ISSUES FOR PROXIMATE USER INTERFACES

Ken Pier

Information Sciences and Technology Laboratory
Xerox PARC
3333 Coyote Hill Rd.
Palo Alto, CA 94304
(415) 812-4861
pier@parc.xerox.com

James A. Landay

School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
(412) 268-3564
landay@cs.cmu.edu

Abstract

In the future world of ubiquitous computing, interactive devices ranging in size from pocket to wall will be strewn about the user's environment. Display and digitizer surfaces will be merged so that users appear to interact directly on display surfaces using an electronic pen or touchscreen. Within some size range of these devices, and for some tasks, traditional "widget" style interfaces may suffice. However, for devices at the extremes of the size range, such as the Xerox Liveboard, widget style interfaces break down. When interacting directly on a very large display surface, static interface widgets will be so situated as to be useless, out of reach on the opposite side of the board, near the ceiling, or near the floor. In this paper, we discuss issues and solutions to this problem and present a prototype example of a *proximate user interface.*

Keywords

User Interface, Interaction Techniques, Stylus, Pen, Toolkits.

1 Introduction

In the future world of ubiquitous computing [29], interactive devices ranging in size from pocket to wall will be strewn about the user's environment, be it office or home. Display and digitizer surfaces will be merged so that users appear to interact directly on the display surface using an electronic stylus (pen) or touchscreen. Within some size range of these devices, and for some tasks, traditional "widget" style interfaces may suffice. However, for *outsize* devices, devices at the extremes of the size range, such as the Xerox ParcTab in Photo 1 and the Xerox Liveboard [6] in Photo 2, widget style interfaces break down. The elements of interfaces required for outsize devices are different than the interface elements in use today.

Existing stylus-based interfaces [5, 16, 19] have made some strides in addressing the needs of users in a stylus-centric interface, but they have an unstated underlying assumption; that is, that they are in use on traditional size display and interaction surfaces, whose diagonal dimension approximates the human forearm in length. As soon as interaction surfaces get quite small (ParcTab) or quite large (say, twice the size of the existing Xerox Liveboard: the 2X Liveboard), problems arise. The problems stem from the fact that current user interfaces contain *static* interaction elements, that is, continuously displayed elements such as pull-down menu bars, buttons, sliders, and scroll bars, in static positions on screen. But there is no room for static, continuously displayed elements on a ParcTab. On a 2X Liveboard, in Photo 3, there is room for static elements but they will be so situated as to be useless, out of reach on the opposite side of the board, near the ceiling, or near the floor.

For example, if we display a Macintosh screen on an existing Liveboard, its pull-down menus will be above the reach of many, and above the reach of all on a 2X Liveboard. If we run the GO Corp. stylus-based Notebook User Interface [5] on a 2X Liveboard, the screen image will scale up, thanks to ImagePoint resolution-independent imaging, but the BookShelf, a fixed palette of icons at the bottom of the screen, will be below our knees, and the pull-down menu bar above our reach. A potential solution, tearing off these static interface items, only complicates the problem by forcing the user to manage these interface elements as well as the application objects necessary for the user's task.

Photo 1: ParcTab

Photo 2: Actual Xerox Liveboard

Photo 3: 2X Liveboard mock up

Thus, we propose the idea of a *proximate user interface* [1] that has no interface elements in fixed locations. Two means of interaction that can fulfill the requirements of proximate user interfaces are the electronic stylus combined with non-static interface elements and the human voice. With the stylus, user interface elements flow out of the stylus tip at the user's command and are invisible unless they are immediately in use. Use of voice requires only that the input generated by the vocal tract be received by a microphone, recognized in real time, and that recognition results be quickly transmitted to the computing device at hand.

Examples of proximate interface elements for the stylus include pie menus [4], hierarchical pie menus [9], pop-up menus, pop-up property sheets [11], pop-up handwriting recognition pads [5], and gesture-based interfaces [5, 12, 23]. Vocal user interface elements have been demonstrated by Schmandt [25], Bush and Wilcox [30], and are available in commercial speech-based user interface builders [10].

This paper addresses issues raised by proximate user interfaces on a large display surface as exemplified by the Xerox Liveboard. We summarize related work both early and contemporary, enumerate some of the challenges raised by proximate user interfaces, discuss the problem of *interfering applications,* and finally relate our first experience in attempting to build and use an application with a proximate user interface.

[1] This was referred to as a location-independent interface in an earlier technical report: Pier, K. & Landay, J. A., "Issues for Location-Independent Interfaces." Technical Report ISTL92-4, Xerox Palo Alto Research Center, Dec. 1992.

