XIML: A Universal Language for User Interfaces

Angel Puerta and Jacob Eisenstein
RedWhale Software
277 Town & Country
Palo Alto, CA USA
+1 650 321 0348
{puerta, jacob}@redwhale.com

ABSTRACT

In recent years, there have been a number of industry and academic efforts to standardize the representation of many types of data in order to facilitate the interoperability of applications. There is, however, no comparable effort aimed at *interaction data*, the data that relates to user interfaces. We introduce XIML (eXtensible Interface Markup Language), a proposed common representation for interaction data. We claim that XIML fulfills the requirements that we have found essential for a language of its type: (1) it supports design, operation, organization, and evaluation functions, (2) it is able to relate the abstract and concrete data elements of an interface, and (3) it enables knowledge-based systems to exploit the captured data. In this paper, we introduce the characteristics of XIML, its scope and validation, and a proposed path for industry adoption.

Keywords
User interface languages, model-based systems, user-interface management systems, interface models

INTRODUCTION

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 were 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) [6,12]. These and many other efforts are being channeled through standards organizations such as the World Wide Web Consortium [12] and the Organization for the Advancement of Structured Information Systems [6].

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 also 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 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 this paper, we propose a solution for the representation and manipulation of interaction data. We introduce XIML (eXtensible Interface Markup Language), an XML-based language that enables a framework for the definition and interrelation of interaction data items. As such, XIML can provide 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.

In the following sections, we first discuss the requirements elicited for XIML. Second, we present the structure, organization and scope of the XIML language. Third, we continue by describing the validation process followed and by presenting examples of user-interface engineering functions enabled via XIML. Fourth, we proceed to discuss a proposed plan for the dissemination and adoption of XIML. Finally, we conclude by discussing related and future work.

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.

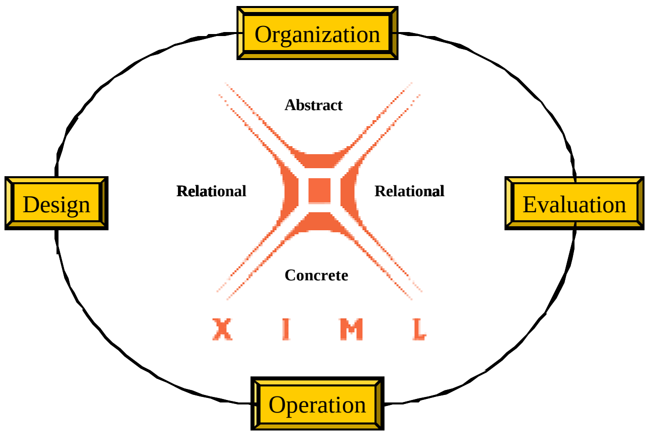


Figure 1. XIML represents abstract, concrete and relational interface data items. It also enables user-interface engineering functions of design, operation, evaluation, and organization.

- **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 of 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 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.
- **Relational support.** The language must be able to effectively 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 [7,9]. 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.

THE STRUCTURE OF XIML

The XIML language draws mainly from two foundations. One is the study of ontologies and their representation [4], and the other one is the work on interface models [8,9,10]. 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 will be made available with the XIML documentation [15]. 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. Following, we examine each of its main representational units.

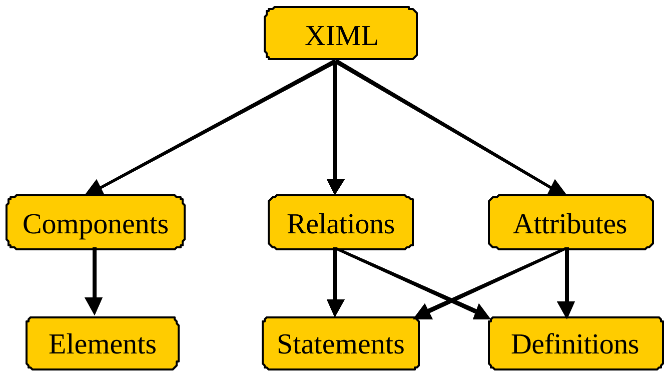


Figure 2. The basic representational structure of the XIML language.

Components

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 [4] 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 [8,9]. 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.

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 [7].

XIML supports relation *definitions* that specify the canonical form of a relation, and relation *statements* that specify actual instances of relations. It should be noted that XIML does not specify the *semantics* of those relations. Those are left up to the specific applications that utilize XIML.

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.

VALIDATION OF XIML

In order to validate the expressiveness and usefulness of XIML, we undertook a number of tests and projects. These activities have the main goal of allowing us to assess the feasibility of XIML satisfying the requirements that we elicited for the language. The validation activities included among others

- Hand coded representation of interfaces
- Multi-platform interface development
- Intelligent interaction management
- Task modeling
- Reverse Engineering

Hand Coded Interface Definition

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 implementation problems.

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 [8] and many user-interface management systems [5] have not had this separation established clearly and therefore developers ended up mixing up interface logic with interface definition.

