SEFAGI: SIMPLE ENVIRONMENT FOR ADAPTABLE GRAPHICAL INTERFACES

Generating user interfaces for different kinds of terminals

Tarak Chaari and Frédérique Laforest

Lyon Research Center for Images and Intelligent Information Systems, INSA Lyon, 20 Avenue Albert Einstein, 69621 Villeurbanne Cedex, France

Keywords: user interface adaptation, adaptation to terminal, code generation, XML

Abstract: The SEFAGI project takes place in domains where many different user interfaces are needed in the same application. Instead of manually developing all the required windows, we propose a platform that automatically generates the needed code from high level descriptions of these windows. Code generation is done for standard screens and for small screens on mobile terminals. New windows are automatically taken in charge by an execution layer on the terminal. Data adaptation to the different terminals is also provided. A platform-independent window description language has been defined.

1 INTRODUCTION

In the information systems components, the user interface takes a great importance. In fact, it is the external facet that guarantees the interaction between the user and the application. In many domains, developers have to design and implement a large number of user interfaces for many users and many terminals. For example, in medical applications we distinguish many types of users: doctors, secretaries, nurses, administrators, paramedical practitioners, patients… Each user accesses the same electronic patient records but he/she has different medical skills and different access rights. Moreover, the same application can be accessed by different kinds of terminals (PDA, Desktop PC, mobile phone…).

The literature (Myers Rosson, 1992) proves that user interfaces development requires a high percentage of the entire application development time. This constitutes a bottleneck for application's development life cycle. This awkward problem grows as the user interfaces number grows in the information system.

The SEFAGI project tends to solve this problem by providing a platform that allows:

1. The automatic generation of windows without any human coding,
2. The integration of new windows into applications,
3. The adaptation of windows to terminal capabilities.

For the first point, SEFAGI provides a library of predefined composite visual widgets (called panels) that guarantees the graphical construction of windows. The user associates Web services to panels to define an abstract window description stored in an XML file. SEFAGI automatically generates the executable code from this description. Then the window can be downloaded into the terminal.

For the second point, SEFAGI guarantees an incremental application development by adding new windows on runtime. No need for the application to be entirely rebuilt before providing it to end-users. Moreover, modifications on existing windows can also be done by modifying the abstract window description, re-generating the corresponding code and replacing the window code in each terminal when they connect to the system.

For the third point, code generation ensures adaptation to the terminal characteristics. For the moment, we have defined three main profiles: classical PC screens, very limited mobile terminals (mobile phones) and common mobile terminals (PDAs). Windows code is generated from the abstract window description taking in consideration hardware and software characteristics of the

Chaari T. and Laforest F. (2005).
SEFAGI: SIMPLE ENVIRONMENT FOR ADAPTABLE GRAPHICAL INTERFACES - Generating user interfaces for different kinds of terminals.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 232-237
DOI: 10.5220/0002525702320237

terminal. To ensure this adaptation, we have defined transformation rules to adapt the code and the presented data to different terminals.

2 RELATED WORKS

User interface automatic generation is not a new domain of research. A lot of architectures have already been proposed. These architectures are known as UIDE (User Interfaces Design Environments) (Foley et al, 1991) or UIMS (User Interface Management Systems) (Kasik et al, 1989). However, there are some differences between these two terms in the literature. The first one describes architectures for designing user interfaces using abstract models and eventually generating the corresponding source code. The second one focuses on managing generated code of many user interfaces using abstract script languages. Thus, an UIMS can be seen as a part of an UIDE. Tadeus (Elwert et al, 1995) and Mecano (Puerta, 1996) are examples of UIDE including an UIMS.
But all these approaches require providing a lot of information in complex models and descriptions (Da Silva, 2000). These environments are so heavy that they can not be used by non specialists. Their complexity explains why they are not widely used.

Simpler user interface development environments exist. For example, Borland Jbuilder and IBM Visual Age for Java help the developer by writing the code skeleton. But the major part of the code remains to be hand-coded. That's why we are not satisfied by these products: we need a product ensuring no hand-coding at all.