2 Related Work

Many of the widgets that we consider proximate interface elements were proposed years ago, yet have seen little use in modern user interfaces. The main reason is that the problems encountered on outsize devices, like the Liveboard, do not occur on standard workstations. In addition, the recent popularity of pen-based computing has resulted in several new interface styles based on the stylus. In the next two subsections we give an overview of both the earlier work and the more recent stylus-based interfaces.

2.1 Early Proximate User Interfaces

Many of the early proximate user interface elements can be characterized as forms of "pop-up" menus. These include the movable menus and pie menus described below.

2.1.1 Movable Menu

Proximate user interfaces date back to 1969, when Wiseman [31] first proposed the idea of a "movable menu" displayed close to the cursor and appearing only when the user needs it.

2.1.2 Markup at Xerox PARC

In 1976, in the Markup [20] graphical editing program on the Xerox Alto [28], a mouse button was reserved to pop up a single large icon-based menu, part of which is shown in Figure 1, centered around the display cursor. The user could then invoke an operation from that menu by releasing the mouse button over the desired function icon. The menu subsequently disappeared.

Figure 1: Markup menu

2.1.3 Pie Menus at the University of Maryland

In 1986, Don Hopkins [4] produced a pie menu interface element for the X10 window system. Implementations for Sunview and NeWS window systems were produced in subsequent years. Figure 2 illustrates a pie menu. Pie menus pop up under the cursor in response to a particular mouse button press, or a mouse button press without cursor motion for a short period of time (i.e., press and hold still). The advantage of the pie menu over the linear pop-up menu is that all of the choices in the pie wedges are equally close to the cursor when the menu appears, and very little cursor motion is needed to move into a pie wedge. In the linear menu, the choice may be all the way at the end of the list and, if the list is even moderately long, choosing the last entry can become tedious.

Figure 2: Pie menu

2.2 Modern Stylus-based User Interfaces

There have been three major attempts at producing commercial systems with stylus-based user interfaces. The systems produced by GO, Microsoft, and Momenta are discussed below.

2.2.1 GO

The stylus-based Notebook User Interface by the GO Corp. [5] is nearly entirely gesture based. There are about a dozen basic gestures for navigating through and displaying documents (i.e., applications), and many more gestures available for document-specific editing of such elements as graphics and text. Most gestures are made directly on an application window, often directly over the objects to be edited, and thus provide excellent proximate interfacing. For text editing and small amounts of text entry, pop-up handprint recognition widgets appear into which the user directly prints characters. Other pop-up user interface elements such as property sheets and command sheets contain fill-in fields which are themselves handprint recognition widgets and can be directly written on.

The GO Notebook User Interface does contain some fixed elements which would make it impossible to use on a very large or small screen. In particular, Notebook icons are stored in an area called the BookShelf located in a horizontal strip at the bottom of the display. A standard pull-down menu bar appears horizontally at the top of the display. On a wall-sized display, the BookShelf would appear near the floor and the pull-down menus would be located near the ceiling. The GO system uses a proprietary, scalable graphics interface (ImagePoint) which can deal with scaling and rotating the entire user interface display, but the interface was not designed to support outsized devices.

2.2.2 PenWindows

Microsoft has added pen capabilities to its Windows 3.1 graphical interface for IBM compatible computers. The use of the pen is completely backwards compatible with the use of the mouse, so that existing applications for the Windows environment may simply substitute pen for mouse. Some applications have been modified to accept handprinted input for recognition directly over their text regions or via pop-up recognition pads. Microsoft has provided a software tool called the Pen Palette that is a pop-up handprint recognizer that can direct its recognized text to conventional text-oriented applications. Text editing can be accomplished through the use of gestures and handprint recognition. Other applications may be written that are truly pen-centric.

Microsoft Windows uses conventional icons, overlapping windows and pull-down menus at the top of each window, making it unsuitable for implementing proximate user interfaces. Microsoft makes no claims for scalability of Windows to larger or smaller display surfaces.

2.2.3 Momenta (R.I.P. 1988-1992)

In the stylus-based interface for the notebook computer by the Momenta Corp. [16], a pie menu called the Command Compass is the main interactor for Momenta applications. There is partial proximation in this interface since selecting an object, like a graphic or text, causes an icon for the application-specific command compass to appear on or near the selected object. The user may then press on the icon with the stylus to make the Command Compass appear. The proximation is only partial in that each application window can have a Command Compass fixed in its caption bar, and the overall Momenta user interface has a Command Compass fixed in the upper left corner of the display. Momenta makes no claims for scalability of its user interface components to larger or smaller display surfaces.

