

Diagnostic Affordances: Heuristics for Enhancing Interface Information

Alex Paul Conn

Compaq Computer Corporation

110 Spit Brook Road

Nashua, NH 03062-2698

+1 603 884 0459

alex.conn@digital.com, alexconn@acm.org

ABSTRACT

As applications become more powerful and developers add new features, user interfaces often grow more complex. Through the use of style guides, designers have been able to achieve consistency along some important dimensions on a given platform. Yet interfaces remain confusing for many users, especially when functions they know must exist are hard to recognize, and even when found, do not seem to work. This paper introduces an affordance model based on two related interface concepts: mini-modes and diagnostics. Mini-modes are related states that must be active for a given set of operations to make sense. Diagnostic affordances provide heuristics both for displaying mini-modes unambiguously and for helping the user navigate between states so desired functions become active. This paper discusses the implications of the affordance model and presents a set of heuristics for applying it.

Keywords

User interface design, usability, affordances, diagnostics, heuristics, principles, design rationale, guidelines, mini-modes, agents, states

INTRODUCTION

User interfaces are becoming increasingly complex and powerful. The average person cannot be expected to understand even a small fraction of the possible states a given application can exhibit. Yet successful manipulation of such interfaces requires some understanding of the states or modes these applications may exhibit.

In this paper, we assert that user interfaces actually have a number of mini-modes during which certain kinds of operations and input are accepted. These mini-modes tend to be independent; if one mode is active, it does not

necessarily prevent other modes from being active simultaneously.

Without visual cues, multiple concurrently active mini-modes can cause serious confusion to average users. Such visual cues already exist in many well-known applications today. However, the implementations of these visual cues are inconsistent because developers do not have heuristics for consistent use of graphical user interface (GUI) components. As applications become "web-enabled," the same user interface issues will apply not only to browsers, but also to the web pages—including applets, controls, and plug-ins.

This paper explains the concept of affordances, which provides the key for evaluating the use of GUI components and developing a set of consistent heuristics.

THE PROBLEM AND RELATED WORKS

An affordance is information that provides clues on what something can do or how to use something without having to try it (or be familiar with it). An affordance can be visual, auditory, tactile, or even olfactory. There are affordances in everyday life, as well as computer displays and other high-technology devices. For example, a door handle provides an affordance if it visibly indicates whether somebody should open the door by pulling, pushing, or sliding. Another affordance for a door might tell visitors enough information that they do not try to open the door (for example, an occupied sign on an airline restroom door).

The concept of affordance was first introduced by Gibson [3]. Don Norman popularized the concept [7] and William Gaver applied it to computer interfaces [2]. According to Gaver, affordances exist whether the user cares about them or not. "Making affordances perceptible is one approach to designing easily-used systems."

While Norman, Gaver, and others provide a strong conceptual framework, no clear-cut affordance model with heuristics has emerged for applying and evaluating affordances in human interfaces. In addition, while Fitzmaurice [4] and others have discussed the problems

with mode affordances, the ideas of mini-modes and diagnostics have not been formally tied to affordances.

In an earlier paper, we defined an affordance model for time delay in interfaces [1]. A "time affordance" captures the important properties of a time delay on a computer, and tells the controller (user or computer) what to expect and how to proceed. It is a "gestalt" indication of whether "everything is proceeding properly" or "something may be wrong." This paper places time affordances in the context of the broader diagnostic-affordance framework.

THE AFFORDANCE MODEL

We define an affordance model based on mini-modes and consisting of eight components. These mini-modes define the states in which the operations identified by the affordances become available.

Mini-Modes

Icons portray functions that might be available under some circumstances. The trouble is, it is often hard to tell when or why such circumstances occur. Sometimes icons or menu items are grayed out, presumably because they are unavailable (feature is not installed, object is not selected, file is not open, etc.). Without additional help, there are too many possibilities for many users to deal with.

A particular function is usually unavailable because the computer program is not in a state where that function would make sense. Where two or more states are related because their operations or functions are naturally tied together, we define the set of states to be a "**mini-mode**." For example, text in a box often has (at least) two states associated with it: (1) the state in which users can enter or modify the text, and (2) the state in which users can modify attributes of the box and/or the text (size, shape, color, etc.).

