Representing Interaction Data for Multiple User Interface Design

Angel Puerta, Jacob Eisenstein

November 1, 2002

1 Introduction: Reusability and Adaptability

Mobile computing offers to meet the computing needs of users in a variety of new environments. But the recent proliferation of mobile devices poses new and unique challenges for user interface design and development [19, 20]. **add more to this paragraph**

- Interactive applications and their user interfaces must run on many different computing platforms, ranging from the powerful workstation to the tiny cellular phone. These differences in computing power impose 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).
- 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.

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 [21] 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.

1.1 Reusability and Adaptability

All of these new and complex challenges to the development of multiple user interfaces can be traced to two underlying issues: reusability and adaptability. *Reusability* means that some or all of the features of a user interface design for a specific device or scenario can be reused in another design. Reusability does not necessarily imply that one design can be automatically transformed into another. Reusability can also be achieved through the use of an interactive redesign toolkit, which would help designers modify a user interface for the constraints of a new context. Such a toolkit should save the designer from repetitious work as much as possible, and help ensure that the underlying design principles that motivated the original user interface are applied as closely as possible in every new context of use. The current set of commercially available user interface design toolkits provide little, if any, support for reusability.

Adaptability refers to the capacity of a user interface to change itself, with respect to the device, context of use, or user **REWORK**. An adaptive user interface might restructure its presentation structure to meet the constraints of a new device, or switch to speech-based interaction to support a visually-impaired user. Adaptive user interfaces essentially perform automatic self-redesign. While automatic redesign may have disadvantages compared with human-guided redesign [10], there are bound to be situations in which human redesign is not possible. For example, the ever-increasing proliferation and variety of mobile devices is creating a situation in which only the most popular and profitable websites can afford to offer a human-created redesign for every device that might be used. Other websites that want to offer at least some functionality to any user might find adaptation to be a useful solution. Moreover, user groups can always be subdivided into smaller, more precisely defined subgroups. To support customization on a personal level, adaptation seems like the only

feasible solution.

Research on adaptability for multiple user interfaces is proceeding on a number of fronts. Adaptive user interfaces is a mature field, encompassing a wide variety of research directions [18]. However, relatively little research has focused on applying adaptive techniques to mobile devices or universal usability, and no general technique has emerged for providing this sort of adaptation. On a commerical level, CC/PP [28] and and UAProf [24] provide standards for describing user agents (e.g. web browsers, cellphones, etc), and researchers have begun to explore the possibility of providing variable content depending on characteristics of the web browser [3, 8, 14]. However, this kind of adaptation is rarely undertaken in a principled way, with respect to the original intentions of user interface designer. Although there has as of yet been little evaluation of this type of adaptation, we will claim that a successful, general approach to adaptation for multiple user interfaces is impossible without knowledge about the rationale that guided the original design.

1.2 Representation of Interaction Data

Interaction data is a formal, declarative, platform neutral description of the user interface, including both its structure and function. This includes, but is not limited to, a description of the visual appearance of the UI, the commands available at each stage of interaction, and the characteristics of the devices that support the UI. In addition, interaction data can include a formal description of the task structure that the UI is intended support, as well the relevant domain and user groups. Most importantly, interaction data should capture the relationship between all of these different types of components. We will argue that representation of interaction data is a critical feature of any solution to the problem of achieving reusability and adaptability.

1.2.1 Reusability

One key piece of interaction data is the relationship between the task structure and the implementation of the user interface, which we will describe in detail later on. It is this mapping that best captures the rationale behind the user interface design, since it indicates the role that each concrete UI component plays in helping the user to achieve the stated goals. Although the creation of this relationship–or even the task model itself–is usually not explicit in the UI design process, the ability to do this mapping well is in many ways the essence of good user interface design.

If the mapping from task structure to UI design is consistent across devices or contexts of use, then users should be able to move between platforms without any learning curve at all. If this mapping is inconsistent, then users will struggle to relearn the application. Current practice leaves this mapping implicit, opening the door for inconsistency, especially when there are multiple user in-

terface designers. By providing allowing designers to formalize this mapping in an explicit representation, we can lay the groundwork for tool support that can ensure consistency across multiple user interfaces.

1.2.2 Adaptability

Similarly, any general approach to adaptability will likely require knowledge about the design rationale of the UI. Consider, for example, the problem of automatically adapting a web page from full screen desktop display to a small screen mobile device. Typically, this requires breaking up a single page into multiple smaller pages. Web pages may contain a variety of different types of content and controls: e.g., navigation bars, related links, links to more detailed information, etc. Without any knowledge of which controls correspond to which tasks, it would be extremely difficult to divide the page up in a way that makes sense to the user. Here again, maintaining a formal, declarative representation of the relationship between the actual design and the user's task structure can play a critical role in performing a successful adaptation.

1.3 Summary

Multiple user interface design, which stems from the desire to support a variety of mobile devices and provide universal usability, poses difficult new challenges. To help designers meet these challenges, toolkits must provide support for both reusability and adaptability. Such support is beyond the scope of the current generation of commercial user interface design tools. Comprehensive representation of interaction data is the key missing element. In the remainder of this chapter, we will describe XIML, a representation framework for interaction data. We claim that XIML is sufficiently expressive to support both reusability and adaptability, and we will provide examples to show how this might work.

2 XIML: A Universal Language for User Interfaces

In the previous section, we argued that there is a compelling need for a common representation framework with which to describe interaction data. We will now present our proposal for such an interaction framework. First, we will describe a key feature of our framework: an emphasis on interoperability and open standards that has led us to propose XIML as a new XML-based standard for the representation of interaction data. Next we will detail what we feel to be essential requirements of any such representation framework. We will then describe the specifics of our language, and show why we feel it meets those requirements. This section includes a detailed description of the syntax and

semantics of XIML, as well examples of how some common interaction patterns are be modeled.