3 Proximate User Interface Issues

We have identified several issues that proximate user interfaces must address. In this section we discuss what some of the advantages are of large-scale devices like the Liveboard, problems with simply running existing toolkits on these devices, and some of the problems that proximate user interfaces must overcome.

3.1 Functionality

Many software applications, such as CAD packages and illustration packages, may be enhanced by execution on a large display surface. Collaborative design can be facilitated by large display surfaces visible to everyone in a room and available for interaction by one or more designers. However, typical CAD-like user interfaces [1, 15, 21] are highly dependent on buttons, pop-up menus, scroll bars, and other static user interface elements around the top and side of the display surface. Is it possible to transmute these interfaces to proximate user interfaces without swamping the user in a deep hierarchy of proximate interactors?

3.2 Scalability

Many existing user interface toolkits are layered on top of graphics standards, such as Microsoft Windows and X Windows [24], which are inadequate for use on outsize devices. Microsoft Windows and X Windows do not have a resolution-independent imaging model; they deal in units of pixels rather than in some absolute, scalable unit. As a result, users are at the mercy of the display size that they happen to be working on at any given moment, as toolkits do not perform the painstaking task of scaling their interface elements so that the interface element size is appropriate to the display in use. When display sizes range from palmtop to palm tree, this approach simply will not work. Proximate user interfaces must be flexible and able to respond to the variety of work surfaces available. The interface elements that we have identified as proximate are flexible enough to work well on a variety of work surfaces.

3.3 Ease of Discovery and Use

Emphasis on paper-like interfaces [5, 22] for individual use and on "walk up and use" interfaces for collaboration (in the Xerox Liveboard community) raise fundamental questions concerning ease of discovery and use. In current mouse-based interfaces, users might click on static elements to attempt to discover functionality, or push buttons labeled "Help" or icons displaying the international "information" symbol. However, iconic buttons often fail to convey their purpose in spite of the best efforts of interface and graphics designers. A proximate user interface presents a new twist: there is nothing visible to click on! How does such an interface provide prompting, feedback, and guidance for the new user?

3.4 Compatibility with Existing Interfaces

In the immature phase of proposals for stylus-based interfaces there were attempts to directly substitute the stylus for the mouse in existing applications, viewing these instruments as essentially two forms of an equivalent pointing device with buttons. In fact, the Xerox Liveboard [6] stylus has three rather awkward buttons on the stylus itself, intended to emulate the three-button mouse attached to conventional workstations. Although providing this mouse mimicry allows existing software to run unchanged, it is a poor substitute for a true stylus-based interface and in fact hinders development of new interfaces. The hindrance comes in two forms: first, people learn to make do with the stylus and with existing applications and, second, people get disgusted with the stylus because it is so badly matched to mouse-based applications that they lose interest in stylus-based computing.

For example, many mouse-based user interfaces depend on the location of the mouse cursor to provide input focus and to direct user actions such as keystrokes to the application window within which the cursor lies. When a user moves a mouse cursor and then lets go of a mouse on a horizontal surface, the mouse remains still and the cursor remains put. When a user puts down a stylus while working directly on a surface, the stylus cursor goes away unless something special is done in the interface software or the user is required to do something awkward, such as lift the stylus straight up until it is well away from the digitizer surface. This awkwardness is unacceptable. Stylus-based interfaces cannot rely on cursor placement as conventional workstation interfaces now do.

Similarly, much emphasis was placed on so-called "handwriting recognition," which was both mislabeled and misleading. What is at best available today is recognition of carefully printed glyphs, usually arbitrarily divided into "characters" and "gestures." Although recognition technology seems to have improved within the last two years, it is still far from robust enough to handle the scrawls that many of us make for communication. In contrast to typing, handprinting is slow and handprint recognition is inaccurate.

Much is made of the stylus interface as the entrance to computing for those who may not type, either for lack of skill or for physical circumstances. But for existing users who have mastered typing and character-based user interfaces, the novelty of handprint recognition soon wears off. It is simply impractical to "print in" even a moderate amount of text. So stylus-based user interfaces need to deemphasize handprint input and instead rely on a small set of robust gestures coupled with the ability to cope with ink as a data type, so that most messages intended for human reading go on their way uninterpreted by machines. For example, a stylus-based electronic mail system might have a pop-up scrollable browser for common mail destinations, a pop-up handprint pad for the rare instance when a new destination must be input, and a blank window on which a message is written and drawn in ink. Such "multimedia" mail messages are becoming more common in the Internet community [2].