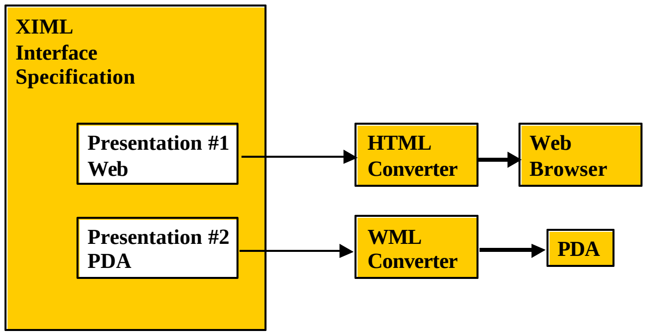
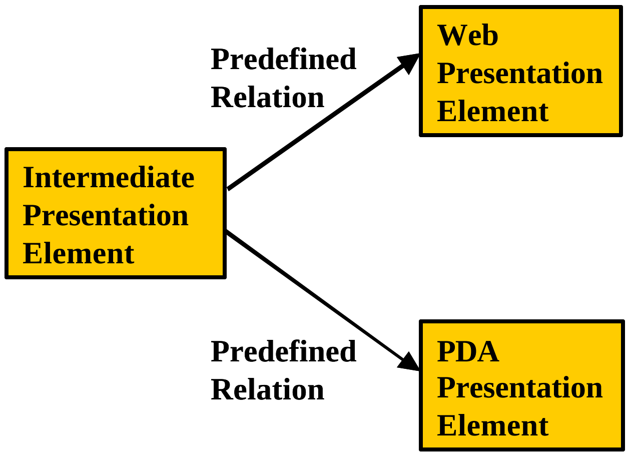


Figure 3. XIML provides a framework for the development of user interfaces that have multiple target displays.

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 shown case, 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 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 offers 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, is to 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. As part of the validation process for XIML, we have reported in detail elsewhere on an intelligent system that can automate to a large degree the development of interfaces for multiple devices [2]. We have shown in that work an example of displaying a map-annotation user interface on a desktop, a PDA, and a cell phone. The intelligent system automates the multi-platform interface-development process by managing the relations between intermediate and device elements taking into consideration a number of device and design constraints and rules.

Intelligent Interaction Management

One of the main goals of XIML is to provide 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.

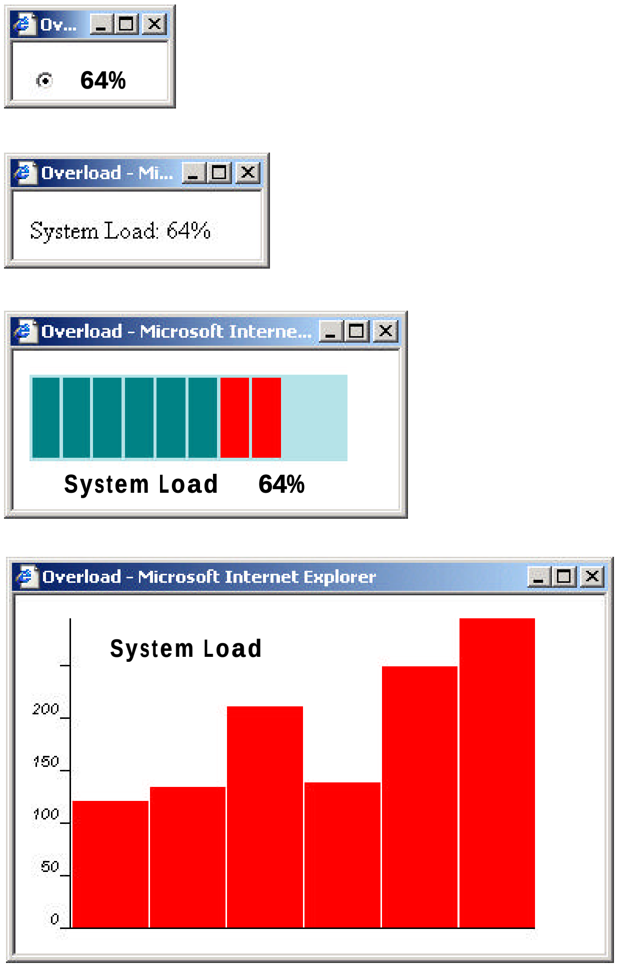


Figure 5. Dynamic presentation reorganization based on available display area.

- **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.

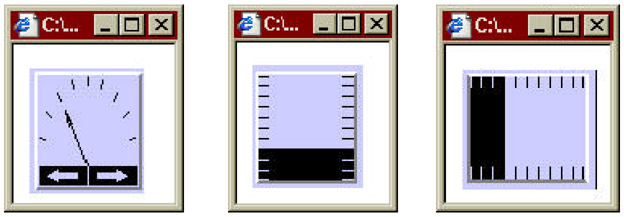


Figure 6. Various widgets available for personalization in an XIML specification.

- **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 provides 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.

