

Adapting to Mobile Contexts with User-Interface Modeling

Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta

RedWhale Software Corporation, Town and Country Village Suite 273-277, Palo Alto, CA 94301

E-mail: jacob@redwhale.com, jeanvdd@redwhale.com, puerta@redwhale.com

Abstract

Mobile computing offers the possibility of dramatically expanding the versatility of computers, by bringing them off the desktop and into new and unique contexts. However, this newfound versatility poses difficult challenges for user-interface designers. We propose three model-based 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. All three techniques depend on the development of a user-interface model which 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. User-interface models allow automatic and automated tool support that will enable UI designers to overcome the challenges posed by mobile computing.

1. Introduction

As mobile computing today grows more complicated and offers a variety of access devices for multiple contexts of use, it poses a series of unique challenges for user-interface (UI) design and development [20]:

- Interactive applications and their UIs must run on many different computing platforms [21], ranging from the powerful workstation to the tiny cellular phone (fig. 1). These differences in computing power pose restrictions on the user-interface that is to be implemented.
- Mobile computing platforms have unique and challenging input and output constraints. For example, some support extensive graphical capabilities (e.g., a large monitor), while others only provide very limited display resolutions (e.g., a cellular phone); some are equipped with enhanced input/output devices (e.g., a trackball or keyboard), while others are constrained by limited input (e.g., a touch pen) [7, 12].
- The user-interface should be designed to accommodate and take advantage of the varying contexts of use for which mobile devices are suited. For example, a home-based Internet Screen Phone is immobile, while a PDA is mobile; thus, the PDA is suited for different tasks than the Screen Phone.

- Mobile computing increases the probability of environmental or contextual 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; the user receives a phone-call while working, forcing the system to disable voice-based interaction and provide visual substitute interactors.



Figure 1. Some mobile access devices.

The problem becomes even more complex when we consider that users may move between different platforms while carrying out a single task [4] or many tasks [8]; e.g., a reader who wants to buy a book might initially search for it from her desktop computer, download the reviews of the book on a PDA to read them on a train, and then order it on a cellular phone. Additionally, users may want to collaborate on a task while using heterogeneous platforms. For example, a traveling user reviews a presentation plan with a supervisor: the supervisor would use a desktop workstation, while the mobile user would use a PDA. In handling all of these platform and contextual constraints, it is critical to preserve consistency and usability.

In summary, mobile computing requires that UIs be sensitive to [1,7,12]:

- *Platform*: by having presentations that adapt to screen surface, color depth, screen resolution and dialogs that adapt to network bandwidth;
- *Interaction*: by having mechanisms that remember previously used interaction techniques, windows sizes and locations, and respect user preferences for these presentations;
- *User*: by adapting to user experience level, system and task experiences, skills, conventions, and preferences.

To meet these challenges, the most frequently adopted practice consists in developing unique UIs for each case. This poses further problems. A similar and consistent UI should be implemented for several platforms, thus raising

the need for the cross-platform development, while maintaining compatibility both at design and at development times. This repetitive process can be resource-consuming and complex because each platform presents its own set of constraints to be satisfied. A slight change in the UI requirements may induce many important modifications in the resulting code.

It is almost inevitable that this cross-platform design will be performed by several individuals, who may have different backgrounds and experiences, thus reinforcing the difficulty of coordinating styles and consistency across the different UIs. Moreover, the design is never quite finished: as new mobile devices continue to proliferate, software developers will need to accommodate them quickly and cheaply. This process will be substantially decelerated if the UI must be completely redesigned and re-implemented for each device. This problem is exacerbated by the fact that UI code is often among the last portable of all source code for an application.

Finally, the variations of the contexts of use and the different devices are subject to several usability issues that designers may not be aware of. The current set of commercially available software tools for developing UIs offer little or no guidance in helping developers to navigate these issues.

For these reasons, we believe that current practices for UI design for mobile computers are in need of significant

improvement. We believe that **user interface modeling [22] will be an essential component of any effective long term approach to developing UIs for mobile computing.** User-interface modeling 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 exploited by a UI-development toolkit to help designers to produce UIs matching the requirements of each context of use. Many design decisions can even be automated by intelligent UI design software.