The fact that the computer might be in a mini-mode is of no use to users unless they can recognize the mini-mode and distinguish it from other related mini-modes. We believe that appropriately designed affordances are key to portraying such mini-modes.

The affordances that indicate some mini-modes are familiar to most users, such as the reshaping handles on a selected object. Other mini-modes are harder to discern, such as the text entry versus text box attribute mode in Microsoft PowerPoint. Many users fail to see that the bounding box changes from a diagonal (text entry) to a hash pattern (text box). If an application must be in a particular mini-mode for certain operations to be available or make sense, that correlation must be very clear to ensure the interface is not confusing or inconsistent to users.

A well-designed affordance can provide the important correlation between a mini-mode and available functions. For example, the reshape handles can provide an affordance that says to the user "you are in a mini-mode in which an object has been selected and you can change its properties." The flashing I-beam cursor provides an affordance that says, "you are in text entry mode." The consistent use of affordances can significantly help the user recognize and distinguish mini-modes. Thus, the usability of GUI components is directly related to how well they conform to a consistent affordance model.

Note that mini-modes do not "own" functions. A particular function might be available (make sense) in more than one mini-mode (e.g., changing point size for whole text box as well as for a selected word during text input mode). The affordance indicator might be a special element (bounding box, pointer change, etc.) or it might be a group of related icons (e.g., text alignment).

Affordance Components

Every affordance will consist of some or all of the following components:

1. Existence: Indicates the existence of a function, operation, or potential state (for example, an icon on a task bar or the task bar itself, if it appears as a result of a mini-mode).

The existence component is the affordance "syntax"; that is, the user will perceive but not necessarily recognize the entity (e.g., the metaphor is not clear). Perception might not always be visual (for example, the dot on the 5 of a numeric keypad for orientation is usually felt rather than seen).

2. Purpose: Indicates (usually visually) both the purpose and use of a function, operation, or state. Purpose is usually indicated by text, such as font name, shape, icon, or associated bubble or task bar help. Purpose is also indicated by layout, such as an index on the left side of Web pages.

The purpose component is the affordance "semantics." A user must recognize some if not all of the meaning of an affordance for it to be useful. The quality of the affordance is based in part on the fidelity of the inferred meaning to the intended meaning.

3. Availability: Indicates whether the user can activate the function, operation, or potential state in the current context (for example, grayed out if not available).

An application can show availability with a pointer change, such as the browser changing to a finger over a hyperlink. Availability is most strongly indicated if there is a diagnostic component associated with it, such as bubble help. For example, if the user holds the pointer next to the grayed-out undo button, help might say "nothing to undo."

Many GUI elements exhibit "positional stability" to minimize surprises for users. That is, an icon stays in the same position whether available or grayed out. Such visual on/off cues help to identify mini-modes.

4. Activation: Indicates (visual, auditory, tactile, etc.) the system recognizes or accepts the selection.

Often a button is animated to look as though it is being pressed. Auditory aids (beeps or clicks) may be needed if the feel is uncertain (e.g., such as the membrane interface on a microwave oven or the surface of a touchpad).

There is a close relationship of activation to time affordances. If there is no visible time affordance (e.g., hourglass) and yet a button will not cancel or a hyperlink will not click, the user is likely to assume a malfunction and take "appropriate" action.

5. Acceptance: Displays the current state of an attribute or mini-mode (for example, selected or not selected).

State may be inferred by a clustering of affordances. For example, the text alignment icons in Microsoft Office function like radio buttons (only one can be pressed at a time) and together indicate state.

If there are other obvious indications of state, the acceptance component may not be needed. Thus, a button to activate a separate application need not remain pressed (unless clicking again is the recommended way of deactivating the application).

6. Default: Displays current default properties of an attribute or state. A default visually indicates stickiness attributes the user can change, such as text color.

Default is a particularly valuable component of an affordance, telling the user which of several possibilities will occur if the element is activated.

7. Guarantee: Assures the user that what the affordance says is really true. This may involve controls over who gets to present the affordance, especially in secure operations, such as the locked padlock a browser displays.

Guarantee is a quality measure. If an affordance is not available but appears to be, the user learns not to trust the interface fully. While animation during activation can help, some affordances cause events that cannot easily be verified (such as sending mail or a fax). The visible addition of an entry in the sent log can help reassure the user.