Shneiderman [26] has demonstrated that a competent typist can achieve up to 25 words per minute typing speed on a pop-up touchscreen keyboard. The pop-up keyboard consists of a picture of a conventional QWERTY keyboard upon which the user pecks out characters with the stylus. The typing speed and accuracy obtainable varies with the size of the target keys. A pop-up keyboard is implemented in almost all commercial stylus-based environments, implying that there are always applications in which falling back to typing is desirable. Other arrangements of characters on pop-up keyboards would be possible, and users could tailor such arrangements to their individual preferences.

David Goldberg, at Xerox PARC, has devised a clever alphabet called Unistrokes for what he calls "touch typing with a stylus" [7]. He eliminates the need for hard or soft keyboards, substituting what he claims is an easy-to-learn shorthand alphabet using simple, single stroke characters. Because each character is a single stroke, there is no need for either forcing users to print characters in boxes or for clever software to do character segmentation. Unistrokes, like touch typing, can be very fast, once learned, and also allows eyes-free "heads up" writing, since all of the strokes can be written on top of each other; recognized characters appear immediately in text being created or edited.

3.5 Toolkits and User Interface Builders

Existing toolkits [8, 14] and user interface builders [27] do not readily extend to the proximate user interface paradigm. They are targeted at arranging a set of interactors that are constantly visible in fixed places on a display. They do not readily deal with mostly invisible interactors, or with interactors that float above application canvases and require more direct cooperation between applications and the user interface (see Interfering Applications, below). Our attempts to extend existing toolkits have met with difficulties. It is too often the case that the toolkit must be circumvented or thwarted by extensions reaching deep down to the lowest levels of interaction, such as Xlib, or it has proved simply too baroque to implement the extensions required.

3.6 Summary of Proximate Interface Issues

Thus we have seen that outsize devices improve the functionality of some software applications and that proximate user interfaces can help solve the scalability problems associated with these outsize devices. It is also clear that the we should not simply substitute the stylus for the mouse in these applications, but instead rely on the strengths of the stylus. Unfortunately, it is unclear what the best solution is to the ease of discovery problem faced by interfaces based on non-visible interface elements such as gestures. In addition, interfaces with no static elements must solve the interference problem discussed in the next section.

4 Interfering Applications

Proximate user interfaces raise a problem that we term the *interference* problem. Simply stated, the underlying application can interfere with the proximate interface if a given stylus motion is potentially meaningful to both the interface and the application. For example, in a freehand drawing program on a stylus-equipped computer, any stylus input might be treated as a gesture or command, aimed at the interface, or as a stroke to be drawn, aimed at the drawing program. In contrast, static user interface elements spatially disambiguate input; i.e., stylus activity over a button is destined for the interface and stylus traces drawn on the application window are destined for the application. It is therefore impossible to draw over a button or to have a button in the midst of the drawing surface. This is the manner in which conventional drawing programs are organized. Interference does not occur in programs such as GEdit [12], which uses a combination of gestures and pie menus for interaction, because the stylus input is always aimed at the interface and does not interfere with the synthetic graphical objects that GEdit creates and modifies. In freehand drawing or markup applications, interference occurs.

4.1 Modeless Interfaces

There are at least two methods for dealing with the interference problem: moded and modeless interfaces. Modeless interfaces have the advantages of requiring no effort on the part of the user to learn, switch among, track, and understand mode states. However, in a so-called modeless interface, that effort is not gone, but actually moved into both the interface manager and the interfering applications. Software must now determine, via some hierarchical, heuristic, or "blackboard" analysis technique, whether a stylus stroke is to be interpreted by the interface or by the implementation (viz. PenPoint operating system from GO Corp.) This technique is computationally intensive and cannot be right all of the time. For example, in a synthetic drawing application, it should be possible to draw a circle on top of an existing object by laying down a circular ink track, but if a circular gesture means "select the objects in this circle," the user will not be able to create that circle in the most natural manner. Instead, some mode must be introduced that the user can invoke to mean "send my strokes to the application, not the interface manager, until I tell you differently" or the user must contrive some artifice such as drawing two half circles whose end points match up in order to get a circular shape (but not a circular object!).

Noted problems with moded interfaces are that there are too many different modes, modes are arcane, modes do not provide sufficient feedback of the moded state to the user, and modes cause modal errors. But in the particular case of directing stylus input for interfering applications to the interface or to the application, a simple mode seems to us the best solution to this problem.