The remainder of this paper is structured as follows: we begin in Section 2 with a detailed scenario of a mobile computing application whose development would benefit significantly from a model-based approach. Section 3 will describe our ontology for user-interface modeling. Section 4 will introduce and discuss three techniques that exploit these models to facilitate UI development for mobile computing applications. Section 5 will put forth some ideas on how this work can be expanded and developed in the future. Section 6 will discuss related efforts in the user-interface modeling community, and will show where our work stands in relation to the existing literature. Section 7 will draw some conclusions about the model-based techniques that we have described and their application to mobile computing.

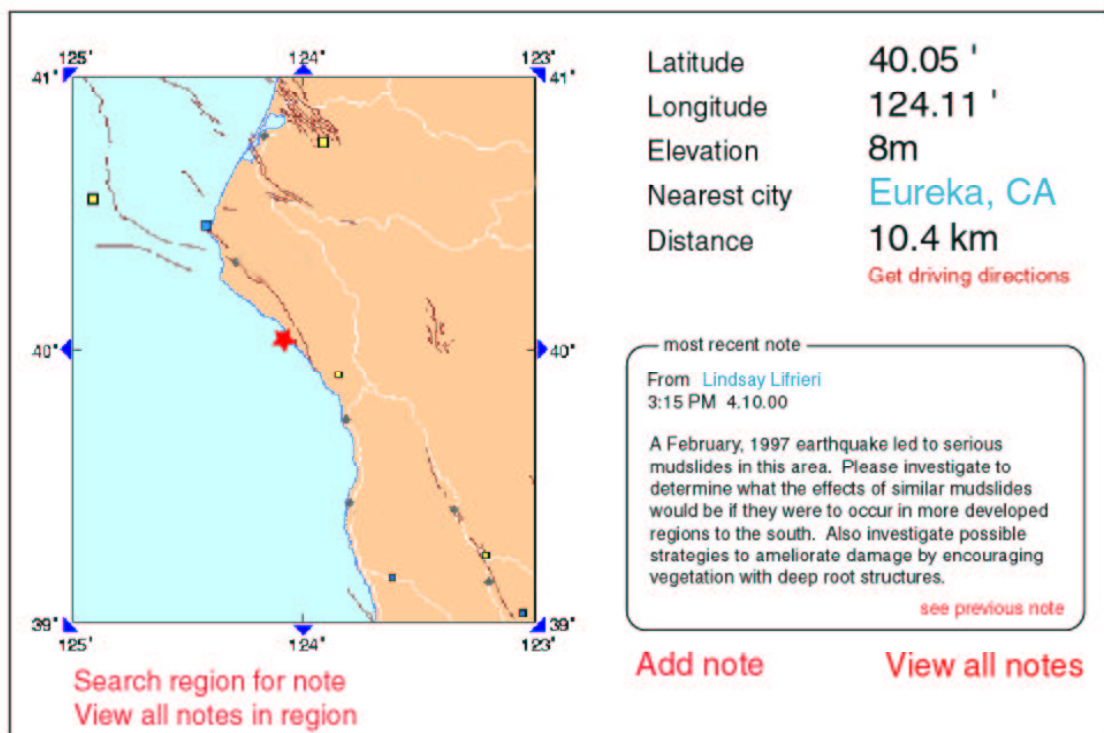


Figure 2. The desktop UI design for MANNA. Hyperlinks are indicated by color, and are underlined only when the mouse rolls over them.

2. 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 existing maps and reports on the area to prepare for her visit (fig. 2). 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 (**Figure 3**). 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 completing 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

