On-the-fly Generation of UIs for Ubiquitous Applications

Davide Carboni, Andrea Piras, Stefano Sanna, Gavino Paddeu
CRS4 – Center for Advanced Studies, Research and Development in Sardinia
VI strada Ovest, Z.I. Macchiareddu
I – 09010 Uta (CA), Italy
{dadaista, piras, gerda, gavino}@crs4.it

ABSTRACT

As software applications become ubiquitous they must adapt seamlessly to different devices, from desktop PC to cellular phones, from web pages to voice interfaces. This paper addresses the problem of platform independent user interfaces (UI); it introduces some of the challenges related to ubiquitous applications requirements and presents an original approach to UI generation. This approach has driven the development of MORE, Multimodal Object REnderer, a framework for automatic runtime generation of UI for desktop workstation, set-top boxes, PDA, web browsers, and mobile phones.

Categories and Subject Descriptors

D.1.m [**Programming Techniques**] Miscellaneous – *Java, XML, HTML, WML, VoiceXML*; D.2.2 [**Software Engineering**]: Design Tools and Techiques - *User Interfaces*; H.5.m [**Information Interfaces and Presentation**]: Miscellaneous.

General Terms

Design, Experimentation, Human factors, Languages.

Keywords

Platform Independent User Interfaces, XML, Java, Device Adaptation.

1. INTRODUCTION

The last decades have witnessed a radical transformation in the way that people use computers. From the mainframe age, in which a single machine was shared among dozens of expert users, to early '90s where each user had a whole machine available on his desktop, full equipped with data, programs and peripherals. Nowadays, the computers/people ratio has flipped over and one single user often uses many computers, in different contexts and situations: at work, always connected to the broadband Internet; in the car or during a trip, connected or disconnected to narrowband networks depending on the physical availability of radio channels.

Some requirements emerge from the above scenario [1]. The computer is no longer a repository for data and programs. It becomes a window to the environment services. Applications do not cease to work if the user turn-off the device: the application exists in a given place, adapts itself to the user, and is accessible from any device. Applications are not simply on-line, but rather they enhance the physical surroundings giving the user the perception to move in an empowered environment: virtual spaces must enhance physical spaces.

2. CHALLENGES

To-date, development environments are aimed for building applications whose execution context is known a priori. The Java language is aimed to be a platform-independent language, but, nevertheless, at the time it was conceived the number of devices able to host a Java Virtual Machine (JVM) was really small. Indeed, the slogan "Write once run anywhere" is today too optimistic and fails in its literal intent and is meaningful only for classes of devices: an application user interface based upon a component library like Swing cannot be executed in a device with small footprint and minor hardware capabilities. Moreover, there are devices that cannot run a JVM at all or devices with mono-dimensional input like a voice interface.

3. APPROACH

A way to manage the complexity in the design of platform independent user interfaces is through the Model-View-Control design pattern [3]. Such a pattern decomposes a system in a set of models that are strictly bound to the data either persistent or transient that are managed by the system and shared with the user. Our approach is based upon models defined as objects in a given programming language (i.e. Java) and must be distinguished from model-based design, in which tasks and dialogues are modeled in ad-hoc declarative languages [5]. For each model, the designer can provide one or more views that give the user the perception of data objects and, eventually, the possibility to edit them. A peculiarity of device independent user interfaces is that we can indeed define device independent models but cannot define views because views are concrete "interactors" and must be defined by composing widgets belonging to a specific toolkit such as Swing, AWT, Tcl/Tk and so forth. In our approach, we have developed a framework for defining and composing models (i.e. a string value is a model of a text object and can be part of an enclosing model such as the model of a form). Therefore, such a composition can be transformed at runtime into a structure of concrete and interactive objects by means of a "rendering process". In such a process, widgets such as text fields, radio buttons, and more sophisticated controls, are deployed in place of the data they refer: strings, booleans, and more complex data types. The advantage of this approach is that the same model can be used from any channel for which is available a rendering engine. According to the physical features of possible user devices, we grouped them in three fundamentals categories: fat clients, devices with high processing power, reliable Internet connection (i.e. PCs,

notebooks, workstations) that can dynamically download and run mobile code; thin clients, devices with limited power processing and narrowband Internet connection (i.e. PDAs, smartphones, set-top boxes) that can run custom applications but they cannot execute dynamic mobile code; null clients, devices with Internet connection equipped with a 3rd party browser (i.e. cellular phone WAP-enabed, Internet kiosks and terminals, VoiceXML browsers).

4. MORE – MULTIMODAL OBJECT RENDERER.

According to the approach described above, we implemented a modular architecture called Multimodal Object REnderer (MORE).