4.2 Moded Interfaces

Moded interfaces are generally decried, but in fact are nearly unavoidable in real interfaces. Workstation users are familiar with moded interfaces even though they might not even be aware of modes. In particular, keyboard modifiers (e.g., Control, Shift, Alt, Meta, ...) of mouse button actions introduce modes, as do the use of multi-click actions with mouse buttons themselves, an instance of a *temporal* mode. Of course, the most common mode is the spatial mode, in which screen real estate is divided into many different interface and application plots, within which user actions may be interpreted nearly arbitrarily. So the issue becomes how to introduce and use modes conveniently. In the case of the stylus there are several possibilities: stylus buttons, temporal modes, speech, and auxiliary devices.

4.2.1 Zero, One, and Two Button Styli

We have observed stylus designs incorporating from zero to two buttons on the stylus (in addition to a switch on the tip of the stylus): the prototype notebook from the GO Corp. (0), the Momenta Corp. stylus (1), the Xerox Liveboard stylus[2] (2)

Stylus buttons have empirically been observed to be awkward to use [6]. In particular, a stylus has very little surface area near the fingertips not covered by the user's grip, so buttons tend to be placed close together on the "top" or "side" of the stylus case. Both the buttons and the spacing between them are considerably smaller than the tip of the finger. Thus, too much care is required in contacting the desired button and even more care is required in avoiding unintentional activations of buttons at undesired times and places. This problem is eliminated by having no buttons at all, but the no-button case leads directly to the problem of interfering applications outlined above.

With a single stylus button, the question of what meaning to assign to a button press or click arises. In a stylus-based world, we recommend that the button have a single, universal meaning which is to direct stylus strokes to either the interface software or the application. In particular, the stylus should be designed so that the user can comfortably hold down the button while stroking, so that:

- The user gets immediate and constant tactile feedback (by holding down the button) as to what mode stylus input is in;
- No other visual or audible feedback for the mode is required;
- The button is not overloaded with a variety of meanings.

[2] The Xerox Liveboard stylus actually has three buttons, but one of them duplicates the function of the tip switch, so that only two independent buttons are available.

In particular, avoid the temptation to allow a button click (as opposed to pressing and holding) to toggle between "the strokes go to the interface" mode and "the strokes go to the application" mode. This is tempting to implement, but it replicates the original circumstance which led to the condemnation of modes in early text editors that had a "command" mode and an "insert" mode. These modes determined whether the command interpreter (user interface) or the editor (application) interpreted keystrokes.

4.2.2 Temporal Modes

When interaction space or mechanism is limited, temporal modes can be employed. With a temporal mode, the amount of elapsed time since an initiating action (e.g., stylus down) or between actions indicates mode switching. An example of an early temporal mode is the "repeat key" mode on electric typewriters, in which the pressing and holding of a keyboard key for a relatively long period of time initiated the simulation of "repeating" keystrokes until the keyboard key was released.

One could use a temporal mode in stylus-based interfaces to solve the interference problem. That is, if the stylus is pressed and held still for a "sufficient" period, an interactor pops up and interaction with the user interface is initiated. If the stylus is moved "quickly enough" after the stylus is pressed down, then the application receives the input. This mechanism seems particularly attractive for markup or freehand drawing applications. Of course, the meaning of "quickly enough" will differ among users, but user-controlled preferences can deal with that. Another difficulty is the interpretation of the meaning of "held still." Is a motion of a single pixel really meant as a motion? What about two pixels? Three? A millimeter? Lack of a rock-steady hand should not bar the user from interacting.

Hopkins [9] and Kurtenbach [12] report the use of temporal modes as accelerators for pie menus. In these systems, depressing the stylus or mouse button initiates a temporal mode in which, should a sufficiently quick stroke be completed, the stroke is interpreted as invoking the command which would have been chosen had the pie menu popped up. The pie menu will pop up if a sufficiently long period elapses without stylus motion. A similar form of temporal accelerator appears in the Cedar system pop-up buttons [21], in which a static button can be quickly clicked on using a combination of mouse buttons and keyboard modifiers to invoke an operation, instead of pressing down and waiting for the pop-up button to appear with menu choices and documentation.

Temporal modes can also be used to alter a stroke, in mid-glide, from operand selection to operation invocation. In GEdit, selecting and operating on synthetic graphic objects is achieved with a single stroke which pauses in mid-glide to switch from selecting to operating. This requires that the user interface always receive stroke information and constantly be evaluating partial input as the stroke progresses. Thus, initiating a stroke that first encircles a set of objects, then pauses, indicates the selection of the enclosed operands. Continuation of the stroke indicates the Move command in which the stroke now defines the motion vector. Near the end of the stroke, an optional concluding motion in the shape of the letter "C" transforms the Move command into a Copy command.

