Developing a Multiple User Interface Representation Framework for Industry

Angel Puerta and Jacob Eisenstein
RedWhale Software

1. Introduction

As many chapters of this book testify, developing an efficient and intelligent method for designing and running multiple user interfaces is an important research problem. The challenges are many: automatic adaptation of display to multiple display devices, consistency among interfaces, awareness of context for user tasks, and adaptation to individual users are just some of the research problems to be solved. In the past few years, significant progress had been made in all of these areas and this book reports on many of those achievements.

There is, however, a challenge of a different kind for multiple user interfaces (MUIs). This challenge is that of developing a technology for multiple user interfaces that is acceptable and useful in the software industry. A technology that not only brings efficiency, consistency, and intelligence to the process of building MUIs, but that does so also within an acceptable software engineering framework. This challenge is no doubt compounded by the fact that throughout the relatively short history of the software industry, the user interface and its engineering have been its poor cousins. Whereas significant engineering advances have been made in databases, applications, algorithms, operating systems, and networking, comparable progress in user interfaces is notable for its absence.

The road to build a solution for MUIs in industry is long. There can be many possible initial paths and in technology development sometimes choosing the wrong one dooms an entire effort. We claim that the essential aspect that such as solution must have is a common representation framework for user interfaces; common from a platform point of view and also from a domain point of view. In this Chapter, we report on our process and initial results on our effort to develop an advanced representation framework for MUIs that can be used in the software industry. The eXtensible Interface Markup Language (XIML) is a universal representation for user interfaces that can support multiple user interfaces at design time and at runtime [27]. This Chapter describes how XIML was conceptualized and developed, and how it was tested for feasibility.

1.1 Special Challenges for MUIs Solutions for Industry

Developing a technological framework for MUIs useful to industry imposes a number of special considerations. These requisites, named below, create tradeoffs between purely research goals and practical issues.

- **Common representation.** It is crucial for industry that any key technological solution for MUIs be based on a robust representation mechanism. The representation must be widespread enough to ensure portability. A common representation ensures a computational framework for the technology, which is essential for the development of supporting tools and environments, as well as for interoperability of user interfaces among applications.

- **Requirements engineering**. Definition of the representation must not be attempted without a clear understanding of industry requirements for the technology. In short, the types of applications and features that the representation enables must be in sync with the needs of industry. This may mean that the intended support of the representation may go beyond MUIs if the requirements dictate it.

- **Software engineering support.** Any proposed MUI technological solution for industry must define a methodology that is compatible with acceptable software engineering processes. If that is not the case, even a successful technology will find no acceptance among industry groups.

- **Appropriate foundation technologies**. The software industry is highly reluctant to incorporate any technology that is not based on at least one widely implemented foundation technologies. This is the reason why a language like XML is considered an excellent target candidate for MUI representation mechanisms.

- **Feasibility and pilot studies.** MUI technologies for industry must undergo substantial feasibility studies and pilot programs. These naturally go beyond strictly research studies and into realistic application domains.

All of these requirements create a long development cycle. It can be expected that any successful effort towards MUI technology in industry will demand a multi-year process.

1.2 Foundation Technologies

As we mentioned previously, we state that developing a representation framework for MUIs is the first step in developing a successful MUI technology for industry. To that effect, we have chosen two foundation technologies to build such a framework: model-based interface development and XML. These two technologies combine to effectively allow us to satisfy the industry requirements enumerated in the previous section.

Model-based interface development [17] provides an excellent foundation for the creation of declarative models that capture all relevant elements of a user interface. As such, it provides: (1) organization and structure to the definition of a user interface, (2) an engineering methodology for user interface design, and (3) a software engineering approach to user interface development. These three items take

on special importance within our effort as current user interface technologies in industry have considerable shortcomings in all of these areas.

XML has gain wide acceptance within industry in the last few years. It offers a very portable representation mechanism that effectively separates data from content. It is also the preferred technology for implementing interoperability among disparate applications. Many advanced industry efforts, such as Web Services, are using XML as a foundation. In addition, XML representations enjoy the support of various independent organizations that guide the process of their definition and standardization.

1.3 Summary of Chapter

The rest of this Chapter is divided into two main sections. In the first section, we describe the process that we applied to create the XIML representation framework. We examine the structure of the language and discuss its potential uses for both basic and advanced user interface functionality. We also detail a number of feasibility exercises that we conducted in order to evaluate XIML. In the second section, we present a pilot study in which we build an MUI platform for a realistic domain using XIML. We conclude the Chapter with an examination of related work, a proposed plan for future work, and a set of conclusions about the XIML framework.

2. The XIML Representation Framework

An industry project, especially one dedicated to the development of new infrastructure technologies, must be subdivided into a series of phases. Each phase must have an *exit criteria*, meaning a set of findings and results that justify moving the project into the next phase. These criteria may include many aspects such as strategic, technological, and financial ones. For the purpose of this Chapter, we will focus only on the technological aspects of the project.

With a general goal of creating a representation framework for multiple user interfaces, the logical initial phase of that project is that of a feasibility assessment. In short, we would need to create an initial representation and evaluate whether it can potentially fulfill the requirements that we set at the beginning of the project. This section reports on the feasibility assessment for XIML. The assessment included the following steps:

1. Industry Computing-Model Evaluation. This is a study of what computing models are prevalent in industry now and in the near future. Our representation must target one of these models to improve its chances of realizing its potential.
2. Requirements Elicitation. An understanding of what are the functions and features that the framework must enable, and what general objectives it must meet in order to be successful.
3. Representation Development. A language development effort based on the target requirements and computing model.
4. Validation Exercises. A series of manual and/or automated test exercises that allows us to determine the feasibility of the technology.

2.1 Target Computing Model

The software industry is making a substantial effort to lay the 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 where the main focus of the development process. The model is for now aimed at web-based applications but it is nevertheless extensible to future integration with workstation environments.

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), the definition of standard protocols for application interoperability (SOAP), and a number of proposed standard definitions for various types of data, such as data for voice-based applications (VoiceXML), and data for directory services (DSML) [13,25]. These and many other efforts are being channeled through standards organizations such as the World Wide Web Consortium [26] and the Organization for the Advancement of Structured Information Systems [13]. For now, one of the most important examples of this new computing model is the area of *web services*, a platform that enables the building of applications by integrating mostly black-box functional units from multiple providers. All major software companies support the web services platform.

The benefits of the interoperability of software applications and the ease of data interchange among those applications are self-evident. Not only integration of these applications is facilitated in a significant manner, but 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.

We feel therefore that our effort in building a representation framework is best targeted at this new computing model. By targeting this model, we take advantage of an existing, viable industry model plus we are spared the difficulty of retrofitting a new technology for user interfaces into the admittedly limited older computing models.

