Applying Model-Based Techniques to the Development of UIs for Mobile Computers

Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta

RedWhale Software Corporation
Town and Country Village Suite 273-277, Palo Alto, CA 94301
{jacob, jeanvdd, puerta}@redwhale.com

ABSTRACT

Mobile computing poses a series of unique challenges for user interface design and development: user interfaces must now accommodate the capabilities of various access devices and be suitable for different contexts of use, while preserving consistency and usability. We propose a set of techniques that will aid UI designers who are working in the domain of mobile computing. These techniques will allow designers to build UIs across several platforms, while respecting the unique constraints posed by each platform. In addition, these techniques will help designers to recognize and accommodate the unique contexts in which mobile computing occurs. Central to our approach is the development of a user-interface model that serves to isolate those features that are common to the various contexts of use, and to specify how the user-interface should adjust when the context changes. We claim that without some abstract description of the UI, it is likely that the design and the development of user-interfaces for mobile computing will be very time consuming, error-prone or even doomed to failure.

Keywords

User-interface modeling, mobile computing, task model, platform constraints, plastic user-interface, adaptive user-interface

INTRODUCTION

Mobile computing poses a series of unique challenges for user interface (UI) design and development [20]. UIs must run on many different computing platforms [21], ranging from the powerful workstation to the tiny cellular phone. Each computing platform has its own constraints: some devices are immobile (e.g., a home-based Internet Screen Phone) while others are mobile (e.g., a Personal Digital Assistant – PDA); some support extensive graphical capabilities (e.g., a large monitor), while others only provide limited interaction capabilities (e.g., a cellular phone); some are equipped with enhanced input/output devices (e.g., a trackball), while others are constrained by limited input [7, 12]. In addition, mobile computing increases the probability of environmental change while the user is carrying out the task: e.g., the train may go into a dark tunnel, forcing the screen of the PDA to dim; the surrounding noise level may rise, forcing the volume of audio feedback to increase so it can still be heard.

To meet these challenges, the most frequently adopted practice consists in developing unique UIs for each case. This poses further problems. Foremost is the unnecessary repetition involved in implementing a UI again and again, for each platform and usage case. In addition, a consistent UI design must be implemented across several platforms, even though that design will likely be implemented by many different designers, each with unique skills, experiences, and preferences. Revisions to the proposed design must be implemented multiple times, and the introduction of a new device requires a re-implementation of the UI.

Clearly, current practices for UI design for mobile computers are in need of significant improvement. We believe that **user interface modeling will be an essential component of any effective long term approach to developing UIs for mobile computing**. User-interface modeling [22] involves the creation of knowledge bases that describe various components of the user-interface, such as the presentation, the dialog, the platform, the task structure, and the context. These knowledge bases can be further exploited to automatically produce a usable UI matching the requirements of each context of use.

MANNA: THE MAP ANNOTATION ASSISTANT

In this section we describe MANNA, a hypothetical software application that reveals many of the challenges posed by user-interface development for mobile computing. MANNA is a multimedia application that must run on several platforms and can be utilized collaboratively over the internet. It is intended to be used by geologists, engineers, and military personnel to create annotated maps of geographical areas. Annotations can include text, audio, video, or even virtual reality walk-through.

In our scenario, a geologist from the United States Geological Survey has been dispatched to a remote location in northern California to examine the effects of a recent earthquake. Using a desktop workstation, our geologist downloads

IUI'01, January 14-17, 2001, Santa Fe, New Mexico, USA.

existing maps and reports on the area to prepare for her visit (fig. 1). The desktop workstation poses few limiting constraints to UI development, but unfortunately, it is totally immobile. The documents are downloaded to a laptop, and the geologist boards a plane for the site.

On the plane, the laptop is not networked, so commands that rely on a network connection are disabled. When the geologist examines video of the site, the UI switches to a black-and-white display, and reduces the rate of frames per second. This helps to conserve battery power. In addition, because many users find laptop touch pads inconvenient, interactors that are keyboard-friendly are preferred, e.g., drop-lists are replaced by list boxes.

After arriving at the airport, the geologist rents a car and drives to site. She receives a message through the MANNA system to her cellular phone, alerting her to examine a particular location. Because a cellular phone offers extremely limited screen-space, the map of the region is not displayed. Instead, the cell phone shows the geographical location, driving directions, and the geologist's current GPS position. A facility for responding to the message is also provided.

Finally arriving at the site, our geologist uses a palmtop computer to make notes on the region (fig. 6). Since the palmtop relies on a touch pen for interaction, interactors that require double-clicks and right-clicks are not permitted. Screen size is a concern here, so a more conservative layout is employed. Having completed the investigation, our geologist prepares a presentation in two formats. First, an annotated walk-through is presented on a heads-up display (HUD). Because of the HUD's limited capabilities for handling textual input, speech-based interactors are used instead. A more conventional presentation is prepared for a high-resolution large-screen display. Since this is a final presentation, the users will not wish to add information, and interactors that are intended for that purpose are removed. The layout adapts to accommodate the larger screen space, and important information is placed near the center and top, where everyone in the audience can see it.