2.1 A Standard for Interaction Data

Earlier, we described how the recent profileration of heterogenous computing platforms has created a need for software and representational support for multiple user interface design. During this same period of time, the software industry has begun to build a foundation for a new computing model that will enable a standard way for applications to interoperate and interchange data. This is a substantial shift from previous computing models where individual-application capabilities and data manipulation were the main focus of the development process.

Over the past few years, both industry and academia have contributed a number of building blocks to this new computing model. These efforts include, among others, the dissemination and adoption of a common data representation format (XML) [28], the definition of standard protocols for application interoperability (SOAP) [28], and a number of proposed standard definitions for various types of data, such as data for voice-based applications (VoiceXML) [27], and data for directory services (DSML) [11]. These and many other efforts are being channeled through standards organizations such as the World Wide Web Consortium [28] and the Organization for the Advancement of Structured Information Systems [11].

The benefits of the interoperability of software applications and the ease of data interchange among those applications are self-evident. Not only is integration of these applications significantly facilitated, but in addition, integrated software support can now be devised for many complex and multi-step workflows and business processes that previously could not be supported.

There is, however, a problem that the user interface software community faces as this new computing model emerges. A standardization effort has not yet emerged for representing and manipulating interaction data–the data that defines and relates all the relevant elements of a user interface. This failure is problematic in at least two fronts. One is that an opportunity is being lost, or delayed, to provide a mechanism to bridge the gaps that exist between the user-interface engineering tasks of design, operation, and evaluation (which are the three critical aspects of the user-interface software cycle). The second one is that without a viable solution for interaction-data representation, user-interface engineering will be relegated to the same secondary plane that it has suffered in basically every previous computing model prevalent in industry.

Admittedly, one key reason why interaction data has not been effectively captured yet is because doing so entails a high level of complexity. Interaction data deals not only with concrete elements, such as the widgets on a screen, but

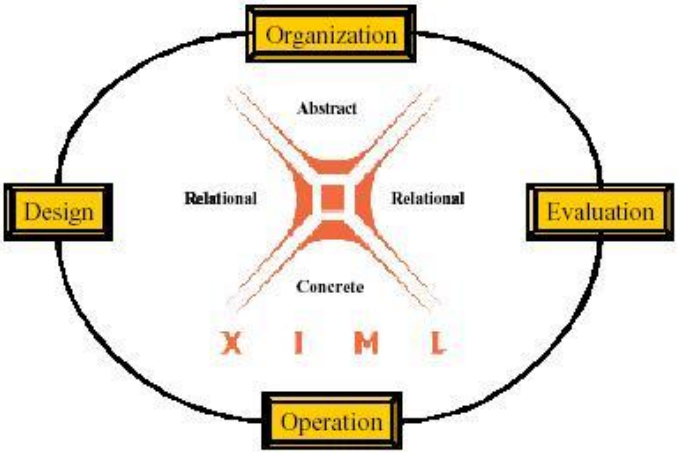

Figure 1: XIML represents abstract, concrete and relational interface data items. It also enables user-interface engineering functions of design, operation, evaluation, and organization.

also with abstract elements, such as the context in which the interaction occurs. Therefore, capturing and relating these distinct elements into a cohesive unit presents difficult technical challenges.

We propose XIML as a solution for the representation and manipulation of interaction data. We hope that XIML will offer a standard mechanism for applications and tools to interchange interaction data, and to interoperate within integrated user-interface engineering processes, from design, to operation, to evaluation. Such a standard should synchronize the efforts of the research community, freeing researchers from having to continually reinvent representation schemes, and allowing them to focus on building a new generation of interoperable user interface tools.

2.2 Requirements

Reusability and adaptability are the two main goals that have led us to pursue a representation framework for interaction data. However, if our design is based solely on achieving these goals, we run the risk of producing an ad hoc solution that does not scale beyond our test cases. In this section we propose a set of requirements for any representation framework that claims to support reusability and adaptability. We have designed XIML with these requirements in mind, with the aim of finding a general, scalable solution to the problem of

managing interaction data.

2.2.1 Central Repository of Data

We require that our language enable a comprehensive, structured storage mechanism for interaction data. These repositories of data may cover in scope one user interface or a collection of user interfaces. In this manner, purely organizational or knowledge management functions can be supported by XIML.

For example, a cell-phone manufacturer could use XIML to store and manage all the characteristics and design data relevant to the user interfaces of its entire line of products. A software program, such as a user interface development toolkit, could then access this repository to help guide the migration of user interface designs across the various cellphone platforms.

Similarly, a web-based content provider could use XIML to create a repository of presentation styles and interaction techniques that could be shared across all of its webpages. In this way, XIML can provide similar functionality to traditional stylesheet techniques, such as CSS and XSL [28]. But whereas conventional stylesheet techniques only allow the customization of presentation features such as fonts and colors, XIML allows the development of stylesheets that describe all characteristics of interaction.

2.2.2 Comprehensive Lifecycle Support

The language must enable support functionality throughout the complete lifecycle of a user interface. This includes conceptual design, concrete design, operation, and evaluation phases. This requirement is critical because it will afford an engineering framework to connect the now disjoint stages in the life of a user interface. With current technology, there is little tool support at all for the conceptual design and evaluation phases. Programs that do support these phases do not produce output that can be integrated with the other phases.

We require that XIML support conceptual design–for example, by allowing designers to model the task structure that the UI is intended to support. We also demand that the output from this conceptual design phase be relevant to the concrete design phase, which may use the task model as a starting point for UI layout. The interace specification that is produced in the concrete design phase can then be used at runtime for the management of interaction, and can also be the basis for subsequent usability engineering activities.

