Representation Languages For User Interfaces To Appliances

Krzysztof Gajos
kgajos@cs.washington.edu

January 30, 2003

1 Motivation For Developing Universal UI Languages

1.1 UI Languages

A number of XML-based platform independent user interface description languages have been proposed in recent years. Some of the most desirable features of proposed to motivate this research are summarized here:

Cross-platform interoperability Currently developers need to invest a lot of effort to port user interfaces from one platform to another even if the general layout and mechanics of the interface does not change. The ideal UI description language should make it easy to port an application from one system to another. The most common approach to this problem is to design a platform-neutral description of the interface and then pass it to a platform-specific renderer.

Support a range of display devices The sizes of displays range from very small monochrome displays on cell phones to large on-wall projected displays with millions of pixels. Some applications need to interact with the user in any of those environments. Ease of deployment and application's portability would be improved if the user interface could be specified once and deployed on any kind of device.

Support a range of interface styles Even on the same kind of a device, a number of interface styles may need to be supported to reflect different needs of different users or just a different context the device is used in. A tablet computer may be used in a laptop mode where keyboard is available or in the tablet mode where long text input may be inconvenient (thus favoring option menus over free-form input). Also, different interface styles may be required by users with disabilities.

Support for dynamic choice of interaction modalities Some efforts exist to build UI description languages that could be rendered through means other than graphics. Speech is the most commonly mentioned other modality though some groups suggested also new non-standard devices such as haptic interfaces or virtual reality.

Facilitate deployment in distributed and heterogeneous software environments Strict separation of the interface and the application enables interaction with the application through a remote interface. This potentially enables delivering the UI to the client through a web browser and instantly enabling interaction with a server-side application. It can also display limited devices such as household appliances to export their user interfaces to other computationally and graphically more powerful devices such as the user's handheld computer.

1.2 Appliance Description Languages

Several projects have attempted to build universal interfaces to physical devices. A lot of this work extends some of the concepts of the universal UI languages. The motivation for this work is summarized below:

Providing a better interface than the original Everyday appliances are growing in complexity and their button-based hardware interfaces can hardly support the exploding number of features offered. It is hoped that more dynamic interfaces could expose the available features and make them easier to use.

Figure 1: Comparison

Replace numerous custom remote controls with a single universal control Every A/V appliance as well as some other household or office items (such as motorized drapes, X10 light control, etc) come with a remote control. Some times a relatively simple task, such as playing a movie from a VCR, may involve interacting with several different remote controls (TV, VCR, audio) but certainly requires finding the right one among many that are available. Universal remote control projects hope to replace all of these gadgets with a single multi-function UI on a PDA, for example.

Providing additional or improved functionality The universal controls are intended not only to replicate the controls already present on the device's interface. They are supposed to make it easier to complete tasks regardless of how many simple actions they would ordinarily require with the device's own interface. For example erasing all old messages on an answering machine may require multiple button presses on the phone's interface but only a single command on the universal control. Additionally, the universal control may provide easy ways to invoke tasks that require coordination of several devices. For example, a single button press may be sufficient to configure the TV, VCR and the audio system for playing a movie.

Provide alternative interface for users with disabilities This is related to the multi-style and multi-modal requirement for UI description languages. Given a sufficiently abstract of appliance's capabilities, one can render the UI in was accessible to users with various cognitive disabilities.

A uniform interface for gaining access to new appliances or gaining control of a new space In the age when mobile and nomadic computation is becoming more and more of a common place, user's personal devices should be able to discover and interact with the surrounding environment. One aspect of it would be to automatically provide the control UIs for the surrounding appliances to the nomadic user upon entering a space.

2 Evaluation Of Existing Solutions

A brief summary of this section is provided in Figure 1.

2.1 XUL

The XML-based User Interface Language (XUL) [5] is the language used by Mozilla-based browsers for specifying the layout and some of the behavior of the graphical user interface. It is XML-based and allows the same interface to be rendered on a variety of platforms. The interface description is composed on a number of files, each containing different aspects of the interface. The main file contains all of the widgets, their organization and references to the underlying application code which should handle user's interaction with particular parts of the UI. Another file can contain definitions of entities (such as all text objects) thus allowing for easy internationalization. Finally the style file contains more fine grain information about the interface's look and feel.

XUL seems to be most useful for adding or modifying UI elements of Mozilla-based browsers. Unfortunately it appears that the language cannot really be used for building new applications independent of the Mozilla code base[1].

Because XUL forces the programmer to describe the GUI at fairly detailed level, the XUL-based interfaces cannot automatically adapt to devices with radically different display capabilities. It is possible, however, to build one's own interface to Mozilla for any platform as long as the base program would run there.

[1] I have just found out that an open-source project jXUL (http://jxul.sourceforge.net/) is working on building a swing-based renderer for XUL. I am not sure yet how far the project has progressed and what subset of XUL it is able to render.