8. Diagnostic: Provides additional information about the availability or state of an affordance.

The diagnostic component is often the key to helping users decipher where they are and why a function/state is

unavailable. Ideally, the diagnosis provides both an explanation and a way to find out more (such as how to fix or change the state/mode).

The diagnostic may indicate the effect of the operation without forcing the user to try it, making the user interface less threatening to a user.

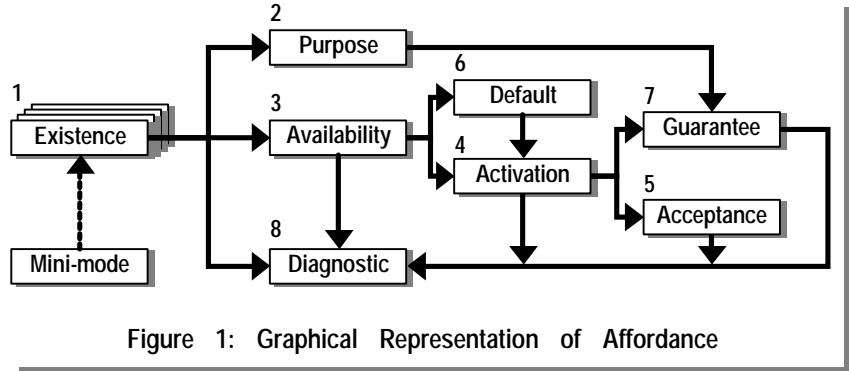


Figure 1: Graphical Representation of Affordance

Diagnostics may exist as bubble or taskbar phrases, or they can be incorporated into context-sensitive help. For example, the diagnostic associated with a grayed out floppy icon may say "Save [no changes yet]."

Affordance Relationships

Figure 1 shows the affordance model components and their relationships. The dashed line from *mini-modes* to *existence* illustrates that a mini-mode may result in the appearance of an affordance (such as a toolbar appearing). In a properly structured affordance, if an affordance exists, it should convey a *purpose*, indicate whether the affordance is *available*, and provide *diagnostics* (e.g., bubble help).

The *availability* component may illustrate a *default* state (e.g., current color if pressed), cause *activation* if selected (pressed) and provide contextual *diagnostic* information about what will happen upon activation (e.g., what will the undo button cause to happen?).

For critical affordances, such as those associated with security, the *guarantee* component can provide additional assurance that the state is as displayed. For example, clicking on the locked padlock in a browser results in a diagnostic component that displays X.509 v3 certificate information.

When a user attempts to *activate* an affordance, ideally there is a visual and/or audible indication that the computer recognizes the action (e.g., an animated button press). Activation may result in additional information that *guarantees* the expected state change (e.g., the advisory security window in a browser). Sometimes, the button clicks but nothing seems to happen. If in fact activation took place (something happened), there should be visual cues indicating transition to the *acceptance* state. If nothing happened, *diagnostic* information should

provide help. However, a properly designed affordance should not appear available if it cannot be activated.

Acceptance may cause the system to enter a state in which significant time elapses before additional input is accepted. A time affordance [1] would provide ongoing *diagnostic* information. During the time affordance, the system should properly portray unavailable affordances (for example, gray out icons, hide a toolbar, etc.)

Relationship of the Affordance Model to Style Guides

Style guides promote or enforce consistent behavior across an application, framework, or platform. However, they might not enforce consistency across a given dimension, such as affordances, unless they are developed with the concept of affordances in mind. Thus, once affordances are understood, style guides can enforce them.

APPLICATION OF THE AFFORDANCE MODEL THROUGH SCENARIOS

The following three actual scenarios demonstrate affordance issues where one or more affordance components are missing or inadequately presented to the user. These scenarios help illustrate components of the affordance model.

Scenario 1

Situation: A user is trying to use a computer program for scoring competitive synchronized swimming. The program requires the user to press a **T** to enter data on technical routines and to press the **Enter key** to enter data on free routines. The resulting displays are essentially identical. Because the program lacks clear affordances, past users were often confused when entering meet data, sometimes "losing" data because they entered it in the wrong category without realizing it.