2.2.3 Abstract and Concrete Elements

XIML must be able to represent both the concrete aspects of the user interface, such as the choice and location of specific widgets, and also the abstract features of the design, such as the expected contexts of use and the task structure.

Since the abstract elements of the user interface are most important early in the design process, this requirement is almost a corollary of the previous one; comprehensive liefcycle support would not be possible without it. This requirement also a signifies a recognition that interaction decisions–be it in design or in operation of a user interface–are dictated in great part by items such as the task flow of a target business process or the characteristics of a specific user type.

Nonetheless, this requirement is not met by some well-known languages from the commercial sector, including UIML [1] and XUL [4]. Such languages are simpler than what we propose, and may seem more "practical." However, we will argue that without representation for the abstract features of the user interface, a language lacks the expressive power to support modularity and adaptability.

2.2.4 Relational Modeling

The language must be able to effectively model relations between the various elements captured within the scope of its representation. This is particularly important in the case of relating abstract and concrete elements of interaction data. From a certain standpoint, the entire process of user interface design can be viewed as creating a set of relations, or mappings, between various prespecified models [16]. The abstract models, such as the task and domain structure, are given by the client or customer. The concrete models, such the presentation and interaction techniques, are constrained by the device and implementation. The task of the UI designer is find the set of relations between abstract and concrete components that yields the most intuitive, usable, and efficient interaction.

The relational capabilities of the language enable the development of knowledge-based support throughout the lifecycle of a user interface [22]. Such tools include task elicitation at the conceptual design stage [23], intelligent assistance to help the designer conform to existing usability standards [6, 25], automated help generation and agent support [7, 13], and automated usability evaluation [12]. Later in this chapter, we will show that the same relational models that are exploited by all of these existing tools can also be used to support multiple user interface design across heterogenous devices.

2.2.5 Flexibility

To support new contexts of use and to advance the goal of universal accessibility, user interface designs are going to have to become more flexible. That is, rather than producing a single, fixed design, designers will have to create user interfaces that change to reflect new circumstances. For a representation framework to support this new kind of design, it must also be flexible; it must allow for design decisions that take the form of contingencies and preferences. Whereas in the past it might have been acceptable to represent only straightforward statements such as, "use widget w_1 to help the user achieve task t_1, we must now support a much broader notion of a "design decision." A design decision now may take

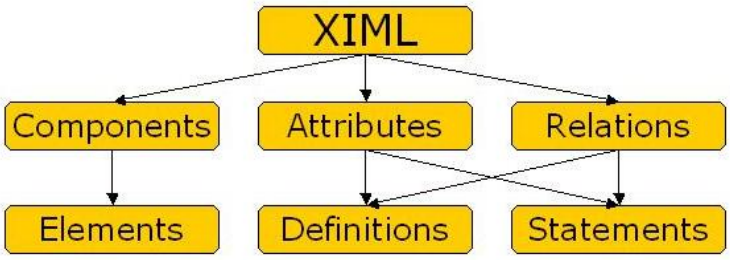

Figure 2: The basic representational structure of XIML.

the form, "use widget w_1 if pen-based input is available; otherwise use w_2", or "use dialog d_1 if a full-screen display is available; if a small-screen display is available, use d_2; if only a screen reader is available, then use d_3." The flexibility to support this type of complex design preference is at the heart of multiple user interface design.

2.2.6 Underlying Technology

Earlier in this section, we described a new computing model based on interoperability and open, standard protocols. In order to be useful within this new, industry-based computing model, XIML must adhere to at least two implementation requirements. First is the use of an underlying technology that is compatible with that computing model. This points to the use of XML–the representational centerpiece of the new computing model–as the base language for XIML. Second, the language must not impose any particular methodologies or tools on the design, operation, and evaluation of user interfaces. It must be able to coexist with existing methodologies and tools (limited, of course, by any compatibility issues external to XIML between those tools and methodologies and the chosen underlying technologies). It should nevertheless be noted that implementation issues are strictly a practical consideration for the language. They impose certain limitations as to what can be achieved in practice, but they do not detract from the theoretical principles of the language and its applicability to different underlying technologies.

2.3 The Structure of XIML

The XIML language draws mainly from two foundations. One is the study of ontologies and their representation [9], and the other one is the work on interface models [17, 21, 22]. From the former, XIML draws the representation principles it uses; from the latter it derives the types and nature of interaction data.

A discussion of the entire XIML schema, or of the specific language constructs would be beyond the scope of this paper, but is available with the XIML

documentation [29]. For the purpose of this paper, we focus within this section on describing the organization and structure of that schema. Figure 2 shows the basic structure of XIML. In the remainder of this section, we examine each of the main representational units.

2.3.1 Components

From a high level perspective, XIML can be viewed as an organized collection of interface elements that are separated into one or more major interface components. The language does not limit the number and types of components that can be defined. Nor is there a theoretical limit on the number and types of elements under each component. In a more practical sense, however, it is to be expected that an XIML specification would support a relatively small number of components with one major type of element defined per component.

In its first release version (1.0), XIML predefines five basic interface components: task, domain, user, dialog, and presentation. In addition, we will describe the platform component, which will appear in the next release of XIML. These six components correspond to what we feel are the most basic categories of interaction data. The first three can be characterized as contextual and abstract while the last three can be described as implementational and concrete. We now examine each of these components in detail.

Task The task component captures the business process and/or user tasks that the interface supports. The component defines a hierarchical decomposition of tasks and subtasks, and also defines the expected flow among those tasks and the attributes of those tasks. It should be noted that when referring to a business process that is captured by this component, we are referring to that part of the business process that requires interaction with a user. Therefore, this component is not aimed at capturing application logic. The granularity of tasks is not set by XIML so examples of valid tasks can for example include *Enter Date*, *View Map*, or *Perform Contract Analysis*.