The Abstract Window Description language that we have written is based on XML. Other projects have used XML to describe user interfaces. UIML (Abrams, 1999) is a complete platform independent language. Harmonia is a web-based user interface generator using UIML descriptions (Boshernitsan, 2001). An UIML description requires so many details for each widget. This makes the description so heavy and cannot be useful for non specialists. These considerations have encouraged us to develop new user interface description language with a higher granularity level.

Adaptation is another research topic close to our work. In (Satyanarayanan, 2001), Satyanarayanan says that adaptation is necessary when there is a considerable disparity between the supplied resource and the requested one. Many other works showed the importance of the adaptability or plasticity (Calvary et al, 2002) of user interfaces to the contexts and the environments of their usage.

Two main directions are defined in adaptation:

- Static adaptation consists in writing as many code versions as terminal types. Most of time, static adaptation has concerned existing applications for standard PCs and adapted them to mobile terminals. (Larson, 2002) and (Benjaminsen et al, 2002) are examples of this approach. In this case, written code is highly optimized for each terminal. But the drawback of such methods is the explosion of code versions due to the growth of terminal types. Manual coding of all the code versions is heavy and time consuming for the developers.
- Dynamic adaptation makes adaptations on the fly: the interface is built and sent to the client on his demand. (Lemlouma et al, 2001) and (Menkhaus, 2002) are projects of web based user interface dynamic adaptations. The inconvenient of such approaches is the latency due to page generation at each query. Another disadvantage of the dynamic adaptation is the efficiency of the adaptation. In fact, hardware and software characteristics of the terminals grow fast and adaptation rules must follow this evolution.

For a perfect adaptation to the terminal characteristics we consider mixing the two methods with the maximum automation of the adaptation process.

3 PANELS AND WINDOWS

3.1 Definition

User interface descriptions are often very heavy because of the huge quantity of details required to specify all parameters of each widget (size, position, color, events…). This is due to the low level description of reusable components (also called widgets). In this work, we have defined a higher level of reusable components: panels. A panel contains a set of grouped widgets that assures a typical functionality. For example, we have defined a panel to scroll image lists. Panels have been defined after parctical user interfaces studies on the medical domain and we think they can answer many domain needs. Nevertheless, new panel descriptions can be added for specific needs. We have defined and implemented six panel types:

- The table panel type ensures capturing and presenting data in a grid
- The text panel type ensures capturing and presenting a multi-line text
- The graphic panel type presents graphs or histograms for 2D values
- The image panel type presents a list of images and shows the selected image

- The video panel type presents a list of videos and plays the selected video
- The command panel type groups buttons that allow to execute actions on panels.

Each panel is associated to Web services that allow to:

- Fill in the panel widgets (output services),
- Update data from end-user captures (input services).

A window is thus defined as a list of panels witch are associated to Web services. More precisely a window is defined as a list of panels. A panel is composed of a list of widgets which are associated to Web services input/output data. When a Web service is associated to a panel type, each input/output data is automatically associated to a panel widget.

3.2 XML Abstract Window Description (AWD)

We have chosen to use an intermediate internal format to store Abstract Window Descriptions. This AWD is platform-independent: an AWD may provide different codes for the different types of terminals. This intermediate format is based on XML: each window is described by an XML document. This intermediate AWD can be reused to create new windows or to update an existing window that requires modifications.

The Abstract Window Description used in our project allows generating documents that are smaller than other description languages that we can find in the literature. Many implementation details are not provided in AWDs as it remains at an abstract level. The interface rendering on the screen is not specified. This is an advantage for new windows construction as it guarantees efficiency (descriptions are rapid and the designer does not get lost in details) and consistency (no risk for widgets to overlap for example). But it can also be seen as a drawback: generated panels are “pre-formatted” and cannot be highly personnalized. But the advantages it provides have largely covered this restriction in the case studies that we have done.