THE USER-INTERFACE MODEL

The highly adaptive, multi-platform user-interface described in this scenario would be extremely difficult to realize using conventional UI-design techniques. We will describe a set of model-based techniques that can greatly facilitate the design of such UIs.

All such techniques depend on the development of a user-interface model, which we define as a formal, declarative, implementation-neutral description of the UI. A UI model is expressed by a modeling language; that language should be declarative, so that it can be edited by hand, but it should be formal so that it can be understood and analyzed by a software system. The MIMIC modeling language meets these criteria, and it is the language we have chosen to use for UI modeling [16].

MIMIC is a *comprehensive* UI modeling language; ideally, all relevant aspects of the UI are included in a MIMIC UI model. However, we will focus on the three model components that are relevant to our design techniques for mobile computing: platform model, presentation model, and task model.

A *platform model* describes the various computer systems that may run a UI [19]. This model includes information regarding the constraints placed on the UI by the platform. The platform model contains an element for each platform that is supported, and each element contains attributes describing features and constraints. The platform model may be exploited at design time and be used as a static entity. In this case, a set of user-interface can be generated: one for each platform that is desired. However, we prefer the dynamic exploitation of the platform model at run-time, so that it can be sensitive to changing conditions of use. For example, the platform model should recognize a sudden reduction in bandwidth, and the UI should respond appropriately.

A *presentation model* describes the visual appearance of the user interface. The presentation model includes information

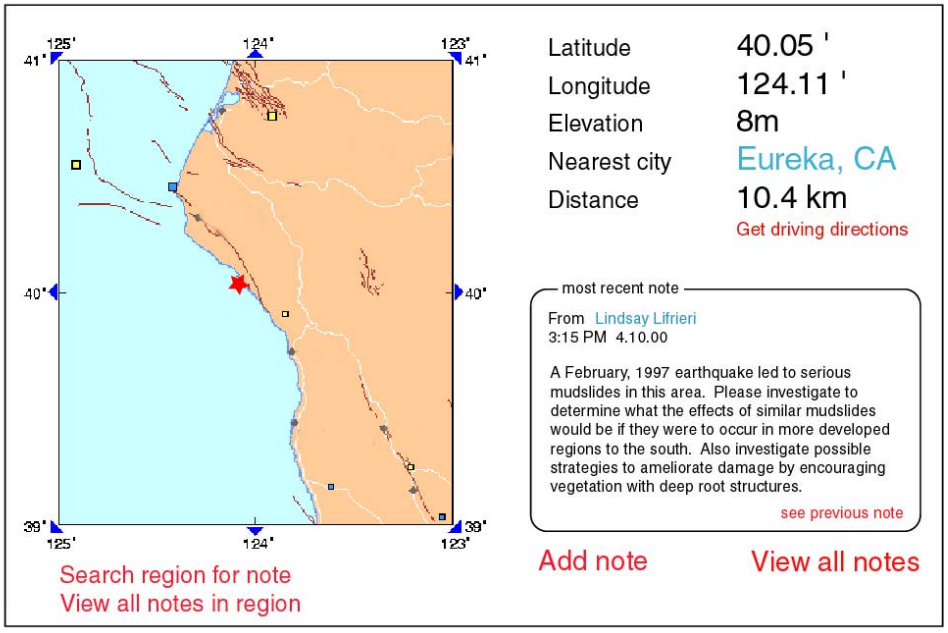

Figure 1. The desktop UI design for MANNA. Hyperlinks are indicated by color, and are underlined only when the mouse rolls over them.

describing the hierarchy of windows and their widgets (e.g., sliders, list boxes), stylistic choices, and the selection and placement of these widgets. Widgets are described in accordance with the traditional distinction between AIOs and CIOs [25]. Each widget is modeled abstractly as an AIO: an abstract interaction object, which is platform-neutral. Then each AIO is associated with several CIOs: concrete interaction objects, which are executable on a specific platform. CIOs inherit some of their behavior from AIOs, and may supply some additional parameters. The CIO/AIO distinction allows our UI models to run on any computing platform, as long as the appropriate CIOs are present. Figure 2 describes the relationship between an AIO (the Push Button), several related CIOs, and the platforms on which they are instantiated.

A *task model* is a structured representation of the tasks that the user of the software may want to perform. The task model is hierarchically decomposed into subtasks, and information regarding goals, preconditions, and postconditions may be supplied [23]. In addition, we model features such as whether a task is optional, whether it may be repeated, and whether it enables another sub-task.