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 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. In turn, solving the abstract-concrete dichotomy becomes one of the key requirements that our representation framework must satisfy.

2.2 XIML Requirements

In order to effectively define a representation mechanism for interaction data, it is necessary to clearly establish the requirements of such a representation in terms of expressiveness, scope, and underlying support technologies. Figure 1 graphically summarizes the major types of requirements that we have found essential for XIML. In this Section, we discuss each of those types in detail.

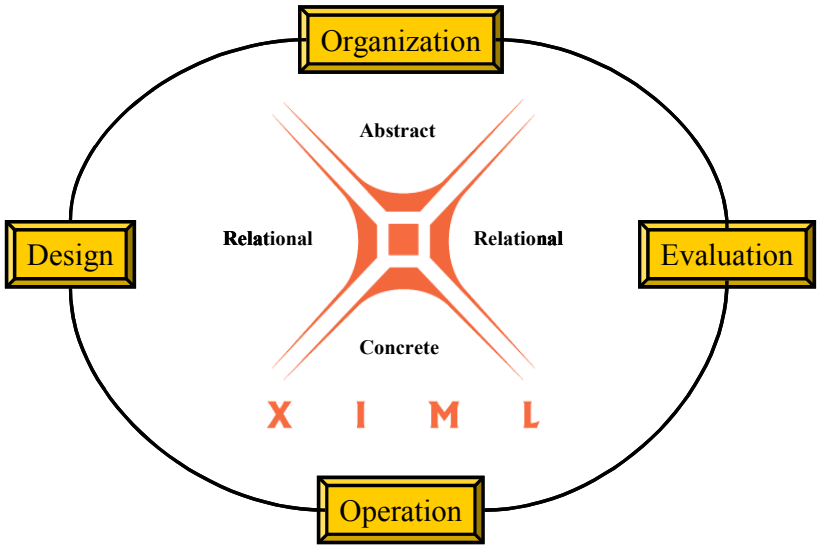

Figure 1. XIML represents abstract, concrete and relational interface data items. It also enables user-interface engineering functions of design, operation, evaluation, and organization.

- **Central repository of data**. The language must 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 for its entire line of products.

- **Comprehensive lifecycle support**. The language must enable support functionality throughout the complete lifecycle of a user interface. This includes 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. For example, an interface-design tool could output an XIML interface specification that can then be used at runtime for the management of interaction, and that can also be the basis for usability engineering activities.

- **Abstract and concrete elements**. XIML must be able to represent the abstract aspects of a user interface, such as the context in which interaction takes place, and the concrete aspects, such as the specific widgets that are to be displayed on a screen. This requirement is almost a corollary of the previous one as comprehensive lifecycle support would not be possible without it. It is also a recognition that interaction decisions—be it in design or 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.

- **Relational support**. The language must be able to effective relate 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. The relational capabilities of the language are what enable the development of knowledge-based support throughout the lifecycle of a user interface [16,20]. For example, model-based interface development tools, interface agents, and intelligent ergonomic critics are some of the technologies that can take advantage of these relational capabilities within their reasoning processes.

- **Underlying technology**. In order to be useful within an industry-based new 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. In this case, 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 be nevertheless 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 Structure and Organization of XIML

The XIML language draws mainly from two foundations. One is the study of ontologies and their representations [12], and the other one is the work on interface models [17,19,20]. 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 Chapter, but is available with the XIML documentation

[27]. In addition, Section 3 includes selected XIML language samples created for our pilot application. For the purpose of this Chapter, we focus within this section on describing the organization and structure of the XIML schema. Figure 2 shows the basic structure of XIML. Following, we examine each of its main representational units.

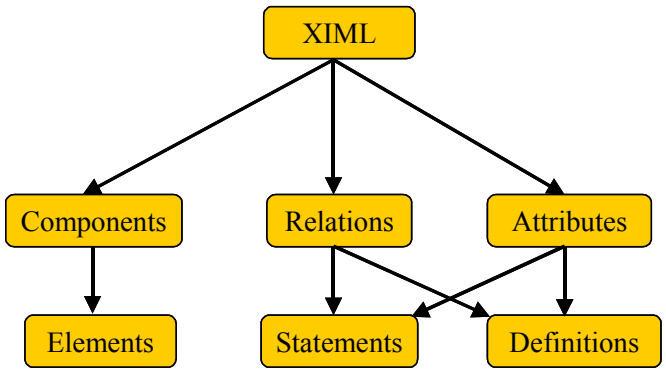

Figure 2. The basic representational structure of the XIML language.

2.3.1Components

In its most basic sense, XIML is an organized collection of interface elements that are categorized into one or more major interface components. The language does not limit the number and types of components that can be defined. Neither there is 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 version (1.0), XIML predefines five basic interface components, namely *task, domain, user, dialog, and presentation*. The first three of these can be characterized as contextual and abstract while the last two can be described as implementational and concrete. We now examine each of these five components.

- **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 that 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”.

- **Domain.** The domain component is an organized collection of data objects and classes of objects that is structured into a hierarchy. This hierarchy is

similar in nature to that of an ontology [12] but at a very basic level. Objects are defined via attribute-value pairings. Objects to be included in this component are restricted to those that are viewed or manipulated by a user and can be either simple or complex types. For example, "Date", "Map", and "Contract" can all be domain objects.

- **User.** The user component defines a hierarchy—a tree—of users. A user in the hierarchy can represent a user group or an individual user. Therefore, an element of this component can be a "Doctor" or can be "Doctor John Smith". Attribute-value pairs define the characteristics of these users. As defined today, 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.

- **Presentation.** The presentation component defines a hierarchy of interaction elements that comprise the concrete objects that communicate with users in an interface. Examples of these are a window, a push button, a slider, or a complex widget such as an ActiveX control to visualize stock data. It is generally intended that the granularity of the elements in the presentation component will be relatively high so that the logic and operation of an interaction element are separated from its definition. In this manner, the rendering of a specific interaction element can be left entirely to the corresponding target display system. We will expand on the practical impact of this separation below when we discuss the issue of cross-platform interface development.

- **Dialog.** The dialog component defines a structured collection of elements that determine the interaction actions that are available to the users of an interface. For example, a "Click", a "Voice response", and a "Gesture" are all types of interaction actions. The dialog component also specifies the flow among the interaction actions that constitute the allowable navigation of the user interface. This component is similar in nature to the Task component but it operates at the concrete levels as opposed to the abstract level of the Task component.

The components predefined in the first version of XIML were selected by studying a large variety of previous efforts in creating interface models [17,20]. There are other components that have been identified by researchers in the past as being potentially useful, such as a workstation component (for defining the characteristics of available target displays), or an application component (for defining the links to application logic). We have found that in most practical situations, we have been able to subsume all necessary definitions for a given interface into the existing components. XIML is in any event extensible so that other components can be added in the future once their presence is justified.

