Filling the gap between users and objects: a multichannel interactive environment

Davide Carboni, Andrea Piras, Stefano Sanna, Gavino Paddeu
CRS4 - Center for Advanced Studies, Research and Development in Sardinia
Parco Scientifico e Tecnologico, POLARIS, Edificio 1, 09010 PULA (CA - Italy)
Email: {dcarboni, piras, gerda, gavino}@crs4.it

***Abstract*— The design and the implementation of software for mobile computers and for pervasive computing environments entail several issues and lead to new requirements. Applications must handle at run-time the heterogeneity of delivery contexts in terms of devices features, network bandwidth, input and output modalities, operating systems, and so forth. In this paper, we propose a practical approach for enabling platform independent application development and on-the-fly generation of user interfaces based on object introspection and device adaptation.**

I. Introduction

The design and the implementation of software for mobile computers and for pervasive computing environments [1] entail several issues. Furthermore, software developers spend up to 50% of a whole project time-resources [2] in the design, implementation and testing of friendly and intuitive user interfaces. Despite such a considerable work, applications initially developed for a traditional Web browser or running in personal computers must often be re-implemented from scratch in order to reach new terminals such as set-top boxes, voice browsers and smartphones. It would be very useful if a system could automatically generate an interface by the analysis of an application logic and data models. Unfortunately, such a system is still in the wish list of most programmers. Yet development environments are aimed for building applications whose execution contexts are known a priori. For one, the Java programming language is aimed to be platform-independent. Nevertheless at the time it was designed 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: the user interface of an application based upon a component library like Swing cannot be run on a device with a small footprint and few hardware capabilities, like a WAP-phone. Moreover, there are devices that cannot run a JVM at all or devices with mono-dimensional input like voice. In this paper, we describe the Multichannel Object REnderer (MORE), a framework able to generate user interfaces from a reflective analysis of data and Java code used for defining the application model.

II. Device Categories

According to their physical features, we grouped devices in three fundamental categories: fat clients, thin clients, and web-like clients.

Fat clients are devices with high processing power and reliable Internet connection. They are able to dynamically download and run mobile code. Examples of fat clients are PCs, notebooks, tablet-PCs and workstations.

Thin clients are devices with limited power processing and narrowband Internet connection. They can run custom applications but can not load and execute dynamic mobile code. Such category includes Personal Digital Assistants (PDAs), hi-end programmable mobile phones and set-top boxes for interactive television.

Web-like clients are equipped with a third party browser and they include low-end WAP-enabled mobile phones, Internet kiosks and VoiceXML browsers.

III. An Object-driven Conceptual Design

A way to manage the complexity in the design of platform independent user interfaces is through the Model-View-Control (MVC) design pattern [3]. Such a pattern decomposes a system in a set of models that are strictly bound to the persistent or transient data. The data are managed by the system and shared with the user. For each model, the designer can provide one or more user interfaces, the View-Control (VC) subsystem. The VC part provides the user with a representation often graphical of data objects and may also offer the possibility to edit them. A peculiarity of device independent user interfaces is that we can indeed define device independent models but cannot define device independent views because views are concrete ”interactors” and must be defined in terms of widgets belonging to a specific toolkit, for example Java Swing. In our approach, we have developed a framework for defining and composing models. For instance, 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, the composition of objects can be transformed at runtime into a structure of concrete and interactive objects by means of a ”rendering process”. In this process, widgets (called run-time editors) such as text fields, radio buttons, and more sophisticated controls, are deployed in place of the data they refer to: strings, numbers, booleans and more complex data types like object arrays and collections. Complex models are viewed as composite graphical objects and the rendering process is applied recursively till atomic interactors are obtained.

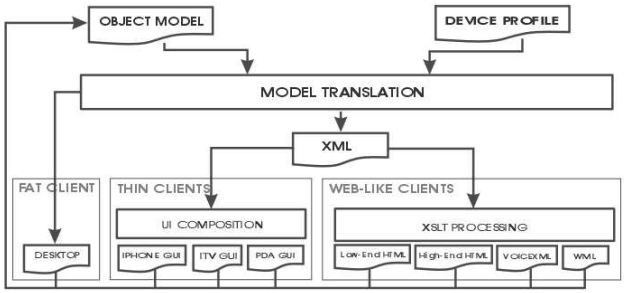

Fig. 1. Multichannel Object REnderer architecture.

