, p. 143]. Many ecological researchers rely on this definition, with one notable specialization: an emphasis on affordances for action. Kirlik and his colleagues, for example, define an affordance as "a relationship between properties of the environment and properties of an organism's action capabilities" [Kirlik et al. 1993, p. 934]. Static relationships between an agent and its environment, such as support or containment, are of less interest than the active behavior that the environment affords the agent, such as walking or sitting [Mark et al. 1990]. For researchers and designers interested in issues beyond perception, this is a natural shift of focus.

—*Affordances are perceived properties.* Baecker and Buxton write, "Affordances are the perceived properties of an artifact that indicate how it can be used" [Baecker et al. 1995]. This view, common in human-computer interaction (HCI) circles, was popularized by Norman [Norman 1988]. It moves the emphasis from properties that exist independent of the intentions and perceptions of an agent, toward the idea that affordances are *meant* to be perceived, in order for an artifact to be used properly. Thus the perceptions themselves are sometimes described as being the affordances. While this may seem a minor difference, it is important from an ecological psychology point of view. As Flach writes, "[this] confuses the affordances of an object with the information that specifies the affordances" [Flach 1995].

—*Affordances are actions.* Some researchers work at a level of abstraction in which the distinction between affordances and actions is blurred. Mark and colleagues, in testing the ability of human subjects to perceive whether a seat can be sat on, define affordances as "those actions which the particular arrangement of environmental substances and surfaces will support" [Mark et al. 1990]. In a similar vein, Gaver describes affordances as "potentials for action" [Gaver 1991, p.79]. It is clear that these authors are using "action" and "potential for action" as a kind of shorthand, but it raises a question about whether the distinction between action and affordance is always necessary.

—*Affordances are mental constructs.* A final view is that affordances are subjective in nature. Vera and Simon see affordances as "internal representations of complex configurations of external objects" [Vera and Simon 1993a, p. 41]. Gibson's original definition holds that affordances are objective, measurable properties in the external world. Vera and Simon's view does not deny the existence of "external" affordances; for example, Newell and Simon observe that "adaptive devices shape themselves to the environment in which they are embedded" [Newell and Simon 1972, p. 789]. Instead, their position is that for regularities in the world to be perceptible by an agent, some change in mental representation must occur, and it is the resulting mental structure that deserves to be called an affordance. These mental affordances are the internal encodings of symbols denoting relationships, rather than the external situations that evoke the symbols [Vera and Simon 1993b].

It would be difficult to bring all these differing viewpoints into agreement, and in fact there has been some controversy in the cognitive science literature over just this issue [Vera and Simon 1993a; Greeno and Moore 1993; Suchman 1993; Agre 1993; Vera and Simon 1993b]. Instead, we will extract appropriate elements from each viewpoint, concentrating on the first and fourth definitions. Our goal is to construct a framework in which we can analyze the affordances of an interface, concentrating especially on affordances for action, to decide how to improve them and where to implement new ones. We want to provide methods to answer the questions an ecologically-minded designer might ask: Does a given combination of factors in the

environment constitute an affordance? How so? Does one design provide better affordances than another? Is the difference between affordances even amenable to measurement? How can we decide where affordances are needed? Can we predict the benefit of adding an affordance?

Three lines of research have made relevant contributions toward this goal, by either proposing an operational definition of affordance or explicitly showing how affordances contribute to design.

The approach of Kirlik's research group is most similar in spirit to our own. They examined the supervisory and manual control of a simulated scout craft in a dynamic uncertain environment containing friendly and hostile agents [Kirlik et al. 1993]. Tasks included locomotion, sighting, and searching. A central part of their study was the definition of affordance distributions. An affordance distribution is a multi-dimensional array, overlaying a map of the environment, in which numerical entries reflect opportunities for action. A high value in one entry of a locomotion affordance distribution, for example, indicates that the craft can easily move from that position in the environment. Lower values are found along the edges of obstacles and in cul-de-sacs. All values in an affordance distribution are constructed through analysis or simulation, and tuned if necessary during a model validation phase. To evaluate their system, Kirlik's group developed process models that exploited the information provided by affordance distributions to guide their behavior. Based on standard measures of performance, the process models generated behavior very similar to that of human crew in the simulation. Their results support the view that affordances can play a significant role in constraining and guiding behavior in a complex environment.

Rassmussen and Vicente's Ecological Interface Design (EID) framework addresses the relationships between affordances, rather than affordances themselves [Rassmussen 1988; Vicente and Rasmussen 1990]. Their approach fits affordances into a means-ends hierarchical structure. They begin for illustration with a list of affordances identified by Gibson: survival, nurturing, falling-off, communicating, locomotion, sitting-on, swimming-over, support, bumping-into, walking-on, manufacture, getting-underneath, lifting, cutting, obstacle, sinking-into, carrying, graspable, copulation, standing-on, shelter, nutrition, drinking, breathing, and throwing [Vicente and Rasmussen 1990]. Intuitively, we tend to group these affordances by their similarity to one another. Cutting and lifting, for example, are more similar than cutting and locomotion. Locomotion, in turn, is more similar to communicating than to survival. A coarse decomposition classifies affordances into three categories: *how*, *what*, and *why*. For example, a surface might provide locomotion (*what*), with walking the means (*how*) and survival the abstract goal (*why*). A system in which affordances are directly related to one another, and in which these relationships can be directly perceived, gives an agent a way to concentrate on important problems in the system and the information necessary to solve the problems. The hierarchy lets the agent move from abstract overviews to the details of the environment, guided by specific relationships between levels in the hierarchy.

A third direction is suggested by Vera and Simon's view of affordances as mental constructs. By identifying and categorizing invariants, an automated system might determine their relevance to its own actions, and thus capture the affordances of an environment [Kalish 1993]. Cohen's work is a good example of this approach. The Neo project is an attempt to build an agent in a simulated environment that can learn conceptual structures through its interactions with the environment [Oates and Cohen 1996; Cohen et al. 1996; Cohen et al. 1997]. At the core of the Neo system is a machine learning algorithm that monitors streams of symbolic information elements that change discretely with time and in response to the actions taken by Neo. As information elements and combinations of elements recur over time, patterns can be detected and stored in "fluent" structures. Fluents are the mental constructs that represent invariant relationships in the environment, the affordances, in Vera and Simon's terminology. As Neo learns new fluents, these become new, virtual streams of information which can be observed and learned from in turn. Fluents can be built on top of fluents to create a hierarchy of affordances at different levels of detail. Comparable approaches have been taken in projects described in the situated action literature [Agre and Chapman 1987; Suchman 1987; Agre and Chapman 1990].

3. DESIGNING AFFORDANCES

We begin with a general definition of affordances in physical environments:

> An *affordance for action* is a property of the relationship between an agent and the environment, a property that passively allows or actively facilitates the execution of an action or set of actions.

The properties called out in this definition have two limiting cases. If we are interested in a set of uniform agents with fixed properties, then the agent/environment relationship will be determined by the characteristics of the environment, and we may informally refer to affordances as "properties of the things in the environment that are relevant to their contributions to interactions that people have with them" [Greeno and Moore 1993]. This is just shorthand, though, for the more general case of affordances as ecological properties. A similar shorthand holds for agents when the environment is held fixed; these affordances are sometimes called effectivities [Turvey and Shaw 1979]. Otherwise, the definition depends on the key notions of properties that allow and facilitate actions.

—*Properties that allow actions:* Any action that an agent finds possible to perform in an environment is afforded (for that agent in that environment.) We call a relationship that no more than allows an action to proceed a *simple affordance*. Simple affordances are nonrestrictive in the extreme. For example, the action of walking is afforded by a flat, extended surface, such as a floor in a room. If we were to change the properties of the surface, by inclining it, or adding obstacles, or making it less rigid, we might finally reach a point where we would agree that walking was not possible at all. Only then does the simple affordance for walking disappear.

Simple affordance is defined in black and white terms: it either exists or it does not. Though this can be hard to determine in real-world, physical examples such as the one above, for some kinds of environments it is more straightforward. For example, a conventional jigsaw puzzle has a simple affordance for being put together in only one configuration, because the puzzle pieces will fit together in no other way. Some toys are designed to be easily assembled—the affordance is identical [Norman 1988]. As we will see in our discussion of user interface affordances, the existence of a simple affordance for many actions can be easily determined.

—*Properties that facilitate actions:* Some affordances, beyond simply making an action possible, make that action easier to perform. We call this a *facilitating affordance*. Unlike simple affordances, facilitating affordances admit different degrees of effectiveness. Consider the affordance of a hammer for striking (a nail). Imagine choosing a hammer, hefting it in one's hand, raising it high in the air, and swinging downward until the head of the hammer hits the nail. A more powerful swing of the arm is possible with the hammer in hand than with an empty hand. The swinging action is facilitated by several factors. First, part of what makes a hammer a hammer is its massive head, the solid properties of its wood or metal components, and its relatively unrestricted mobility (in contrast, say, to a branch attached to a tree.) These are properties common to all conventional hammers, and are relevant regardless of the properties of the agent swinging the hammer. Second, the properties of the agent also come into play: humans have hands to grasp hammers and arms to swing them, without which abilities the swinging activity would not be possible. Third, and most importantly, the affordance depends on the sometimes complex relationships between the attributes of the object and the agent. The handle of the hammer must be narrow enough to be grasped by the agent; the hand of the agent must be flexible enough, strong enough, not too slippery, and so forth, to do the grasping; the hand must be empty before grasping the hammer, the arm unencumbered. Taken all together, these properties facilitate the striking of a nail with the hammer. Remove or alter a property in some way, and the facilitation is reduced: a hammer made of gelatin will not produce a solid impact; a hammer without a head will not sufficiently increase the force of the swing; a slippery hand might let the hammer to fly out of control.