By no means are these the only models that we consider relevant to mobile computing. For many applications, it is essential to model the users themselves, especially when there are multiple users with different preferences, abilities, and privileges. It is also often appropriate to model the domain characteristics of the tasks supported by the UI. Such information often guides the selection of widgets [9,13,25]. However, the techniques that we have developed for UI design for mobile computing depend only on the three models already described.

Although the internal structure of each model component plays a significant role in the overall design of the UI, our approach places special emphasis on the connections between the various model components. We feel that it is these mappings that determine the interactive behavior of a UI. For mobile UIs, we are most concerned with the connections between the platform model and the presentation model. These connections describe how the constraints posed by the various platforms will influence the visual UI appearance. We are also interested in the task model, and how it relates to the other two models. The next section will describe how some platforms naturally lend themselves to specific tasks, and how this information can be used to tailor the presentation accordingly.

TECHNIQUES FOR SUPPORTING MOBILE UIs

In this section, we describe a spectrum of model-based techniques that can be used to support mobile computing. Each technique involves creating mappings between the various model components that we have discussed. These mappings are interpreted to produce a UI that is specially customized for the relevant device and context of use.

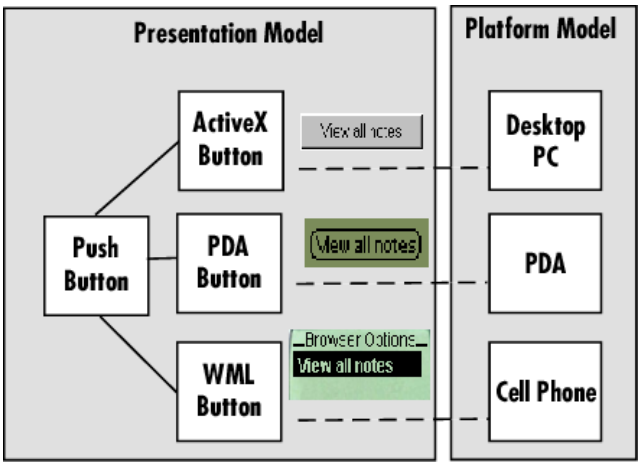

Figure 2. Concrete Interaction Objects are subclassed from an Abstract Interaction Object, and are then mapped on to specific platforms.

Handling Platform Constraints

One obvious constraint that is often posed by mobile computing platforms is the display resolution, which can range from a wall-size flat screen to a cellular phone to a head-mounted immersive environment. We will use this screen resolution constraint as an example case, and describe some methods for dealing with it, while showing how these methods might also be applied to other constraints. The screen resolution constraint was chosen because we feel that in many ways it is the most difficult constraint to deal with; whereas other constraints, such as limited interaction capabilities, can be handled through interactor selection, the screen space constraint often requires a more global solution.

The screen resolution is typically expressed in pixels (e.g., 1024x768). It should not be confused with the screen surface area; two displays having different sized surfaces can share the same resolution. Because of screen resolution differences, an optimal layout for one display may be simply impossible to render on another device. This problem is particularly salient for mobile computing, because so many mobile platforms possess small-size, low-resolution displays. There are three parameters that contribute to the amount of display size required by a user-interface design: size of the individual interactors, layout of interactors within a window, and the allocation of interactors among several windows.

Interactor Size.

There are two possible methods for accommodating screen resolution constraints by adjusting the size of the interactors. The first possibility is simply to shrink the interactors, while observing usability constraints related to the AIO type. For example, the length of an edit box can be reduced to a minimum (e.g., 6 characters visible at the same time with horizontal scrolling) while its height cannot be decreased below the limit of the smallest font size legible (e.g., 8

pixels); usability experiments have determined that the minimum size for an icon is roughly 8 by 6 pixels.

Of course, many interactors simply cannot be shrunk to any significant extent. An alternative to reducing the size dimensions of an interactor is to replace that interactor with a smaller alternative. For example, a Boolean checkbox typically requires less screen space than a pair of radio buttons. The technique of automatically selecting an appropriate interactor while considering screen resolution constraints has already been investigated and shown to be feasible [9,13,26].

These techniques can easily be applied to constraints other than screen resolution. Interactors can be parameterized to reduce bandwidth usage or battery consumption–for example, a video player can reduce the frame rate or the number of colors. This parameterization is similar to reducing the display size of interactors. Likewise, interactor selection can be performed to accommodate reduced interaction capabilities. For example, a handwriting-based interactor from a PDA could be replaced with a voice-based interactor on a cellular phone. When there are several criteria that need to be optimized – e.g., screen size, usability, and power consumption – interactor selection can become a multidimensional optimization problem. An algorithm for solving this problem would be beyond the scope of this paper, but we hope to explore it in the future.

Returning to the matter of screen resolution, we see that both techniques for adjusting the size of the interactors in a UI can help us to find an appropriately-sized presentation. However, we believe that in many cases, a more global solution is necessary. A WAP-enabled cellular phone can display only one or two interactors at a time, no matter how small they are. To achieve the amount of flexibility necessary to support all mobile devices, we will have to examine window layout and the allocation of interactors among windows.