4.2.3 Speech

Speech recognition has been sought by linguists and computer scientists since the 1950s. Most efforts at speech recognition in real time have produced limited results until recently. It is now possible to do real time, speaker dependent, isolated word, small vocabulary recognition with sufficiently high hit rates, in software, to be useful in user interfaces. Systems such as SimonSays [10] provide an interactive user interface builder for voice commands to window applications. SimonSays builds a macro-like event sequence by capturing standard user input to an application, and allows the user to assign a word or short phrase to name the macro. The user then trains the voice recognizer with three samples of the vocal input for that macro name. Subsequently, when the voice input system is enabled in recognition mode, vocal commands are recognized and the corresponding macro is invoked to drive the application. SimonSays can recognize on the order of a three hundred word vocabulary in a given application context before degrading from real time performance on a NeXT or Sun workstation. In particular, SimonSays can be used to invoke mode commands in editors.

In Liveboard environments, voice recognition is complicated by the potentially large distance of the speaker from the microphone, and by the fact that the training environment is likely to be acoustically different than the use environment. Further, it is undesirable for a user to wear a headset microphone in order to solve acoustic problems. A clip-on, possibly wireless, microphone is acceptable when controlling Liveboard applications from near the Liveboard. Our preliminary experiments with wireless FM microphones indicate that acceptable voice command hit rates can be acquired in a small, relatively quiet conference room.

4.2.4 Auxiliary Devices for Mode Invocation

Finally, we note that when using stylus input on a device that is freestanding or fully supported, the weak hand is free and could be put to good use. Buxton [3] has long advocated the use of two-handed interfaces. Mouse, keyboard, and trackball are an example of a multi-handed input suite that works on conventional workstations. In the case of the Liveboard, we can use ParcTab buttons (viz. Photo 1) instead of stylus buttons or keyboard modifiers for directing stroke interpretation. The ParcTab is held in the weak hand. ParcTab has a wireless infrared connection to either a dedicated machine or a backbone server network. When we use ParcTab buttons as an auxiliary user interface device, synchronization and response time currently require that the ParcTab wireless receiver be interfaced directly to the Liveboard workstation. This is an acceptable limitation for the present.

A cheap, simple button device for the weak hand is a three-button serial mouse plugged into the serial port of a workstation. The user simply holds that mouse in the hand and uses the buttons as required by the interface.

4.3 Summary of the Interference Problem

We have seen that proximate user interfaces introduce a new problem, the interference problem, which interfaces based on static interface elements do not face. The interference problem is essentially an ambiguity as to where stylus motion has been aimed. As shown above, this problem can be solved by having moded interfaces, either through a stylus button, a slight time delay, speech, or by using another device with the weak hand.

5 Prototype and Results

We have shown that large display surfaces present special problems to interactive applications. In order to explore solutions to these problems, we built some prototype proximate interface elements and a demonstration application that incorporates them. Our application addresses the problems faced by large screens by using gestures and pie menus. In addition, these interface elements address the scalability problems discussed in section 3.2, and by using a stylus button for indication of the current mode, the interference problem discussed in section 4.

The *Proximate* toolkit is a collection of interface elements written in Common Lisp and added to the Garnet User Interface Development Environment within the X11 Window System [17, 24]. The interface elements implemented, *pie menus* and *gesture recognizers*, are not generally available in user interface toolkits for X11. Garnet was chosen because it is designed specifically with our kind of task in mind: the exploration of advanced and novel user interface techniques. In addition, the Garnet interactor abstraction [18], which separates interaction mechanisms from graphics presentation, is ideal for creating an implementation of pie menus. Our pie menus use much of the same interaction code as do other Garnet menus (e.g., pull-down menu bars). Most other toolkits we examined would have required us to write all of this interaction code ourselves. In addition, the existing gesture interactor [13] in Garnet made doing gesture recognition in our application quite easy.

The application that we examined for scaling problems was GarnetDraw, a simple synthetic-graphics drawing program, shown in Figure 3. This application is typical of many conventional drawing applications found on X11, the Macintosh, and Microsoft Windows. These applications are characterized by pull-down menus at the top of the window and a number of palettes (for selection of drawing tools, line styles, colors, and fill patterns) along the bottom and sides. There is a relatively large canvas area in the middle of the window where the actual drawing is created. As was previously discussed, drawing programs use this spatial layout of static interface elements to solve the interference problem.

Figure 3: GarnetDraw static interface

Figure 4: PrGarnetDraw proximate user interface

Once we ran GarnetDraw on the Liveboard, we were confronted with several of the problems that have been previously discussed (e.g., difficulty reaching interface objects and awkwardness in using a stylus with an interface that required the use of multiple buttons). We encountered additional problems:

- The need to step back from the Liveboard in order to locate and use menus and palettes. Users tend to view the Liveboard from the same distance as one works on a chalk board; at that close distance objects that are not within a one foot radius of the stylus are very hard to read;

- The inability to see mode feedback provided by the altered appearance of menus and palettes, again because they are out of view.