IV. Model Introspection

An application written for MORE contains a set of classes defining the platform independent ”model” for the user interface. Some of these classes extend or implements base classes provided by the framework. Fields of these classes are either ”composite model” or ”basic models”. Basic models do not need to be decomposed to get a user interface for them. Samples of basic models are strings, numbers, dates, and so forth. By means of ”Java Reflection”, a model is inspected at run-time to identify its data (fields) and the actions (methods) the user may have access to. Depending on the current device profile, a concrete view is built. Basic models contained in the model under analysis are directly placed in the view while composite models are represented by hypertext links. When the user requires a representation of the composite model (i.e. pushing on a button or clicking on a link or obtaining it as result of a method invocation) the composite model becomes the next model processed by MORE, and so on.

V. An Example: Personal Agenda

A user interface model consists of a set of classes containing fields and operations the user can interact with. We have implemented a simple personal agenda application to show how MORE works. Figures 2 and 3 show the Unified Modelling Language (UML) Class Diagram and Sequence Diagram of the personal agenda model. The ”UserLogin” class is the entry point of the application; it consist of the ”userName” string field and the ”login” method. Such a methods returns an instance of the ”PersonalAgenda” class. The ”PersonalAgenda” class holds the status of the user (known-unknown), an array of meetings, a method to add a new meeting. Each meeting has a name, a date, a flag for confirmation and a time. The ”NewMeeting” class extends the ”Meeting” class by adding a method for saving the data of the new meeting in the meeting Vector of ”PersonalAgenda”. Given such a model, MORE is able to create concrete user interfaces and chain them with an appropriate event model (see figures 4, 5, 6, 7).

VI. User Interface Generation in Fat Clients

For fat clients, both the model introspection and the user interface construction are performed on client side. The rendering engine directly interacts with the user interface model, builds a concrete user interface in a frame for the actual

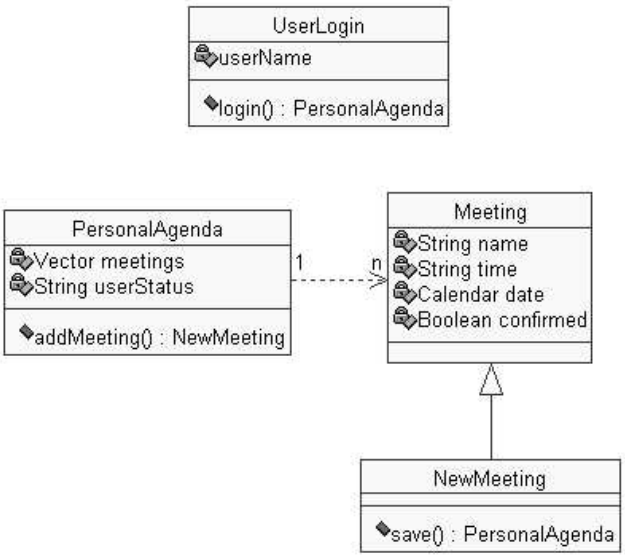

Fig. 2. UML Class Diagram of the model of the personal agenda.

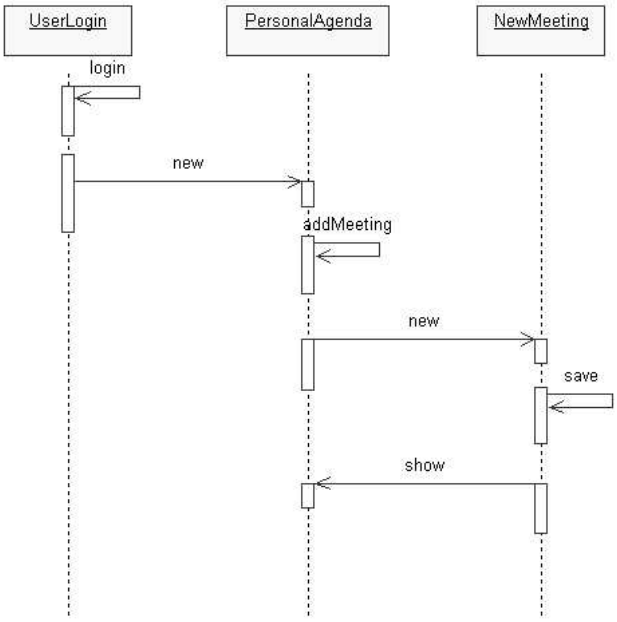

Fig. 3. UML Sequence Diagram of the model of the personal agenda.