2.2 AUI

AUI [8] is a low level language for describing graphical user interfaces in a platform-neutral manner. The kinds of widgets are specified explicitly (window, button, shape) and their exact dimension and size can be specified. The AUI system forwards GUI update information from the application to the renderer and passes GUI events back to the application. Although not explicitly mentioned in the paper, the system could probably be used in a distributed system. Due to the very low level at which GUI elements are specified, AUI provides no support for automatic adaptation to devices of different capabilities or for custom rendering styles.

2.3 UIML

The User Interface Markup Language (UIML) [1] developed by Harmonia has been designed to allow more rapid porting of applications to different platforms whether they differ mainly by their windowing API or more fundamentally at the level of their capabilities (e.g. desktops vs. cell phones). UIML achieves its goal by carefully separating and reusing those aspects of a UI that do not change between different platforms from those that do. In particular, the GUI description is broken into five parts:

- description: lists elements and the classes they belong to
- structure: specifies hierarchical structure of the elements
- data: provides information about the elements such as their names
- style: specifies how to render different classes of elements
- events: specifies what elements generate what events and how those events should be handled

The style information contained in a UIML description is thus language-specific making it nearly impossible to influence how the UI is rendered on the client side.

2.4 ISL

The Interface Specification Language (ISL) [3] was developed as a middleware layer above the distributed object layer. Its goal is to extend the traditional distributed component programming approach by forcing applications to expose a portion of their system state (that pertains to the interaction with the user) as documents. In practice, the interfaces defined in ISL are a mix of two kinds of approaches:

– generic descriptions of individual methods and their parameters that can be rendered in a way preferred by the client-side renderer

– references to pre-build interface objects that need to be downloaded over the network.

This approach allows the client side to build any kind of a user interface since it can easily ignore references to the already provided parts of the UI.

2.5 Roman, et al.

Another architecture comparable to ISL has been proposed by [2]. In that architecture, services methods and their parameter types are described in an XML-based format. Unlike in ISL, though, each of these descriptions is accompanied by the types of widgets that should be used to render the interface to this method. Also, no structural information is provided about which methods are related to one another making automatic layout for complex interfaces nearly impossible (one has no way of knowing that play and pause buttons for a CD player should be close to one another, for example). In this approach, however, this is irrelevant because GUI rendering is done using XSL. Thus different XSL style sheets need to be created for different kinds for displays but since the interface layout is decoupled from element descriptions, new UIs can be created easily. This approach does not support different modalities.

2.6 XIML

The eXtensible Interface Markup Language (XIML) [7] is a very flexible language that allows allows multiple renditions of a user interface on different clients devices. XIML allows GUI elements to be defined abstractly in terms of the type of the information they are supposed to convey or elicit from the user. Multiple rendering options are provided and they can be chosen at run time either according to user's personal preference or by the renderer according to the available screen area and device capabilities.

The above information is based on the conclusions of the creators of XIML. No XIML specification, code samples or live demos are available for public scrutiny.

2.7 iCrafter

ICrafter [6] uses a language loosely based on ISL to describe services offered by physical devices present in an environment. In rendering the UIs, it attempts to merge the contextual information about the space and the user with the device description. For example, it may use the information about physical layout of displays and projectors to generate the GUI for controlling these devices.

ICrafter generates the GUIs through a set of template-based generators. The generators are platform-specific and usually new ones have to be written for each class of physical devices. Thus the total number of generators required by the system is the product of the number of classes of controllable devices and the number of platforms used for control.

2.8 Pebbles

The Pebbles project at CMU has produced a system for automatically constructing remove control interfaces for devices given their abstract description [4]. The devices are represented in terms of state variables and commands – state variables correspond to those aspects of the device functioning that can be both manipulated and observed by the software interface. The commands correspond to those actions, whose effects may be visible in the environment but are not observable by the interface. Individual elements are grouped into a hierarchical structure showing how closely related various parts of the interface are and thus giving some direction in how the interface should be rendered. The descriptions of the elements also include explicit tags with spoken names of the components to facilitate automatic creation of voice interfaces.

References

[1] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. UIML: An appliance-independent xml user interface language. *WWW8 / Computer Networks*, 31(11-16):1695–1708, 1999.

[2] Manuel Román James Beck and Alain Gefflaut. A device-independent representation for services. In *Third IEEE Workshop on Mobile Computing Systems and Applications*, 2000.

[3] Todd D. Hodes and Randy H. Katz. A document-based framework for internet application control. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[4] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joe Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *CHI Letters: ACM Symposium on User Interface Software and Technology, UIST'02*, Paris, France, 2002.

[5] The Mozilla Organization. The XPToolkit architecture, 1999. http://www.mozilla.org/xpfe/xptoolkit/.

[6] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A service framework for ubiquitous computing environments. In *Proceedings of Ubicomp 2001*, pages 56–75, 2001.

[7] Angel Puerta and Jacom Eisenstein. XIML: A universal language for user interfaces, 2002. unpublished paper available at http://www.ximl.org/.

[8] Kevin A. Schneider and James R. Cordy. AUI: A programming language for developing plastic interactive software. In *Proceedings of the 35th Annual Hawaii International Conferece on System Sciences*, 2002.