4.1 Model introspection

An application written for MORE contains some classes defining the platform independent "model" for the UI. Such classes are extensions of base classes provided by our framework. Platform

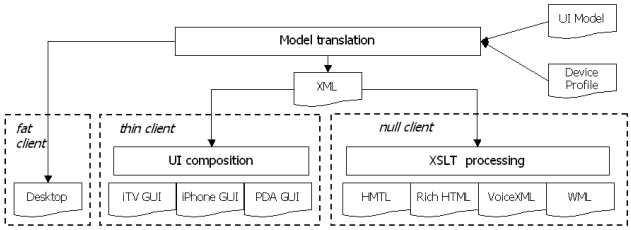

Figura 1. Object Renderer Engine

independent model classes contains fields that are either "composite model" or "basic models" such as strings, numbers, dates, and so forth. By means of "Java Reflection", such models are inspected at run-time and, depending on the actual device profile, a set of concrete views is built, as described in next step.

4.2 From models to views

Based upon actual device profile, a different family of views is loaded. When a model element is found, the corresponding view is created. For fat clients, MORE directly instantiates UI components and assembles them into a container; such an interface is composed by Swing widgets. On the other hand, in the case of thin and null clients, MORE generates an intermediate

XML representation of platform independent models. In such cases the task to build concrete interactors is delegated to a "client renderer" (see §4.3) or to an XSLT processor (see §4.4), respectively. MORE can be easily extended to support new basic model types.

4.3 UI composition on thin clients

Since this kind of devices cannot load mobile code, the runtime generation of user interfaces relies on a two-tiered architecture. The local tier is a component installed on user's device. It receives and the XML representation transmitted by the remote tier, parses it, and generates a UI on-the-fly. To-date, we have implemented local-tiers for iTV (MHP), PDA (PersonalJava), and smartphone (J2ME). The local-tier assembles the concrete UI according to device's specific "Look & Feel"; for instances, operations are rendered as buttons on a PDA, while they'll be soft-button menus on a smartphone. This way, the concrete UI generated is adapted and gets the best of device's characteristics.

4.4 XSL transformation for null clients

For null clients, the XML representation must be translated into in a markup language manageable by the user browser. MORE uses a XSLT processor which takes XML and one XSL style sheets returning a markup language document adapted for the actual device. We have identified three main tag-based clients: web, WAP and voice. Actually, MORE implementation supports four possible target languages: "rich" HTML, for browsers with complete Javascript support; "small" HTML for browsers with limited Javascript support (i.e., mini browsers bundled with PDAs); WML for WAP-enabled mobile phones; VoiceXML for Interactive Voice Responder. For each of them, MORE selects the suitable set of XSL style sheets to perform the translation.

5. CONCLUSIONS

This paper has presented MORE, the Multimodal Object REnderer, a framework aimed to write platform independent user interfaces and the architecture that performs on-the-fly UI generation. It takes an abstract model of application front-end and translates it into a concrete UI, according to physical features and computational capabilities of the target device. To-date, there are eight different implementations of rendering engine, ranging from desktop workstations to mobile phones and voice-based interactors. Considering that design and development of UIs are some of the most expensive tasks in terms of money, time and intellectual energies (up to 50% of whole project duration [4]), the availability of such a framework provides some immediate and considerable advantages: multimodal UI translator makes "design once, display anywhere" a reality. MORE has been used extensively in the e-MATE project [2], where it has been a key component in the development of multimodal distributed applications. MORE has some drawbacks, mainly related to the automatic association between model types and UI components. MORE, in fact, introduces some constraints that limit a fine customization of UI.

6. ACKOWLEDGEMENTS

MORE is a research line of the e-MATE that has been funded by Italian Ministry for University and Scientific Research. We want to thank all people of the Network Distributed Application group at CRS4 for their contributes.

7. REFERENCES

[1] G. Banavar et al., "Challenges: An Application Model for Pervasive Computing," in Proc. MobiCom'2000, August 2000, 266-274.

[2] D. Carboni, S. Giroux, E. Vargiu, C. Moulin, S. Sanna, A. Soro, G. Paddeu. e-MATE: an open architecture to support mobility of users. Proc Fifth Baltic DB&IS, June 3-6, 2002, Tallinn, Estonia.

[3] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-wesley, Reading, MA, 1995.

[4] B. A. Mayers, and M.B Rosson. Survey on user interface programming. Human factors in computing system. Proceedings SIGCHI'92 - Monterey – CA.

[5]: F. Paternò. Model-Based Design and Evaluation of Interactive Applications. Springer, 2000, ISBN 1-85233-155-0