Appeared in the Proceedings of the Fourth Workshop on Workstation Operating Systems, Napa, CA, October 1993

User Interface Issues in Mobile Computing

James A. Landay and Todd R. Kaufmann

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

Abstract

The age of mobile computing with small portable computers is upon us. However, systems researchers continue to focus on transferring their workstation environments to these machines rather than studying what tasks more typical users wish to perform. Furthermore, we argue that even in the unlikely event that these tasks are the same as those performed by researchers on their workstations, desktop environments and applications may still be inappropriate for the wide range of mobile devices that will be marketed. To explore this area, we have built a mobile computing device along with a user interface system that attempts to address some of these issues.

1. Introduction

While researchers have speculated on the advent of ubiquitous computing [18], mobile computing is already upon us. This new technology is becoming a reality mostly as a result of seemingly haphazard forces of the consumer marketplace and not of the careful plans of the research community.

Mobile computing technology is evolving in two directions. Pagers and other hand-held devices are becoming more powerful, and laptop computers are becoming smaller. The paging devices are distinguished by their use of communications technology to access and author data, while laptops generally use a "manual caching" approach. With manual caching, as opposed to the "automatic caching" used in Coda [9], users explicitly copy information between a base computer and their laptop machines. Consumers in both the paging market and the laptop computing market have demanded – and are beginning to receive – the capabilities that are traditionally the strength of the other market: more authoring capabilities in hand-held devices and more communication features in laptops. When the distinction between these devices truly begins to blur, as they are expected to in machines like the AT&T EO Personal Communicator and Apple Newton, mobile computing will attract non-traditional computer users.

The researchers on the BNU project [17] are exploring some of the operating systems and user interface issues that must be addressed to make mobile computing devices more useful. To explore what is already possible with inexpensive and readily available hardware, we have combined an off-the-shelf hand-held device, the HP 95LX Palmtop PC [7] (see Figure 1), with inexpensive infra-red (IR) communications hardware to build a simple mobile computing system. The BNU system is meant for an office building or campus area that is outfitted with IR "cells", *i.e.*, it is primarily intended for "connected" operation. Our mobile computing device might be considered a poor man's ParcTab [18].

Our attempt to build a user interface management system[1] for this small mobile device led us to question the current direction of systems research for these types of machines. Current research assumes that the user of these machines will be performing tasks similar to those now performed on workstations. We think this is a faulty assumption. Although some devices will be used for these purposes, mobile computers may be used for many new tasks that current research neglects. This paper focuses on the user interface issues that must be addressed when building a small mobile computing device and on the infrastructure that we built to support these types of interfaces.

The rest of this paper is organized as follows. Section 2 discusses key user interface issues as they pertain to small mobile computers and explains how these differ from those studied previously for workstations. In Section 3 we present an overview of the BNU system. We follow this with a description of the user interface management system we built for BNU and conclude with the status of the project and our plans for future research.

Figure 1. The HP 95LX is small enough to fit in the user's palm. The machine comes with 1MB of RAM, runs MS-DOS, and has a 2400 baud IR I/O port.

2. User Interface Issues in Mobile Computing

A theme common to much of the past work on mobile computing devices is the desire to run similar computing environments on the mobile machines and on the user's office workstation [4, 8]. Although running many of the same *applications* may be useful and desirable, running the same *environment* may be both undesirable and, for many mobile devices, impossible. In addition, researchers must realize that although we would like our research to guide the development of commercial systems, we are no longer the typical users of these systems. By designing systems for our past work habits and preferences we may help to produce systems that are appropriate and efficient, but not for the intended user base [2].

---

[1]A user interface management system is a software system that helps the designer handle the sequencing aspects of the user interface [10].

2.1 User Interface Processors vs. General-Purpose Computers

Duchamp has constructed a simple four category taxonomy for mobile computers: *terminal* vs. *workstation* and *disconnected-often* vs. *disconnected-rarely* [4]. It is not clear which of these four categories will produce the best mobile device for the users of these machines. While some may argue that mobile devices should be general-purpose computers [4], we opted instead for simple *user interface processors*. A user interface processor (or a "disconnected-rarely terminal" in Duchamp's taxonomy) only performs drawing and low-level event processing, leaving the rest of an application's computation to a remote server. Other "disconnected-rarely terminals" include the Berkeley Infopad [15] and the Xerox ParcTab [18].