2.3.2 Relations

The interaction data elements captured by the various XIML components constitute 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 an 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, "Data type A is displayed with Presentation Element B or Presentation Element C" (relation in italics) is a link between a domain-component element and a presentation-component element.

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 capture the design knowledge about 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].

XIML supports relation definitions that specify the canonical form of a relation, and relation statements that specify actual instances of relations. It can be expected that there are a number of relations that can be of interest in the design and management of user interfaces. Therefore, XIML includes a number of predefined relations and their semantics. However, it also allows users of the language to define custom relations with semantics residing in the specific applications that utilize XIML.

2.3.3 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 data types 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. In addition, relations among elements can be expressed at the attribute level or at the element level. As in the case of relations, XIML supports definitions and statements for attributes, and also predefines some typical attributes of interest for user interfaces.

2.4 Validation Exercises

As mentioned in our introduction, it is necessary to conduct a number of validation exercises that enables us to assess the *feasibility* of XIML as a potential representation for industry. Whereas in the purely scientific world having an initial assessment of feasibility may not be in order, for our purposes the project could not move into the next phase unless we can demonstrate such feasibility. We sought to answer two questions with these validation exercises:

1. Is the language expressive and flexible enough to represent simple user interfaces?
2. Can we build small demo applications based on XIML that enable functionality that is of general interest for industry, but that may not be easily implemented with current capabilities?

In order to proceed with the project, we needed "yes" answers for those questions. This section describes the various validation exercises undertaken.

2.4.1 Hand Coded Interface Specifications

It is useful with any new language schema to hand code a few real-world target samples. This allows language designers to ascertain the range of expressiveness of the language as well as its verbosity—the size and number of expressions that would be necessary to code one example. The first of these properties determines if the language is rich enough to cover common situations, the second one is useful in understanding potential implementation challenges to the language, such as computing resources needed for its storage and processing. It should be noted nevertheless that it is not expected that developers will write XIML directly but rather that they will use tools that will read and write the language.

An XIML specification can contain as few as one of the standard components described in the previous section. Therefore, our hand coded specifications ranged from a single domain component to describe the catalog items of a store, to a task model for a supply-chain management application, to presentation components for simple C++ interface controls for Windows, to entire interface definitions for a number of applications (a geographical data visualization application, a baseball box-score keeper, and a dictionary-based search tool among others). In all of these examples, we found XIML to be sufficiently expressive to capture the relevant interaction data. We did find the language somewhat verbose but well under any threshold that could pose practical implementation problems.

2.4.2 Multi-Platform Interface Development

One of the important uses of XIML can be in the development of user interfaces that must be displayed in a variety of devices. XIML can be used to effectively display a single interface definition on any number of target devices. This is made possible by the strict separation that XIML makes between the definition of a user interface and the rendering of that interface—the actual display of the interface on a target device. In the XIML framework, the definition of the interface is the actual XIML specification and the rendering of the interface is left up to the target device to handle. In the past, many model-based interface development systems [17] and many user-interface management systems [14] have not had this separation established clearly and therefore developers ended up mixing up interface logic with interface definition.

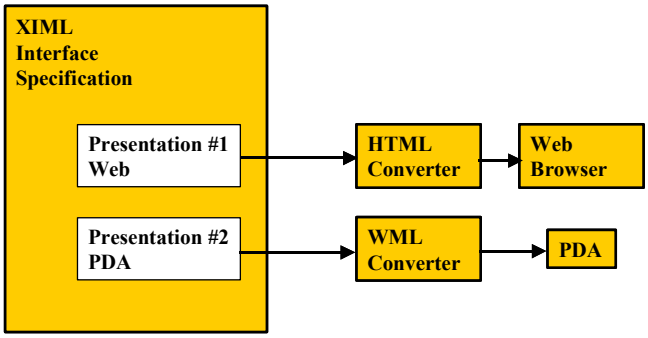

Figure 3. XIML provides a framework for the development of user interfaces that have multiple target displays.

Figure 3 illustrates the XIML framework for multi-platform development for a couple of sample device targets. The language is not restricted to those two types of devices but can theoretically support many types of stationary and mobile devices. In the case shown, there is a single XIML interface specification for the data to be displayed, the navigation to be followed and the user tasks to be supported. Then, by simply defining one presentation component per target device the entire specification can support multiple platforms. Specifying presentation components simply means determining what widgets, interactors, and controls will be used to display each data item on each of the target devices. As far as the rendering of the interface is concerned, an XML-capable device is able to process an XIML specification directly. For the case when the target device is not XML-capable, a converter needs to be used to produce the target language. To support the XIML validation effort, we have developed converters for popular target languages including HTML and WML.

The multi-platform framework described above saves development time and helps ensure consistency. However, there is still the chore of creating a presentation component for each target device. To solve that problem, XIML enables additional capabilities that can provide a high-degree of automation to the multi-platform interface development process. Figure 4 illustrates this automation framework. Instead of creating and managing one presentation component per target device, developers would work with a single "intermediate" presentation component. XIML would then predefine via relations how the intermediate component maps to a specific widget or control on the target display device.

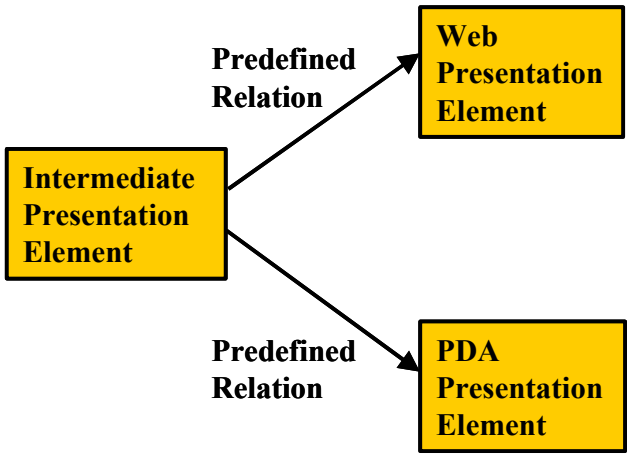

Figure 4. Automation framework for multi-platform interface development in XIML.

As an example, a designer could specify in XIML that a particular data type, say a geographical location, be presented with an intermediate presentation element called "map-location widget". By using the established relations, XIML will then automatically map the map-location widget to an actual graphical-map control for the Web and to a text-based data display on the PDA.

Clearly, specifying the relations between intermediate presentation objects and device presentation objects in a static manner would be too inflexible to be of practical use. There are many considerations that go into selecting an appropriate widget to use in a given instance. These considerations would include screen size, what other elements are on the screen at the same time, user preferences, contextual issues and so on. Therefore, it is expected that intelligent tools would be necessary to handle the task of creating and updating the relations between intermediate and device elements. For the purposes of our project, we first conducted a small-scale validation exercise for a map-annotation user interface to be displayed in two devices (desktop, and cell phone). Then, we used this same problem area to build an XIML-based pilot application, which we report in detail in Section 3.