Analysis: This year, the programmer changed the entry screens so that rather than depending on a keystroke to distinguish free routines from technical routines, the screen layout itself provides this information. As shown in **Figure 3**, there are now two visual groupings that clearly separate the free routines from the technical routine entry areas. In addition, highlighting of areas where entries exist allows the user to spot mistakes easily. For example, one of the TECH DUET 14-15 entries was inadvertently entered as a FREE DUET 14-15. The highlighting makes the mistaken entry stand out.

By providing two distinct groupings, both visible at once, the program told the user of both free and technical routine **existence**, **purpose**, and **availability**. By highlighting entries, the program supplied valuable **diagnostic** information so the user could spot mistakes. **Activation** and **acceptance** are already in the program logic.

Scenario 2

Situation: A user of PowerPoint is attempting to draw a diagram with various shapes of boxes. Each box has associated text. The user first draws a box, then enters text in a separate text box, and finally painstakingly aligns the two boxes. After having to resize some of the diagram, the user is frustrated because the text is no longer aligned correctly with those boxes. There must be a better way!

Analysis: Microsoft PowerPoint users often miss subtle mini-mode cues, so that the **existence** of affordances is not fully recognized. Many shapes in PowerPoint are considered text "boxes." Further, text placed into an object rather than a separate overlaid text box does not require separate alignment and retains proper positioning even when the object is resized. The problem is that the reshaping handles used as the affordance to tell that an object is active give no hint as to potential use for text (the **purpose**).

We created an "Affordance" toolbar in PowerPoint for testing affordances by highlighting **mini-modes**. The toolbar contains icons grouped to illustrate mini-modes using existing PowerPoint icons. **Figure 2** shows the top of that affordance toolbar along with text boxes for two possible selected states: text edit **mini-mode** and text box selected **mini-mode**.

When a user first draws an AutoShape in PowerPoint, a third **mini-mode** is displayed in which the shape is designated only by solid outlines and not surrounded by a rectangular hashed box. The moment the user enters any text, the hashed outline appears (and it never disappears even if all the text is deleted). Clearly, the better affordance would be to have the text input feature of AutoShapes always be visible (to clearly illustrate **availability**) when the object is selected, so the average user would understand the potential for text input. **Figure 2** also shows each text box state and how the various grayed out regions in the Affordances toolbox quickly become "indicator lights" for mini-modes.

The PowerPoint developers have missed additional opportunities for enhancing affordances. For example, the symbols for "bring to front" or "bring forward" and their counterparts do not provide as rich an affordance as is possible. While the set does gray out when no object is selected (not shown), the designers missed the chance to gray out the corresponding buttons if the selected object is already as far forward as possible. At that position the **availability** component of the affordance is no longer present. (**Figure 4** shows the recommended icons when the user cannot move the object any further forward).

The buttons for selecting fill color, line color, text color, etc. also only partially capture the potential affordance. While they do provide the **default** color state (the color

that will be used if the center of the button is pressed), they appear to be always **available**, simply beeping at the user (there is no **diagnostic**) if no object is selected for which the given button could apply. **Figure 5** provides a recommended change that shows **default** without indicating **availability**.

Figure 6 shows yet another way to illustrate **default** styles and fonts in otherwise inactive affordances.

Scenario 3

Situation: A Web page for purchasing and downloading software does not use the visual affordances for security (such as a padlock). Users progress through various browser screens, choosing the software version that is compatible with their operating system. A pop-up window appears asking for credit card information. The dialog box claims to guarantee security; yet, the visual affordance for security (padlock) on the browser is not visibly set (locked). Many users fail to purchase the software because the normally recognized security affordance (locked padlock) is not present.

A user who is not as concerned with security enters the credit card as requested. On the resulting web page, the user expects to see a download screen, but the user cannot find any hyperlink for downloading. The user does not notice that the browser display is unusually wide. By scrolling to the right, the user would have seen a download button. Instead, after trying several links and finding no clear way to download, the user calls the company in frustration.

Analysis: There are two fundamental parts to this web-based scenario: the security lock and the download button. In the World Wide Web, there is a de facto standard in which a key or lock is used to portray security. The **existence** of a key or lock is now recognized by most users to indicate security status. When the key becomes solid or the padlock locks, users may assume that its **purpose** is to visually indicate that the browser security is in effect—especially if they read the advisory dialog box that explains the secure state.