operating system. The engine analyzes the properties (fields of object) and the methods of the model and for each property it searches a run-time editor. If a suitable run-time editor is found, it is added to the frame. The developer can also provide a sort ”layout description” to the engine, in order to show editors in a certain order and to change the displayed name of fields and methods (that usually have to follow certain naming conventions not suitable for the end-user presentation). If no editor has been found the property is displayed as a link. By selecting such a link the user requests to re-iterate the rendering process for the property object. Finally, for each method the engine adds a trigger to the Frame (e.g, a JButton

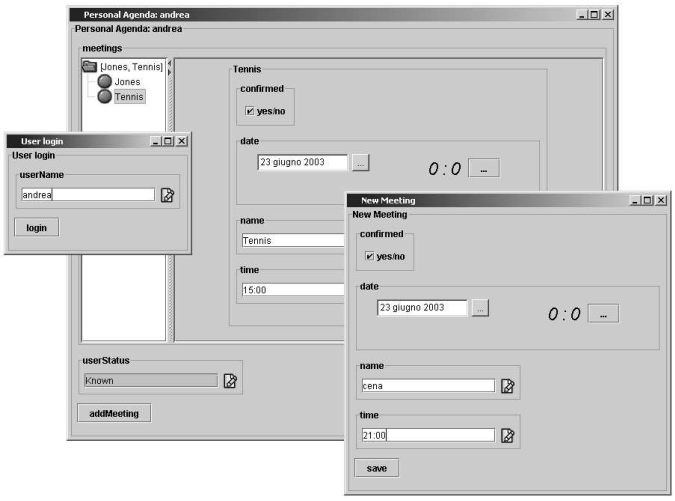

Fig. 4. The agenda example rendered on a fat client, using the components of the Java Swing library.

in the Swing implementation).

VII. User Interface 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 remote tier creates a XML representation of the model and sends it to the local tier. The local tier is a component installed on user's device, receives the XML representation, parses it, and generates a user interface on-the-fly.

To-date, we have implemented local-tiers for iTV (MHP), PDA (PersonalJava), and smartphone (J2ME). The local-tier assembles the concrete user interface 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 user interface generated is adapted and gets the best of device's characteristics. The local-tier sends back a XML document to remote tier specifying which next task has to be performed and the list of the modified values of the fields in the actual user interface that must updated in the remote model.

VIII. XSL Transformation for Web-like Clients

The same XML representation sent from the remote tier to local one about the thin clients is used by the Web-like clients to generate a document in the mark-up language manageable by the user browser. MORE uses a XSLT [4] processor which takes the XML representation of the current model and one selected set of XSL style sheet and returns a mark-up language document adapted for the actual browser. The browser is identified analyzing the headers sent in the HTTP request and the set of XSL style sheets is selected according it.

We have identified three main tag-based clients: web, WAP and voice. Actually, MORE implementation supports four possible target languages: "high-end" HTML [5], for browsers with complete Javascript and CSS support; "low-end" HTML

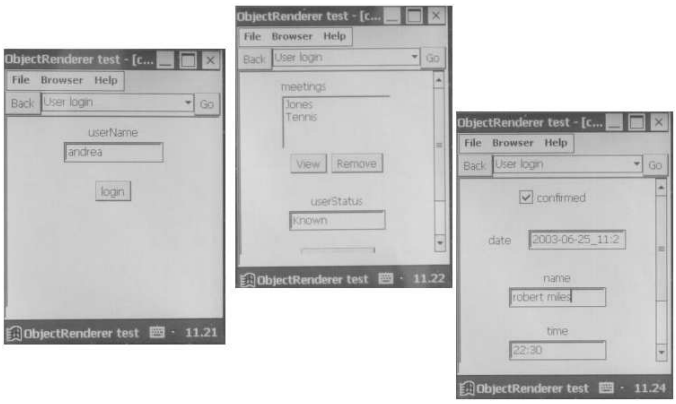

Fig. 5. The agenda example rendered on a thin client (PersonalJava on a Compaq iPAQ).