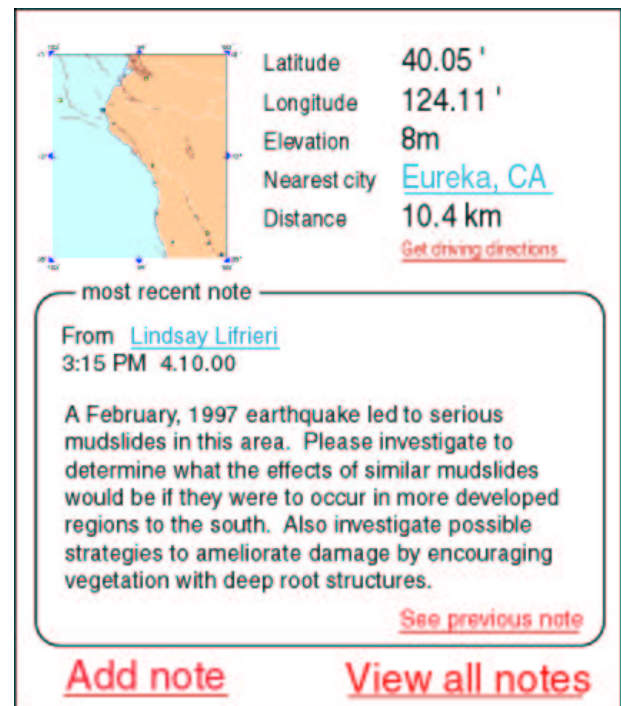


Figure 3: Proposed PDA UI design for MANNA. There is no color display, so all hyperlinks are underlined. Clicking on the map will bring on a full-screen view.

placed near the center and top, where everyone in the audience can see it.

3. The user interface model

The highly adaptive, context-sensitive, 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 should be expressed by a modeling language. A UI modeling language should be *declarative*: it should be comprehensible to humans, even if they are not familiar with the software tools that support the language. However, it should also 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].

A user-interface modeling language should be entirely platform-independent. For each platform to be supported, a tool must convert from the modeling language to some kind of runnable program code: e.g., Swing, HTML, WML. This conversion can be performed at compile-time, to generate static user interfaces, or at run-time, to generate dynamic user interfaces.

MIMIC is a *comprehensive* UI modeling language; ideally, all relevant aspects of the UI are included in a

MIMIC UI model. This represents the major difference between the model-based approach and the UIMS approach. A UIMS only deals with information about presentation and dialog structure; it does not include a model of the context, task structure, or user. 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 may act as a static entity, in which case it can be exploited at design time. This enables designers to generate a set of user-interfaces, 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 describing the hierarchy of windows and their widgets (e.g., sliders, list boxes), stylistic choices, and the selection and placement of these widgets [25].

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 post-conditions 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 three 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 tell us how the constraints posed by the various platforms will influence the visual UI appearance. We are also concerned with the task model, and how it relates to the other two models. We will see in the next section how some platforms naturally lend themselves to specific tasks, and this allows us to tailor

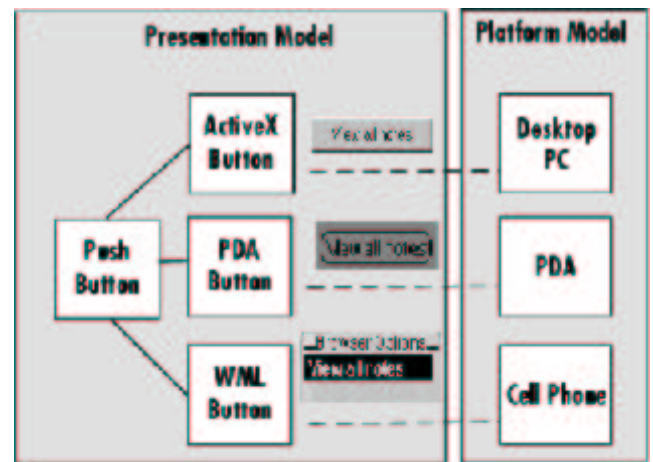


Figure 4. Concrete Interactor Objects are subclassed from an Abstract Interactor, and are then mapped onto specific devices.

the presentation accordingly.

4. Adaptation in mobile devices

The proliferation of mobile devices has led to a dramatic increase in the number of variety of contexts that must be accommodated by user-interface software. When the user-interface changes dynamically in response to a new context, it is said to be *adaptive*. Adaptation may occur at design time, in the form of automated design support. It may also occur at run-time, creating a dynamic user-interface.

In this section, we describe a spectrum of model-based techniques for adapting to the varying contexts of mobile computing. Each technique involves creating mappings between the various model components that we have discussed. When these mappings are interpreted, a static or dynamic UI is created, specially customized for the relevant device and context of use.