Although most users do not realize it, the security icon is **available** for selection. If the user clicks on the icon, it will display the security certificates and other **diagnostic** status information. Most important, users also assume that the padlock affordance is a **guarantee** that the appropriate security mechanisms have been reliably executed (i.e., SSL, the secure socket layer).

In this scenario, the Web developers should have used the standard mechanisms to retain the all-important security affordance. In a complex frame environment where the key/lock affordance does not always work properly, it may be necessary to spawn a new frameless browser window that is tied specifically to the sensitive information.

In this scenario, the inability of the user to spot the download button illustrates an important affordance principle in Web design: assume the user is set up with a standard-sized browser window. The **existence, purpose, and activation** of important functions (usually with an HTML forms submit button) should be visible in this initial window. In this case, the infrequently needed horizontal scrollbar exacerbated the problem.

DISCUSSION: IMPLICATIONS OF THE MODEL

Adding Diagnostics

Recent releases of interfaces have begun to provide the beginnings of diagnostic affordances. Even the early Mosaic browsers provided an effective use of graying out in the forward and back buttons to indicate the existence of additional URLs on the list in the respective directions. In recent browser releases, such as Netscape Communicator Version 4.0, if the user positions the graphics pointer over the back or forward buttons, bubble help will give the title of the Web page that will be retrieved if the button is clicked. Such information is a useful addition to the affordance. Microsoft employs similar “active bubble help” in Windows NT 4.0 Explorer.

The next step is to provide help for those buttons that are grayed out. The diagnostic engine, perhaps in concert with a user interface agent, could provide the short explanations of why the operation is not available (for example, active only in 3D mode). With sufficient context information, the system could remember what previous selection caused the state to be active. It might even be possible to allow the user to carry out a special activation of an inactive function (such as a control-click) causing it to activate that state that it was previously in.

Recognizing and Designing Mini-Modes

Current modeling practice places considerable emphasis on the use of state diagrams in analysis and design. Recent nested state diagramming techniques, introduced by David Harel [5] and described by Rumbaugh [10] and in the Unified Modeling Language [8], show the use of nested states for capturing complex systems. Using this approach, designers can model the relatively independent mini-modes in today’s applications. Once they fully specify interactions between mini-modes, designers can determine how many states a given interface component might need. Where testing reveals that an interface component cannot support the number of states proposed, designers might use two components instead. For example, it may turn out that users respond better to separate indications of whether a microphone is functioning and whether the microphone is active. Designers should carry out state diagram analysis and testing to determine whether the affordances are recognizable to the average user and whether they are consistent.

Affordance versus Help and Training

An affordance is not a substitute for help and/or training. Icons that can display the presence or absence of a mini-mode depend on the user's recognition of the icon and what it represents. Just as the affordance of Don Norman's modified light switch panel depends on the user's recognition of the purpose of a light switch, so too the affordance offered by the icon group for text alignment depends on the user's familiarity with text formatting.

HEURISTICS

In the course of our work, we have derived the following heuristics:

Availability of affordances:

- **If lack of readiness is unknown, show affordance as available until first try.** Example: a scanner is shown as available until the system cannot find one. Then the icon is grayed out.
- **Show affordance as unavailable when the end of a range is reached.** Example: gray out the bring-to-front icon if the object is already at the front. See **Figure 3**.
- **Show affordance as unavailable when a singular event is completed.** Example: gray out floppy icon when save is completed and nothing has changed.
- **Show all icons as unavailable until application is really ready to accept input.** Example: if a table is not selected, none of the table modification icons should look available.
- **If actions are unavailable, tell user why and how to make them available.** Example: In the "delete columns" entry in the table menu, add bubble help that says "select column to activate" to explain why the icon is grayed out.
- **Show affordances as available only if users can activate them.** Example: in a Web page, do not make an image look like it can be clicked on if it cannot. Do not underline anything except links. On navigation bars, do not make a click go to the same page as you are on (unless it is an anchor on the same page).

Use mini-modes consistently.