2.4.3 Intelligent Interaction Management

Another important useful application of XIML in industry may be as a resource for the management of a user interface at runtime. By centralizing in a single definition the interaction data of an interface, it is hoped that we can build tools that will similarly centralize a range of functions related to the operation of that interface. In order to validate the feasibility of using XIML for interaction-management functions, we explored three runtime functions: (a) dynamic presentation reorganization, (b) personalization, and (c) distributed interface management.

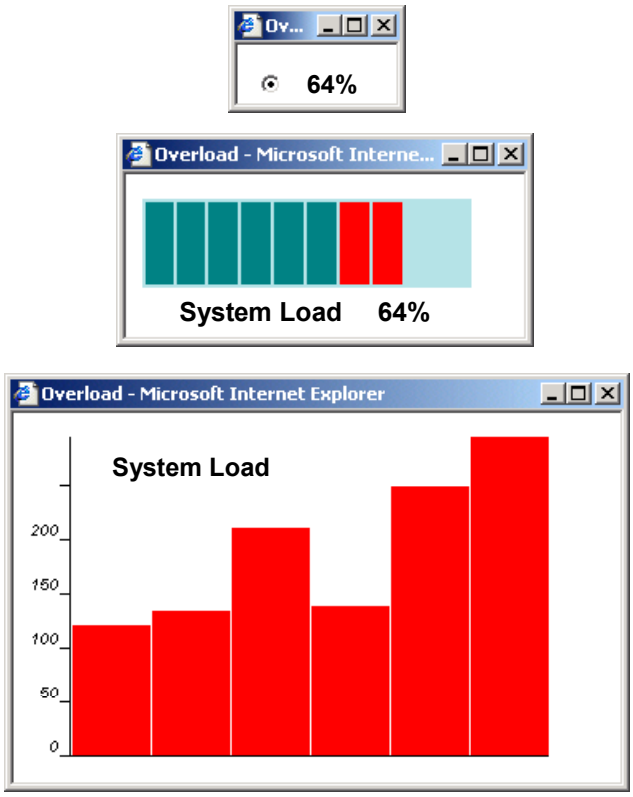

Figure 5. Dynamic presentation reorganization based on available display area.

- **Dynamic presentation reorganization.** Figure 5 shows a sequence of views of a single web page that displays the system load of a server. The page displays different widgets or controls according to the screen area available for display. When that area is minimal, the page displays the most basic data item. As the area increases, additional text and then a graphical view is added. Finally, when the display area is maximized, the page displays the most sophisticated control available for that target data item. To implement this function we built a simple application that read the XIML specification for the interface and dynamically adjusted the presentation component of the specification according to a set of thresholds on the value of the display area available. It is clear that a system that would support sophisticated dynamic presentation reorganization would probably need a good degree of sophistication itself. However, our goal at this point was not to build such a system, but rather to validate that this type of problems can be represented and solved using XIML as an interaction data repository and in a very straightforward manner.

Figure 6. Various widgets available for personalization in an XIML specification.

- **Personalization.** Figure 6 shows a simple example of a personalization feature. The widgets display a reading of a data source (in this case system load as in the previous example). The corresponding XIML specification indicates that there are a number of widgets that can be used to display that data source. In addition, the widgets can be oriented in various manners on the screen. For this feature, we wrote a small application that selected the widget to display according to criteria based on the user component of the XIML specification. As in the previous example, the sophistication of the personalization issue was not the focus of the experiment. The focus was to ensure that XIML has capabilities to support personalization features and that, as the various examples are added together, that the XIML framework can offer the value of a single repository of interaction data to support many user-interface management functions.

- **Distributed interface management.** One of the drawbacks of any client-based software application is that the update of the client software is problematic since each individual client needs to be updated. Server-side applications reduce that problem to a large extent but then have the tradeoff that a server update affects every user of the application at the same time whether these users desire the change or not. In either case, an update is not a trivial task and can be initiated solely by the software provider.

XIML could be used to provide a mechanism for the distributed update of user interface components. Figure 7 illustrates this mechanism. As we saw in the previous two examples, the widget or control being displayed on a page can be changed easily via an XIML specification. That framework assumes that the widget to be displayed is available, but it does not confine it to be on a specific server or a client. It simply treats the widget as a black box that performs a function. We have taken advantage of that flexibility to allow the widget to simply be available somewhere on the network be it on a client, a peer, or a server machine. The XIML specification can be set to link to providers of the widget or it can rely on a search-and-supply application. In this manner, for example, a calendar widget on a travel-reservations page can be provided by any number of XIML-compliant calendar-widget suppliers. The choice of suppliers can be made dependant on any XIML-supported criteria such as user preferences.

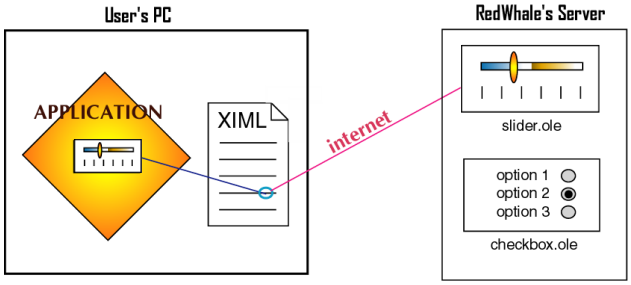

Figure 7. XIML mechanism for distributed interface management.

2.4.4 Task Modeling

One of the critical requirements that we set for XIML was the ability to represent abstract concepts such as user tasks, domain objects, and user profiles (a process that can be referred to as task modeling). Our group has previously developed a number of model-based interface development tools. These tools included, among others, an informal user-task model specification tool called U-TEL [21], and a formal interface-model development environment called MOBI-D [17]. Both of these tools have advanced modeling facilities to represent interface models, including the contextual concepts of user tasks, domain objects, and user profiles. The tools have been used to model a wide variety of applications such as a military-logistics management tool, a medical-data visualization application, and a supply-chain management tool, among others. The interface modeling language used by U-TEL and MOBI-D is a frame-based language that shares some characteristics with XIML. To verify that the task-modeling capabilities of XIML were at least at the same level as those of MOBI-D, we successfully built a converter that can take any MOBI-D model specification and convert it into an XIML specification. We applied the converter successfully to all models previously built with MOBI-D.

2.4.5 Reverse Engineering

While the benefits of XIML are potentially many, practical reality indicates that a very substantial amount of code has been written in HTML. It would be ideal that in the same way that XIML can be converted into HTML, that HTML code could be reverse engineered into XIML. In this manner, the benefits of XIML could be brought to existing applications through some level of automated support. The reverse engineering of HTML into XIML has successfully been accomplished by a research group—working independently from us [4]. The implementation is currently at the prototype level and it has been applied to simple examples such as converting the CHI conference online registration form to XIML.