4.1. The abstract interaction object

The first technique we will discuss is the use of *abstract interaction objects* (AIOs). The use of AIOs has been documented elsewhere [25], although not in conjunction with mobile computing. An interaction object (sometimes called a *widget*) is any element that allows users of an application to visualize or manipulate information, or to perform an interactive task. From a modeling perspective, these objects are often viewed as the atomic building blocks of a UI.

We can draw a distinction between Concrete Interaction Objects (CIOs) and Abstract Interaction Objects (AIOs). CIOs are interactors that are executable on some platform without any additional processing. Although a CIO is executable, it is not necessary to understand its implementation details, and its behavior does not depend on program code elsewhere in the application, except that it

can be parameterized.

Unlike a CIO, an AIO is not executable on any platform; it is agnostic as to how it is to be implemented. For this reason, AIOs are completely portable. However, that implementation information must be found elsewhere in the UI, or else it must be implicit in the design of the tool that interprets the UI modeling language.

We define a presentation model architecture that is very similar to an object hierarchy; each AIOs is the *parent* of several CIOs, which inherit all the properties of the AIO while supplying some additional information. Each CIO is then mapped on to the platform(s) on which it is supported. The user-interface designer need only think in terms of the AIOs desired. When the UI model is interpreted, the appropriate CIO is rendered automatically, depending on the platform. A single AIO may be rendered through one or many CIOs depending on the platform considered. This architecture can be visualized in fig. 4.

AIOs are one piece of the puzzle: they allow us to implement the *same* UI on several platforms. But as we have seen in the scenario, each platform may pose special constraints on UI design—especially in the case of mobile computing. It is not enough to implement the same UI on each platform, if that UI will violate some platform-based constraint. Rather, we need to implement a UI that has been customized for each platform.

4.2. Adapting to Interaction Constraints

Each mobile platform imposes constraints on the input and output capabilities of the software. Many of these constraints were discussed in the introduction. 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. 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. 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 computing platforms involve small-size, low-resolution displays.

We will focus on screen resolution because we feel that screen-size adaptation is in some ways more difficult than many other forms of adaptation. The reasons for this will become apparent as we discuss possible solutions in the next section. Not only does the screen-space constraint make the most use of our arsenal of model-based techniques, but there is the additional benefit that the goal of screen-space adaptation is unambiguous and easy to understand. In addition, screen-space adaptation is a good example because it is useful for almost any time of mobile application. We believe that the techniques used to han-

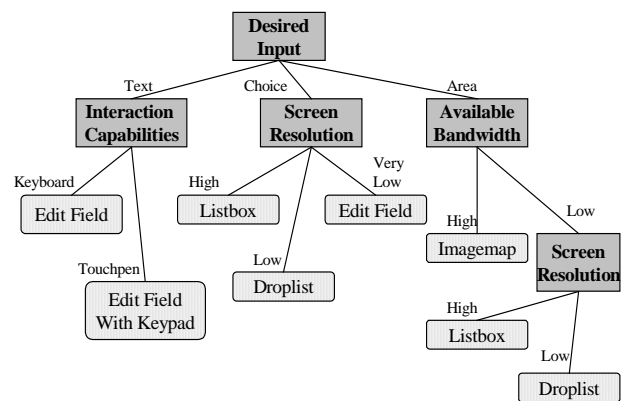


Figure 5: A simplified decision tree for adaptive interactor selection based on context.

dle this constraint can easily be applied to other cases of context-sensitive adaptation.

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.

4.2.1. Adaptive Interactor Selection. There are two possible methods for accommodating screen resolution constraints by reducing 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 with an eye to screen resolution constraints has already been investigated and shown to be feasible [13,26].

This technique can easily be applied to constraints other than screen resolution. It is similarly possible to select interactors that minimize bandwidth usage or battery power consumption. On a PDA or similar restricted-interaction device, inconvenient interactors can be exchanged for interactors that are more suitable to the capabilities of the relevant device.