Selecting the Appropriate Presentation Structure

The remaining two parameters—layout of interactors within a window, and allocation of interactors between windows—can be grouped together under the mantle of *presentation structure*. We want to select the appropriate presentation structure, given the constraint of the amount of screen-resolution afforded by a platform. To solve this problem, we need a set of alternative presentation structures, which can be generated by the designer or with the help of the system. The simplest solution would then involve creating mappings between each platform and an appropriate presentation structure.

However, in this case a more dynamic solution is possible. Recall that the screen resolution of each device is represented declaratively in the platform model. Similarly, it is possible to represent the amount of screen space required by each presentation structure in the presentation model. We can exploit this knowledge by constructing an intelligent *mediator* agent [1,2]. This mediator should determine the maximum usable screen resolution for the relevant device, and evaluate the amount of screen resolution required by each presentation structure alternative. It can then select the presentation structure that consumes an amount of screen resolution that falls just under the maximum (fig. 3).

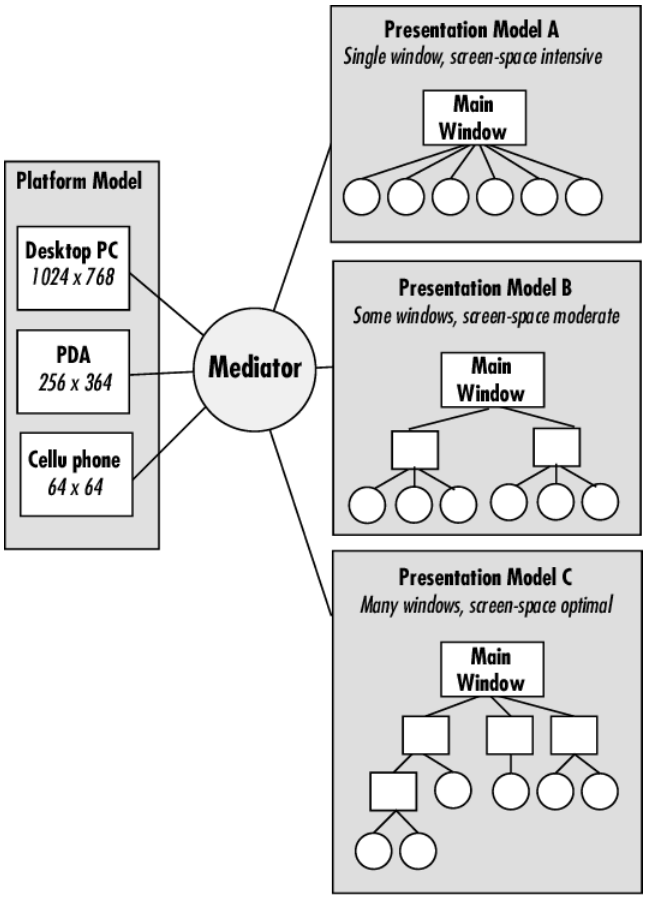

Figure 3: A mediating agent dynamically selects the appropriate presentation model for each device.

This more dynamic solution is preferable because it accounts for the fact that the screen resolution of a device may change while it is in use. Moreover, it eases the integration of new devices into the platform model. Rather than forcing the user to explicitly specify the appropriate presentation structure for a new device, the user needs only to specify the amount of available screen space, and the mediator will find the correct mapping.

Generating the Appropriate Presentation Structure

Under our proposed architecture, it is still left to the interface designer to specify a set of alternative presentation structures. However, it would be better if the correct presentation structure could be generated by the system, given a set of user-defined constraints [1,13,14,26]. For this purpose, we need to consider additional elements in our presentation

model besides AIOs and CIOs. Two new abstractions need to be defined:

1. *Logical Window* (LW): this can be any grouping of AIOs—a physical window, a subwindow area, a dialog box or a panel. Every LW is itself a composite AIO as it is composed of other simple or composite AIOs. All LWs should be physically constrained by the user's screen.
2. *Presentation Unit* (PU): a PU is defined as a complete presentation environment required for carrying out a particular interactive task. Each PU can be decomposed into one or many LWs, which may be displayed on the screen simultaneously, alternatively, or in some combination thereof. Each PU is composed of at least one window called the *main window*, from which navigation to other windows is allowed. For example, a tabbed dialog box can be described as a PU, and it should be decomposed into LWs corresponding to each tab page.

The abstractions can be structured into a hierarchy (fig. 4) that serves to expand modeling capabilities of a presentation model. We can use this hierarchy to construct an automated design tool that generates several platform-optimized presentation models from a starting presentation model that is platform independent. This tool could function by employing one of the redesign strategies described below.