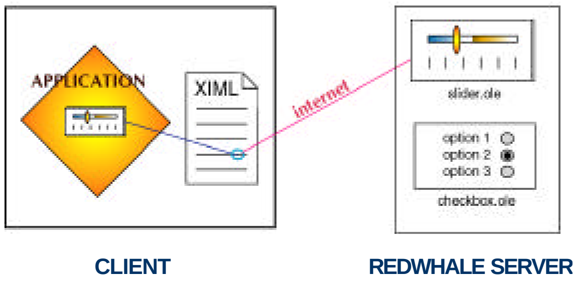


Figure 7. XIML mechanism for distributed interface management.

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 [11], and a formal interface-model development environment called MOBI-D [8]. 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.

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 [13]. 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.

Summary of Validation

The validation tests performed for XIML allow us to conclude that there is enough evidence to justify the engineering feasibility of XIML as a universal interface-definition language. This will enable us to move the development of XIML into its second phase as discussed in the following section.

THE XIML ROADMAP

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 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.
2. **Validation.** Experiments are conducted on the language to assess its expressiveness and the feasibility of its use. This is the phase being reported on this paper.
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. We do consider, however, that the evidence produced so far seems to indicate that further efforts are warranted.

RELATED WORK

The work on XIML draws principally from previous work on three areas: model-based interface development [8] user-interface management systems [5], and knowledge representation for domain ontologies [4]. In general, XIML shares some of the goals of these fields, but it is not directly comparable in nature to them. For example, the main focus of model-based interface development systems over the years has been the design and construction of the user interface. For XIML, this is just one aspect but the goal it to have a language that can support runtime operations as well. In this point, it mirrors the aims of user-interface management systems but 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 additional efforts in the area of creating XML-based user-interface specification languages. UIML [1] is a language geared towards multi-platform interface development. However, it does not capture context data, it is not intended to support knowledge-based system functions, does not target operation and evaluation functions, and it does not clearly separate the rendering of the interface from the definition of it. XUL [3] is a language developed with the Netscape 6 browser for the definition of user-interface elements on a web page. It is much more limited than XIML and even UIML in scope and therefore not directly comparable. In addition, some other groups have implemented individual interface design and/or operation functions using an XML-based representation [14]. In general, those efforts are aimed at solving the particular problem at hand and do not have the generality sought for XIML.

CONCLUSIONS

We have introduced XIML, an interface representation language for universal support of functionality across the entire lifecycle of a user interface: design, development, operation, management, organization, and evaluation. We have described the various language-validation activities undertaken, including among others intelligent-interface management functions, user-task modeling activities, and multi-platform interface development. We claim that there is enough evidence to support the continuation of this effort to its next phase, which is the dissemination of the language within the academic and industry research communities.

ACKNOWLEDGMENTS

We thank the following individuals for their work on XIML: Hung-Yut Chen, Fred Hong, Yicheng Huang, James Kim, Simon Lai, Tunhow Ou, Justin Tan, Mark Tong and Jean Vanderdonckt.

REFERENCES

1. Abrams, M. et al. “Appliance-Independent XML User Interface Language”. In Proc. *Eighth International World Wide Web Conference*, Toronto, Canada, 1999.
2. Eisenstein, J., Vanderdonckt, J. and Puerta, A. “Applying Model-Based Techniques to the Development of Uis for Mobile Computers”. In *IUI01: 2001 International Conference on Intelligent User Interfaces*. Santa Fe, NW, pp. 69-76.
3. Mozilla. http://www.mozilla.org.
4. Neches, R. et al. “Enabling Technology for Knowledge Sharing”. In *AI Magazine*, Volume 12, Number 3; Fall 1991, pp. 36-56.
5. Olsen, D. *User Interface Management Systems: Models and Algorithms*. Morgan Kaufmann. San Mateo, CA 1992
6. Organization for the Advancement of Structured Information Systems. http://www.oasis-open.org
7. Puerta A. and Eisenstein, J. “Towards a General Computational Framework for Model-Based Interface Development Systems”. In *Knowledge-Based Systems*, Vol. 12, 1999, pp. 433-442.
8. Puerta, A.R. “A Model-Based Interface Development Environment”. In *IEEE Software*. 1997. pp. 40-47.
9. Szekely, P. et al. “Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach”. In *Engineering for Human-Computer Interaction*, L.J. Bass and C. Unger (eds), Chapman & Hall, London, 1995, pp 120-150.
10. Szekely, P., Luo, P. and Neches, R. “Beyond Interface Builders: Model-Based Interface Tools”. In *Proc. of InterCHI’93*, ACM Press, New York, 1993, pp. 383-390.
11. Tam, R.C.-M., Maulsby D., and Puerta, A. “U-TEL: A Tool for Eliciting User Task Models from Domain Experts”. In Proc. *IUI98: 1998 International Conference on Intelligent User Interfaces*. San Francisco, CA. ACM Press.
12. The World Wide Web Consortium. http://www.w3c.org.
13. Vanderdonckt, J., Bouillon, L., and Souchon, N., “Flexible Reverse Engineering of Web Pages with Vaquita”. In Proc. *WCRE'200: IEEE 8th Working Conference on Reverse Engineering*. Stuttgart, October 2001. IEEE Press.
14. Vanderdonckt, J. (ed.). Proceedings of *CADUI 2002: Computer-Aided Design of User Interfaces*. In Press.
15. XIML. http://www.ximl.org.