4 LOGICAL ARCHITECTURE OF THE SEFAGI PLATFORM

Our SEFAGI platform is based on three main entities, presented in fig 1Application server

The application server contains four main entities:

- **Data servers** that include database management systems and file management systems. They store the application domain data.
- **Business servers** include procedures called by terminals. We have used the W3C Web services standard to design and develop the business knowledge of the application. We have developed generic services (SQL querying, image adapting…) that facilitate the creation of the business functionalities of the application.
- A **service repository** provides a hierarchical description of all available services in the application
- **Adaptation servers** adapt data according to adaptation rules. All these services communicate with the client using the XMLRPC standard. With every service call, the terminal provides its hardware and software capabilities in CC/PP format.

4.1 Terminals

We have defined three main software components on terminals:

- A **repository** of generated windows coming from the interface server,
- A service invocator which invokes distant application server procedures and gets their returned values.
- An **execution layer** we have specifically designed to manage user interfaces. It contains the necessary graphical components for the execution of the generated windows.

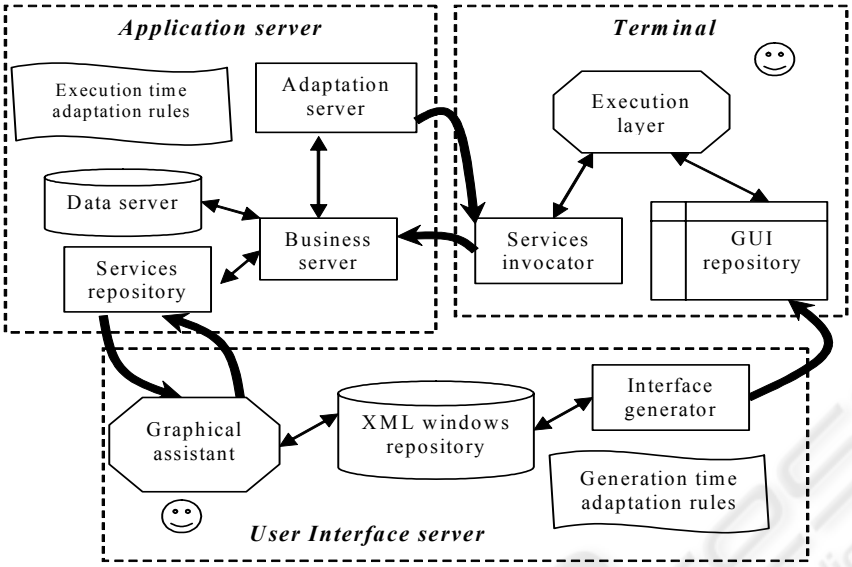

Figure 1: SEFAGI general architecture

5 FROM XML ABSTRACT WINDOWS DESCRIPTIONS TO EXECUTABLE CODE

5.1 Window code Organization

Windows Java code is generated from abstract windows descriptions. To structure this task, we have chosen to write the code according to the MVC standard structure. Each window code is divided into three classes:

- the Model class manages user identification and the dialogue with Web services,
- the View class includes the definition of the graphical components of the user interface their initializations and their setter and getter methods to set or to get data from them,
- the Controller class includes events management and transmission of services calls to the Model class.

5.2 Inter-window data exchange

Navigation between windows during their execution is ensured by navigation menus and by exchanging data between windows. For example, selecting a line in a patient list provides a patient id that can be used in the next window (temperature graphics window and lab results window for example). To do so, we have defined a generic common data exchange structure that can be accessible by all windows. This is implemented as a two-level structure: the first level defines groups that include second level elements (attribute / value pairs). The view class of the window uses this structure to refresh its data.

5.3 Adaptation rules

In this section, we will present the set of rules that we have defined to ensure these adaptations.