In order to address some of these problems, we built a new application, PrGarnetDraw, shown in Figure 4. PrGarnetDraw encompasses all of the functionality that is in GarnetDraw, yet it removes the static interface components and replaces them with our proximate interface elements. All of the items that were reachable from the GarnetDraw menu bar are now reachable from a pop-up hierarchical pie menu, which is activated by holding down one of the stylus buttons. The pop-up pie menus are by their nature exactly within reach and view of the user. In addition, the tools palette, fill color palette, fill pattern palette, and line style palette are all reachable from the main pie menu. A user can access and learn all of the functionality of PrGarnetDraw through this one menu.

PrGarnetDraw uses gestures that take advantage of the power of the stylus. We have initially implemented a simple "X" gesture for deleting graphical objects, a "C" gesture for copying objects, flick gestures for moving objects, and box and circle gestures for creating new boxes and circles, respectively. These gestures are indicated as such by holding down another of the stylus buttons. In the near future we will add a caret gesture to bring up a recognizer for handprinted text, and a circling gesture to select objects.

Our initial experience with PrGarnetDraw suggests the need to remove the initial mouse button press that brings up the pie menu; we would like to implement the temporal technique previously discussed. This would allow our interface to work with a stylus with only one button (in addition to any stylus tip button). Initial experience with a different application that uses a pie menu invoked by pressing and waiting has been positive.

In addition, we would like to try combinations of pie menus with gestures. One combined technique is to have a new gesture that will immediately bring up a pie submenu that is below the main pie menu in the menu hierarchy. For example, we could draw a script "E" with a single stroke to bring up the "Edit" pie menu. We could further accelerate the use of this interaction by allowing the user to draw the initial gesture (e.g., the "E") and then immediately continue the gesture in the direction of the desired pie wedge (e.g., Cut, Copy, etc.) without the pie menu ever coming up (see Figure 5). That is, we use gestures as accelerators for well-known operations. The final combining strategy we would like to try is making gestures that are drawn as the stylus leaves a pie menu slice. This would allow further operation parameterization or operand selection by combining the menu choice with the subsequent gesture (see Figure 6). These two combining techniques are a step toward solving the ease of discovery problem discussed in section 3.3.

Figure 5: Combination of gestures with pie directions

Figure 6: Pie menu combined with gestures leaving slices

6 Conclusions

We have shown that the interface elements required by outsize devices are different than the interface elements in use today. On large display surfaces problems of element visibility and accessibility are the primary difficulties to overcome. We have suggested several techniques that may overcome some of these problems (e.g., speech, gestures, pie menus, combinations of gestures and pie menus) and examined new problems that may be introduced, i.e., the interference problem. We have built some of these new interface elements, but we would like to build more and do user testing to see what works best.

Acknowledgments

The authors wish to thank Polle Zellweger, Eric Bier, and Brad Myers for supporting this work. Production credit for this paper and many thanks to: Cathy Marshall and Scott Elrod for realistic modeling; Andy Aronson, Brian Tramontana, and Dan Murphy for photographic services; Jade Goldstein for commenting on several drafts. Roy Want designed the ParcTab hardware and made it available to us. This work was done under the auspices of the Xerox PARC Information Sciences and Technologies Laboratory.

References

1. Adobe Systems Inc. Adobe Illustrator (TM) User's Manual Version 1.0, Jan. 1987.

2. Borenstein, N. Multimedia electronic mail: will the dream become a reality? In *Communications of the ACM* 34, 4. (April, 1991), pp. 117-119.

3. Buxton, W. and Myers, B. A Study in Two-Handed Input. In *Proceedings of the CHI '86 Conference on Human Factors in Computing Systems.* (Boston, MA. Apr. 13-17, 1986). ACM Press, New York, 1986, pp. 321-326.

4. Callahan, J., Hopkins, D., Weiser, M., and Shneiderman, B. An Empirical Comparison of Pie vs. Linear Menus. In *Proceedings of the CHI '88 Conference on Human Factors in Computing Systems*. (Washington, DC. May. 15-19, 1988). ACM Press, New York, 1988, pp. 95-100.

5. Carr, R. and Shafer, D. The Power of PenPoint. Addison Wesley, New York, 1991.

6. Elrod, S., et. al. Liveboard. In *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems.* (Monterey, CA. May. 3-7, 1992). ACM Press, New York, 1992, pp. 599-607.