- **Cluster icons that relate to the same mini-mode or attribute.** Example: group all icons having to do with text alignment, table manipulations, etc.
- **Place affordances in view by default whenever their corresponding mini-mode is visible.** Show less frequently used affordances when mini-mode is activated. Example: in Microsoft Office, during picture edit, a special toolbar appears with image editing icons.

- **Place important Web page activation buttons within the initial view of a standard browser window.** Assume that many of your users have their browsers sized to fit a VGA screen and will not necessarily see or use scroll bars—especially horizontal ones.
- **Make sure mini-mode affordances adhere to emerging de facto standards.** Example: what looks like reshape handles should enable the reshape operation.
- **Define or describe mini-modes in help dialog boxes.** Include how to turn them on and how to recognize that they are present.

Allow users to customize toolbars to group affordances and to create new affordances.

- **Retain affordance customizations across platforms and upgrades. Provide a method for importing/ exporting affordances from one computer to another.** Customizations should be in a file similar to templates. The user should be able to point to a centrally located customization file (e.g., on a server) or ensure that the same file is available for all computers used by the user. Even without an import/export mechanism, a user should be able to customize affordances in an application once for use on several platforms.

Indicate when a state is active or on.

- **Show a button as pressed if the attribute is active.** Example: the bold text button in Microsoft Office.
- **Make different states easily distinguishable.** Example: a failed search or inexact match should look noticeably different from a successful one.
- **Make the affordances for related mini-modes easily distinguishable.** Example: in Microsoft Office, make the bounding box modes that distinguish between text entry and box attributes more easily distinguishable.
- **Indicate whenever there is a default or "sticky" setting, and allow users to control stickiness whenever appropriate.** Call out changes from the defaults that are due to stickiness. Example: the standard print icon should not print 10 copies left over from the previous setting without a clear indication.
- **Indicate what will happen if the affordance is activated.** Example: show color, style, font, pitch, etc. of text and show what the font will look like using a preview, if possible.

Use consistent affordance paradigms in the same application.

Example: in Microsoft Word, "OVR" means overstrike mode (as opposed to insert mode). When the mode is

insert, the user sees a grayed out overstrike. That implies that the mode is not available at this time, which is wrong. Ideally, it should follow the lead set by other icons in Word: it should be a button that when pressed goes into overstrike mode. It should look pressed and not pressed when the user toggles the Insert key.

Use cooperative or coupled affordances to help reduce ambiguity.

Example: the shape handles of close proximity objects may overlap. If the pointer changes to a text I-bar, this may identify the box having text as the one that was selected. Thus, the pointer affordance “cooperates” with the selection affordance to help decipher “on canvas” layout ambiguities.

ADDITIONAL DISCUSSION OF AFFORDANCES

It is all too easy to trivialize the affordance problem as one of icon design and minor GUI capabilities. Yet the real problem is one of a complex state machine in which a relatively limited number of mini-modes, once recognized, are the key to simplifying the interface for the user. In other words, the mini-modes can be seen as a much simpler state machine. Each mini-mode state accepts only input from components that are “lit up” when the mode is active. How to transition from one state to the next is also part of the affordance. A fully realized affordance is able to explain which operations are associated with which mini-mode(s) and can thus provide the all-important diagnostic help for a user to navigate the interface.

Many other aspects of the user interface exhibit some characteristics of an affordance. When a program requests a directory path name, the browse button is essentially a diagnostic affordance (give me valid choices). Mail programs that have directories provide diagnostic affordances by checking names either by command completion or a mechanism similar to a real-time spell checker.

Graphic pointers (mice) have for years changed shape as an affordance, indicating whether the pointer is properly positioned to resize an object or whether a drag and drop operation is valid. Drag and drop could be enhanced with a diagnostic capability indicating why a given target is not considered acceptable (e.g., no converter available). Internet browsers have standardized on a representation of a finger to indicate that the underlined text or the graphic is a URL link.

As agent technology continues to take off, having a diagnostic affordance custom tailored to a user becomes a possibility. For example, an interface agent [6] could anticipate why a user is trying to use inactive operations and activate the most recent state that used them or provide a choice. Users of complex interfaces in which operations are aggregated into composite forms would

benefit from an autonomous interface agent. This agent would clearly portray what is feasible and provide feedback for operations that look possible but in fact cannot be carried out. Part of a diagnostic affordance might be, in agent terminology, a discourse segment [9], in which the pause or special click signals the agent to take the initiative and seek an explanation (diagnosis) for the unavailable operation.