User-centered design [11] and task analysis [5] techniques stress that we should first determine what tasks the intended user population would likely perform on mobile devices. It was argued at a recent mobile computing panel [1] that much of the usage of mobile devices will be to access rapidly changing data at the home office, rather than authoring new data. The current uses of communications technology (*i.e.,* pagers and telephones) seem to support this conclusion. Placing most of the computational power on the mobile device seems to be overkill if this is how the machines are actually used. In designing a device that has stringent constraints on price, power consumption, and size, one must omit components that are not absolutely necessary.

Running applications on a remote host and using the mobile device primarily for the user interface solves some of the hard problems faced by designers of mobile computer operating systems. First, the system reinitialization problem becomes less complicated. When the mobile device crashes, it can be rebooted and the user can continue the application with no loss of data or state. This assumes that the remote hosts are more reliable than the mobile devices.

The second major problem that the user interface processor approach solves is the data consistency problem. Instead of needing intelligent caching and consistency protocols as in Coda [9], a user will have access to the most current versions of their private and home office data as long as their communications link is stable. This is a reasonable assumption in a single building or campus setting. The user interface processor approach fits well with the idea that autonomous local file systems will become increasingly undesirable [14]. Although we focus on our experience in building a user interface management system for BNU, much of this discussion applies to all mobile computers, regardless of where they fit in Duchamp's taxonomy.

2.2 Mobile Tasks are Different

As suggested above, an analysis of mobile computing tasks may produce a vastly different set of application priorities than an analysis of the use of a typical office workstation. It will certainly produce a different set of tasks than those performed by a typical computer scientist. For example, an analysis may indicate that the primary tasks to be performed on mobile devices are: scheduling, short note taking, email reading and composition, large database browsing, and filling out forms that must be incorporated into large home office databases. Compare this set of tasks with the typical tasks performed on office workstations by researchers: large document preparation and layout, programming, email reading and composition, and the creation of graphics presentations. Most of the workstation tasks involve a large amount of data entry, whereas the mobile tasks mainly involve small amounts of data entry and the presentation of existing information.

If an actual task analysis produced these disparate sets of tasks, we would like to have mobile devices optimized for the typical mobile task set. For example, these mobile tasks would favor a system that includes strong communications capabilities and therefore needs little local storage, yet does not require high-rate input devices (*e.g.*, a keyboard). The workstation tasks would favor just the opposite. We have not actually carried out this task analysis, but we present this example to illustrate how a different set of user tasks can affect systems design decisions.

2.3 Mobile Device Sizes

Furthermore, the scale of mobile machines (particularly palmtops) will preclude our workstation environments from being useful or even possible to implement. The primary reasons for this are the small size of the displays and the wide variety of input devices used on mobile computers. For example, the small amount of screen real estate on the HP 95LX, about nine square inches, would tend to favor interfaces that have few static interface elements occupying space, such as scroll bars, palettes, and icons. Unfortunately, the systems that are considered easy-to-use (*e.g.*, the Apple Macintosh and Microsoft Windows) rely on static interface components. Further research must be conducted on designing easy-to-use interfaces without static elements.

For example, to conserve the precious screen space, a small mobile machine might be better outfitted with popup menus. In addition, the small keyboard of the HP 95LX would favor a more menu-oriented interface since the maximum possible typing speed is greatly reduced. Many other machines have as their primary input technique either gestures (*e.g.,* the PenPoint-based [3] machine from EO and the Apple Newton) or voice, as on the "wearable" computer being developed at CMU [16]. The use of gestural and vocal input will also make our current environments and applications incompatible with the new machines. An earlier paper discusses these issues for outsized devices, both those with extra small and extra large screens, with an emphasis on the large-screened Xerox Liveboard [13].

3. The BNU System

The BNU project began by exploring the terminal / disconnected-rarely category of Duchamp's taxonomy, with our mobile device acting as a simple keyboard and display processor for applications running on a remote host. Excess communication overhead led us to focus on a category that might be considered half-way between a terminal and computer, with much of the processing done on a remote host. The mobile device performs computation necessary for implementing the user interface, and potentially, an arbitrary computation via downloaded functions. The user interface support is supplied by the RUSE system discussed in Section 4.