7. Goldberg, D. Touch Typing With a Stylus. To appear in *Proeedings of INTERCHI '93.*

8. Heller, D. XView Programming Manual. O'Reilly Associates, Inc. 1990.

9. Hopkins, Don. The design and implementation of pie menus. *Dr. Dobb's Journal.* Vol. 16, No. 12, (Dec 1991), pp. 16-26.

10. HSD Microcomputer Inc. SimonSays User's Manual, Version 1.1, Mountain View, CA.

11. Johnson, Jeff, et. al. The Xerox Star: A Retrospective. In *IEEE Computer*, Vol. 22, No. 9, (Sep 1989), pp. 11-29.

12. Kurtenbach, G., and Buxton, W. Issues in Combining Marking and Direct Manipulation Techniques. In *Proceedings of UIST'91, the Fourth Annual Symposium on User Interface Software and Technology* (Hilton Head, SC. Nov. 11-13). ACM Press, New York, 1991, pp. 137-144.

13. Landay, James A., and Myers, Brad A. Extending an Existing User Interface Toolkit to Support Gesture Recognition. In *Adjunct Proceedings of INTERCHI* (Amsterdam, Netherlands. April 24-29). ACM Press, New York, 1993.

14. Linton, M.A., Vlissides, J.M., and Calder, P.R. Composing user interfaces with InterViews. In *IEEE Computer,* Vol. 22, No. 2, (Feb 1989), pp. 8-22.

15. Lubow, M. Working out with AutoCAD. New Riders Publishing, P.O. Box 4846, Thousand Oaks, CA 91360 U.S.A., 1987.

16. Momenta Corp. Momenta User's Manual Version 1.0, Oct. 1991.

17. Myers, Brad A., et al. Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces. In *IEEE Computer,* Vol. 23, No. 11, (Nov 1990), pp. 71-85.

18. Myers, Brad A. A New Model for Handling Input. In *ACM Transactions on Information Systems*, Vol. 8, No. 3, (Jul 1990), pp. 289-320.

19. NCR Corp. Microsoft Windows For Pen Computing: Guide to Pen Computing, 1992.

20. Newman, W. Markup. In the *Alto User's Handbook,* Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304, 1979.

21. Pier, K.A., Bier, E.A., and Stone, M.C. An Introduction to Gargoyle: an Interactive Illustration Tool. In *Proceedings of EP88, the International Conference on Electronic Publishing, Document Manipulation and Typography* (Nice, France. Apr. 20-22). Cambridge University Press, Cambridge, 1988, pp. 223-238. Also available in Xerox PARC Technical Report EDL-89-1, 1989.

22. Rhyne, J.R., and Wolf, C.G. Paperlike User Interfaces. IBM T. J. Watson Research Center, Yorktown Heights, New York, 1991.

23. Rubine, D. Integrating Gesture Recognition and Direct Manipulation. In *Proceedings of the Summer '91 USENIX Technical Conference.* (June, 1991). USENIX Association, 1991, pp. 95-100.

24. Scheifler, R.W., and Gettys, J. The X window system. *ACM Transactions on Graphics* Vol. 5, No. 2, (Apr 1986), pp. 79-109.

25. Schmandt, C., et al. Observations on Using Speech Input for Window Navigation. In *Proceedings of IFIP INTERACT '90: Human-Computer Interaction,* 1990, pp. 787-793.

26. Shneiderman, B., et al. A New Era for Touchscreen Applications. In *User Interface Strategies '91, A Live, Interactive Satellite Broadcast.* (College Park, MD. Dec. 5). University of Maryland Instructional Television System, 1990.

27. Sun Microsystems, Inc. Open Windows Developers Guide 1.1, User's Manual, 1990.

28. Thacker, C.P., et al. Alto: a Personal Computer, in *Computer Structures: Principles and Examples,* eds. D. P. Siewiorek, C.G. Bell and A. Newell. Mcgraw-Hill, 1982.

29. Weiser, M. The Computer for the 21st Century. In *Scientific American* 265, 3 (Sep. 1991), pp. 94-104.

30. Wilcox, L.D., and Bush, M. A. HMM-based Wordspotting for Voice Editing and Indexing. In *Proceedings of Eurospeech '91* (Genova, Italy, Sep. 24-26). ESCA, 1991, pp. 25-28.

31. Wiseman, N. E., et al. PIXIE: A New Approach to Graphical Man-Machine Communication. In *Proc. 1969 CAD Conf., IEE Conf. Pub. 51*, (Southampton, 1969), p. 463.