CONCLUSIONS AND FUTURE WORK

This paper has presented an affordance model based on eight components and mini-modes. This model appears to be very effective for analyzing and characterizing the user interfaces of today’s applications. The paper illustrated the application of the model in three familiar scenarios and discussed the implications of the model. Finally, it presented a set of affordance heuristics for applying the model.

Future work should explore mini-modes in more detail. How can mini-modes be more effectively displayed to users? What are the heuristics for arranging and evaluating related affordances? Are there mini-modes that are common across various applications and suites? Research into the eight affordance components could result in important aspects of each component, defining for example, the characteristics of effective diagnostics or guarantees. Finally, empirical studies can both validate and extend the model and the heuristics.

ACKNOWLEDGMENTS

We would like to thank members of the Compaq Services NSIS Technology Group for their various affordance ideas, particularly Sri Raghavan for help with conceptual organization, Dennis Wixon for ideas on mini-modes, and Pat Srite for editing help.

REFERENCES

1. Conn, Alex Paul, “Time Affordances: The Time Factor in Diagnostic Usability Heuristics,” CHI ‘95 Proceedings, pp. 186-193.
2. Gaver, William W., “Technology Affordances,” CHI ‘91 Proceedings, pp. 79-84.
3. Gibson, J. J., *The Ecological Approach to Perception*, Houghton Mifflin, Boston, MA, 1979.
4. Fitzmaurice, George W., *Graspable User Interfaces*, Doctoral Dissertation, University of Toronto, 1996. <http://www.dgp.toronto.edu/people/GeorgeFitzmaurice/thesis/Thesis.gf.html>
5. Harel, David & Eran Gery, “Executable Object Modeling with Statecharts,” February, 1997, section 7.
6. Lieberman, Henry, *Autonomous Interface Agents*, CHI ‘97 Proceedings. (<http://www.acm.org/pubs/articles/proceedings/chi/258549/p67-lieberman/p67-lieberman.pdf>)

7. Norman, Donald A., *The Psychology of Everyday Things*, Basic Books, New York, 1988.
8. Rational Software Corporation: Documents on UML (the Unified Modeling Language), *UML Notation Guide*, Chapter 8, State Diagram, section 8.7
9. Rich, Charles and Candace L. Sidner, "Adding a Collaborative Agent to Graphical User Interfaces," Ninth Annual Symposium on User Interface Software and Technology (UIST'96), Seattle, Washington. (<http://www.merl.com/reports/TR96-11/index.html>)
10. Rumbaugh, James and others, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991, pp. 94-98.