1. *Re-modeling LWs within a PU*: the contents of initial LWs are redistributed into new LWs within a single PU. Basic operations involve: ungrouping an LW into several smaller LWs; grouping the contents of several LWs into a single, comprehensive LW; and moving AIOs from one LW to another. For example, if ample space is available, then the LWs representing several tab sheets can be grouped into a single LW. Alternatively, if space is at a premium, the contents of a single window can be broken into pages of a tab sheet. Some existing tools apply some of these operations. SEGUIA suggests configurations based on the semantics [26]: minimal (as many logical windows as possible), maximal (one logical window for a PU), input/output (one logical window gathering input while another gathers output for a same function), functional (LWs depending on the functional analysis), and free (as many LWs as the designer wants to). TIMM [18] enables designers to graphically specify how many LWs they want by sliding a cursor on a scroll bar. At one end of the scroll bar, only one LW will be used in a PU; at the other end, one LW is used for each AIO for each LW.
2. *Re-modeling AIOs within a LW*: according to existing constraints, the contents of an initial LW are redistributed into new AIOs, composite or simple. For example, AIOs contained in a group box are split into smaller group boxes or individual AIOs. A group box can also be replaced by a push button that displays the AIO contained in this box on demand. This technique remains unexplored today. Moreover, we are aware of no unambiguously successful algorithm for the automatic generation of a presentation structure based on these abstractions that has yet been documented.

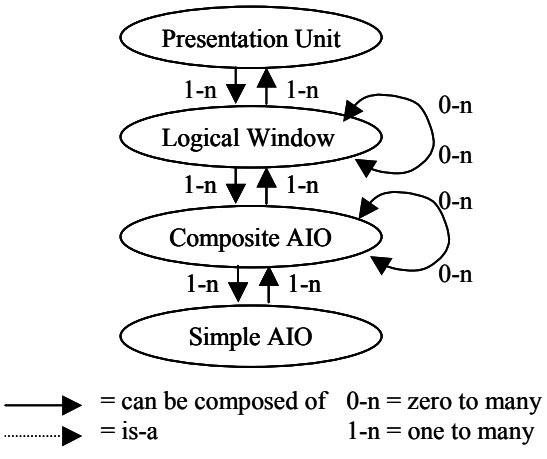

Figure 4. Hierarchy of presentation concepts.

Obviously, both of these techniques leave out the difficult question of dialog layout. MOBILE is a model-based layout tool which designers could use for this purpose [17].

As we have said, screen resolution is not the only constraint that must be negotiated. Other constraints include bandwidth usage, battery power consumption, number of display colors, and interaction capabilities. However, unlike the screen-space constraint, these problems do not appear to require a global strategy; they can usually be accommodated through AIO selection. Previous research has addressed the problem of dynamic interactor selection at length, and the issue of platform-specific input constraints has been considered [9,13,25]. We realize that additional constraints may present themselves in the future. While the techniques described in this paper cannot be expected to handle any and all platform-based constraints arising from mobile computing, we feel that this work can serve as a starting point for the development of further model-based solutions.

Focusing on Contexts of Use

So far we have assumed that on each device, the user will want to accomplish the same set of tasks. Often this is not the case. A device may be especially suited for a specific subset of the overall task model. For example, in the scenario in Section 2, we know that the cellular phone is especially suited for finding driving directions, because we can assume that if the user were not driving, she would use the PDA. The desktop workstation cannot be brought out into the field, so it is unlikely that it will be used to enter new annotations about a geographical area; rather, it will be used for viewing

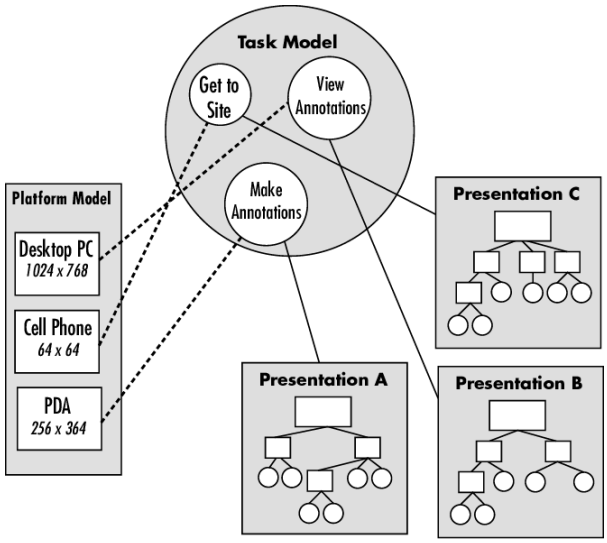

Figure 5. Platform elements are mapped onto high-level task elements that are especially likely to be performed. In turn, the task elements are mapped onto presentation models that are optimized for the performance of a specific task.

annotations. Conversely, the highly mobile PDA is the ideal device for entering new annotations.