2.4.6 Summary of Validation Exercises

The validation exercises performed for XIML allowed us to conclude that there is enough evidence to justify the engineering feasibility of XIML as a universal interface-definition language. The next step therefore is the development of a pilot application of XIML, which we report in Section 3.

3. An XIML Pilot Application

In order to satisfy our exit criteria for the first phase of the development of XIML, we must (1) successfully complete the validation exercises, and (2) be able to build a proof-of-concept prototype of interest. When we say "of interest", we refer principally to two items:

- The prototype must implement a solution to a problem that the software industry is currently facing and for which there is no efficient solution under current technologies
- The prototype must demonstrate a new set of features and/or methodologies that are valuable and that could be incorporated into mainstream industry frameworks within a short period of time (one or two years at most)

It is clear that the theme of this book—that of multiple user interfaces—fulfills the requirements of the first interest item. The ubiquity of internet-capable devices, both mobile and stationary is now a reality. Users of these devices demand an integrated interactive experience across devices but the software industry lacks methods and tools to efficiently design and implement such an experience. For the second interest item, we rely on the widespread acceptance of XML to transition XIML into industry. In addition, we will demonstrate via the proof-of-concept prototype that there are a number of valuable middleware functions that can easily be based on the XIML framework.

In the remainder of this section, we first present the problem application domain and then we illustrate how we define the abstract components of the corresponding XIML representation for the problem domain. Next, we detail various middleware functions useful within the context of our sample problem.

3.1 MANNA: The Map Annotation Assistant

MANNA is a hypothetical software application that reveals many of the challenges posed by user-interface development for multiple user interfaces, including mobile ones. MANNA is a multimedia application that must run on several platforms and can be utilized collaboratively over the Internet. It is intended for use 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.

For this application domain, we created an interaction scenario that demanded various display devices and posed a number of presentation and dialog constraints. 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 (see Figure 8). The desktop workstation poses few limiting constraints to UI development, but obviously it is a stationary device. The documents are therefore downloaded to a laptop, and the geologist boards a plane for the site.

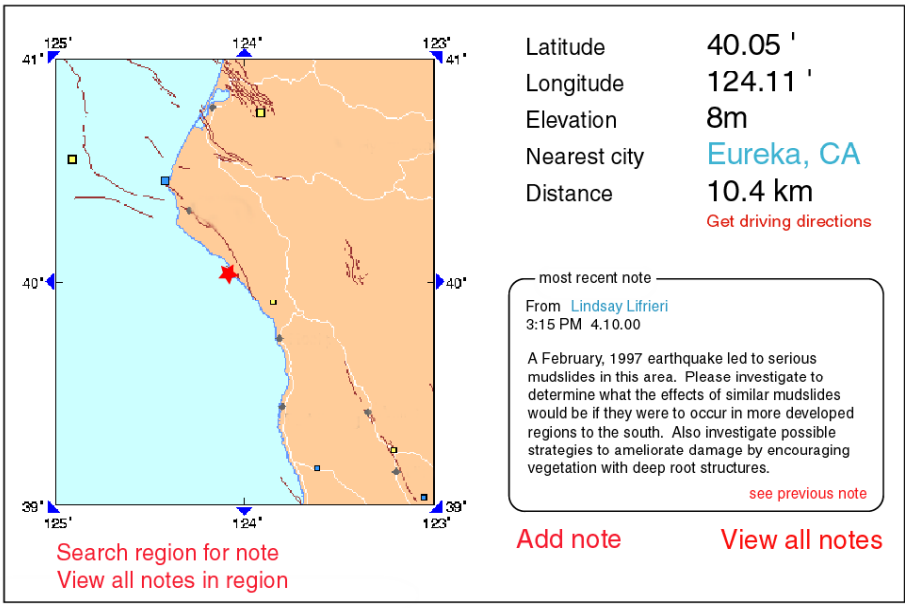

Figure 8. The desktop UI design for MANNA. Hyperlinks are indicated by color.

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., list boxes replace drop-lists.

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 (see Figure 9). 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.

Figure 9. The cell phone UI design for MANNA.

Finally arriving at the site, our geologist uses a palmtop computer to make notes on the region (see Figure 10). Since the palmtop relies on a touch pen for interaction, interactors that require double-clicks and right-clicks are not permitted. Screen size is a concern here, so a more conservative layout is employed. Having completed the investigation, our geologist prepares a presentation in two formats. First, an annotated walk-through is presented on a heads-up display (HUD). Because of the HUD's limited capabilities for handling textual input, speech-based interactors are used instead. A more conventional presentation is prepared for a high-resolution large-screen display. Since this is a final presentation, the users will not wish to add information, and interactors that are intended for that purpose are removed. The layout adapts to accommodate the larger screen space, and important information is placed near the center and top, where everyone in the audience can see it.

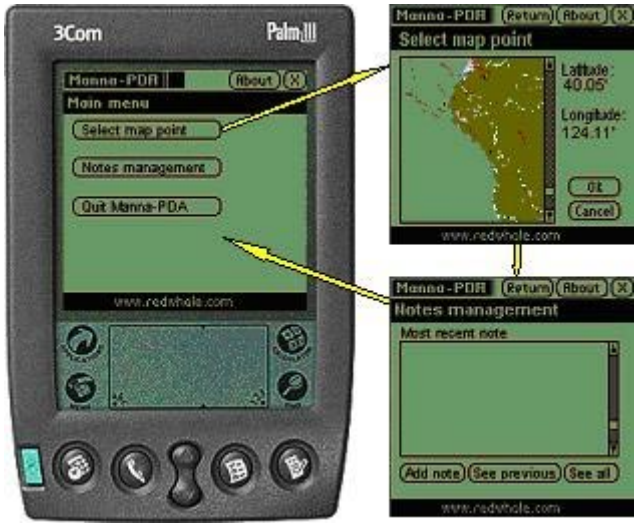

Figure 10. The PDA UI design for MANNA.

Clearly in this scenario we are faced with quite a number of presentation and dialog constraints. Obviously, we could build each interface separately, but that would duplicate work multiple times and would make it difficult to maintain consistency among interfaces. Ideally therefore, we can use XIML to define as single abstract interface specification (see Section 2.4.2), and then implement appropriate middleware that can correctly generate each interface, under each set of constraints, from that single specification.

3.2 The MANNA Abstract XIML Components

The first step in building our proof-of-concept prototype is to define the abstract XIML components for the MANNA application. These components are: task, domain, and user. With these abstract components in hand, we will be able to develop middleware that allow us to manage the creation of multiple target interfaces. For this task, we used a basic XML editor. Following are samples of the resulting MANNA abstract XIML components.