Throughout this section, we will develop an example UI specification in XIML. The target application is MANNA, the Map Annotation Assistant. Later, we will show how the MANNA user interface can be redesigned for multiple devices and contexts of use. For now, we offer a simplified model of part of the MANNA task structure.

```
<TASK_MODEL ID='tm1'>
 <TASK_ELEMENT ID='t1' name='Make annotation'>
  <TASK_ELEMENT ID='t1.1' name='Select location'/>
  <TASK_ELEMENT ID='t1.2' name='Enter note'/>
  <TASK_ELEMENT ID='t1.3' name='Confirm Annotation'/>
 </TASK_ELEMENT>
</TASK_MODEL>
```

Domain The domain component is an organized collection of data objects and classes of objects. Objects to be included in this component are restricted to those that are viewed or manipulated by a user; internal program data structures are not represented here. Objects can be typed, using either simple types–e.g. integer, enumeration–or more complex structures.

The domain model is typically structured into a "contains" hierarchy. In this case, inheritance relations are specified using *relation statements*, as described in Section 2.3.3. However, as with all model components, the structure of the domain component can be set by the user. We feel that this is a knowledge representation detail that is beyond the scope of this paper; for more information, see the XIML specification [29].

```
<DOMAIN_MODEL ID='dm1'>
 <DOMAIN_ELEMENT ID='d1.1' name='map annotation'>
  <DOMAIN_ELEMENT ID='d1.1.1' name='location'/>
  <DOMAIN_ELEMENT ID='d1.1.2' name='note'/>
  <DOMAIN_ELEMENT ID='d1.1.3' name='entered_by'/>
  <DOMAIN_ELEMENT ID='d1.1.4' name='timestamp'/>
 </DOMAIN_ELEMENT>
</DOMAIN_MODEL>
```

Note that the type of each domain element is not indicated here. In XIML, type is represented using *attributes*, a generic formalism for describing characteristics of elements. The representation of attributes will be described in Section 2.3.2.

User The user component describes relevant characteristics of the various individuals or groups who will use the interface. User elements are organized in an inheritance hierarchy; more general user types are represented by high-level elements, while their children have more specific features. For example, the highest level user type might be *Doctor*, with a child element *Doctor Octopus*. Attribute-value pairs define the characteristics of these users. As currently defined, the user component of XIML does not attempt to capture the mental model (or cognitive states) of users but rather data and features that are relevant in the functions of design, operation and evaluation. Relevant features include user preferences, permissions, and level of expertise.

```
<USER_MODEL ID='um1'>
 <USER_ELEMENT ID='u1.1' NAME='field researcher'>
  <USER_ELEMENT ID='u1.1.1' NAME='field supervisor'/>
  <USER_ELEMENT ID='u1.1.2' NAME='field geologist'/>
 </USER_ELEMENT>
 <USER_ELEMENT ID='u1.2' NAME='analyst'>
</USER_MODEL>
```

Presentation The presentation component describes the visual appearance of the user interface. It includes information describing the hierarchy of windows and widgets, stylistic choices, and the selection and placement of these

widgets. Widgets range from canonical WIMP interactors like push buttons and sliders to more complex and non-traditional interactors like voice menus and data visualization components.

The presentation component is typically organized in one of two ways, depending on its purpose. If it is organized as a class hierarchy, then the intention is usually to represent a library of widgets. Properties are passed down from parents to children in a conventional inheritance scheme. This type of organization can be used to represent the distinction between Abstract Interactor Objects and Concrete Interactor Objects [26]. For example:

```
<PRESENTATION_MODEL ID='pm1' NAME='Widget catalog'>
 <PRESENTATION_ELEMENT ID='p1.1' NAME='button'>
  <PRESENTATION_ELEMENT ID='p1.1.1' NAME='push_button'>
   <PRESENTATION_ELEMENT ID='p1.1.1.1' NAME='HTML_push_button'/>
   <PRESENTATION_ELEMENT ID='p1.1.1.2' NAME='Swing_push_button'/>
  </PRESENTATION_ELEMENT>
  <PRESENTATION_ELEMENT ID='p1.1.2' NAME='radio_button'/>
 </PRESENTATION_ELEMENT>
</PRESENTATION_MODEL>
```

The presentation component may also be organized in a "contains" hierarchy– this organization is usually used to describe the presentation structure of a specific user interface. In this case, the children of a presentation element are contained within that element. For example, a collection of widgets might be contained within a single window, and that window might be part of a larger multi-window user interface. Presentation elements in this type of presentation component are usually linked to the relevant widgets in a "catalog" presentation component, using relation statements. They need not be linked to "leaf" nodes; this allows an interaction management system to select the appropriate interactor depending on various constraints. For example, a presentation element called "password entry field" might be mapped to a high-level presentation element, "text field," with two children, "Swing text field" and "HTML text field." If the UI is used in a web application, the HTML text field is chosen; in a desktop setting, the Swing text field would be used instead.

```
<PRESENTATION_MODEL ID='pm2' NAME='Map Annotation Assistant'>
 <PRESENTATION_ELEMENT ID='p2.1' NAME='Annotation Creation Dialog'>
  <PRESENTATION_ELEMENT ID='p2.1.1' NAME='location map'>
   <PRESENTATION_ELEMENT ID='p2.1.1.1' NAME='move north button'/>
   <PRESENTATION_ELEMENT ID='p2.1.1.2' NAME='move south button'/>
   <PRESENTATION_ELEMENT ID='p2.1.1.3' NAME='move east button'/>
   <PRESENTATION_ELEMENT ID='p2.1.1.4' NAME='move west button'/>
  </PRESENTATION_ELEMENT>
  <PRESENTATION_ELEMENT ID='p2.1.2' NAME='notation textarea'/>
  <PRESENTATION_ELEMENT ID='p2.1.3' NAME='notation complete button'/>
 </PRESENTATION_ELEMENT>
</PRESENTATION_MODEL>
```

Dialog The dialog component defines the structure of the user's interactions with interface. Dialog elements define the specific actions that are available to

the user at any given point, and also describe the relationship between those actions. One way to think about the role of the dialog component is to consider the "look and feel" of a UI; the presentation component defines the look, and the dialog component defines the feel.