As the number of constraints under consideration multiplies, we face the problem of selecting appropriate interactors while respecting several different types of constraints simultaneously. In previous work, we have shown

that decision trees can be a useful mechanism for performing interactor selection while respecting several constraints [9, 26]. Decision trees have many advantages: they require few computational resources; they are easy for humans to read, write, and comprehend; there are existing algorithms for learning decision trees from example data. Existing decision trees for interactor selection can be extended to handle the unique constraints posed by mobile computing. **Figure 5** is a simplified decision tree for interactor selection that accounts for some of the constraints posed by mobile devices. This decision tree handles a static constraint – the availability of a keyboard. When the mobile device does not offer a keyboard, a software keypad interactor is displayed, as shown in **Figure 6**. The more dynamic constraint of bandwidth availability is also handled. In low-bandwidth situations, the imagemap is not downloaded; instead, a textual choice is displayed.

Adaptive interactor selection is a powerful method for dealing with many of the constraints posed by mobile computing. However, we believe that in some cases, a more global solution is necessary. This is particularly true when we return to the example of screen resolution. 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.

4.2.2. Selecting a presentation structure. We have seen that the interactor-size parameter can provide only limited flexibility in meeting the screen-resolution constraint. 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-space afforded by a platform. To solve this problem, a set

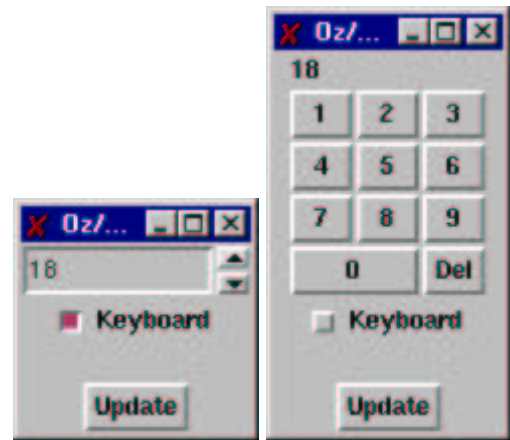


Figure 6: A textbox interactor is replaced by a keypad when no keyboard is available. In this case, the adaptation is initiated by the user, who sets the value of the “keyboard” checkbox.

of alternative presentation structures is needed. These 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. This is similar to the technique that we applied to abstract interactor objects. The main drawback to this approach is that it is rather static; if the available screen resolution changes at run-time, or even if a new device with new screen resolution is introduced, no support is provided.

We prefer a more dynamic solution. 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. Then we can build an intelligent *mediator* agent that dynamically selects the appropriate presentation model for each device (fig. 7) [1,2]. We are currently developing a language for specifying mapping rules that can be applied by a rule-based mediator.

4.2.3. Generating a Presentation Structure. Under our proposed architecture, it is still left to the human 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 constraints about which AIOs should be grouped together [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.

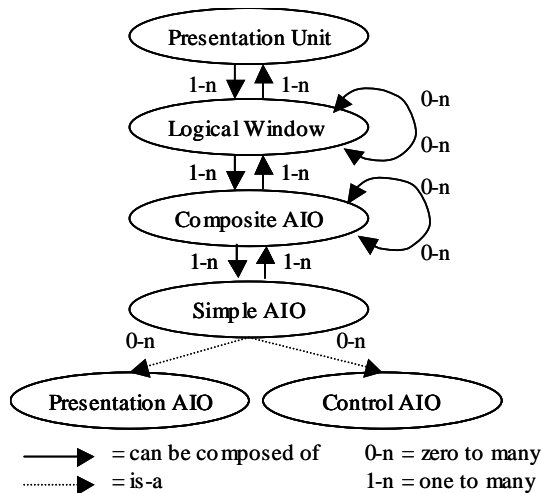


Figure 7. Hierarchy of presentation concepts.

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.