Through the application of a task model, we can take advantage of this knowledge and optimize the UI for each device. The designer should create mappings between platforms (or classes of platforms) and tasks (or sets of tasks). Additional mappings are then created between task elements and presentation structures that are optimized for a given set of tasks. We can assume these mappings are transitive; as a result, the appropriate presentation model is associated with each platform, based on mappings through the task model. The procedure is depicted in fig. 5. In this figure, the task model is shown to be a mere collection of tasks. This is for simplicity's sake; in reality, the task model is likely to be a highly structured graph where tasks are decomposed into subtasks at a much finer level than shown here.

There are several ways in which a presentation model can be optimized for the performance of a specific subset of tasks. Tasks that are thought to be particularly important should be represented by AIOs that are easily accessible. For example, on a PDA, clicking on a spot on the map of our MANNA application should allow the user to enter a new note immediately (fig. 6). However, on the desktop workstation, clicking on a spot on the map brings up a rich set of geographical and meteorological information describing the selected region, while *showing* previously entered notes. On the cellular phone, driving directions are immediately presented when any location is selected (fig. 7). On the other devices, an additional click is required to get the driving directions. The "bottom arrow" button of the cellular phone enables the user to select other options by scrolling between them.

In this way, our treatment of optimizations for the task structure of each device is similar to our treatment of the screen-space constraint: global, structural modifications to the presentation model are often necessary, and adaptive interactor selection alone will not suffice. Here, we propose to solve this problem through the use of several alternative user-generated presentation structures. In many cases, automated generation of these alternative presentation structures would be preferable. We hope to explore the automated generation of task-optimized presentation structures in the future.

FUTURE WORK

In the introduction, we mentioned a lack of support for the observation of usability guidelines in the design of user-interfaces for mobile computing. We have experience in implementing model-based UI development environments that incorporate usability guidelines [25,26]. However, the lack of a substantial existing literature on usability guidelines for mobile computing [3,5] prevents us for implementing such a strategy here. As the research literature grows, we hope to provide support for applying these guidelines to mobile UI development.

Much of the information found in the platform and task models can be expressed as UI constraints: e.g., screen size, resolution, colors, available interaction devices, task structure, and task parameters (e.g., frequency, motivation). It is therefore tempting to address the problem of generating alternative presentation structures as a constraint-satisfaction problem. In this case, there are two types of constraints: first, features that are common to the user-interface across platforms and contexts of use; second, platform-specific constraints such as screen resolution and available bandwidth. In our future research, we hope to build a constraint-

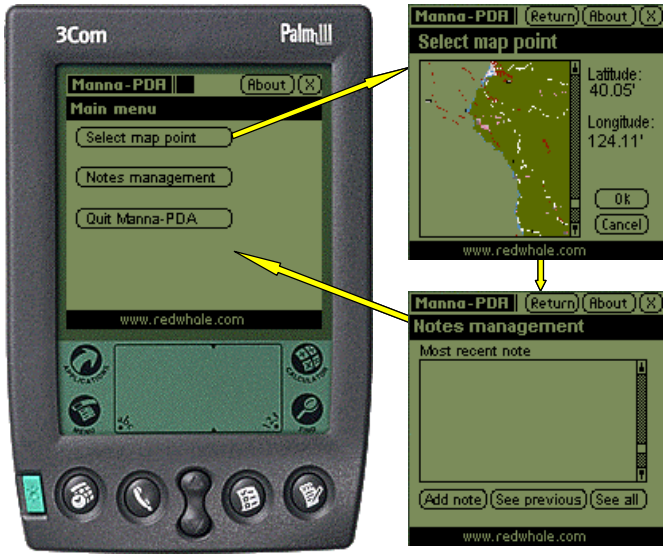

Figure 6. Final presentation for the PDA.

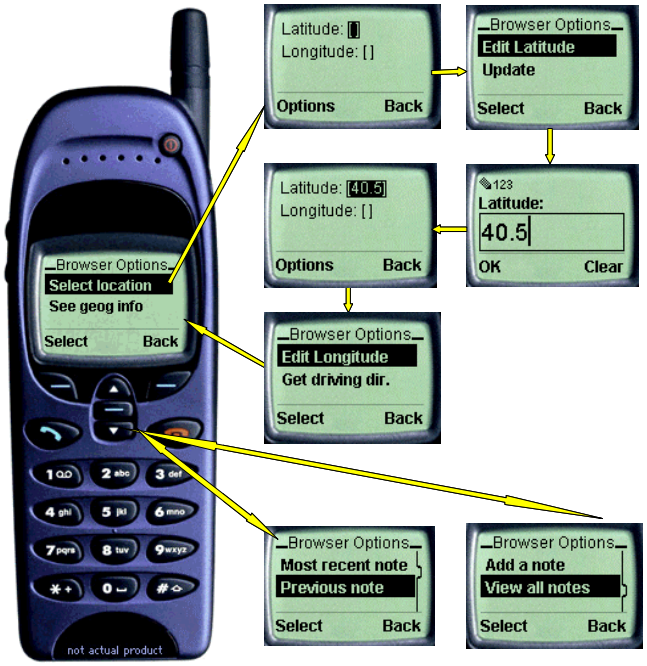

Figure 7. Final presentation for the cellular phone.

satisfaction system that automatically generates presentation structures.