If the user interface is patterned directly after the task model, then the structure of the dialog component will be nearly identical to the task component. In this case, there will be exactly one action in the dialog component for every task in the task component, and the actions will be structured in the same way. For example, if there is a set of tasks that is required to be performed sequentially, then there will be an analagous set of sequential actions in the dialog action. Whereas the task component is abstract, the dialog component is concrete; dialog elements specify exactly how each action is to be performed. Examples of such "interaction techniques" include: sending a "click" message to a push-button, giving voice input, or making a gesture.

For more complex user interfaces, the dialog component must also model "navigational" actions that are not an essential part of the intrinsic task structure, but are required by the user interface design. Most realistic applications include such navigational actions. The dialog component models various navigational techniques, such as modal or modeless dialogs, and sequential, indexed, or global navigation. This type of modeling can become complex, and is not described in detail in this paper; see [2].

```
<DIALOG_MODEL ID='im1'>
 <DIALOG_ELEMENT ID='i1.1' NAME='Make annotation'>
  <DIALOG_ELEMENT ID='i1.2' NAME='Select location'>
   <DIALOG_ELEMENT ID='i1.2.1' NAME='Select map point'/>
   <DIALOG_ELEMENT ID='i1.2.2' NAME='Specify latitude'/>
   <DIALOG_ELEMENT ID='i1.2.3' NAME='Specify longitude'/>
  </DIALOG_ELEMENT>
  <DIALOG_ELEMENT ID='i1.3' NAME='Enter note'/>
  <DIALOG_ELEMENT ID='i1.4' NAME='Confirm annotation'/>
 </DIALOG_ELEMENT>
</DIALOG_MODEL>
```

Note that the dialog component is very similar to the task component, but that it contains additional details. From the perspective of the task component, it is irrelevant if the location is selected by choosing a point on the map or by manually specifying a latitude and longitude. This choice of actions is not a part of the abstract task structure, but it is of course highly relevant to the concrete design of the user interface that there are multiple sets of actions that can accomplish a given task.

The dialog elements shown above are related with each other–and with elements from the other interface components–in a number of ways. For example, we will require that *Choose location* and *Enter note* are executed before *Confirm annotation*. We will also specify that if *Select map point* alone is executed, then *Select location* is satisfied; alternatively, the combination of *Specify latitude* and

Specify longitude can satisfy the *Select location*. In addition, each dialog element corresponds to a presentation element, which provides the interaction technique for that specific action. Finally, the dialog component is of course related to the task structure. All of these relations will be formalized later, using relation statements.

Platform The platform component describes the various computer systems that may run a UI [15]. This component includes information regarding the constraints placed on the UI by the platform. It contains an element for each supported platform, and each element includes attributes describing features and constraints. These elements are organized hierarchically, so that children inherit features from their parents. For example, an *Ericsson T68 Cellphone* would inherit the properties of the abstract device *Cellphone*, but it would override some of these and also provide additional details. Device characteristics are specified by attribute-value pairs, which can be used to describe screen resolution, number of colors, available input modalities, etc. We have not yet developed a canonical list of the properties that might be relevant to the device model. The UAProf [24] standard, which supports the specification user agent profiles, provides some guidance in this respect. We are currently investigating ways in which XIML, our proposed standard for interaction data, can be integrated with this standard for user agent profiling.

```
<PLATFORM_MODEL ID='am1'>
 <PLATFORM_ELEMENT ID='a1.1' NAME='Desktop Computer'>
  <PLATFORM_ELEMENT ID='a1.1.2' NAME='Linux Box'/>
 </PLATFORM_ELEMENT>
 <PLATFORM_ELEMENT ID='a1.2' NAME='Cellphone'>
  <PLATFORM_ELEMENT ID='a1.2.1' NAME='WAP-Enabled Cellphone'>
   <PLATFORM_ELEMENT ID='a1.2.1.1' NAME='Nokia 6210'/>
  </PLATFORM_ELEMENT>
 </PLATFORM_ELEMENT>
</PLATFORM_MODEL>
```

Other Components The components predefined in the first version of XIML were selected by studying a large variety of previous efforts in creating interface models [17, 22]. There are other components that have been identified by researchers in the past as being potentially useful, including representations of the application logic, or the various possible contexts of use. However, we have found that the chosen set of the components are sufficient to model a wide variety interaction situations. XIML is easily extensible so that other components can be added in the future, either as customizations by individual users, or as part of the evolving language standard.

2.3.2 Attributes

In XIML, *attributes* are features or properties of elements that can be assigned a *value*. The value of an attribute can be one of a basic set of datatypes, or

it can be an instance of another existing element. Multiple values are allowed, as well as enumerations and ranges. The basic mechanism in XIML to define the properties of an element is to create a number of attribute-value pairs for that element. XIML supports attribute *definitions* that specify the canonical form of an attribute, including its type, and the elements to which it may apply. Attribute *statements* specify actual instances of attributes.