3.2.1 Task Component

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 valid tasks can range from simple one-step actions (e.g., Enter Date, View Map) to complicated multi-step processes (e.g., Perform Contract Analysis). For MANNA, we created a task/subtask

decomposition of the user tasks based on our interaction scenario. A portion of this task component is as follows:

```
<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>
```

3.2.2 Domain Component

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 using more complex structures.

The domain model is typically structured into a "contains" hierarchy. In this case, inheritance relations are specified using relation statements. However, as with all model components, the structure of the domain component can be set by the user [27]. Here is a portion of the XIML domain component for MANNA:

```
<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. 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 data types, 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.

3.2.3 User Component

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, a highest-level user type might be "Medical Personnel", containing a child element "Technician", which in turn contains

a child element “Nuclear Medicine Technician”. 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 levels of expertise. The following is a portion of the user component for MANNA:

```
<USER_MODEL ID='umode1'>
        <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>
```

3.2.4 Using XIML relations

The three abstract components detailed above constitute a basic representation of the MANNA application domain from a user interface point of view. There is however, a further body of knowledge that is not captured as of yet by those components. Namely, how do we answer a basic question such as what domain objects are involved in completing a defined user task.

To answer those questions and more, XIML uses relations. In short, a relation links two or more XIML elements (e.g., tasks, users) with each other and may define a semantic meaning to the relation. We have identified a number of relations among XIML elements that are of interest [16], such as for example relations between task elements and domain elements. The semantics of a relation are not mandated by XIML. Instead, the language allows the application handling the relation to process it according to its own semantics. It is expected, however, that a set of basic XIML predefined relations will be applied with similar semantics across applications. The mechanism to manage and enforce such basic semantics is currently under conceptualization.

3.3 XIML-Based Middleware for MANNA

Once the abstract components of MANNA are defined in XIML, we can proceed to build the user interface for each of our target platforms (Desktop, PDA, and cell phone. Basically, this entails defining complete XIML presentation and dialog components for each of the platforms. By defining those concrete components and relating them to the abstract components we already have a fairly positive gain as our three target user interfaces are consistently linked to a single set of task, domain objects, and user types.

However, we are more interested in means to automate—at least to some degree—the process of creating the three target presentation and dialog components. To that effect for the MANNA application, we defined three *middleware units*. These units are able to examine a partial XIML specification and augment and/or modify that

specification based on their internal logic and available knowledge base. Of the various middleware units considered [7,9,10] we focus here in particular on three units to support the following functions: (1) interactor selection, (2) presentation structure definition, and (3) contextual adaptation.

3.3.1 Interactor Selection

The presentation component of XIML is used to predefine the presentation elements and attributes of display devices. For each device, XIML captures all of the interactors (widgets) available in that device as well as any associated parameters and attributes (e.g., screen resolution). As part of the building process for a user interface, it is necessary to select an appropriate interactor for each user action and domain object to be displayed. Clearly, for multiple user interfaces this can turn out to be a challenging, if not tedious task. Therefore, our first middleware unit automates the selection of interactors for a wide range of situations.

First, the unit simplifies the selection of interactors by managing sets of Abstract Interaction Objects (AIO) and Concrete Interaction Objects (CIO) [24]. An AIO is a platform-neutral XIML presentation element that symbolizes an action or a domain object display. A CIO is an actual widget available in a target platform. Figure 11 illustrates the AIO-CIO relationship.

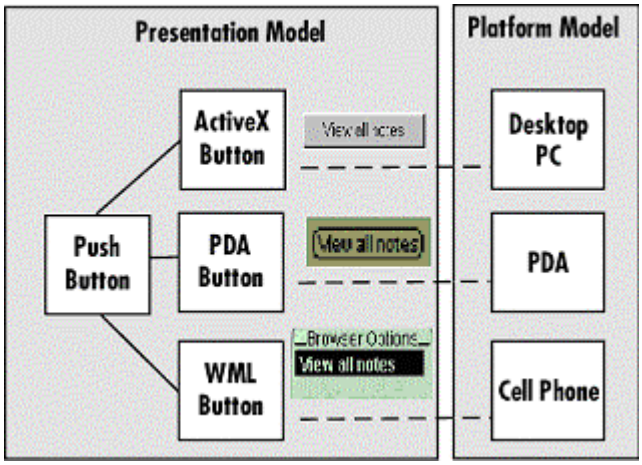

Figure 11. An Abstract Interaction Object ("Push Button") is subclassed into Concrete Interaction Objects available in specific platforms.

The middleware unit manages the set of AIO-CIO relationships. It can set or cancel those relationships based on designer input or by examining the XIML platform specifications for a device and using its knowledge base to determine the appropriate CIOs for a given AIO. In this manner, a user interface designer can create a single design based on AIO and the middleware unit completes the interaction selection process.

Second, the unit helps with dynamic aspects of interactor selection. There are a number of constraints that can affect the selection of an interactor at runtime. These include attributes such as screen resolution, available display area, current bandwidth, user preferences and so on. For MANNA, we explored adaptation based on two constraints: available display area and screen resolution. Based on the CIO established by the unit as appropriate for a given presentation situation, the unit attempts to change the value of the CIO's XIML attribute values to meet the applicable constraint values. For example, this can take the form of shrinking a widget. Alternatively, it may replace one valid CIO with another based on the stated constraints (see Figure 5). In this manner, a developer of multiple user interfaces gains a considerable degree of freedom by not having to account explicitly for constraint-satisfaction issues in interactor selection. [8,11,23]

3.3.2 Presentation Structure Definition

Another key aspect of building a user interface is the definition of the presentation structure. For example, in a desktop device this would entail defining how many windows will be created, what user tasks will be completed in each window, and correspondingly, what interactors will be placed in each window. Our second middleware unit helps automate this task. The unit helps us deal with two interrelated issues: (1) How do user tasks translate into a presentation structure, and (2) how to choose an optimal solution for a device among various presentation structure options.

In XIML, we call the presentation structure the union of the language's two concrete components: the presentation component and the dialog component. The first one represents the hierarchy of presentation elements in the interface (e.g., windows, widgets) whereas the second one defines the actions between user and system (i.e., navigation, input and responses).

The middleware unit supports the translation of user tasks into presentation structures in two steps. First, it creates an initial XIML dialog component that is based on the abstract user task component. The following is an example of this initial dialog component (Note the similarities with the task model in 3.2.1):

```
<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>
```

Furthermore, it is necessary to link the dialog elements to the interactors selected as in Section 3.3.1. The XIML sample below exemplifies a dialog component after interactors have been linked. Note that in Section 3.3.1 we linked interactors to domain objects, which in turn are linked to user tasks. By now linking tasks with dialog components, we can complete the full circle and link the dialog elements with

the presentation elements (interactors). In the sample XIML dialog component, the <ALLOWED_CLASSES> tag indicates which types of elements are allowed to participate in the dialog-interactor relation; in this case, any child of the dialog model component i1 can appear on the left side, and any child of the presentation model component p2 can appear on the right side. Note the presence of an attribute definition inside the relation definition; this indicates that the relation itself can be parameterized. 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'>
   <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>
```

Note that to this point, we have linked many of the elements of the XIML components, but do not have yet a definition for how, for example, we would distribute the user tasks and selected interactors among windows in a desktop device. This is a design problem that must take into consideration various factors, among them target device and its constraints, interactors selected, and distribution strategy (e.g., many windows vs. single window). The middleware unit can give support to this function via a mediator agent [3]. As Figure 12 shows, a mediator can examine the XIML specification, including the device characteristics and offer a user task distribution based on an appropriate strategy for the device in question.