Each LW can be recursively decomposed into composite AIOs (e.g., a group box surrounding labels and edit boxes) and simple AIOs (e.g., a list box). Figure 7 graphically depicts the hierarchy that we have described. We can use this hierarchy to construct an automated design tool that generates several platform-optimized presentation models from a starting presentation model which 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. In our previous work, we have presented tools that can 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 the same 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

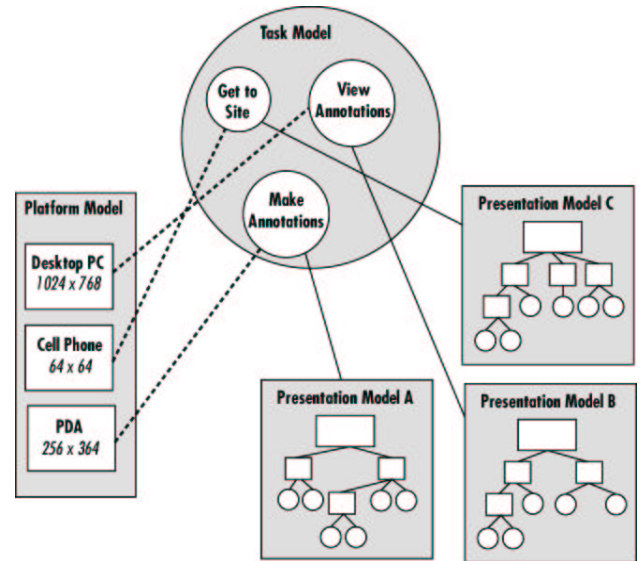


Figure 8. Platform elements are mapped onto task elements that are especially likely to be performed. In turn, the task elements are mapped onto presentation models that are optimized for the performance of a specific task.

these abstractions that has yet been documented.

4.3. Focusing on Contexts of Use

We have now addressed the issue of optimizing the UI for each specific device. Thus far, we have assumed that on each device, the user will want to accomplish the same set of tasks. However, 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 if the user were not driving, she could be using 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 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 to optimize the UI for each device. The designer creates mappings between platforms (or classes of platforms) and tasks (or sets of tasks). Additional mappings are then created between task elements and presentation layouts 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 **Figure 8**. In this figure, the task model is shown to be a shallow list 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. 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. This final presentation is illustrated in **Figure 9**. On the cellular phone, driving directions are immediately presented when any location is selected. On the other devices, an additional click is required to get the driving directions. The “bottom arrow” button of the cellular phone (**Figure 10**) 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 feel that the techniques described in the previous section can be applied to this problem as well.

5. 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 from implementing a strategy here. As the research literature grows, we hope to provide support for applying these guidelines to UI development.

The rest of the future work that is described in this section pertains to the problem of automatically generating presentation structures. 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 con-

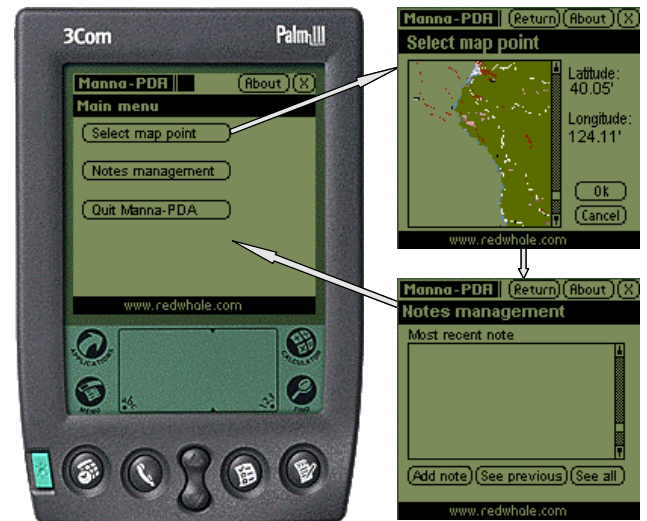


Figure 9: Final presentation for the PDA.

straint-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.

The allocation or re-allocation of LWs within a PU or AIOs within a LW as analyzed in Section 4.2.3 can be located at a rather syntactic level. A more semantic level could be achieved by basing the allocation of LWs on the related task objects and their parameters. First-class task objects should be presented before other objects. By “first-class”, we refer to those objects that are most frequently used, represent most important information, are the most critical to the performance of higher-level tasks, and those objects decided by convention among stakeholders. If we need to reallocate LWs, we can then group together the first-class objects in a highly-accessible LW, while providing a one or more additional LWs that will display the less important objects on demand. The first LW should achieve the UI property of observability (what you see is what you need) while the others should achieve the UI property of browsability (what you will see is what you ask for) [6,11].