Screen Resolution of a Device In this example, we use attributes to represent the screen resolution of two devices in a Platform Model. The attribute definition is included under the `<definitions>` tag, which also may include relation definitions. Definitions usually occur inside a model component, but may also occur inside an element. The attribute statement is included under the `<features>` tag, which also may include relation statements. Attribute statements may occur inside model components or elements. Note that the features of the *WAP-Enabled Cellphone* are inherited by the *Nokia 6210*.

```
<PLATFORM_MODEL ID='am1'>
 <DEFINITIONS>
  <ATTRIBUTE_DEFINITION NAME='horizontal resolution'>
   <TYPE>integer</TYPE>
  </ATTRIBUTE_DEFINITION>
  <ATTRIBUTE_DEFINITION NAME='vertical resolution'>
   <TYPE>integer</TYPE>
  </ATTRIBUTE_DEFINITION>
 </DEFINITIONS>
 <PLATFORM_ELEMENT ID='a1.1' NAME='Desktop Computer'>
  <PLATFORM_ELEMENT ID='a1.1.2' NAME='Linux Box'>
   <FEATURES>
    <ATTRIBUTE_STATEMENT DEFINITION='horizontal resolution'>1200</ATTRIBUTE_STATEMENT>
    <ATTRIBUTE_STATEMENT DEFINITION='vertical resolution'>1024</ATTRIBUTE_STATEMENT>
   </FEATURES>
  </PLATFORM_ELEMENT>
 </PLATFORM_ELEMENT>
 <PLATFORM_ELEMENT ID='a1.2' NAME='Cellphone'>
  <PLATFORM_ELEMENT ID='a1.2.1' NAME='WAP-Enabled Cellphone'>
   <FEATURES>
    <ATTRIBUTE_STATEMENT DEFINITION='horizontal resolution'>200</ATTRIBUTE_STATEMENT>
    <ATTRIBUTE_STATEMENT DEFINITION='vertical resolution'>320</ATTRIBUTE_STATEMENT>
   </FEATURES>
   <PLATFORM_ELEMENT ID='a1.2.1.1' NAME='Nokia 6210'/>
  </PLATFORM_ELEMENT>
 </PLATFORM_ELEMENT>
</PLATFORM_MODEL>
```

2.3.3 Simple Relations

The interaction data captured by the various XIML components and their attributes constitutes a body of explicit knowledge about a user interface that can support organization and knowledge-management functions for user interfaces. There is, however, a more extensive body of knowledge that is made up of the relations among the various elements in the XIML specification. A *relation* in

XIML is a definition or a statement that links any two or more XIML elements, either within one component or across components. For example, "Task $T_{1.3}$ is instantiated through Dialog Element $I_{1.4}$, using the Interaction Technique *onClick* for Presentation Element $P_{2.1.3}$, which uses Widget $P_{1.1.1.2}$ on Device $A_{1.1}$."

By capturing relations in an explicit manner, XIML creates a body of knowledge that can support design, operation, and evaluation functions for user interfaces. In particular, the explicit nature of the relations enables knowledge-based support for those interaction functions. In a sense, the set of relations in an XIML specification captures the *design knowledge* for a user interface. The runtime manipulation of those relations constitutes the *operation* of the user interface. A more in-depth study of the nature of relations in a declarative interface model can be seen in [16].

Linking Commands to Widgets This example shows how to link commands, as represented by dialog elements, with widgets, as represented by presentation elements. The `<ALLOWED_CLASSES>` tag indicates which types of elements are allowed to participate in this relation; in this case, you can any child of the dialog model component i_1 on the left side, and any child of the presentation model component p_2 on the right side. Note that the presence of an attribute definition *inside* the relation definition; this indicates that the relation itself can be paraterized. In this case, the parameter indicates the specific interaction technique that is required to perform the command indicated by the dialog element on the left side of the relation. The relation definition is applied to the dialog element "Select map point", stating that to perform this command, the user must double click on the map.

```
<DIALOG_MODEL ID='im1'>
 <DEFINITIONS>
  <RELATION_DEFINITION NAME='is_performed_using'>
   <ALLOWED_CLASSES>
    <CLASS REFERENCE='im1' INHERITED='true' SLOT='left'/>
    <CLASS REFERENCE='pm2' INHERITED='true' SLOT='right'/>
   </ALLOWED_CLASSES>
   <ATTRIBUTE_DEFINITION NAME='interaction_technique'>
    <TYPE='enumeration'>
    <DEFAULT>onClick</DEFAULT>
    <ALLOWED_VALUES>
     <VALUE>onClick</VALUE>
     <VALUE>onScroll</VALUE>
     <VALUE>onChange</VALUE>
     <VALUE>onDoubleClick</VALUE>
    </ALLOWED_VALUES>
   </ATTRIBUTE_DEFINITION>
  </RELATION_DEFINITION>
 </DEFINITIONS>
 <DIALOG_ELEMENT ID='i1.1' NAME='Make annotation'>
  <DIALOG_ELEMENT ID='i1.2' NAME='Select location'>
```

	Conditions	Targets	Criteria	Objects
Binding	1	1	None	None
Simple Preference	Any Number	1	None	None
Ordered Preference	Any Number	Any Number	None	None
Abstract Preference	Any Number	Any Number	Logical and preferential	None
Design Guideline	Any Number	Any Number	Logical only	Any Number

Table 1: A Spectrum of Preference Relations

```
  <DIALOG_ELEMENT ID='i1.2.1' NAME='Select map point'>
   <FEATURES>
    <RELATION_STATEMENT DEFINITION='is_performed_by' REFERENCE='2.1.1'>
     <ATTRIBUTE_STATEMENT DEFINITION='interaction_technique'>
      onDoubleClick
     </ATTRIBUTE_STATEMENT>
    </RELATION_STATEMENT>
   </FEATURES>
   <DIALOG_ELEMENT ID='i1.2.2' NAME='Specify latitude'/>
   <DIALOG_ELEMENT ID='i1.2.3' NAME='Specify longitude'/>
  </DIALOG_ELEMENT>
  <DIALOG_ELEMENT ID='i1.3' NAME='Enter note'/>
  <DIALOG_ELEMENT ID='i1.4' NAME='Confirm annotation'/>
 </DIALOG_ELEMENT>
</DIALOG_MODEL>
```

2.3.4 Complex Relations

The relations shown in the examples above are all one-to-one; this is the only type of relation supported in XIML 1.0. We will now discuss a new framework for many-to-many relations. However, since modeling many-to-many relations in the general case would add considerable complexity to the representation language, we have narrowed our focus to the specific phenomenon of *preference relations*.