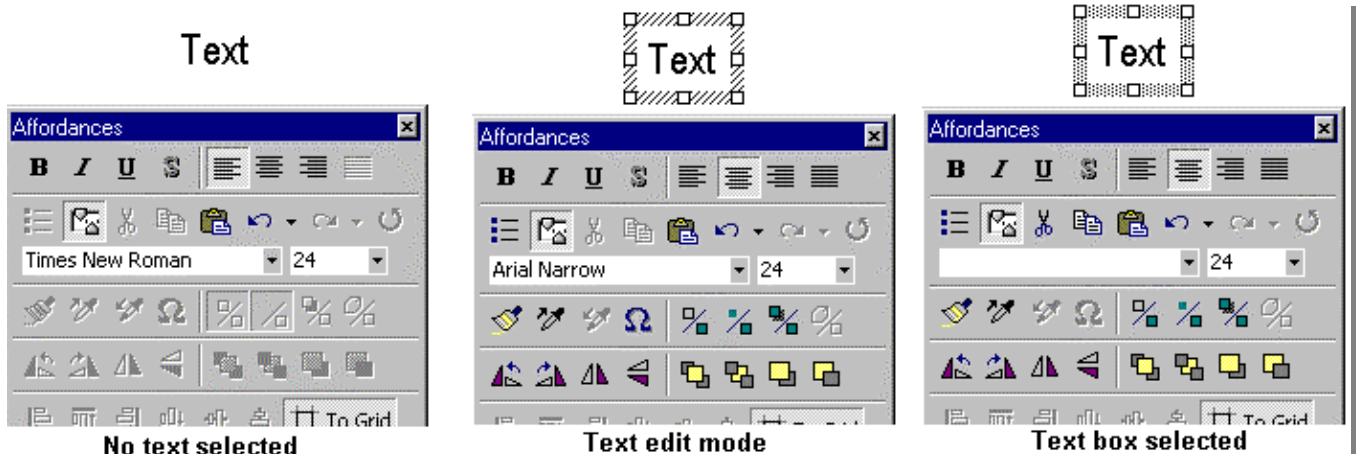


Figure 2: Text portions of the affordance toolbar with no text selected (left), text edit mode with cursor flashing (middle) and text object manipulation mode (right), in which any operations impact all of the text in the object. Note the lack of an affordance to indicate the **default** font (right) even though text was in fact a single font and could have been changed at this point. See **Figure 6**.

HI-LIGHTED	Routine names have routines entered
FREE SOLO	Routines: 11-UN 12-13 14-15 16-17
FREE DUET	Routines: 11-UN 12-13 14-15 16-17
FREE TRIO	Routines: 11-UN 12-13 14-15 16-17
FREE TEAM	Routines: 11-UN 12-13 14-15 16-17
TECH SOLO	Routines: 11-UN 12-13 14-15 16-17
TECH DUET	Routines: 11-UN 12-13 14-15 16-17
TECH TRIO	Routines: 11-UN 12-13 14-15 16-17
TECH TEAM	Routines: 11-UN 12-13 14-15 16-17

Figure 3: New version of DOS synchronized swim scoring program has a diagnostic affordance using highlighting. Entry personnel can easily spot data entry errors.

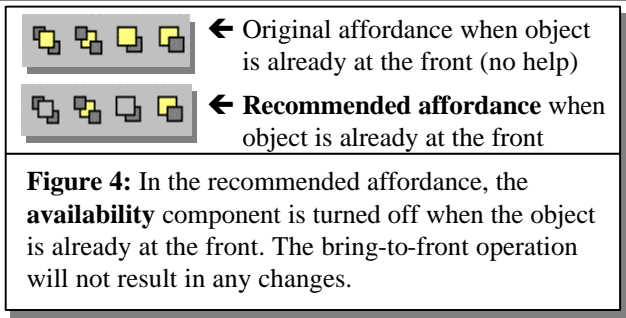


Figure 4: In the recommended affordance, the **availability** component is turned off when the object is already at the front. The bring-to-front operation will not result in any changes.

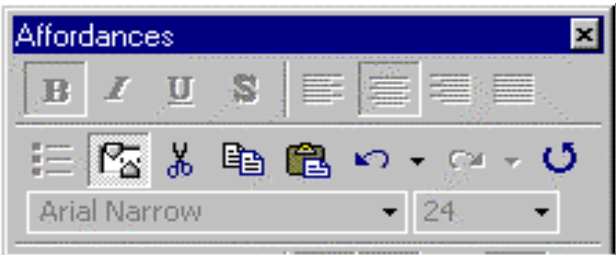


Figure 6: In the recommended affordance for **Figure 2** (right), the **default** font (Arial Narrow) shows up when the text entry state is inactive.

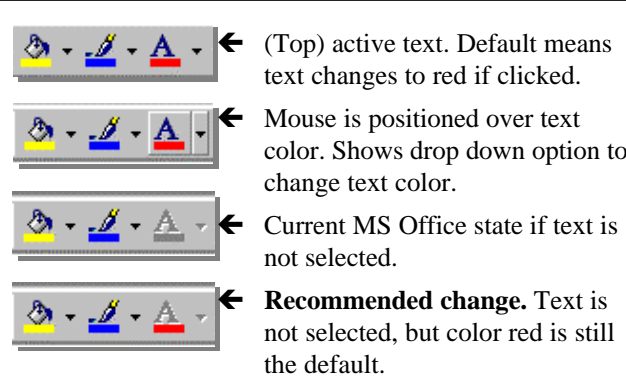


Figure 5: In the Microsoft Office draw capability, the attempt to illustrate **default** interferes with the **availability** affordance component. Clicking on the apparently available font color icon causes the computer to simply beep in error.