Facilitation also applies to affordances of environmental features, in addition to objects. A flat surface, for example, affords support. A flat surface with a high friction coefficient, such as a stretch of sand or an area of corrugated rubber, affords support and lateral stability for objects placed on it. A lower-friction surface, such as a smooth piece of glass or ice, affords support and sliding. In both cases, support is facilitated, while other properties of each surface associate it with either lateral stability or nonstability.

The inverse of facilitation, inhibition, is also relevant. Insofar as an object affords one set of activities, it simultaneously renders it more difficult to accomplish other activities. A specific action may even be afforded

Object	*Property*	*Facilitates*	*Inhibits*
Hammer [Gibson 1979]	Weight	Arm movement	Arresting movement
Ground [Gibson 1979]	Solidity	Support	Penetration
Corridor [Kirlik et al. 1993]	Unobstructedness	Line-of-sight	Concealment
Shelf [Zaff 1995]	Accessible height	Reachability	Child-proof storage
Doorway [Warren 1995]	Width	Passage	Containment
Stair riser [Mark 1987]	Height	Climbing	Walking, rolling
Chair [Mark 1987]	Seat height	Sitting	Leaning
Incline [Mark et al. 1990]	Steepness	Crawling	Walking

Table 1. Objects, properties, facilitation and inhibition

in a relative sense, if other possible actions are not as easily accomplished. For example, the shape and mass distribution of the hammer facilitate grasping at one end and swinging, but also result in an inhibition of other relationships and activities. The hammer head is usually too awkward to be comfortably gripped; the hand holding the hammer becomes no longer available for other tasks; once in motion, the mass of the hammer makes it difficult to stop the swinging arm. Affordances of environmental features follow the same pattern. If a ridged surface lacks an affordance for lateral stability, it may afford heavy objects being slid across it, or playing hockey on it. Conversely, to the extent that a smooth surface inhibits sliding, it will facilitate lateral stability.

Table 1 lists a few examples from the literature on affordances. Each entry lists an object, one of its ecological properties, and one example each of actions the property facilitates and inhibits. An enormous number of other actions are simply afforded by these properties, but neither facilitated nor inhibited.

3.1 The user interface as a problem space

Descriptions of physical affordances draw on our rich understanding of objects and environments, and our experience in interacting with them. Formally representing action in the physical world, with all its complexity, however, is an enormously difficult problem. Instead, we will treat actions as part of abstract problem-solving activity and describe affordances in terms of the space in which these activities take place: a problem space.

Newell and Simon define a problem space as containing a set of elements, U, a set of operators, Q, an initial state of knowledge, u_0, a problem with goal G, and the knowledge available to solve the problem [Newell and Simon 1972]. A problem space describes a problem environment in discrete terms, specifing the properties (often in the form of logical predicates) that hold in the "world", as defined by U, and how the application of different operators will change these properties. A solution to a problem involves reasoning from the state u_0 to to the state G by constructing a sequence of appropriate operators from Q. The application of each operator leads to a new state, in which the next operator can be applied. Because there may be many paths from u_0 to G, and many more paths that do not lead to a solution at all, the process almost inevitably involves search.

Planning is a kind of search appropriate for reasoning in problem spaces [Korf 1987; Tate et al. 1990]. For traditional planners, the set Q contains *STRIPS operators* [Fikes and Nilsson 1971]. STRIPS operators are commonly formulated as having a precondition form and an effect form, both of which specify properties that hold or potentially hold in the world. An operator may be applied only when its preconditions hold in the current state. Its application results in the creation of a new state, in which the effects of the operator hold. A plan is a partially ordered sequence of operators that, when applied, lead from u_0 to G.

A problem space representation gives a planner a simple, controllable environment in which a problem has been refined down to its basic elements. When a problem space simplifies a physical problem to an extreme, it is sometimes called a microworld. Microworlds are common in the introductory AI literature: Towers of Hanoi, Monkey and Bananas, Missionaries and Cannibals, and Blocksworld [Rich and Knight 1991; Russell and Norvig 1995]. The advantage of dealing with a problem space instead of a real physical environment is that a planner, whether human or automated, can reason about the logic of problem solving instead of about difficult but less immediately relevant details.

As planning researchers have found, however, mapping from a solution in a problem space back to the real world can pose some serious pitfalls. A plan generated in an abstract problem space can be only an imperfect match for most real, complex environments; the environment may not proceed in discrete steps, with state changes that exactly match the planner's operators. The relevant distinction is between plan generation

and plan execution. Difficulties in executing a sequence of abstract operators arise mainly from the issues of control and uncertainty. First, the planner does not have complete control over the environment. Other agents may interfere with the planner's activities. Time passes and the environment may change dynamically while the agent acts. Second, the planner may not have enough information to perfectly predict the effects of all actions. This uncertainty means that some plans cannot be guaranteed to succeed in advance of execution.

For example, suppose that we generate a plan for the physical task of hitting a nail on the head with a hammer. Our problem space representation contains the operators *Grip hammer*, *Raise arm*, *Aim*, and *Swing downward*, which the plan orders as given. The preconditions and effects of these operators specify the relative positions of the hammer and the nail, the speed of the hammer, and other relevant factors. The conditions holding in the final state describe a situation in which the nail is embedded in some surface.

A myriad factors can interfere to cause the failure of this plan as it is executed. The hammer could slip out of my hand, causing me to throw the hammer instead of swinging it. While I am raising the hammer, a practical joker might snatch the nail away. An extraneous noise may disturb my aim, or an unforeseen obstacle may jog my arm as I swing the hammer. I might swing too slowly or weakly, hitting the nail with insufficient force to drive it into the surface. All these events are examples of the pitfalls that can arise in the transition between generating a plan in the abstract and executing it in a real environment.

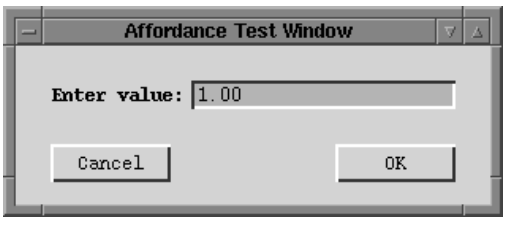
Let's consider now a problem space representation for the user interface, keeping these general caveats in mind. Following AI tradition, we name the problem space GUI World. Every object in the user interface has some set of properties. A text entry widget, for example, might have a *type* property whose value is `text-field`, a *label* property whose value is displayed next to it, a *value* property holding the contents of the field, and an *active* property which determines whether the text may be changed in the current context. Further, sets of interface elements and the interface as a whole have properties that correspond to their settings, modes, and so forth. Figure 1 shows a simple dialog and its problem space representation.

The state of the interface changes dynamically: buttons become active or inactive; the size and typeface of a segment of text can be changed by the user; menu lists may have selected and unselected items. Some operators to effect these changes can be applied by the user, others by the system. We usually concentrate on the activities of the user, and think of the system's behavior as simply responding to the user. The list below informally describes a small set of actions and their pre- and postconditions for a single-button mouse interacting with an interface with an unspecified set of widgets. These actions are abstract operator schemas: in order to be executed, they need to be associated with specific widgets and other properties of the interface.

—*Mouse down on:* The user can press the mouse button on any interface object (including the background), as long as the mouse button is not already down, the pointer is over the object, and it is unobstructed. After execution of this operator, the mouse button is in the down state and any object under the pointer becomes activated.

—*Mouse up on:* The user can release the mouse button under similar conditions. This operator is applicable when the mouse button is down. The effects of this operator vary, depending on any widget the pointer is over at the time of its execution. For example, button widgets become clicked, other objects selected. In all cases, the mouse button returns to its original state.

—*Move:* This operator causes the pointer to move from its current position over some interface object to a




```
((widget Text)
 (widget-type Text-box text-field)
 (property Text-box label "Enter value:")
 (property Text-box text "1.00")
 (widget OK-button)
 (widget-type OK-button button)
 (widget Cancel-button)
 (widget-type Cancel-button button))
```

Fig. 1. A simple interface

new object. It is not applicable when the mouse button is down.

—*Drag:* This operator is the equivalent of the move operator, but with the mouse button in the down state. If the mouse down operator occurred over a movable object, that object is dragged to a given target.

—*Drag select:* For some types of interface widgets, such as pulldown menu items, dragging causes the pointer to move from one item to the next. This operator causes the pointer to change position between drag-selectable items. It is only possible when the mouse button is down and the pointer is over a widget of the appropriate type.

—*Deactivate:* Deactivation is the equivalent of a drag carried out on a non-movable object. With the mouse button down, the user moves the pointer away from an interface object, causing it to be no longer under the pointer, and no longer activated.

—*Reactivate:* Reactivation occurs when the pointer is moved over a non-movable object again, with the mouse button still held down, causing it to be activated once more.

—*Press key:* This operator is used to press a key on the keyboard. It is applicable when the key in question is not already being held down, and results in a state in which the relevant key is in a "down" state. *Release key* is the inverse operator.