Preference relations elude description by one-to-one mappings, but they are particularly important for multiple user interface design. Preference relations allow for a more flexible and adaptable specification of the user-interface; rather than specifying exactly how the UI must perform, the designer can specify what would be preferred, and why. This type of modeling is particularly well suited for user interfaces that adapt across heterogenous devices, or in relation to specific users.

Concrete Preferences Any preference relation includes a set of *conditions*, under which the preference becomes valid. For example, if User U_1 prefers for Presentation Element P_1 (say, a listbox) to represent Domain Object D_1, then U_1 and D_1 are the conditions. P_1 is the *target* of the preference relation–the thing that is preferred. A preference relation that involves exactly one condition

and one target will be referred to as a *binding*; bindings are identical to the one-to-one relations supported by the current release version of XIML. A preference relation that involves a single target and more than one condition will be referred to as a *simple preference*. The example at the beginning of this paragraph is a simple preference. A designer might also want to enumerate a set of interactors, in order of preference. In this case, there are multiple targets. Preferences that involve multiple conditions and multiple targets are referred to as *ordered preferences*.

Abstract Preferences Bindings, simple preferences, and ordered preferences are all concrete preferences, because the targets are specified directly. Rather than specifying the specific element that is preferred, a designer may wish to instead specify the characteristics of the element that is preferred. For example, a designer may prefer whatever presentation element consumes the least screen space, or whatever dialog structure requires the least amount navigation. This kind of preference is referred to as an *abstract* preference.

Abstract preferences have an additional feature: a set of *criteria*. The criteria determine which target is selected; in the example above, the amount of screen space is a criterion. For an abstract preference, there are multiple targets indicating all of the possible selections, and the criteria determine which target is actually chosen.

There are two types of criteria: *preferential* and *logical*. Preferential criteria specify those characteristics that cause one target to be preferred. For example, a preferential criterion might say to choose the dialog structure requiring the least navigation. If there is more than one preferential criteria, then each criterion must have a *priority* to indicate how important it is relative to the others.

Logical criteria limit the allowable targets. A logical criterion might say not to choose any dialog structure with more than three levels of navigation. Logical criteria need no priority; if the value of any feature violates a logical criterion, then the respective target is ruled out.

Design Guidelines Thus far, it has been assumed that the criteria apply only to the targets. That is, distinctions can be made between the targets only on the basis of characteristics of the targets themselves. It is possible to say, for example, "U_4 prefers the least complex dialog structure," but it is not possible to say, "If U4's experience level is less than *intermediate*, then use the least complex dialog structure." To do this, we need to separate the list of targets from the criteria.

Preference relations where the criteria do not necessarily apply to the targets are called *design guidelines*. Design guidelines are the most abstract and

complex preference relations that we will attempt to model. For this class of preference relations, an additional feature must be considered: a list of *objects*. The criteria refer to features of each object. In design guidelines, all criteria must be logical. If the value of every criteria is true for every object, then a mapping is created between the conditions and each of the targets.

Consider the following example: "If the number of colors supported by Platform A_4 is less than intermediate, then use Presentation Element P_3 as the interaction technique for the Dialog Element I_1." In this case, A_4 is the object. The criterion is the "number of colors" attribute. The condition is I_1, and the target is P_3. If the experience level meets the criteria of being less than intermediate, then the target P_3 is chosen and the mapping is made.

The target of a design rule could also be another preference relation. By creating a series of *design guidelines* that target each other, we can implement a decision tree to represent complex abstract preferences. A set of preferences at this level of abstraction might be applicable to *any* user interface; it can be thought of as a high-level, general purpose "stylesheet" of interaction properties. More details on the modeling and use of design guidelines can be found in [5].

Examples of Preference Relations

3 XIML in Action

3.1 MANNA: The Map Annotation Assistant

In this section we describe MANNA, a hypothetical software application that reveals many of the challenges posed by userinterface 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 walkthrough.

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 (Figure 3). 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

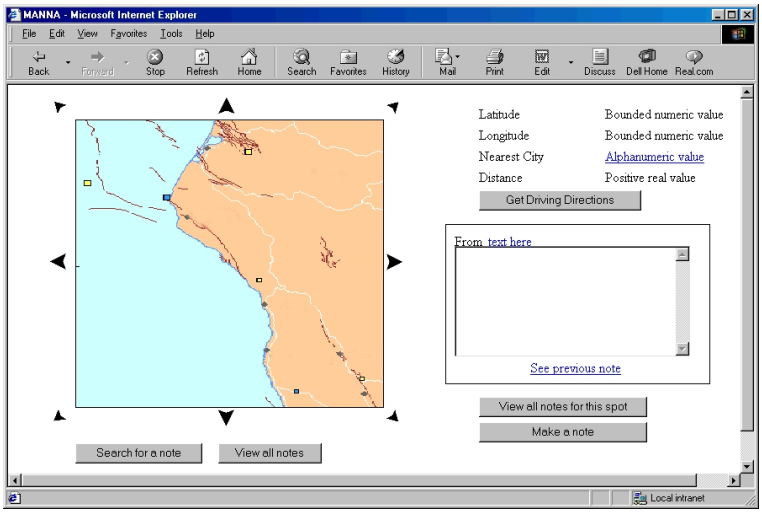

Figure 3: An HTML user interface to MANNA. This example was automatically generated from XIML, using a prototypical XIML to HTML converter.

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 (see Figure 4), 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 (see Figure 5). 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

Figure 4: The cellphone user interface to MANNA

Figure 5: A Palm Pilot user interface to MANNA

top, where everyone in the audience can see it.