5.3.1 Generation-time adaptation rules

Generation-time adaptation rules are executed while writing the executable code of a window from its AWD. We have defined three classes of rules for adaptation at generation time:

1. **Widget transformation rules** These rules consist in a table providing for each abstract widget (from the AWD) its corresponding platform specific class (e.g. an abstract button corresponds to a JButton on a large screen and to menu items on a small screen).
2. **Layout transformation rules** allow defining the position of the widgets inside a panel and the position of the panels inside a window. For example, the layout of the table panel is grid – like on a large screen; it is decomposed into two screens on mobile terminals: the first one serves as an index to access the second one that presents the selected line. Layout transformations provide rules for each panel type and for each terminal type.
3. **Navigation transformation rules** are the third type of rules for code generation. They also depend on the screen size: when an AWD

window is decomposed into many screens (for mobiles), navigation between the implemented windows should be included. Navigation transformations concern two elements: the widget to be selected to make the navigation, and the exchange of information between the implemented windows.

5.3.2 Execution-time adaptation rules

Other adaptations have to be made at execution time. Such adaptations concern data. Data adaptations depend on the terminal capabilities and on network transfer characteristics. We have defined three classes of execution-time adaptation rules:

1. **Content transformation rules** concern data format and terminals capabilities. For example, images are transmitted only if the terminal can display them: data formats are transformed into other ones supported by the target terminal (e.g. JPEG to PNG for images).
2. **Coherence transformation rules** concern Web services output and input. We gather the returned data from a Web service into a single block. We have standardized the data block structure so that any exchanged data is streamed in a standard format. We decorticate this structure at reception. To avoid data loss, transactions have to be maintained between terminals and servers. To do so, we used checkpoints in data transmission.
3. **Transmission transformation rules** concern data transfer between the terminals and the services. They consist in adapting this data to the XMLRPC standard stream.

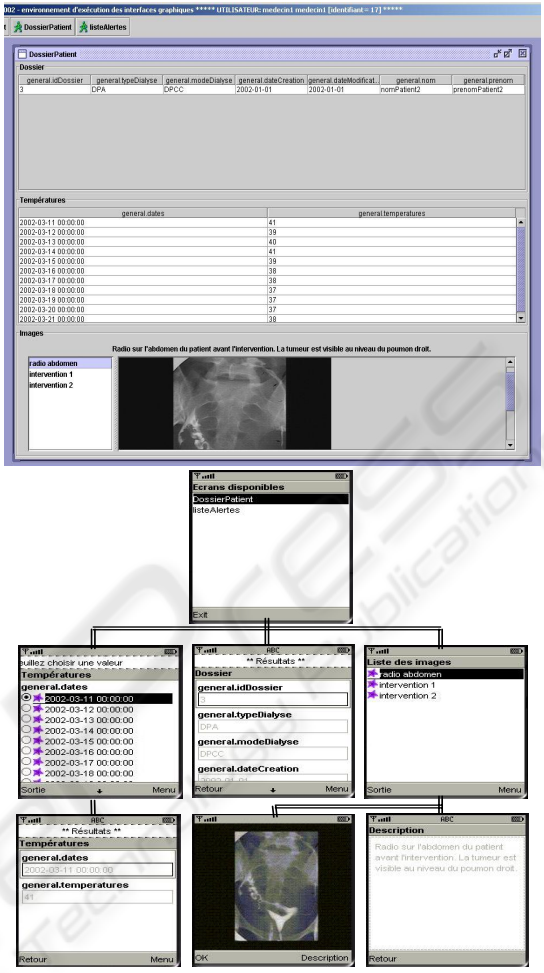

Figure 2: Generated window for standard screens and its equivalent for mobile phones

6 CASE STUDY

We have validated the SEFAGI adaptation process in the scope of the SICOM project (Robardet et al, 2001) witch aims to improve the healthcare quality by managing generic electronic patient records. In fig 2, we present a SICOM window containing three panels: the first is a table panel presenting general information about the patient record. The second is another table panel which presents a temperature data list of the concerned patient. The third is an image panel.