By eventually outfitting our building with several infra-red "cells", we could use our mobile device to run applications (*e.g.*, reading email) from anywhere in the building. BNU supports mobility through several different mechanisms. A name server acts as an initial contact point for location information during application bootstrapping. Each mobile device has a proxy representative on a UNIX workstation in the LAN. This proxy presents a fixed location to the applications which run on the workstation. The communications layer provides reliable, sequenced messages and RPC handling between applications, proxies, and the mobile machine. Messages are automatically forwarded through dynamic adjustment of route information as palmtops relocate. For more information on the low-level systems details of BNU, see [17].

BNU has all of the advantages discussed above for user interface processors: a better fit for applications that access rapidly changing remote data, ease of system reinitialization, and a trivial solution to the data consistency problem. In addition, BNU includes a user interface management system that takes advantage of the small size of the HP 95LX.

4. The Remote User-interface Software Environment

In order to explore interaction techniques that might prove useful for small devices with strong remote communication capabilities, we have designed a simple user interface management system. The Remote User-interface Software Environment (RUSE) improves the performance of BNU applications by running the user interface code on the HP itself. RUSE and the applications that use it are written in the Scheme programming language. RUSE was designed to address four problems faced by BNU applications: programming is difficult, performance is slow, transparency of remote execution is hard to maintain, and the small size of the HP 95LX is limiting.

4.1 Design Goals

The initial BNU applications were built using only a very low-level graphics/input interface (*i.e.*, print-text-at, get-keypress, *etc*.). By supplying some simple user interface elements or *widgets*, we could make the life of the application developer easier. In addition, we wanted to support the types of interaction advocated above for small machines: popup menus, scrolling lists, *etc.* Also, we believe that applications are much easier to develop in a clean interpretive language like Scheme.

The second problem faced by BNU applications was performance-related. The low-level graphics/input interface caused a large amount of network traffic. For example, typing a line of text on the HP generated two network messages for each character: one to get the keypress to the UNIX host and another to ask the HP to display the character. With a low speed IR link, this communications cost is prohibitive and causes the user interface to be essentially unusable. By locally processing the input and then sending the entire string of text to the application in one message, we can improve the interaction significantly.

Finally, RUSE was designed to hide from the developer the fact that the widgets are running at a different location from the application. The programming interface is the same, regardless of where the widget implementation resides. This fact was quite useful for debugging the widgets and is useful for debugging applications without using an HP 95LX. Another advantage gained by using RUSE is the ability to continue local processing when the communications link is temporarily lost. For example, a simple form filling application (*e.g.*, the type of application an insurance adjuster would use in the field) might not send the data back to the host application until the entire form (or a field of the form) has been filled in. This allows the effects of short-term communications interruptions, as when switching between “cells”, to be mitigated.

Unfortunately, there is one drawback to using RUSE. One benefit of BNU’s stateless design was the ease of recovering from a crash on the mobile device. When using RUSE there is now state on the HP (*i.e.*, widget instances) that must be recreated after reboot. Interfaces can be built to minimize the amount of state that can be lost by an individual widget at the expense of limiting the amount of local processing that can continue during communications interruptions. For example, it might be wise to send the data from an input field to the application (via a callback) when widget focus leaves that field. If applications are designed in this way, little data will be lost. This sort of design is facilitated by the event-based programming model used by RUSE applications.

4.2 Supported Widgets

RUSE supports several widgets that are commonly found in the types of applications we envision for the HP 95LX. These widgets include scrolling and non-scrolling text input fields, single and multi-line text output fields, dialog boxes, scrolling popup menus, and multi-field forms. Each of these widgets can also respond to arbitrarily defined events. Many of the widgets were designed specifically to help solve the size constraints of the small screen and keyboard of the HP 95LX. For example, popup menus allow users to input commands or other data with a few keystrokes.

In addition to these widgets, RUSE also supports a set of management functions providing support for initialization and cleanup of applications, screen refresh, modal widgets, widget focus, and navigation (the ability to specify the widget that should have focus). The navigation and focus functions were necessary since there is no mouse pointer on the HP 95LX. Element focus within an application is determined by checking application-supplied focus navigation lists. These lists give an ordering of elements to use for each specified focus-changing event, *e.g.*, pressing the tab, return, or arrow keys. Developers can easily specify which events are focus-changing events for each widget.