3.2 Reusability and Adaptability in MANNA

3.3 Handling platform constraints with XIML

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.

3.3.1 Display Resolution

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 holistic solution.

The screen resolution of a device 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. Another way to reduce 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 [6, **?**, **?**].

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 chapter, 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 WML-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. **TALK ABOUT HOW THIS IS REUSABILITY**

However, in this case a more dynamic solution is possible **ALSO REUSABILITY** 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 [**?**]. 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 (see Figure **??**.

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 [**?**, **?**, **?**]. For this purpose, we need to consider additional elements in our presentation model besides AIOs and CIOs. Two new abstractions need to be defined **These can be handled within XIML**:

Logical Window (LW) A logical window 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.

Presentation Unit (PU) A presentation unit 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 (see Figure **??**) that serves to expand the 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.

Remodeling 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 [**?**]: 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 [6] 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.

Remodeling 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.

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 [**?**].

3.3.2 Other constraints

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 [6, **?**, 26]. 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. **TALK ABOUT THE LANGUAGE**

3.4 Optimizing for Contexts of Use

3.5 Other Applications of XIML

3.5.1 Personalization

3.5.2 Reverse Engineering

3.5.3 Evaluation

3.5.4 Agent Support

References

[1] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster. UIML: An appliance-independent XML user interface language. *Computer Networks*, 31:1695–1708, 1999.

[2] L. Bouillon, J. Vanderdonckt, and J. Eisenstein. Model-based approaches to reengineering web pages. In *Proceedings of 2002 Workshop on Task Models and Diagrams for User Interface Design (TAMODIA 2002)*, 2002.

[3] M. H. Butler. Implementing content negotiation using CC/PP and WAP UAProf. Technical Report HPL-2001-190, Hewlett Packard Laboratories, 2001.

[4] T. Cheng. Xul - creating localizable xml gui. In *Fifteenth Unicode Conference*, 1999.

[5] J. Eisenstein. Modeling preference for abstract user interfaces. In *First International Conference on Universal Access in Human-Computer Interaction, in Proceedings of HCI International*. Lawrence Erlbaum Associates, 2001.

[6] J. Eisenstein and A. R. Puerta. Adaptation in automated user-interface design. In *Proceedings of the 5th international conference on Intelligent User Interfaces*, pages 74–81. ACM Press, 2000.

[7] J. Eisenstein and C. Rich. Agents and guis from task models. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 47–54. ACM Press, 2002.

[8] V. Korolev and A. Joshi. An end-end approach to wireless web access. In *International Workshop on Wireless Networks and Mobile Computing*. IEEE Press, 2001.

[9] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, pages 36–56, Winter 1991.

[10] D. A. Norman. How might people interact with agents. *Communications of the ACM*, 37(7):68–71, 1994.

[11] Organization for the advancement of structured information systems. http://www.oasis-open.org.

[12] L. Paganelli and F. Patern. Intelligent analysis of user interactions with web applications. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 111–118. ACM Press, 2002.

[13] S. Pangoli and F. Paterno. Automatic generation of task-oriented help. In *ACM Symposium on User Interface Software and Technology*, pages 181–187, 1995.

[14] T. Phan, G. Zorpas, and R. Bagrodia. An extensible and scalable content adaptation pipeline architecture to support heterogeneous clients. In *22nd International Conference on Distributed Computing Systems*, page 507. IEEE Press, 2002.

[15] S. Prasad. Models for mobile computing agents. *ACM Computing Surveys*, 28(53), december 1996.

[16] A. Puerta and J. Eisenstein. Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems*, 12:443–442, 1999.

[17] A. R. Puerta. A model-based interface development environment. *IEEE Software*, 14(4):41–47, July/August 1997.

[18] A. R. Puerta, E. Cheng, T. Ou, and J. Min. MOBILE: User-centered interface building. In *CHI: Human Factors in Computing Systems*, pages 426–433. ACM Press, May 1999.

[19] S. Rogers and W. Iba, editors. *2000 AAAI Spring Symposium on Adaptive User Interfaces*. AAAI Press, Menlo Park, California, 2000.

[20] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 1–7. ACM Press, May 1996.

[21] P. Szekely. Retrospective and challenges for model-based interface development. In F. Bodart and J. Vanderdonckt, editors, *Design, Specification and Verification of Interactive Systems '96*, pages 1–27, Wien, 1996. Springer-Verlag.

[22] P. Szekely, P. Luo, and R. Neches. Beyong interface buildings: Model-based interface tools. In *InterCHI'93*, pages 383–390. ACM Press, 1993.

[23] P. Szekely, P. N. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools: the mastermind approach. In L. J. Bass and C. Unger, editors, *Engineering for Human-Computer Interaction*, pages 120–150, London, 1995. Chapman and Hall.

[24] R. Tam, D. Maulsby, and A. Puerta. U-TEL: A tool for eliciting user task models from domain experts. In *Proceedings of the 3rd international conference on Intelligent User Interfaces*, pages 77–80. ACM Press, January 1998.

[25] User agent profile specification. Technical Report WAP-248-UAPROF-20010530-p, WAP Forum Wireless Application Group, 2001.

[26] J. Vanderdonckt. Advice-giving systems for selecting interaction objects. In *User Interfaces to Data Intensive Systems*, pages 152–157, 1999.

[27] J. Vanderdonckt and F. Bodart. Encapsulating knowledge for intelligent interaction object selection. In *Proceedings of InterCHI'93*, pages 424–429. ACM Press, 1993.

[28] Voice extensible markup language (VoiceXML) version 1.0. Technical report, The VoiceXML Forum, 2000.

[29] The world wide web consortium. http://www.w3c.org.

[30] The ximl consortium. http://www.ximl.org.