An example of a problem in this representation would be to activate a confirmation button widget in a window. Beginning in some initial state with the pointer over the background of the window, an appropriate plan would contain the appropriate pointer movement and mouse actions. A representative subset of these actions is given in a more formal representation in Figure 2, in the popular planning language of UCPOP [Penberthy and Weld 1992]. This simple problem and its solution are shown in Figure 3.

How faithful is the problem space we have defined for user interaction? It is by no means complete. It describes the pointer position, for example, as simply being over some object, rather than at specific screen coordinates. Most of the functionality of individual widgets is not represented. Nevertheless, though many important elements of user interaction are abstracted away, a useful kernel remains. We can describe the actions of moving the pointer over a button to click on it, moving the pointer to grab a scroll box to scroll a window, pulling down a menu and selecting a menu item, selecting a toolbar icon, and other sequences. Complex behavior can be had despite the simplicity of the operators, through the behaviors and states of the objects in the environment and the context in which the actions are taken. Pressing and releasing a mouse button, for example, has a different effect depending on whether the pointer is over a widget, the type of widget involved, and often the internal state of the widget. This approach to building up complexity on top of simple elements is common to many user interface development aids [Myers et al. 1990; Castells et al. 1997].

With an abstract plan in hand, the next problem we face is the difficulty of executing it in a real user interface. Hutchins and others have referred to this difficulty, in a different context, as the gulf of execution [Hutchins et al. 1986; Norman 1991]. The gulf of execution refers to the difficulty of acting through the interface, of matching one's intentions to the actions that must be carried out. This corresponds exactly to the problem of squaring a problem space solution with a solution in the physical world—that is, plan execution—with the control and uncertainty pitfalls identified earlier: non-deterministic effects of operators, extraneous events, actions of multiple agents, passage of time, and so forth.

Fortunately, and somewhat surprisingly, an abstract problem space representation can be mapped onto a real user interface without great difficulty. In fact, design guidelines for user interfaces, such as the Macintosh Human Interaction Guidelines [Apple Computer 1992; Nielsen and Gentner 1996], can be interpreted as efforts to limit the differences between the abstract representation and the real environment. In each of the cases below, we can rephrase an interface design guideline as an effort to decrease the gap between an abstract plan or action and its execution in the interface.

—*Consistency* [Apple Computer 1992, p. 7]. Predictable behavior is a central design goal for user interfaces [Nielsen and Gentner 1996]. When a user takes an action, such as clicking on a button or activating a different kind of widget, it is expected that the action produces the same effect, as long as the appropriate conditions are in place. In planning terms, this guideline tends to produce operators with deterministic effects.

—*User control* [Apple Computer 1992, p. 9]. From the conventional (i.e., non-agent-based) perspective on user interaction, user control over the interface should be nearly absolute. The interface should not initiate

```
(:operator |Mouse down on|
           :parameters (?object)
           :precondition (and (not (mouse down))
                              (pointer-over ?object)
                              (not (obstructed ?object)))
           :effect (and (mouse down)
                        (when (not (eq ?object background))
                          (object-activated ?object))))
(:operator |Mouse up on|
           :parameters (?object)
           :precondition (and (mouse down) (pointer-over ?object))
           :effect (and (not (mouse down))
                        (when (object-activated ?object)
                          (and (not (object-activated ?object))
                               (when (object-clickable ?object)
                                 (object-clicked ?object))
                               (when (not (object-moved ?object))
                                 (object-selected ?object))
                               (when (object-moved ?object)
                                 (object-dragged ?object))))))
(:operator |Move|
           :parameters (?source ?target)
           :precondition (and (not (mouse down))
                              (pointer-over ?source))
           :effect (and (not (pointer-over ?source))
                        (pointer-over ?target)))
(:operator |Drag|
           :parameters (?object ?target)
           :precondition (mouse down)
           :effect (when (and (object-activated ?object) (movable ?object))
                     (and (object-moved ?object)
                          (object-dragged-to ?object ?target))))
(:operator |Drag select|
           :parameters (?current-item ?next-item)
           :precondition (mouse down)
           :effect (when (and (object-activated ?current-item)
                              (drag-selectable ?current-item next-item))
                     (and (not (object-activated ?current-item))
                          (object-activated ?next-item))))
```

Fig. 2. Operators

```
(define (problem OK-action)
    :domain 'interface-affordances
    :inits ((mouse up)
            (pointer-over background)
            . . .)
    :goal  (object-clicked OK-Button))


Init    : (and (pointer-over Background) . . . )
Step 1  : (Move (from) Background (to) OK-Button)
Step 2  : (Mouse down on OK-Button)
Step 3  : (Mouse up on OK-Button)
Goal    : (Object-Clicked OK-Button)
```

Fig. 3. Problem definition and solution for the simple interface

actions, but rather respond as a tool [Woods and Roth 1988]. If we view the user in the role of a planner, the same ideal of restricting the extraneous variability of the environment holds. This ideal is so well-entrenched that in both the planning and user interface literature the terms "action" and "event" are commonly used interchangeably [Tate et al. 1990]. In a planning context, the user control guideline tends to produce two constraints: the limited occurrence of events not initiated directly by the planning agent, and the elimination of actions taken by other agents in the environment.

—*Perceived stability* [Apple Computer 1992, p. 11]. Accepting user control does not mean ruling out continuous or automatic change in the environment; for process management and virtual reality systems change is a fact of life [Vicente and Rasmussen 1990; Woods 1991]. In conventional user interfaces, however, the user can almost always depend on stability, even as time passes. User interfaces are strongly state-based. Continuous change is unusual, and time-dependent, interface-initiated events (such as pop-up documentation) are rare. Instead, a user action usually leads to an immediate state change in the interface, which remains quiescent until the next user action. In planning terms, this guideline constrains the environment to change slowly enough (if at all) that we can generally rely on a plan being being appropriate for the environment, even if it has been generated some time before.

Our view of the user as a planning agent reflects the traditional interpretation of user interaction as problem-solving behavior [Woods 1991]. Given a set of operators, a human reasoner will select operators to be executed in sequence in order to reach a goal to be satisfied. This view adheres to Card et al.'s rationality principle: "A person acts so as to attain his goals through rational action, given the structure of the task and his inputs of information and bounded by limitations on his knowledge and processing ability" [Card et al. 1983, p. 27]. Intuitively, the rationality principle tells us that an agent will choose the easiest route that results in the completion of the task it has set out to accomplish. In the user interface, or in GUI WORLD, this means that the user will choose those actions that solve the problem at hand.

3.2 Interface affordances

The foregoing discussion has laid the groundwork for a more rigorous understanding of affordances. Recall that we represent interaction with the user interface as a sequence of changes to its state, and an action, or operator, in the interface is a state transition for which specific preconditions must hold before its execution. When we speak of an operator in this context, we mean an operator with its variables bound to specific values. Thus we will distinguish *Move over Widget X* and *Move over Widget Y* as two distinct operators, even though they are derived from the same operator schema. A simple affordance exists when it is possible, in some context, to execute an operator:

> A *simple interface affordance* for an operator A exists if a situation can arise in the interface in which A's preconditions hold.

This definition retains the mutuality property of physical affordances. An affordance depends on the capabilities, or operators, of the agent and the properties of the environment relevant to those capabilities. The same will be true for facilitating interface affordances.

In any realistic environment, even if a simple affordance for an operator exists, extraneous events (which we also represent as operators) can occur that will change some of the relevant state values in the preconditions of that operator. For example, suppose that the operator of interest is *Mouse down on Widget*, with preconditions that include `Pointer over Widget`. Just as we are about to press the mouse button to activate the widget, an inadvertent arm movement pushes the pointer out of the interior of the widget. In other words, an inadvertent *Move* operator causes `Pointer over Widget` to no longer hold, which means that the intended operation *Mouse down on Widget* cannot proceed. Imagine, however, that the interface manages to prevent or even undo the effects of the unintended movement action, so that the pointer remains over the widget. Then this hypothetical interface has a facilitating affordance for the *Mouse down on Widget* action.

> A *facilitating interface affordance* for an operator A is a mechanism that decreases the cost of operators that establish or preserve A's preconditions, or that increases the cost of other operators that do not.

Facilitating affordances are ways of keeping user interaction on track, making sure that it is easier (less costly) to select appropriate operators than inappropriate ones. Keeping in mind the existence of the

gulf of execution, we elect not to depend entirely on the hypothetically rational user who always chooses an appropriate action depending on the goal to be satisfied. The facilitating and inhibiting properties of affordances improve the likelihood that a sequence of abstract operators will be executed properly. In this sense the environment supports, or affords, action.

Our definition of facilitating affordances is reminiscent of a situation that arises in automatic plan generation. Suppose that a planner has selected some operator A_i to establish a precondition p for A_j, resulting in the plan fragment $\{A_i, A_j\}$. If the planner then needs to incorporate an operator A_k that has as one of its effects $\neg p$, then A_k is a *threat* to the relationship between A_i and A_j. A_k might potentially execute after A_i and before A_j, undoing the intended result of A_i, or *clobbering* the necessary precondition p. The usual remedy is to add ordering constraints to the plan to ensure that A_k occurs either before A_i or after A_j. In the case of interface affordances, we must be concerned with the wider range of any operators that may be legally performed, since we lack complete control over which may or may not be performed. Operators can be inserted into the plan, entirely without the intention of the user/planner, clobbering preconditions. This is what happens when an inadvertent mouse movement causes the pointer to move off a widget, or when the user's finger slips and the mouse button is released accidentally. To avoid such problems, we increase the cost of the execution of these threats, or provide new operators to recover from their effects—in other words, we incorporate affordances.