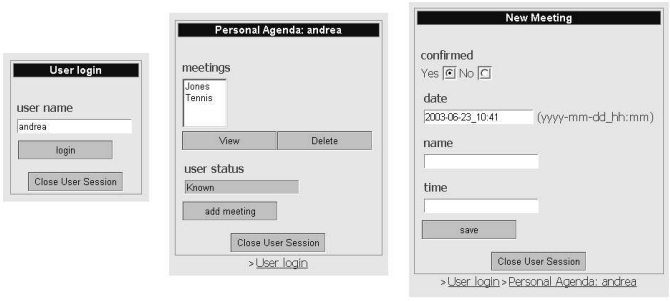

Fig. 6. The agenda example rendered on a "high-end" HTML browser.

for browsers with limited Javascript and CSS support (i.e., mini browsers bundled with PDAs); WML [6] for WAP-enabled mobile phones and VoiceXML [7] for Interactive Voice Responder.

IX. Related Works

A lot of researches on the user interfaces have been carried out about the problem "how to generate multiple user interfaces from a single model". The model-based user interfaces is an important line in such branch. The starting model is usually defined through visual tools able to generate task model. A "task model consists of a set of typed feature structures, referred to as the task descriptions. Informally, a task description serves to specify a minimal amount of

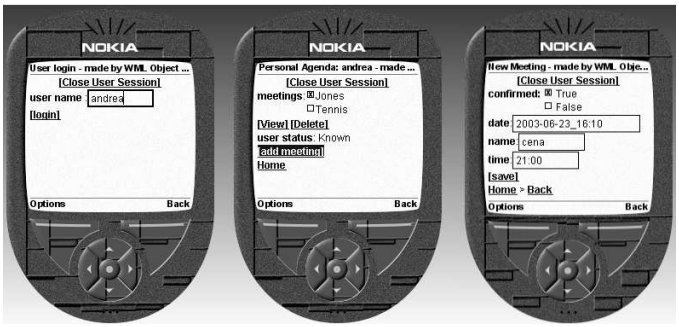

Fig. 7. The agenda example rendered on a WML browser.

information that is necessary in order to perform a specific task, and the conditions that have to be verified in order for the execution of the task to be admissible." [8]. The task model is translated in text document written following an ad-hoc declarative language (often it is a XML document with a custom structure). A processor parses the text and composes the user interface. Pros and cons of model-based user interfaces methodologies are discussed in [9], [10] and [11] and a complete overview is available in [12]. MASTERMIND ([13] and [14]) is another tools and defines a technique based upon the definition of three distinct models: one about the functionalities available by means of the user interface, another defines the presentation structure in terms of widgets and the third model defines the dialog in terms of end-user interaction and how these affect the presentation and the application. In our approach, we follow a different idea based on the decoupling the model from a system that controls and shows it, as indicated by the MCV design pattern. Such model is developed using a high level programming language (C/C++, Java, Smalltalk and so forth), differently by the model-based approach. To solve the problem raised in the first sentence of this paragraph, we join the MCV features to the Java reflection properties. An approach alike to the one of Silica [15], although we have the necessity to expand the user interface environments. Naked Objects [16] and MORE are two frameworks with some common aspects. Both of them analyze a model realized from Java objects but Naked Objects only on one channel (Java Swing) and the models must extend basic classes while in MORE they must be Javabeans [17].