Another possible solution to the problem of generating presentation structures for mobile devices is search. This strategy is motivated by the observation that user-interface guidelines are not always best represented as constraints; sometimes a usability heuristic is a more useful measure. In this case, we can assign a certain penalty to layout decisions, AIO selections, and presentation structures. As there are several usability guidelines, there will be several types of penalties; for example, an AIO may be optimal in regard to usability (and thus incur very little penalty), but may consume too much space (thus incurring a substantial penalty). We hope to design a system that searches for the best possible UI design by minimizing this penalty heuristic. We could then compare the performance of constraint-satisfaction and search as techniques for generating optimal presentation structures.

RELATED WORK

User-interface modeling for traditional, non-mobile GUIs has been explored in depth [1,13,14,16,22,26]. Several papers describe graphical model editing tools that relieve the user of the burden of learning a modeling language [10,14,15,16,17,23,25].

On an implementation level, the well-understood model-based distinction between AIOs and CIOs figures to play a key role in cross-platform development of all kinds, including mobile computers. Of particular interest is [13], which allows AIOs to be mapped to CIOs depending on user parameters, with the aim of accommodating users with special needs (e.g., user with disabilities). Galaxy [10] and Open Interface render a UI definition on different platforms with their native look and feel, while SUIT [15] provides an additional GUI layer for each supported environment on top of which the UI definition is rendered. The rendering is consequently identical on all supported computing platforms. Although these tools contain some presentation abstractions that are expressed in a platform-neutral format, these abstractions do not accommodate platform constraints.CT-UIMS was the first tool to add a platform component to a user-interface model; it supported some AIO [14] redistribution for OSF/Motif large screens and small Macintosh screens. It had the capability of dynamically deciding which part of a PU can be displayed, by filling the screen up to its maximum. Some LWs can then be omitted if screen constraints are too tight to allow further display.Cicero [1] includes a mediator that tries to allocate resources depending on constraints known at run-time. It also uses a model-based approach, but is more intended to support multimedia presentations rather than GUIs. In [26], Vanderdonckt et al. show how PUs and LWs can be used for database-oriented applications. Thevenin and Coutaz [24] are reusing these concepts for analyzing plastic UIs when a presentation and/or a dialog change according to any non-user variation such as the platform, the context, and the interaction capabilities. For example, a UI can change at run-time to accommodate a reduction of screen real estate.TIMM, a system for automatic selection of interactors, is described in [9,18]; it outputs a set of alternative interactors, and uses a decision tree that takes into account constraints such as screen space. In addition, it includes an adaptive component that learns the designer's preferences.In general, the research efforts described in this section were not directed at the problems posed by mobile computing.

CONCLUSION

Technological push and user pull in the domain of mobile computing are increasing the demand for consistent user-interfaces on a variety of platforms and in a variety of contexts. In this paper, we have described a spectrum of techniques for developing a single, consistent user-interface design for several mobile devices and contexts of use. These techniques range from relatively low-level implementation solutions, such as the use of abstract and concrete interactor objects, to high-level task-based optimization of the interface's presentation structure. Each technique requires that the user-interface be described from a position of abstraction. Our central claim is that the use of abstract, platform-neutral models to describe the user-interface greatly facilitates the development of consistent, usable multi-platform user-interfaces for mobile devices.

REFERENCES

1. Y. Arens and E.H. Hovy, "The Design of a Model-Based Multimedia Interaction Manager", *Artificial Intelligence Review*, Vol. 9, Nos. 2-3 (1995), pp. 167-188.