One last piece of the definition of affordances remains: the relationship between plans, plan execution, and the execution cost of individual operators. We need to specify how the sequences of abstract operators in a plan are turned into a sequence of executed actions for our notion of execution cost to make sense.

In the physical world, it is easiest to think of the inhibition or facilitation of operators in terms of the effort involved. A cost measure gives us a way of quantifying the ease or difficulty of executing these operators. The cost of executing an operator could include the time to execute individual actions, the difficulty encountered, the quality of the effects, and other factors.

Conceptually, plan execution proceeds as follows. We treat the combination of the user and the interface as a joint cognitive system, taking actions that depend on the characteristics of both parties [Woods and Roth 1988; Woods 1991]. This system generates a plan and executes it (interleaved generation and execution is possible but irrelevant). At each point in the execution, the joint system probabilistically selects the next operator in the plan *or* an alternative to execute. The probability of an alternative operator being selected is a proportional function of its cost, which represents its associated effort. The higher the cost of an alternative operator, the lower the probability of its being selected for execution. The result is a mostly rational process, but contaminated with noise. For any given plan, this noise can cause the substitution of operators (*Move to background* instead of *Move to widget*), the addition of extraneous operators (*Move to background* preceding *Mouse down*), or even the deletion of planned operators (*Mouse down*/*Mouse up* instead of *Mouse down*/*Drag*/*Mouse up*).

This account implicitly follows Agre and Chapman's assertion that plans are not necessarily programs, but rather provide guidance for decisions about acting [Agre and Chapman 1990]. Even though the decisions required for the interaction are elementary, they are not always correct because the environment is not entirely benign. Affordances bias the execution of actions toward those that are in the plan, and no others.

Examples will clarify the application of our definition of affordances. We will discuss affordances for the operators *Mouse down on*, *Mouse up on*, and *Drag*.

Mouse down on. Consider the task of moving the pointer until it is over a widget, preparatory to selecting or clicking on the widget. Though straightforward, this action is surprisingly unreliable over the long term. In a study of the application of Fitts' Law to performance using different input devices, for example, MacKenzie found that the a small but noticeable proportion of their data was unusable due to pointing errors: "Unadjusted error rates for pointing were in the desired range of 4% with means of 3.5% for the mouse, 4.0% for the tablet, and 3.9% for the trackball" [MacKenzie et al. 1991, p. 164]. A later study found a mean error rate of 1.8% for pointing with the mouse [MacKenzie and Buxton 1994]. Worden and her colleagues found a mean error rate of 5% (for both young and old users) in a study that directly examined targeting difficulties with the mouse [Worden et al. 1997].

In these cases, the mouse down action fails because the preceding action, *Move over Widget*, fails to have its intended effect of placing the pointer over the widget, in up to one out of every twenty trials. To improve this situation, we can reduce the effort needed to move the pointer over the widget. This introduces a facilitating

affordance for the mouse down action, in that the movement action establishes one of its preconditions.

Worden proposes "sticky icons" for this problem [Worden et al. 1997]. When the pointer is over widget that is sticky, the gain is reduced, requiring an increased movement distance to leave. The interface records the speed of the pointer and activates the sticky mechanism when the speed drops below a dynamically set threshold. Münch and Dillmann's haptic mouse implements a comparable affordance [Münch and Dillmann 1997]. On predicting the target widget of a user's mouse movement, the system activates an electromagnetic field to stop the mouse on the target, thus eliminating the possibility of overshooting and the need for detailed adjustment. These mechanisms have an additional benefit: once the pointer is over the widget, it becomes more difficult to move away. In the plan representation, this translates to an inhibition of an extraneous *Move to Background* action before the mouse down action.

Suppose that the system cannot accurately predict the target of a mouse movement and so cannot afford the movement toward a specific widget. If the mouse down action occurs while the pointer is outside a widget's target area, the intended action will fail. In this case another affordance suggests itself: making the selectable area of objects extend beyond their visible edges. An equivalent mechanism is to interpose an additional movement that puts the pointer over the target widget, effectively defining an area of "magnetism" around objects. We will call this a capture effect. Narrow objects such as line segments in graphical drawing interfaces often have such an affordance. Because selecting an arbitrarily fine line segment can be difficult, mousing down near it will cause its selection. Worden et al. take a complementary approach, the area cursor [Worden et al. 1997]. The area cursor has a "hot spot" larger than the conventional single pixel, which provides a greater area for intersection with selectable objects.

These approaches work for objects and widgets well separated spatially, but can run into difficulties in dense or highly structured situations. (Worden et al. found no difference between the standard pointer and the area cursor in discriminating between adjacent widgets—undegraded performance, but also unimproved.) It might seem obvious that if the pointer is over a widget, then that widget is the intended target. This is not always the case, however. A familiar slip when attempting to grab the scroll box of a scrollbar is to select the gray region instead, inadvertently causing rapid scrolling. A related slip in dragging near the edge of a window, to change its size, is to select an underlying window by mistake, with cascading errors possible as the pointer drags over the the contents of the underlying window. Affordances for appropriate behavior in these cases must attend to spatial relationships between potentially unrelated widgets, and could modify selection regions depending on proximity and earlier targeting movements.

Our last example of an affordance for the mouse down action is aimed at selection in a domain-specific environment, a statistical user interface [St. Amant and Cohen 1998]. In an exploratory analysis of a dataset, one often constructs displays such as histograms, scatter plots, and the like, to help visualize patterns in the data. One important part of the process is identifying and if possible accounting for outlying data values [Hoaglin et al. 1983; Hampel et al. 1986]. These can affect nonrobust estimators of various properties of a dataset, including its central tendency (e.g., mean) and spread (e.g. variance). Outliers may also be interesting in their own right as well; identifying outliers in the results of an academic aptitude test, for example, can point out students who may need specialized assistance.

In a direct manipulation statistical user interface, a useful operation is to be able to select individual data points in a potentially crowded display, to be examined in more detail. (Outliers may appear singly or in clusters.) Let's consider the general problem of picking out a data point anywhere in a scatter plot, such as the one in Figure 4a.

The difficulty in selecting an individual point in this plot is the density of data. Consider a problem of selecting a specific point in a cluster of n points, all of which overlap one another. From an affordance point of view, the problem is in selecting from n distinct operators, *Move over Point 1* ... *Move over Point n*. For each of these operators the relevant preconditions are `Pointer over Point i` and `Unobstructed Point i`. We would like to decrease the difficulty in establishing these preconditions, but not at the cost of making other selections more difficult, because we have insufficient information to concentrate on just one of them. Phrased in these terms, the solution again is straightforward: we need a mechanism that separates the points for selection, so that facilitating the selection of one of them does not inhibit the selection of any other.

We do this through a small magnification window that pops up when the pointer pauses over a dense area of the scatter plot, as shown in Figure 4b. The window displays all points in a fixed radius around the pointer position. Points are spread out in the magnification window by increasing their distance from the

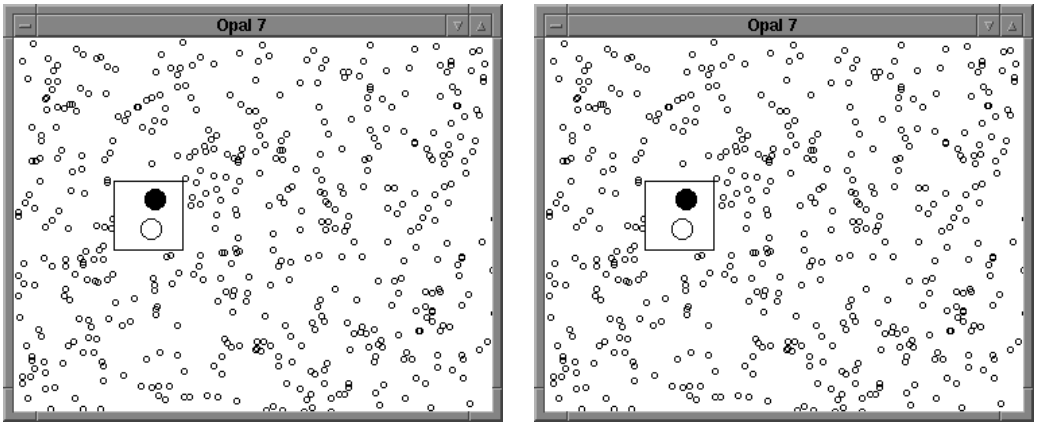


Fig. 4. Data window

central point and by adding small amounts of random "jitter" to their position. The result is a display that preserves the general pattern of points, so that individual ones are recognizable by position, and also makes them more accessible.

The basic idea of this affordance generalizes to other problems that involve unreliable selection as well. For example, cursor placement can be difficult in a word processing system that displays text in different typefaces. Placing the cursor between two narrow-width characters (e.g., *"ti"* or *"fl"*) can result in positioning errors, especially if the displayed characters are small. The usual solution is to require the user to redisplay the document on a larger scale, or to mouse down and then fine tune the placement with the arrow keys. A variant of the affordance we propose above would work equally well here, and could be faster. It would present a small pop up window containing the characters under the pointer, well-separated and in Roman typeface, with an indication (a secondary cursor) of the current selection. This mechanism should be no more obtrusive than the small documentation windows that pop up over some widgets; it is in fact a generalization of that functionality, simply adding an interactive element to the information displayed.