4.3 Programming Model

RUSE supplies an event-based programming model for application development. When widgets are created, they are given the names of callback functions to be executed on the UNIX host when specified events occur. All widgets are registered with RUSE when they call their create function. The create functions cause RUSE to create an instance of the specified widget (with the given parameters) on the HP and to return a unique name for applications to use when referring to the widget (so we can use this name on the UNIX side and map it to the actual object on the HP side). RUSE widgets must be able to respond to the message `handle-event`. In addition, widgets need to respond to keypress events and the following events that are passed with `handle-event`: draw, hide, activate (modal widgets only), take-focus, lose-focus, and destroy.

The RUSE system that runs on the HP implements an event-dispatch function. Every time a key is pressed, the Scheme interpreter running on the HP calls the dispatch function with the key as a parameter. After checking for focus-changing events, RUSE sends a `handle-event` message along with the event to the widget that currently has application focus. The widget then takes the appropriate action for the given event and instance-specific state. This may include remotely executing a function on the UNIX side. Alternatively, the handler can choose to ignore the event. In this case, the event loop will try another appropriate widget (if one exists). An optional application event handler, normally used to recover from a crash, can be called if there are none.

4.4 Scheme Interpreter

An interpretive language eases many of the problems of producing a user interface system that executes at a different location than the application it is servicing. For example, an interpretive system eases system recovery from errors, allows dynamic extension of the interface system, and allows the system to work across heterogeneous processor architectures. This approach has been tried in previous systems. Our use of Scheme in RUSE is very similar to Sun's use of PostScript in the NeWS windowing system [6]. A BNU remote procedure call (RPC) reader with a linked in Scheme interpreter runs on the HP, while the UNIX side executes a BNU application, also with a linked in Scheme interpreter. We use RPC to send strings and (non-circular) s-expressions from the UNIX Scheme interpreter to the HP Scheme interpreter. These s-expressions evaluate to functions on the HP, allowing functions to be sent between the HP and the UNIX host.

The idle loop on the HP is taken over by the Scheme interpreter. This allows the HP to do useful work in Scheme when it is not responding to interrupts. The keyboard interrupt sends the character to Scheme rather than to the IR link. The Scheme interpreter can pass the character up the IR link if it chooses to do so, but if a RUSE application is running, it will send it to RUSE instead.

Our desire for an interpretive environment on the HP did not require that we use Scheme. For example, we could have used something like TCL/TK [12], but there was a desire to be able to download arbitrary code to the HP. In the future, we may switch to using TCL/TK, which would allow developers to use C, depending on the difficulty of doing a port to the HP.

5. Status and Future Research

The entire RUSE system described in this paper, not including the two Scheme interpreters and BNU infrastructure, consists of 1400 lines of well-commented Scheme source code. It implements all of the management functionality described and several of the widgets, *e.g.*, input and output text fields, popup menus, and dialog boxes. One of the key areas for future work is to implement many more widgets, in particular widgets that might be useful for a machine with a limited keyboard. For example, we would like to build widgets that know how to intelligently complete partial input (*e.g.*, email address completion and other history mechanisms.)

In addition, to see how the system performs on a realistic task set, we must build applications more complex than those used to demonstrate RUSE. We have also yet to test RUSE running on the HP with an application running remotely on the UNIX host. One of the reasons for this was the unfinished state of the lower level BNU infrastructure and the infra-red link when the initial implementation of RUSE was completed. Fortunately, we have an excellent simulator that presents a replica of the HP in a window on a standard workstation and experiences the same RPC performance as the HP. We would also like to carefully measure RUSE's resource requirements.

Finally, we would like to use task analysis and user-centered design techniques to get a better idea of what tasks potential consumers will be performing on a range of mobile computing devices. By using this data we can help guide systems and user interface research towards problems specific to mobile devices.

6. Conclusions

We initiated the BNU project in order to explore many of the systems issues faced by mobile computing platforms. Our initial simulations of BNU applications led to the development of a user interface management system that ran on the mobile machine while the rest of the application ran on a remote host. The RUSE system led to a significant improvement in the interactiveness of BNU applications and we believe it will also ease the development of new BNU applications.

Though our initial emphasis was on low-level systems issues, the attempt to build a user interface management system for a small mobile device led us to question the current direction of systems research for these types of machines. We have argued that the physical differences between mobile computers and desktop workstations may produce a different set of potential uses for these machines. Therefore, it may be counterproductive to carry out systems research that emphasizes reproducing researcher's desktop environments on mobile devices. Instead, researchers should use task analysis techniques to further investigate the potential uses of these new devices. Using the results of this analysis will better guide the low-level systems research.