7 CONCLUSIONS AND PERSPECTIVES

In this article, we have presented the SEFAGI project which aims to provide a generic platform for generating adaptive multi-platform user interfaces for information systems. An interface server provides a graphical assistant to describe new windows and a code generator for standard terminals and mobile terminals. We have also developed the necessary software layer for windows management in the terminals to interact with application services.

In this project, user interfaces are adapted at three levels. First, end-users can easily create new screens on the available data through Web services. This can be seen as a static assisted user interface adaptation. Second, the SEFAGI generator ensures the automatic generation of three kinds of codes for standard terminals, limited mobile terminals and

high performance mobile terminals. This can be seen as a static automatic interface adaptation. Third, data content is adapted to terminal capabilities at runtime. This can be seen as a dynamic automatic interface adaptation.

To improve the SEFAGI implementation, we intend to work on a better organization of the AWD documents in the repository. This requires documents indexation and categorization. Another point that could be studied concerns the adaptation services. In the actual version of SEFAGI, adaptation rules are embedded in adaptation services. We are working on the generalization of the adaptation services through external specification of adaptation rules. This will provide much more flexibility to adaptation.

REFERENCES

Abrams M., Phanouriou. C., 1999, "UIML: an XML language for building device independent user interfaces", *XML'99*, Philadelphia.

Benjaminsen S., Djora A., 2002 « JetRek: How organisational identities slowed down speedy requisitions », *BSA medical sociology conference*, september 2002, York. http:// www.sv.ntnu.no/iss/Aksel.Tjora/publications/ Siri-york02-09.pdf

Boshernitsan M., 2001 "Harmonia: A flexible framework for constructing interactive language-based programming tools", Technical Report, University of California, Berkeley

Calvary G., Coutaz J. and Thevenin D., 2001 "Supporting context changes for plastic user interfaces: a process and a mechanism", *Proceedings of HCI-IHM 2001, BCS conference series*, Springer Publ., pp. 349-363

Da Silva P. P.,2000. User Interface Declarative Models and Development Environments: A Survey. In P. Palanque and F. Paterno, editors, Proceedings of DSV-IS'2000.

Elwert T. and Schlungbaum E., 1995, Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In: P. Palanque, R. Bastide (eds.): Designing, Specification, and Verification of Interactive Systems. Wien: Springer, 1995, 193-208.

Foley J., Kim W., Kovacevic S. and Murray K., 1991, UIDE - An Intelligent User Interface Design Environment. Architectures for Intelligent User Interfaces: Elements and Prototypes, Addison-Wesley, Reading MA, pp.339-384.

Kasik D.J., Lund M.A., and Ramsey H.W.. Reflections on using a UIMS for complex applications. IEEE Software, 6(1):54--61, January 1989., M.A. Lund, and H.W. Ramsey. Reflections on using a UIMS for complex applications. IEEE Software, 6(1):54--61, January 1989.

Larson E. D., 2002 Wireless Java Application Saves Women's Cancer Center $500,000 per Year, J2ME case studies, september 2002. http://wireless.java.sun.com/midp/casestudies/wcc/.

Lemlouma T., Layaïda N., 2001 « A Framework for Media Resource Manipulation in an Adaptation and Negotiation Architecture », OPERA project, under submission, INRIA Rhône-Alpes, august 2001

Menkhaus , 2002 « Adaptive User Interface Generation in a Mobile Computing Environment », PhD Thesis, Salzburg University, 2002.

Myers B. A., Rosson M. B., 1992, Survey on User Interface Programming. In: P. Bauersfeld, J. Bennett, G. Lynch (eds.): Striking a Balance. Proceedings CHI'92 (Monterey, May 1992), New York: ACM Press, 195-202.

Puerta A., 1996. The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Presspp. 19-36.

Robardet C., Verdier C., Flory A., "Une nouvelle approche pour la classification de groupes homogènes de patients : application à l'hospitalisation à domicile", AIM'2001, Journée "Télémédecine et Santé", Paris, 2001

Satyanarayanan M., 2001 Pervasive Computing: Vision and challenge, IEEE Communications.