Mouse up on. A mouse up action over a target is subject to the same vagaries as mousing down over a target. Novice users sometimes position the pointer just on the edge of a widget, so that a tiny random movement will cause the mouse up action to be interpreted as a cancelation, rather than a confirmation. Worse, such slips can be difficult to detect, because they involve no direct volition on the part of the user.

An affordance for preventing this behavior is a refinement of the sticky icon: when the state of the mouse button is down, the gain of the pointer increases, specifically to inhibit a *Move* cancelation action when the more usual *Mouse up* is intended. The refinement of a sticky type of affordance is much less intrusive than the original conception. We can easily imagine a dense interface in which constraints on pointer movement with the mouse up would be cumbersome. The refinement has the effect of making only the preliminary gesture of a cancelation (moving the pointer off the widget while the mouse is being held down) more difficult, rather than all movement near the widget.

Drag. Dragging is in principle identical to pointing, though differently parameterized models may be necessary to describe the activities accurately [MacKenzie et al. 1991]. Because dragging involves the same preconditions as for pointing—both are simply mouse movements—the same affordances apply. These affordances can potentially be much more beneficial, however, because dragging is more difficult than pointing. In the same studies cited earlier, MacKenzie and his colleagues found error rates for dragging with different pointer devices to be 10.8% for the mouse, 13.6% for the tablet, and 17.3% for the trackball [MacKenzie et al. 1991]. Their later study found a mean error rate 4.2% for dragging with the mouse [MacKenzie and Buxton 1994]. The affordances for the mouse down operator should have similar benefits here.

Additional affordances can also be helpful in the specialized situations dragging is used. For example, to move rapidly through a text document, the user may grab the scroll box of the scrollbar. In some interfaces, moving the pointer some distance perpendicular to the scrolling direction is interpreted as a cancel operation, a behavior consistent with button widgets. Unfortunately, when one's attention is on the contents of the

Context	*Precondition*	*Affordance*
Mouse down on W	`Pointer over W`	*Move* (toward) facilitated
	`Pointer over W`	*Move* (away) inhibited
	`Pointer over W`	Movement inserted, before *Mouse down*
	`Pointer over W`	Target magnified
Mouse up on W	`Pointer over W`	*Move* (away) inhibited
Drag	`Pointer over W`	*Move* (toward) facilitated
	`Pointer over W`	*Move* (away) inhibited
	`Pointer over W`	*Move* (directional) inhibited
	`Pointer over W`	Movement inserted, before *Mouse down*
	`Mouse down`	*Unclick*

Table 2. Operators and their affordances

scrolling window, unintentional perpendicular movement is difficult to avoid. The cancelation results the disconcerting behavior of the document "jumping" back to its original state. An inhibiting affordance, however, would dynamically change the mouse gain so that perpendicular movement requires more effort than scrolling movement. With this affordance, cancelation would require a directed effort rather than an inadvertent movement.

The difference in difficulty between pointing and dragging can be partly attributed to releasing the mouse button accidentally, as can easily happen when the mouse button is held down for an extended period when dragging the scroll box of a scrollbar. This slip can be expensive in terms of recovery time: an object may need to be retrieved from a container; one's place may be lost in a large document; a large multiple selection may need to be redone.

When examined from the point of view of affordances that preserve or establish preconditions, the solution is obvious: a mechanism should be provided to allow the user to recover the lost state in which the widget or object was grabbed. We can think of this as an "unclick" or "pick up" operator. If a *Mouse up* operator is immediately followed by a *Mouse down* operator (i.e., if the interval between the two events is less than some small value), then the sequence should be ignored, and the state before the *Mouse up* operator restored. Unclicking is an affordance in that it preserves, through an undo type of action, an earlier object-grabbed state, often at a much lower cost than undoing, or canceling and redoing, the sequence of actions. It requires us to add an operator to the representation (or change the interpretation of an existing operator) but it is an easy and potentially useful change.

These operations are summarized in Table 2. While we have described these affordances as being specific to the three operators listed, they are really applicable to any operator with the same preconditions. Notice that this list is much more structured than an ad hoc list of user interface behaviors. Each affordance is targeted not at improving the interface in some general way, but at providing a more direct way of executing a specific operator, by establishing or preserving an appropriate state.

3.3 Affordances and interface dynamics

In our examples we have only touched on the issue of when affordances should be activated. In some cases an affordance can be active at all times, because it does not interfere with other possible actions, as with unclicking, or because it is relatively unobtrusive, as when the mouse gain increases with a mouse down action on a widget. Most situations, however, require some inference on the part of the system about the intentions of the user. Target acquisition is the best example of this, where the system must predict which widget the user intends to select in order to facilitate the action.

We believe that the dynamics of user interaction, as represented in sequences of plan actions, can be exploited to a much greater degree than is common in conventional interfaces. For example, if the user moves the pointer toward a widget, stops one pixel short, and presses the mouse button on the background of the window to no effect, the user's intention is—or should be—unmistakable. For full generality, the interface would need to be able to recognize user plans of all types, an extremely difficult problem. Nevertheless, the affordances we have discussed require only a limited, feasible form of plan recognition. Activating an appropriate affordance simply requires that the system interpret the procedural context in which actions take place, rather than treating them in isolation.

This attention to the dynamics of user interaction can have other benefits as well. Sometimes affordances produce undesired behavior. For example, in a graphical drawing interface with a grid in place, one sometimes wants to place an object in an unaligned position, near but not exactly on an intersection. One tries to drags the object near the intersection, finds that it is warped into position, pulls the object away, and tries again. Eventually one must release the object, turn off the grid, and try again. This is an appropriate place for the deactivation of the capture affordance. Affordance behavior need not be fixed at design time; it can be modified depending on context. A general rule of thumb is clear. Suppose that the user attempts some sequence of operators during which an affordance is activated, and which ends with a cancelation operator. In such cases, the affordance should be deactivated, at least temporarily, under the assumption that it is inappropriate in the current context. In general, consideration of the dynamics of user interaction can give the system useful information about the appropriate activation of affordances.

This leads to the issue of the perception of affordances, in both physical environments and user interfaces. The ecological approach lays a strong emphasis on the exploratory aspect of perceiving affordances [Mark et al. 1990]. While for many objects and environmental features we can immediately identify the interactions they afford, for many others we interact with them, gathering necessary information, to gain a better sense of the uses for which they are appropriate. In an extensive study of the affordances for sitting, for example, Mark and colleagues attached 10cm high blocks to subjects' shoes and asked them to judge whether an adjustable seat was at an appropriate height for sitting [Mark 1987; Mark et al. 1990]. Through exploratory locomotion, head turning, and leaning, subjects were able to retune their judgments to remain highly accurate. These exploratory activities were also necessary in the control condition, with subjects wearing normal shoes, even though sitting is a highly practiced, familiar activity.

In the user interface, the visual properties of many widgets give a direct indication of their behavior. The most commonly cited example of a directly perceivable interface affordance is that of a button widget for being pressed. The suggestive visual properties include its apparent protrusion from a background surface and a clear, outlining border, giving the impression that the button is not directly attached to the surface on which it is mounted [Gaver 1991]. Widgets with ridged or textured surfaces are another example of an immediately perceivable affordance. Such ridges on a physical object provide traction, so that a finger or hand does not slide off when attempting to move the object laterally by pushing down from above. This visual indication provides a perceivable affordance for being dragged, and so can sometimes be found on scroll boxes and grow or drag regions of windows. Based on such observations, several authors have suggested that the affordances of a software object are more readily perceived if it looks and behaves as it would in the real world [Carroll et al. 1988; Gaver 1991; Anderson 1993]. In other words, a mapping between physical objects and interface objects allows knowledge of real affordances to transfer via metaphor to the interface.

The interactive nature of the user interface supports the more dynamic perception of affordances as well. When the pointer is moved over a button widget in some interfaces, for example, the button becomes outlined, to indicate that an action (i.e., pressing the button) is now possible. The pointer cursor may also change shape. This form of highlighting is common for text widgets, radio buttons, and other types of widgets. A more interesting example is the behavior of a scroll box, as part of a scrolling window, when grabbed by the pointer through a mouse down event. Even if the textured surface of the box is insufficient to indicate that the widget is designed to be dragged, any random movement of the hand holding the mouse will cause the contents of the window to scroll. This kind of exploratory activity, even if accidental, provides information to the user about how the widget is meant to be used.

Though our conceptual framework does not directly address the perception of affordances, it does suggest an extension of these visual indications. For example, some widgets are meant to be used by being dragged. The box of a scrollbar, the grow region of a window, and some icons on a desktop behave in this way. The affordance for dragging can be perceived through exploration, in that, if the mouse button is held down, small random movements will suffice to show the functionality of the widget. What if, however, the user executes a single click, only the beginning and end of the necessary sequence of operators? In order to show the effects of its correct use, the widget could execute a do/undo sequence of operators. A window, for example, might momentarily shrink slightly and then return to its original size when its grow region receives a single click. A scrolling text window could scroll down and then up one line. A movable graphical object could move and then return. The important point is that the entire sequence becomes visible to the user, with the state of the system remaining unaltered. This kind of visibility has been suggested for animated interfaces, in which incorrect or illegal operators are not simply ignored, but the interface behaves in such a

way as to allow them but then undo their results.