Acknowledgments

The authors would like to thank Nick Thompson who designed and built the Scheme infrastructure. Thanks to Brian Bershad for allowing us to carry out this research and to Terri Watson for last minute information on the low-level details of BNU. Thanks to Stewart Clamen, Jay Kistler, David Kosbie, Francesmary Modugno, Brad Myers, Jay Sipelstein, Ali-Reza Adl-Tabatabai, and Brad Vander Zanden for giving us comments on the paper. Finally, BNU was originally built as a graduate OS class project and the following individuals designed and built the system: Terry Allen, Shumeet Baluja, Brian Bershad, Claudson Bornstein, Yirng-An Chen, Scott Crowder, Eugene Fink, Geoff Gordon, Tammy Green, Karen Haigh, Todd Kaufmann, Jennifer Kay, James Landay, Gerald Malan, Will Marrero, Paul Olbrich, Robert Olszewski, Manish Pandey, Henry Rowley, Stefan Savage, Motonori Shindou, Nick Thompson, Terri Watson, Stephen Weeks, Bob Wheeler, Hao-Chi Wong, and Masanobu Yuhara.

References

1. Badrinath, B.R. Mobile Distributed Systems Panel at the 13th International Conference on Distributed Computing Systems (Pittsburgh, PA. May 25–28). IEEE Computer Society, 1993.

2. Benel, R.A. Future Systems Development: Limits on Vision. In *Proceedings of the Human Factors Society 35th Annual Meeting* (San Francisco, CA. Sep. 2–6). Human Factors Society, Santa Monica, CA, 1991, pp. 1190–1193.

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

4. Duchamp, D. Issues in Wireless Mobile Computing. In *Proceedings of the Third Workshop on Workstation Operating Systems* (Key Biscayne, FL. April 23–24). IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 2–10.

5. Fath, J.L., and Bias, R.G. Taking the “Task” Out of Task Analysis. In *Proceedings of the Human Factors Society 36th Annual Meeting* (Atlanta, GA. Oct. 12–16). Human Factors Society, Santa Monica, CA, 1992, pp. 379–383.

6. Gosling, J., *et al. The NeWS Book*. Springer-Verlag, 1989.

7. Hewlett-Packard Co. *HP 95LX User’s Guide.* April 1992.

8. Honeyman, P., *et al.* The LITTLE WORK Project. In *Proceedings of the Third Workshop on Workstation Operating Systems* (Key Biscayne, FL. April 23–24). IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 11–14.

9. Kistler, J.J. and Satyanarayanan, M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, vol. 10, no. 1, February 1992, pp. 3–25.

10. Myers, B.A. User Interface Tools: Introduction and Survey. *IEEE Software*, vol. 6, no. 1, January 1989, pp. 15–23.

11. Norman, D.A., and Draper, S.W. (Eds.). *User Centered System Design: New Perspectives on Human-Computer Interaction*. Erlbaum Associates, Hillsdale, NJ, 1986.

12. Ousterhout, J.K. An X11 Toolkit Based on the TCL Language. In *Proceedings of the Winter 1991 USENIX Conference* (Dallas, TX. Jan. 21–25). USENIX Association, Berkeley, CA, 1991, pp. 105–115.

13. Pier, K.A. and Landay, J.A. Issues for Location-Independent Interfaces. Technical Report ISTL92-4, Xerox Palo Alto Research Center, Dec. 1992.

14. Redell, D.R. Workstation Autonomy is a Dead Issue. In *Proceedings of the Third Workshop on Workstation Operating Systems* (Key Biscayne, FL. April 23–24). IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 15–16.

15. Sheng, S., Chandrakasan, A., Brodersen, R.W. A Portable Multimedia Terminal. *IEEE Communications Magazine*, vol. 30, no. 12, December 1992, pp. 64–75.

16. Smailagic, A. and Siewiorek, D.P. A Case Study in Embedded System Design: The VuMan 2 Wearable Computer. *IEEE Design and Test of Computers*, vol. 10, no. 4, 1993.

17. Watson, T. and Bershad, B. Local Area Mobile Computing on Stock Hardware and Mostly Stock Software. To appear in *Proceedings of the 1993 USENIX Symposium on Mobile and Location Independent Computing* (Cambridge, MA. Aug. 1–3).

18. Weiser, M. The Computer for the 21st Century. *Scientific American*, vol. 265, no. 3, September 1991, pp. 94–104.