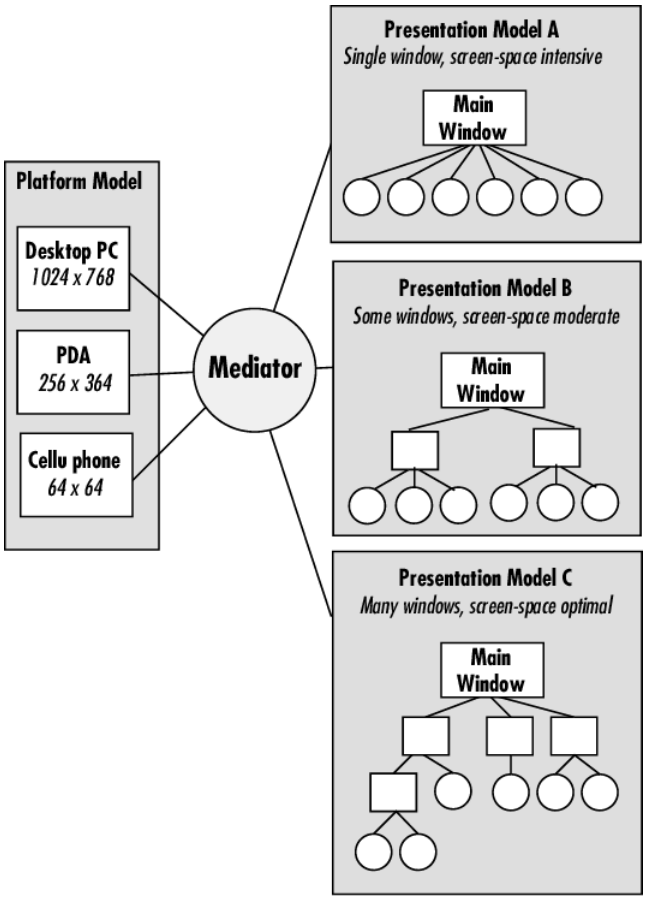

Figure 12. A mediating agent dynamically selects the appropriate presentation model for each device.

After the process shown in this section, we have a comprehensive XIML specification for a user interface (concrete and abstract components). The specification is fully integrated and can now be rendered into the appropriate device. The middleware unit significantly simplifies the development work associated with completing the specification.

3.3.3 Contextual Adaptation

The context in which interaction occurs ha an obvious impact on what user tasks may or may not be performed at any given point in time. For example, in the scenario in Section 3.1, 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.

A middleware unit can take advantage of this knowledge and optimize the user interface for each device. The designer creates mappings between platforms (or classes of platforms) and tasks (or sets of tasks) at the abstract level. 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 fig. 7. In this figure, the task model is shown to be a simple collection of tasks. This is for simplicity's sake; in reality, the task model is likely to be a highly structured graph where tasks are decomposed into subtasks at a much finer level than shown here.

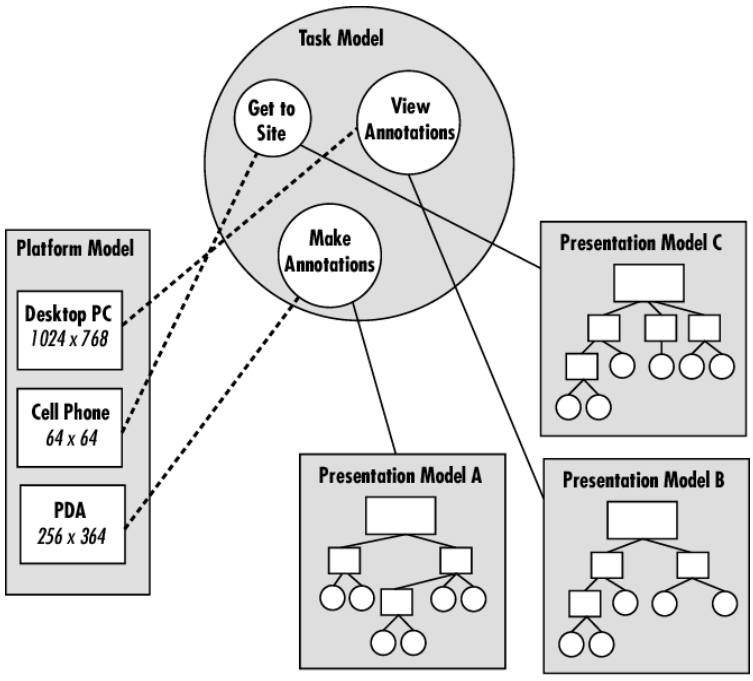

Figure 13. Platform elements are mapped onto task elements that are especially likely to be performed. In turn, the task elements are mapped onto presentation models that are optimized for the performance of a specific task.

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 are represented by AIOs that are easily accessible. For example, on a PDA, clicking on a spot on the map of our MANNA application allows 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 (see Figure 10). 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 (see Figure 9). The "bottom arrow" button of the cellular phone enables the user to select other options by scrolling between them.

This middleware unit therefore benefits designers and developers by managing the localized contextual changes that apply across devices for a given task. The treatment by this unit of optimizations for the task structure of each device is similar to our treatment of screen-space constraints: global, structural modifications to the presentation model are often necessary, and adaptive interactor selection alone will not suffice.

4. Discussion

We conclude by offering a perspective on the ongoing development of the XIML framework, examining related work, and highlighting the main claims of this Chapter.

4.1 The XIML Roadmap

We consider that our exit criteria for the initial phases of the development of the XIML framework have been satisfied. Therefore, we plan to continue this effort. We have devised a number of stages that we plan to follow to build and refine XIML into an industry resource. Each of the stages constitutes a development and evaluation period. The stages are as follows:

1. **Definition.** This phase includes the elicitation of requirements and the definition of language constructs, which we have completed.

2. **Validation.** Experiments are conducted on the language to assess its expressiveness and the feasibility of its use. This phase is being on this Chapter.