X. Conclusion

This paper has presented MORE, the Multichannel Object REnderer, a framework aimed to write platform independent user interfaces and the architecture that performs on-the-fly user interface generation. The framework provides immediate and significant advantages. MORE makes "design once, display anywhere" closer to reality. A programmer solely has to write the model in Java and the reflection of Java code allows extracting the structure of Java types and our system uses this structure to generate a user interface. The model then can be deployed on any device for which there is a rendering engine. MORE is easily extensible as new devices categories emerge. So as technology evolves, it is not necessary to rewrite the user interface but simply to adapt the rendering engine to this device category. To-date, there are eight different implementations of rendering engine, ranging from desktop workstations to mobile phones, PDA and voice-based interactors. MORE has been developed at CRS4 and adopted as framework for the development of multichannel applications in the e-MATE project [18], co-funded by the University and Scientific Research Ministry of the Italian Government. The developers that have used MORE have given us valuable feedback and interesting comments about how the system should be improved to become really effective in the design of user interfaces for industrial applications. Among these suggestions, MORE should provide a clear mechanism to customize the user interface for a given platform: developers need not only to define a layout in terms of XY position whenever applicable, but also to control the dimensions of widgets in terms of percentage of the whole screen. The current implementation of MORE addresses partially this aspect. In fact, a complete control of the user interface presentation is a requirement that conflicts with the platform independency of user interface itself. A promising way to cope with this problem is to find a trade off between platform independent specification and full-custom specification. MORE has some drawbacks, mainly related to the automatic association between model types and user interface components, in fact, introduces some constraints that limit a fine customization of user interface. Although the research about the generation of multiple user interfaces from a single model groups a consistent number of people and resources, a complete, effective and easy to use methodology is still to come.

References

[1] M. Weiser, Some Computer Science Problems in Ubiquitous Computing," Communications of the ACM, pp.75-84, 1993.

[2] B. A. Mayers and M.B Rosson. Survey on user interface programming. Human factors in computing system. Proceedings SIGCHI'92, Monterey, 1992.

[3] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns, Addison-Wesley, 1995.

[4] XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/xslt.

[5] HTML 4.01 Specification, W3C Recommendation 24 December 1999, http://www.w3.org/TR/html401.

[6] Wireless Application Protocol Forum, Ltd, Wireless Markup Language Version 2.0, Wire-less Application Protocol WAP-238-WML-20010911-a, 11 September 2001,http://www1.wapforum.org/tech/documents/WAP-238-WML-20010911-a.pdf.

[7] Voice eXtensible Markup Language (VoiceXML) version 1.0, W3C Note 05 May 2000, http://www.w3.org/TR/voicexml.

[8] Matthias Denecke, The task model, http://www.dfki.de/etai/SpecialIssues/Dia99/denecke/ETAI-00/node7.html.

[9] C. Janssen, A. Weisbecker, J. Ziegler, Generating user interfaces from data models and dialogue net specifications. In: Ashlund S., et al. (eds.): Bridges between Worlds. Proceedings InterCHI'93 (Amsterdam, April 1993). New York: ACM Press, pp. 418-423, 1993.

[10] A. R. Puerta, The Study of Models of Intelligent Interfaces, Proceedings of the 1993 International Workshop on Intelligent User Interfaces, Orlando, pp. 71-80, 1993.

[11] E. Schlungbaum, T. Elwert, Automatic user interface generation from declarative models, Vanderdonckt J. (ed.): Computer-Aided Design of User Interfaces. Namur: Presses Universitaires de Namur, pp. 3-18, 1996.

[12] F. Patern, Model-based design and evaluation of interactive applications, Springer, 2000.

[13] T. P. Browne et al., Using declarative descriptions to model user interfaces with MASTERMIND. In F. Paterno and P. Palanque, editors, Formal Methods in Human Computer Interaction.

[14] Szekely, P. et.al. Declarative interface models for user interface construction tools: the MASTERMIND approach. In Proc. EHCI'95 (Grand Targhee Resort, August 1995).

[15] R. Rao, Implementational Reflection in Silica, ECOOP 1991, LNCS 512, Springer-Verlag, pp. 251-267, 1991.

[16] R. Pawson and R. Matthews, Naked Objects, John Wiley and Sons Ltd, 2002.

[17] JavaBeans, Sun Microsystems, http://java.sun.com/products/javabeans.

[18] D. Carboni, S. Giroux, E. Vargiu, C. Moulin, S. Sanna, A. Soro, G. Paddeu. E-MATE: An Open Architecture to Support Mobility of Users. Databases and Information Systems II-Fifth International Baltic Conference, Baltic DB&IS'2002, Tallinn, Estonia, June 3-6, 2002 Kluwer Academic Publishers. ISBN 1-4020-1038-9.