As a final note for this section, in discussing these visual indications we must take care not to confuse the information that specifies the affordance with the affordance itself. For example, we say that button widgets have a affordance for being pressed. Notice that this is a simple affordance. The user is free to carry out a mouse down action anywhere on the screen; button presses are allowed, but not facilitated. The affordance of button widgets for being pressed is not specific to buttons, and thus the background of the screen has the identical affordance for being pressed. The difference between the two cases is that the affordance for pressing the button is visible, whereas there is nothing to indicate that a mouse down action on the background is possible or will have any useful effect.

4. CALIBRATING AFFORDANCES

At this point we have a useful framework for building affordances, but only in a qualitative sense: given a difficult operation, we can devote our design skills to facilitating its correct execution. If we observe, for example, an undesirably high frequency of errors in clicking on a button widget, we implement a sticky version of the button to capture nearby mouse clicks. But there remain a number of practical problems to be solved, including setting the range of the capture effect and deciding how it should interact with other nearby widgets.

The most direct approach, user testing, would be a massive undertaking for most of the facilitating affordances we have discussed so far. Any affordance in the interface may have a large number of parameters that need to be set correctly for its effective use. Further, these affordances often handle relatively rare events, what are sometimes considered user errors. If in a given scenario we expect an error in 5% of the cases, and an affordance may have five plausible settings for a parameter, then we may need hundreds or even thousands of trials to distinguish these settings in evaluating the affordance.

What we would like instead is a way to map out the performance space of an affordance automatically. This can be done with a programmable user model, or PUM [Young et al. 1989; Kieras and Meyer 1997]. A PUM simulates a user, to some appropriate degree of accuracy, in order to provide a designer with feedback before taking on more extensive (and expensive) usability testing. The core of our PUM will exploit one of the most obvious benefits of a planning implementation: the ability to generate courses of action automatically. We have already mentioned the planner UCPOP and given an example of the relatively abstract plans it can generate. These plans incorporate very little information about widgets beyond their names and types. Even the position of the pointer is represented in symbolic rather than numerical terms. A plan can say, "Move to the OK button, mouse down and then mouse up," but not, "Move to position (432, 624) and mouse down, pause for 200 milliseconds, then mouse up." For a PUM to use an interface, however, this level of detail may need to be specified.

This functionality is provided by a simple model of the user. The model uses Fitts' law and other robust findings from the human factors literature to estimate the performance characteristics of plan operators as they execute. Like all PUMs, our model relies on a large number of simplifying assumptions. PUMs are incomplete models for a number of reasons. Practically speaking, building a complete PUM would require building an artificial user, including cognition and sensory/motor activities, not an immediate prospect in AI and robotics research. From a theoretical viewpoint, an incomplete PUM is a natural and desirable outcome of good experiment design. By dealing with a PUM that simulates only a few, isolated factors of a real user, we are able to make better predictions about the effects of the interface on those factors.

The last piece of the picture is an accessible user interface that can run in the same computational environment as the PUM, including its internal user model and the planner. For this we use GARNET, a flexible and powerful user interface development system for Common Lisp [Myers et al. 1990].

In the following sections we will illustrate how we can use a PUM to map out the space of affordance parameters, to calibrate an affordance to a first approximation, giving us a much smaller space we need to examine in testing with real users. The task will be very simple, for the sake of exposition: clicking on buttons in a dialog box. The affordance we propose to add is sticky behavior for those buttons. Our analysis will be relatively informal, but will identify useful techniques for evaluating and calibrating an affordance.

4.1 The problem

The complexity of an intelligent agent interacting with a simulation can be broken down into four factors: the environment, the agent, its behavior in the environment, and its task [Cohen et al. 1989]. The first three of these factors form what Cohen calls the behavioral ecological triangle. For our domain this translates into the user interface (a GARNET application), the programmable user model and the activities it entails, and a specific task.

The environment. The environment is the simple dialog window shown in Figure 5. The dialog, a Motif window, can be may be moved and reshaped by dragging its borders. The list box can have different single items selected and will scroll vertically. The user can click the "OK" button or the "Cancel" button in any state. A double click interactor provides a shortcut for the "OK" confirmation.

The agent and its behavior. The agent, the PUM, is the most complex part of the system. It has two components, a plan execution algorithm and a user model.

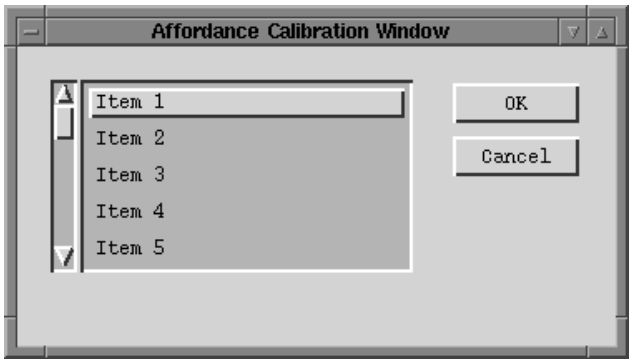


Fig. 5. A simple dialog

Plan execution proceeds as follows. A plan is an ordered sequence of operators that specify an action to be performed and the relevant objects in the enviroment. A movement operator, *Move from Background to OK-Button*, for example, specifies that the pointer should be moved from its current position on the background of the window to a position over the OK button widget. To make this operator executable we attach to it a method that computes the current location of the pointer on the window, a target location based on the position of the button widget, and moves the pointer to that target.

We also attach to each plan operator a monitoring method that evaluates its successful completion. For example, we test whether the movement operator above has completed successfully by checking whether the coordinates of the pointer are within the boundaries of the widget. Another method associated with each operator computes the duration of its execution. These two methods give us two basic performance measures for the execution of a plan: whether it completes successfully and how long it takes.

The planner executes in an open loop mode: it generates a plan to satisfy a given top-level goal and executes it without responding to the success or failure of its operators. Once the plan has been executed, the planner checks whether any operators have failed, to indicate the failure of the entire plan. If this has happened, a new plan is generated based on the current state of the interface. It begins execution, and eventually the task is completed. In general this is a poor approach to interacting with the real world, but in our domain it reflects natural behavior. At the level of detail we are modeling, users do not spend the time or effort, for example, to see whether the pointer is entirely within a widget before clicking. They point and click, and discover errors retrospectively.

Execution, success, and duration methods vary with the individual operators. Duration of pointer movement is based on a well-known finding from the psychology literature, Fitts' law [Fitts 1954]. Fitts' law is an extremely robust, empirical rule governing the movement of rapidly aimed limb movements [MacKenzie 1995]. Modern researchers use the following version:

$$T = a + b \log_2 (A/W + 1).$$

T is the total time to move a limb (e.g., a finger, an arm, foot, or mouse) to a target W units wide and A units distant. The values of the constants a and b are determined empirically for a specific task. For pointing with the mouse, $a = -0.107$ and $b = 0.223$ are appropriate [MacKenzie 1995]. Under Fitts' law conditions, subjects must make targeting movements that maintain high accuracy (a 95% target hit rate requirement is common) while minimizing the duration of the movements. We assume that execution of the *Move* operator follows this model.

We make a few additional assumptions for this execution method. First, we use the distance between the starting position of the pointer and the center of the target widget for A. Second, we assume that movement is in a straight line through the center, with no positioning error perpendicular to the axis of movement. This has the effect of changing a two-dimensional targeting problem into a single-dimensional problem amenable to Fitts' law analysis [MacKenzie 1995]. Third, we assume a symmetrical, normal distribution of positioning error along the axis of movement. This assumption of normality is consistent with other modeling efforts for similar tasks and is common in most kinds of regression modeling, though it is not entirely accurate [Meyer et al. 1988]. The mean of the distribution is the center of the target widget, and its variance is such that 95% of the distribution is contained within the borders of the widget along the axis of movement. This distance between the borders is then W. With A, W, and the constants given above, we can compute the duration of

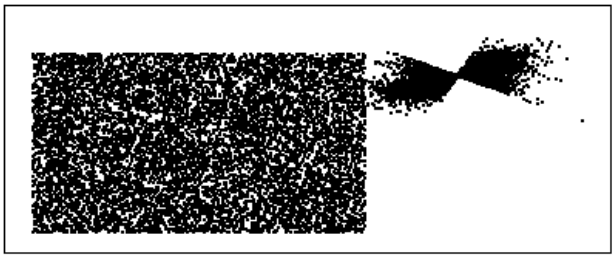


Fig. 6. Paths

the mouse movement operator. Movement distance toward the target widget is modeled by sampling from the normal distribution. In accordance with Fitts' law conditions, accuracy does not depend on the distance from the target, but is fixed in advance.

The other operators for the task we will consider have much simpler behavior. Each *Mouse down* and *Mouse up* operator takes 0.2 seconds [Eberts 1994]. If the composite click takes place on the target widget, the OK button or the Cancel button, there is no additional penalty. If the click is on the background, however, the model takes a fixed 0.5 seconds to react to lack of the expected system response, and a further 0.2 seconds for the decision to try again [Card et al. 1983]. If the click is on an incorrect button, the penalty is an arbitrary 5.0 seconds, which represents the time either to recover from an incorrect cancelation or to undo the results of an incorrect confirmation. Pointer movement is constrained to be within the border of the window, to eliminate the complexity of multiple windows and applications.