2. Y. Arens, L. Miller, S.C. Shapiro, N. Sondheimer, "Automatic Construction of User-Interface Displays", *Proc. of AAAI'98* (St. Paul, 21-26 August 1988), AAAI Press / The MIT Press, 1988 pp. 808-813.
3. C. Bey, E. Freeman, J. Ostrem, "Palm OS Programmer's Companion", Palm Inc. 1996-2000.
4. K.A. Bharat, L. Cardelli, "Migratory Applications Distributed User Interfaces", *Proc. of UIST'95* (Pittsburgh, 14-17 November 1995), ACM Press, New York, 1995, pp.133-142.
5. S.A. Brewster, P.G. Cryer, "Maximising Screen-Space on Mobile Computing Devices," *CHI'99 Extended Abstracts* (Pittsburgh, 15-20 May 1999), ACM Press, New York, 2000, pp. 224-225.
6. J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, R. Young, "Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE Properties", Proc. Of IFIP Int. Conf. on Human-Computer Interaction Interact'95, Chapman & Hall, London, 1995, pp. 115-120.
7. N. Davies, K. Mitchell, K. Cheverst, and A. Friday, "A Caches in the Air: Disseminating Tourist Information in the Guide System", *Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications* (New Orleans, September 1999), IEEE Computer Society Press, Los Alamitos, 1999, pp. 11-19.
8. M.R. Ebling, M. Satyanarayanan, "On the Importance of Translucence for Mobile Computing", *Proc. of the 1st Workshop on Human Computer Interaction with Mobile Devices* (Glasgow, 21-23 May 1998), Ch. Johnson (ed.), GIST Technical Report G98-1, Glasgow, 1998, p. 11.
9. J. Eisenstein and A. Puerta, "Adaptation in Automated User-Interface Design", *Proc. of IUI'2000* (New Orleans, 9-12 January 2000), ACM Press, New York, 2000, pp. 74-81.
10. "Galaxy Application Environment", Ambiência Information Systems, Inc., Breckenridge, 2000.
11. Ch. Gram and G. Cockton, G. (Eds.), "Design Principles for Interactive Software", Chapman & Hall, London, 1996.
12. Ch. Johnson, "The Impact of Time and Place on the Operation of Mobile Computing Devices", *Proc. of the HCI'97 Conference on People and Computers XII* (Bristol, 1997) .H. Thimbleby, B. O'Conaill, P.J. Thomas (eds.), Springer-Verlag, London, 1997, pp.175-190.
13. S. Kawai, H., Aida, T., Saito, "Designing Interface Toolkit with Dynamic Selectable Modality", *Proc. of ACM 2nd Int. Conf. on Assistive Technologies ASSETS'96* (Vancouver, April 11-12, 1996), ACM Press, New York, 1996, pp. 72–79.
14. Ch. Märtin, "A UIMS for Knowledge Based Interface Template Generation and Interaction", *Proc. of IFIP Int. Conf. on Human-Computer Interaction Interact'90*, Elsevier Science Pub., Amsterdam, 1990, pp. 651-657.
15. R. Pausch, M. Conway, and R. DeLine, "Lessons Learned from SUIT, the Simple User Interface Toolkit", *ACM Trans. on Office Information Systems*, Vol. 10, No. 4, October 1992, pp. 320-344.
16. A.R. Puerta, "The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development", *Proc. of CADUI'96* (Namur, 5-7 June 1996), Presses Universitaires de Namur, Namur, 1996, pp. 19-36.
17. A.R. Puerta, E. Cheng, Ou, T., and Min, J. MOBILE: User-Centered Interface Building. CHI99: ACM Conference on Human Factors in Computing Systems. Pittsburgh, May 1999, in press.
18. A.R. Puerta, J. Eisenstein, "Towards a General Computational Framework for Model-Based Interface Development Systems", Knowledge-Based Systems, Vol. 12, 1999, pp. 433-442.
19. S. Prasad, "Models for Mobile Computing Agents", ACM Comput. Surv. 28 (4), Dec. 1996, Article 53.
20. M. Satyanarayanan, "Fundamental Challenges in Mobile Computing", *Proc of the fifteenth annual ACM symposium on Principles of distributed computing*, ACM Press, New York, 1996, pp. 1-7.
21. P. Szekely, "Retrospective and Challenges for Model-Based Interface Development", *Proc. of 3rd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (Namur, 5-7 June 1996), Presses Universitaires de Namur, Namur, 1996, pp. xxi-xliv.
22. P. Szekely, P. Luo, and R. Neches, "Beyond Interface Builders: Model-Based Interface Tools", *Proc. of InterCHI'93*, ACM Press, New York, 1993, pp. 383-390.
23. P. Szekely, P. Sukaviriya, P. Castells , J. Muthukumarasamy, and E. Salcher, "Declarative interface models for user interface construction tools: the MASTERMIND approach", *Engineering for Human-Computer Interaction*, L.J. Bass and C. Unger (eds), Chapman & Hall, London, 1995, pp 120-150.
24. D. Thevenin and J. Coutaz, "Plasticity of User Interfaces: Framework and Research Agenda", *Proc. of INTERACT'99* (Edinburgh, August 1999), IOS Press, 1999.
25. J. Vanderdonckt, F. Bodart, "Encapsulating Knowledge for Intelligent Interaction Objects Selection", *Proc. of InterCHI'93*, ACM Press, New York, 1993, pp. 424-429.
26. J. Vanderdonckt, P. Berquin, "Towards a Very Large Model-based Approach for User Interface Development", *Proc. of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99*, IEEE Computer Society Press, Los Alamitos, 1999, pp. 76-85.