3. **Dissemination.** The language is made available to interested parties in academia and industry for research purposes (www.ximl.org). Additional applications, tests, and language refinements are created.

4. **Adoption.** The language is used by industry in commercial products.

5. **Standardization.** The language is adopted by a standards body under a controlled evolution process.

There is no single measure of success in this process. The language may prove to be very useful and successful at certain levels but not at others. In addition, the analysis of the functional and theoretical aspects of XIML is just one of several considerations that must be made in order to develop a universal language for user interfaces. It

should be noted first that the meaning of the world "universal" in this context is a language that has broad applicability and scope. The term should not be considered to mean a language that is used by every developer and every application. We do consider, however, that the evidence produced so far seems to indicate that further efforts are warranted.

4.2 Related Work

The work on XIML draws principally from previous work in three areas: model-based interface development [17,18,19], user interface management systems [14], and knowledge representation for domain ontologies [12]. XIML shares some of the goals of these fields but is not directly comparable to them. For example, the main focus of model-based interface development systems has usually been the design and construction of the user interface. For XIML, this is just one aspect, but the goal is to have a language that can support runtime operations as well. In this point, it mirrors the aims of user-interface management systems. However, those systems have targeted different computing models and their underlying definition languages do not have the scope and expressiveness of XIML.

There are also some related efforts in the area of creating XML-based user interface specification languages. UIML [1] is a language geared towards multi-platform interface development. UIML provides a highly detailed level of representation for presentation and dialog data but provides no support for abstract components and their relation to the concrete user interface design. Consequently, while UIML is well suited for describing a user interface design, it is not capable of describing the design rationale of a user interface. Ali et al. have recently begun to explore the integration of external task modeling languages with UIML [2]. However, at present XIML remains the only language to provide integrated support for both abstract and concrete components of the user interface.

There are several existing or developing standards that overlap with one or more of XIML's model components. XUL [6], Netscape's XML-based User Interface Language, provides a thin layer of abstraction above HTML for describing the presentation components of a web page. Just as XUL overlaps with the XIML's presentation component, CC/PP [5,26] and UAProf [5,22] provide similar functionality to XIML's platform component. ConcurTaskTrees provides an XML representation of a UI's task structure [15]. DSML [13], the Directory Services Markup Language, offers an adequate representation for the domain structure, at least in the case of e-commerce applications.

Of course, none of these languages provides support for relational modeling between components, which is the essence of XIML's representation framework. Since all of these languages are based in XML, it is straightforward to translate them into XIML. Consequently, we view the existence of these existing languages as an advantage. In the future, we hope to show how XIML can exploit existing documents in each of these languages.

4.2 Summary of Findings

In this Chapter, we have reported on the following results.

- XIML serves as a central repository of interaction data that captures and interrelates the abstract and concrete elements of a user interface.
- We have established a roadmap for building XIML into an industry resource. The roadmap balances the requirements of a research effort with the realities of industry.
- We have performed a comprehensive number of validation exercises that established the feasibility of XIML as a universal interface-specification language.
- We have completed a proof-of-concept prototype that demonstrates the usefulness of XIML for multiple user interfaces
- We have successfully satisfied our exit criteria for the first phases of the XIML framework development and are proceeding with the established roadmap.

Acknowledgements

Jean Vanderdonckt made significant contributions to the development of the MANNA prototype and to the XIML validation exercises. We also thank the following individuals for their contribution to the XIML effort: Hung-Yut Chen, Eric Cheng, Ian Chiu, Fred Hong, Yicheng Huang, James Kim, Simon Lai, Anthony Liu, Tunhow Ou, Justin Tan, and Mark Tong.

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] M. F. Ali, M. Perez-Quinonez, M. Abrams, and E. Shell. Building multi-platform user interfaces using UIML. In C. Kolski and J. Vanderdonckt, editors, *Computer Aided Design of User Interfaces (CADUI'02)*, 2002.

[3] Y. Arens and E. H. Hovy. The design of a model-based multimedia interaction manager. *Artificial Intelligence Review*, 9: 167–188, 1995.

[4] L. Bouillon and J. Vanderdonckt. Retargeting web pages to other computing platforms. In *Proceedings of IEEE 9th Working Conference on Reverse Engineering (WCRE'200*2). IEEE Computer Society Press, pp. 339-348, 2002.

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

[6] T. Cheng. XUL: Creating localizable XML GUI. In *Fifteenth Unicode Conference*, 1999.

[7] 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.

[8] J. Eisenstein and A. R. Puerta. Adaptation in automated user-interface design. In *Proceedings of the 5th International Conference on Intelligent User Interfaces (IUI '00)*, pages 74–81. ACM Press, 2000.

[9] J. Eisenstein and C. Rich. Agents and GUIs from task models. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI '02)*, pages 47–54. ACM Press, 2002.

[10] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the 6th International Conference on Intelligent User Interfaces (IUI '01)*, pages 69–76. ACM Press, 2001.

[11] S. Kawai, H. Aida, and T. Saito. Designing interface toolkit with dynamic selectable modality. In *Proceedings of Second International Conf. on Assistive Technologies (ASSETS '96)*, pages 72–79. ACM Press, 1996.

[12] 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.

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

[14] D. Olsen. *User Interface Management Systems: Models and Algorithms*. Morgan Kaufmann, San Mateo, CA, 1992.

[15] F. Paterno, C. Mancini, and S. Meniconi. ConcurTaskTrees: A diagrammatic notation for specifying task models. In S. Howard, J. Hammond, and G. Lindgaard, editors, *Proceedings of IFIP International Conference on Human-Computer Interaction (Interact'97)*, pages 362–369. Chapman and Hall, 1997.

[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] P. Szekely. Retrospective and challenges for model-based interface development. In F. Bodart and J. Vanderdonckt, editors, *Computer Aided Design of User Interfaces (CADUI'96)*, pages 1–27, Wien, 1996. Springer-Verlag.

[19] P. Szekely, P. Luo, and R. Neches. Beyond interface builders: model-based interface tools. In *Conference Proceedings on Human Factors in Computing Systems (InterCHI '93)*, pages 383–390. ACM Press, 1993.

[20] 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.

[21] C. 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 (IUI '98)*, pages 77–80. ACM Press, January 1998.

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

[23] J. Vanderdonckt and P. Berquin. Towards a very large model-based approach for user interface development. In *Proceedings of First International Workshop*

on User Interfaces to Data Intensive Systems (UIDIS '99), pages 76–85. IEEE Computer Society Press, 1999.

[24] J. M. Vanderdonckt and F. Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In *Proceedings of the Conference on Human Factors in Computing Systems (InterCHI '93)*, pages 424–429. Addison-Wesley Longman Publishing Co., Inc., 1993.

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

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

[27] The XIML Forum. http: //www.ximl.org.