Some aspects of the PUM could be greatly refined through an analysis of specific tasks we wish to test. For our purposes, however, these rough estimates will be enough to illustrate our approach.

The task. A scenario begins with the pointer positioned at a random spot on the list box, just after the selection of an item. There are two tasks defined, the first moving to and clicking on the "OK" button, the other moving to and clicking on the "Cancel" button. Each task involves only three distinct actions: *Move*, *Mouse down*, and *Mouse up*.

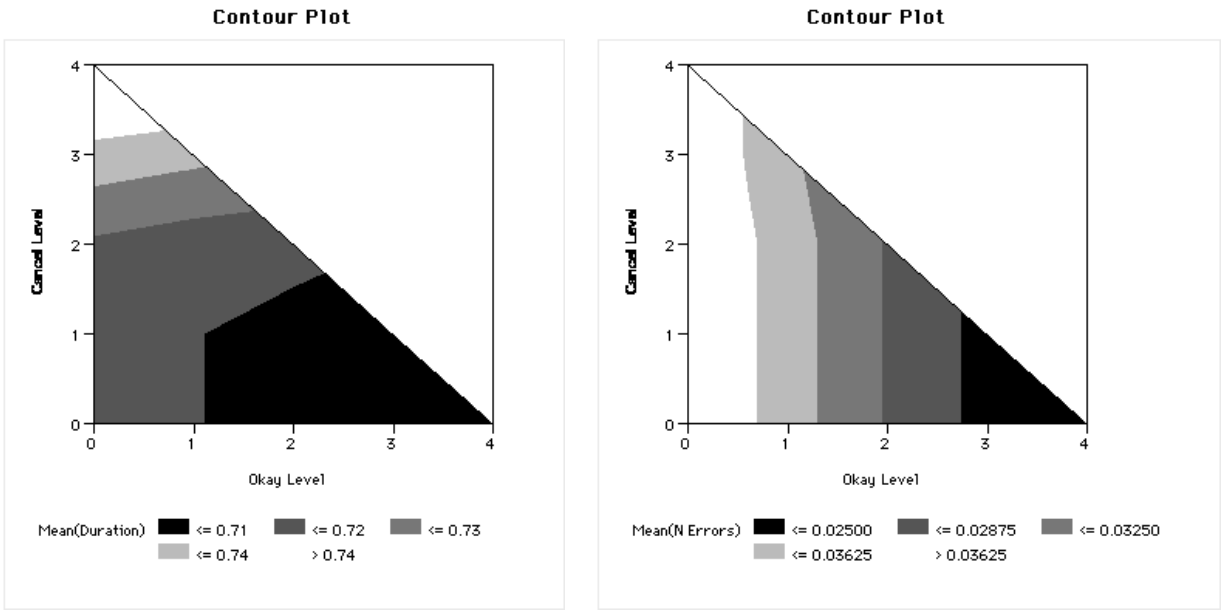
Though simple, the tasks are subject to failure in two different ways: the pointer might end up on the background of the window or its border, where a click has no effect, or the incorrect button may be clicked. The difficulties with *Mouse down* and *Mouse up* operators, discussed in an earlier section, will not arise due to the limitations of the PUM. The distribution of clicks on the OK button, starting from random positions over the list box, is shown in Figure 6.

This problem might be amenable to analytical solution, but it is at the far edge of feasibility. In addition to the non-uniform structure of the interface and its varied responses to errors in positioning the pointer, an analysis must take into account the recovery actions necessary to complete the task. A longer sequence of actions, or the consideration of more varied affordances, would be enough to render the analytical approach impossible. A solution by simulation appears to be far more viable.

4.2 Evaluation

Our affordance is a simple capture effect that activates a button if a click occurs within some short distance of its border. Suppose we add a capture affordance to both of the buttons on the task interface. We let the ratio of confirmation tasks to cancel tasks be 20:1. What should the capture range be for each button for best performance? We will measure performance in three different ways: the mean duration of the task of clicking on either button, the mean number of errors that occur, and whether a button ever incorrectly captures a click. Our evaluation of results depends critically on how we weight these performance measures. We will assume that they are all of equal importance.

We divide the area between the two buttons in the dialog box in Figure 5 into four parts. With different calibrations of its capturing affordance, each button can claim clicks that fall into some number of the four areas. This establishes a set of conditions between which we can vary the capture effect of the affordance. We vary the level of each of the two buttons: $L_o(j)$ refers to the condition in which the OK button captures clicks within any of the i ($i \leq 4$) divisions closest to the button, $L_c(j)$ likewise for the Cancel button. $L_o(0)$ and $L_c(0)$ correspond to the cases in which the OK and Cancel buttons do not capture any clicks outside



a. Duration contours

b Error contours

Fig. 7. Contours

	$L_o(0)$	$L_o(1)$	$L_o(2)$	$L_o(3)$	$L_o(4)$
$L_c(0)$	0	0	0	0	0
$L_c(1)$	0	0	0	0	-
$L_c(2)$	1	1	1	-	-
$L_c(3)$	5	5	-	-	-
$L_c(4)$	10	-	-	-	-

Table 3. Occurrences of incorrect captures

their regions. The capture effects are constrained such that there is no overlap between the buttons. We thus have a two factor, five level, incomplete factorial experiment design.

Describe models.

Figure 7a shows a contour plot of how mean duration varies with L_c and L_o. The mean duration of tasks is lowest with a larger capture range for the OK button and a small or zero capture range for the Cancel button. This is partly an effect of the relative distribution of clicking tasks; confirmation is 20 times as likely as cancelation. Changes in duration are caused mainly by errors in clicking. We can have the greatest effect in reducing the number of errors by altering the affordance for confirmation, because most errors occur during confirmation. The contour graph for errors, shown in Figure 7b, confirms this intuition. There is little to be gained in error reduction by changing the capture range for the Cancel button. Error rates remain the same. Instead, we see contour boundaries only in the differences between OK button levels. Note that the differences in mean duration between conditions are miniscule, on the order of 10^{-2} seconds. A single error, every twenty trials, contributes only a small amount (a 0.7 or 5 second penalty) to the total. The differences in error rates, however, are potentially significant. The graphs tell us that we can reduce a 5% error rate to 2% by adding the appropriate capture affordance to the OK button.

The last measure is of what we can think of as a potentially serious error: clicking the incorrect button. Table 3 shows the frequency of the occurrence during testing. Here again of the distribution of confirmation and cancelation is a factor. In contrast to the above finding that we can reduce error rates by changing the capture range of the OK button, for serious errors we must change the Cancel button.

In summary, this calibration experiment tells us that significant reduction of duration in this task is not a practical goal; the differences we see in mean duration, though statistically significant, are not detectable

in practice. If our goal is to reduce the relative percentage of errors, however, we can achieve a measurable reduction (5% to 3%) simply by setting the capture affordance to its highest level for the OK button, and eliminating any capture effect for the Cancel button. By extending the capture effect beyond the region immediately between the two buttons, we can reduce the error rate even further, to less than 1%. These changes, though small and unobtrusive, can quickly add up for mouse-intensive tasks and environments. We suspect that the reductions in error will also have a disproportionate influence on user perception of the predictability of system behavior. This last conclusion, of course, must await testing with real users.

5. CONCLUSION

Part of this article is structured as an incomplete empirical study: we have developed a tool that purports to improve design practice, and we have shown a few of the changes it suggests and a small, relatively complete example. The control condition, however, is missing: could we have hit upon these improvements without the conceptual mechanisms we have suggested? Of course. Nevertheless, because the suggestions are easily implemented and unobtrusive in operation, have obvious benefits, and yet can be found in no interface that we are aware of, our argument gains some power. We believe that the conceptual framework for affordances is responsible.

This framework gives us a better understanding of how affordances fit into the user interface. It can be used to describe existing interface mechanisms, and provides a more rigorous unification of ideas that were formerly only intuitively related. Further, it suggests new mechanisms and new ways of thinking about how to facilitate actions.

The framework also helps us quantify some aspects of affordances, a neglected area in interface research. The calibration approach outlined in the previous section helps us evaluate affordance parameters in a controlled fashion. In combination with an appropriate user model and interface testbed, we can inexpensively map out the performance characteristics of an affordance, before testing it in a real user interface with real users.

This work concentrates on affordances for low-level actions, rather than, say, affordances for learning, communication, survival, or other complex activities. One advantage of the narrow focus is a surprising generality of application. The situations we consider, mouse movements and gestures for common interface widgets, are ubiquitous in direct manipulation interfaces. Another advantage is that we have been able to ignore user cognition issues almost entirely, with the exception of the affordance calibration phase. Our future work will relax this concentration on low-level activities, to see whether the conceptual framework can be extended to handle more abstract affordances.

REFERENCES

Agre, P. E. 1993. The symbolic worldview: Reply to Vera and Simon. *Cognitive Science 17*, 61–69.

Agre, P. E. and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence* (1987).

Agre, P. E. and Chapman, D. 1990. What are plans for? *Robotics and Autonomous Systems 6*, 17–34.

Anderson, B. 1993. Graphical interfaces considered as representations of the real world: Implications of an affordances-based model. In S. S. Valenti and J. B. Pittenger Eds., *Studies in Perception and Action II* (1993), pp. 89–93. Lawrence Erlbaum.