6. Related work

User-interface modeling for traditional, non-mobile GUIs has been explored in depth: [1,13,14,16,22,26], to

name a few. 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].

The AIO, has been successfully applied for traditional GUIs at design time or at run time. In particular, [13] 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).

Some work has been done on cross-platform development and on handling platform constraints. Galaxy [10] and Open Interface render the same UI on different platforms (typically Windows, Apple Macintosh, and OSF/Motif) with their native look and feel, while SUIT [15] employs a unique UI definition that can be processed on multiple platforms. SUIT provides an additional GUI layer for each supported environment on top of which the GUI is rendered. Therefore, the GUI rendering is similar on all supported computing platforms.

However, none of these research efforts involved a truly model-based approach—although SUIT includes some presentation abstractions rendered in a platform-neutral format. This lack of abstraction, particularly in regard to task structure and platform constraints, limited the flexibility of these applications.

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] is a mediator system 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 UIs. In [26], we 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. TIMM, a system for automatic selection of interactors is described in [18]; this system outputs a set of alternative interactors, and takes into account constraints such as screen space. In addition, it includes an adaptive component that learns the designer's preferences.

7. Conclusion

Mobile computing has dramatically increased the difficulty and complexity of user-interface design because it has forced designers to accommodate a growing variety of devices and contexts of use. The techniques described in this paper can help designers in two different ways:



Figure 10: Final presentation for the cellular phone.

1. As the basis for components of a design toolkit that allows designers to build several user-interfaces out of a single design idea.
2. By leading to the development of adaptive user-interfaces that adjust at run-time to support the necessary context 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, and that the task and context be modeled at the same level as the design and presentation. Our central claim is that the use of abstract, platform-neutral, user-centered models to describe the user-interface greatly facilitates the development of consistent, usable multi-platform user-interfaces for mobile devices.

7. 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 the 7th National Conf. on Artificial Intelligence 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. <http://www.palmos.com/dev/tech/docs/palmos/UserInterface.html#924904>
- [4] K.A. Bharat, L. Cardelli, "Migratory Applications Distributed User Interfaces", *Proc. of the ACM Symposium on User Interface Software and Technology 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. Accessible at http://www.dcs.gla.ac.uk/~johnson/papers/mobile/HCIMD1.html#_Toc420818988
- [9] J. Eisenstein and A. Puerta, "Adaptation in Automated User-Interface Design", *Proc. of ACM International Conference on Intelligent User Interfaces IUI'2000* (New Orleans, 9-12 January 2000), ACM Press, New York, 2000, pp. 74-81. Accessible at <http://lieber.www.media.mit.edu/people/lieber/IUI/Eisenstein/Eisenstein.pdf>
- [10] "Galaxy Application Environment", Ambiência Information Systems, Inc., Breckenridge, 2000. Description accessible at <http://www.ambiencia.com/galaxy/galaxy.htm>
- [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. Accessible at <http://www.hcibib.org/gs.cgi?word=checked&terms=C.ASSETS.96.72>
- [14] Ch. Martin, "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. Accessible at <http://www.cs.virginia.edu/~uigroup/docs/publications/Suit.lessons.paper.ps>
- [16] A.R. Puerta, "The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development", *Proc. of the 2nd Int. W. on Computer-Aided Design of User Interfaces 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. Accessible at <http://www.isi.edu/isd/Interchi-beyond.ps>
- [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 IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'99* (Edinburgh, August 1999), IOS Press, 1999. Abstract accessible at http://iihm.imag.fr/publs/1999/INTERACT99_Plasticite.pdf
- [25] J. Vanderdonckt, F. Bodart, "Encapsulating Knowledge for Intelligent Interaction Objects Selection", *Proc. of InterCHI'93*, ACM Press, New York, 1993, pp. 424-429. Accessible at <http://www.qant.ucl.ac.be/membres/jv/publi/InterCHI93-Encaps.pdf>
- [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. Accessible at <http://belchi.qant.ucl.ac.be/publi/1999/UIDIS99-2.zip>