Apple Computer. 1992. *Macintosh Human Interface Guidelines*. Addison Wesley.

Baecker, R. M., Grudin, J., Buxton, W. A. S., and Greenberg, S. 1995. *Readings in Human-Computer Interaction: Toward the Year 2000*. Morgan Kaufmann.

Card, S. K., Moran, T. P., and Newell, A. 1983. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum.

Carroll, J. M., Mack, R. L., and Kellogg, W. A. 1988. Interface metaphors and user interface design. In M. Helander Ed., *Handbook of Human-Computer Interaction*, pp. 45–65. North-Holland.

Castells, P., Szekely, P., and Salcher, E. 1997. Declarative models of presentation. In *Proceedings of the Third International Conference on Intelligent User Interfaces* (1997), pp. 137–144.

Cohen, P. R., Atkin, M., Oates, T., and Beal, C. R. 1997. Neo: Learning conceptual knowledge by sensorimotor interaction with an environment. In *Proceedings of the First International Conference on Autonomous Agents* (1997), pp. 170–177. To appear in Intelligence Magazine.

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 (Fall), 32–48.

Cohen, P. R., Oates, T., Atkin, M., and Beal, C. R. 1996. Building a baby. In *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society* (1996), pp. 518–522.

Eberts, R. E. 1994. *User Interface Design*. Prentice Hall.

Etzioni, O., Levy, H., Segal, R., and Thekkath, C. 1995. The softbot approach to OS interfaces. *IEEE Software 12*, 4, 42–51.

Etzioni, O. and Weld, D. 1994. A softbot-based interface to the Internet. *Communications of the ACM 37*, 7, 72–76.

Fikes, R. E. and Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*, 3/4, 189–208.

Fitts, P. M. 1954. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology 47*, 381–391.

Flach, J. 1995. The ecology of human machine systems: A personal history. In J. Flach, P. Hancock, J. Caird, and K. Vicente Eds., *Global perspectives on the ecology of human-machine systems*, pp. 1–13. Lawrence Erlbaum.

Flach, J., Hancock, P., Caird, J., and Vicente, K. Eds. 1995. *Global perspectives on the ecology of human-machine systems*. Lawrence Erlbaum.

Fodor, J. A. and Pylyshyn, Z. W. 1981. How direct is visual perception? some reflections on gibson's "ecological approach". *Cognition 9*, 139–196.

Gaver, W. W. 1991. Technology affordances. In *Proceedings of the CHI'91 Conference* (1991), pp. 79–84. Association for Computing Machinery.

Gibson, J. J. 1979. *The Ecological Approach to Visual Perception*. Houghton Mifflin.

Greeno, J. G. and Moore, J. L. 1993. Situativity and symbols: Reply to Vera and Simon. *Cognitive Science 17*, 49–59.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. 1986. *Robust Statistics*. John Wiley & Sons, Inc.

Hancock, P., Flach, J., Caird, J., and Vicente, K. Eds. 1995. *Local applications of the ecological approach to human-machine systems*. Lawrence Erlbaum.

Hoaglin, D. C., Mosteller, F., and Tukey, J. W. 1983. *Understanding Robust and Exploratory Data Analysis*. John Wiley & Sons, Inc.

Hutchins, E. L., Hlann, J. D., and Norman, D. A. 1986. Direct manipulation interfaces. In D. A. Norman and S. W. Draper Eds., *User Centered System Design*, pp. 87–124. Lawrence Erlbaum.

Kalish, M. 1993. Affordance learning as a problem in information integration. In S. S. Valenti and J. B. Pittenger Eds., *Studies in Perception and Action II* (1993), pp. 130–133. Lawrence Erlbaum.

Kieras, D. E. 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.

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*, 929–952.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence 33*, 65–88.

MacKenzie, I. S. 1995. Movement time prediction in human-computer interfaces. In R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg Eds., *Readings in Human-Computer Interaction: Toward the Year 2000*, pp. 483–493. Morgan Kaufmann.

MacKenzie, I. S. and Buxton, W. 1994. The prediction of pointing and dragging times in graphical user interfaces. *Interacting with Computers 6*, 213–227.

MacKenzie, I. S., Sellen, A., and Buxton, W. 1991. A comparison of input devices in elemental pointing and dragging tasks. In *Proceedings of the Human Factors Society* (1991), pp. 161–166.

Mark, L. S. 1987. Eyeheight-scaled information about affordances: A study of sitting and stair climbing. *Journal of Experimental Psychology: Human Perception and Performance 10*, 683–703.

Mark, L. S., Balliett, J. A., Craver, K. D., Douglas, S. D., and Fox, T. 1990. What an actor must do in order to perceive the affordance for sitting. *Ecological Psychology 2*, 4, 325–366.

Meyer, D. E., Abrams, R. A., Kornblum, S., Wright, C. E., and Smith, J. E. K. 1988. Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review 95*, 3, 340–370.

Münch, S. and Dillmann, R. 1997. Haptic output in multimodal user interfaces. In *Proceedings of the Third International Conference on Intelligent User Interfaces* (1997), pp. 105–112.

Myers, B. A., Giuse, D., Dannenberg, R. B., Zanden, B. V., Kosbie, D., Pervin, E., Mickish, A., , and Marchal, P. 1990. Garnet: Comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer 23*, 11.

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

Nielsen, J. and Gentner, D. 1996. The anti-mac interface. *Communications of the ACM 39*, 8 (August), 70–82.

Norman, D. A. 1988. *The Design of Everyday Things*. Doubleday.

Norman, D. A. 1991. Cognitive artifacts. In J. M. Carroll Ed., *Designing interaction: psychology at the human-computer interface*, pp. 17–38. Cambridge University Press.

Oates, T. and Cohen, P. R. 1996. Searching for structure in multiple streams of data. In *Proceedings of the Thirteenth International Conference on Machine Learning* (1996), pp. 346–354.

Penberthy, J. and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning* (1992), pp. 103–114. Morgan Kaufmann.

Rassmussen, J. 1988. A cognitive engineering approach to the modeling of decision making and its organization in process control, emergency management, CAD/CAM, office systems, and library systems. *Advances in Man-machine Systems Research 4*, 165–243.

Rich, E. and Knight, K. 1991. *Artificial Intelligence* (2nd ed.). McGraw Hill.

Russell, S. and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

St. Amant, R. and Cohen, P. R. 1998. Interaction with a mixed-initiative system for exploratory data analysis. *Knowledge-Based Systems*. In press.

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

Suchman, L. 1993. Response to Vera and Simon's situated action: A symbolic interpretation. *Cognitive Science 17*, 71–75.

Tate, A., Hendler, J., and Drummond, M. 1990. A review of AI planning techniques. In J. Allen, J. Hendler, and A. Tate Eds., *Readings in Planning*, pp. 26–49. Morgan Kaufmann.

Turvey, M. T. and Shaw, R. E. 1979. The primacy of perceiving: An ecological reformulation of perception for understanding memory. In L. G. Nilsson Ed., *Perspectives on memory research*, pp. 167–222. Lawrence Erlbaum.

Turvey, M. T., Shaw, R. E., Reed, E. S., and Mace, W. M. 1981. Ecological laws of perceiving and acting: In reply to Fodor and Pylyshyn. *Cognition 9*, 237–304.

Ullman, S. 1980. Against direct perception. *Behavioral and Brain Sciences 3*, 375–415.

Vera, A. H. and Simon, H. A. 1993a. Situated action: A symbolic interpretation. *Cognitive Science 17*, 7–48.

Vera, A. H. and Simon, H. A. 1993b. Situated action: Reply to reviewers. *Cognitive Science 17*, 77–86.

Vicente, K. 1995. A few implications of an ecological approach to human factors. In J. Flach, P. Hancock, J. Caird, and K. Vicente Eds., *Global perspectives on the ecology of human-machine systems*, pp. 54–67. Lawrence Erlbaum.

Vicente, K. J. and Rasmussen, J. 1990. The ecology of human-machine systems II: Mediating "direct perception" in complex work domains. *Ecological Psychology 2*, 3, 207–249.

Warren, W. H., Jr. 1995. Constructing an econiche. In J. Flach, P. Hancock, J. Caird, and K. Vicente Eds., *Global perspectives on the ecology of human-machine systems*, pp. 210–237. Lawrence Erlbaum.

Woods, D. D. 1991. The cognitive engineering of problem representations. In G. R. S. Weir and J. L. Alty Eds., *Human Computer Interaction and Complex Systems*, pp. 169–188. Academic Press.

Woods, D. D. and Roth, E. M. 1988. Cognitive systems engineering. In M. Helander Ed., *Handbook of Human-Computer Interaction*, pp. 3–43. North-Holland.

Worden, A., Walker, N., Bharat, K., and Hudson, S. 1997. Making computers easier for older adults to use: Area cursors and sticky icons. In *Proceedings of the CHI'97 Conference* (1997), pp. 266–271.

Young, R. M., Green, T. R. G., and Simon, T. 1989. Programmable user models for predictive evaluation of interface designs. In *Proceedings of the CHI'89 Conference* (1989), pp. 15–19.

Zaff, B. S. 1995. Designing with affordances in mind. In J. Flach, P. Hancock, J. Caird, and K. Vicente Eds., *Global perspectives on the ecology of human-machine systems*, pp. 238–272. Lawrence Erlbaum.