Preserving Rich User Interface State in Web Applications across Various Platforms

Fabio Paternò, Carmen Santoro, and Antonio Scorcia

ISTI-CNR, Via G. Moruzzi, 1
56124 Pisa, Italy
{Fabio.Paterno,Carmen.Santoro,Antonio.Scorcia}@isti.cnr.it

Abstract. This paper aims to provide thorough discussion of the aspects that compose the state of a Web application user interface, and show how it can be preserved across multiple devices with different interaction resources when the user interface dynamically migrates. The approach proposed exploits a migration server along with logical user interface descriptions.

1 Introduction

The Web is the most common user interface. There are currently hundreds of millions of Web sites and it is increasingly rare to find someone who has never used a Web application. In the meantime, Web technologies have evolved in many directions: the Web 2.0, Rich Interactive Applications, Multimodal Interfaces, … Another important technological trend is the increasing availability in the mass market of many types of interactive devices, in particular mobile devices, which has enabled the possibility of ubiquitous applications.

In such environments migratory interfaces are particularly interesting. They allow users to move about freely, change device and still continue the interaction from the point where they left off. Thus, in order to obtain usable migration two aspects are important: preserving the user interface state across multiple devices and adaptation to the changing interaction resources. In this paper, we focus on the former aspect (state preservation) in the context of Web applications, identify a broad set of relevant aspects, and show how they can be addressed by our migration environment.

After discussing related work, we first identify seven relevant aspects that can be used to define the state of Web User Interfaces, including Web 2.0 applications with Ajax scripts. Then, we introduce our architecture for supporting Web application migration, and explain how it has been extended in order to be able to support the various aspects that have been deemed useful for defining the user interface state.

2 Related Work

ICrafter [1] is a solution to generate adaptive interfaces for accessing services in interactive spaces. It generates interfaces that adapt to different devices starting with XML-based descriptions of the service that must be supported. However, ICrafter is

P. Forbrig and F. Paternò (Eds.): HCSE/TAMODIA 2008, LNCS 5247, pp. 255–262, 2008.

limited to creating support for controlling interactive workspaces by generating user interfaces for services obtained by dynamic composition of elementary ones and does not provide support for migration and, consequently, continuity of task performance across different devices.

Aura [2] provides support for migration but the solution adopted has a different granularity. In Aura for each possible application service various applications are available and the choice of the application depends on the interaction resources available. Thus, for example for word processing, if a desktop is available then an application such as MS-Word can be activated, whereas in the case of a mobile platform a lighter editing application is used. Thus, Aura aims to provide a similar support but this is obtained mainly by changing the application depending on the resources available in the device in question, while we generate interfaces of the same application that adapt to the interaction resources available.

Bharat and Cardelli [3] addressed the migration of entire applications (which is problematic with limited-resource devices and different CPU architectures or operating systems) while we focus on the migration of the user interface part of a software application. Kozuch and Satyanarayanan [4] identified a solution for migration based on the encapsulation of all volatile execution state of a virtual machine. However, their solution mainly supports migration of applications among desktop or laptop systems by making copy of the application with the current state in a virtual machine and then copy the virtual machine in the target device. This solution does not address the support of different interaction platforms supporting different interaction resources and modalities, with the consequent ability to adapt to them. Chung and Dewan [5] proposed a specific solution for migration of applications shared among several users. When migration is triggered the environment starts a fresh copy of the application process in the target system, and replays the saved sequence of input events to the copy in order to ensure that the process will get the state where it left off. This solution does not consider migration across platform supporting different interaction resources and modalities and consequently does not support run-time generation of a new version of the user interface for a different platform. We follow a different approach: we assume that the desktop version of an application exists, without posing any restriction on the method or tool used for its development. Then, during the user session, we dynamically generate the version for the platform at hand exploiting model-based techniques.

We introduced some preliminary ideas on how to obtain automatic generation of migratory Web interfaces in [6]. In this paper we are able to present a solution supporting migration of rich information state, including application with Ajax scripts.

3 The Many Aspects of the Web Interface State

In our study we have identified at least eight aspects that can be relevant for defining the state of Web user interfaces and that can have an impact on the overall user experience. The first element is associated with the *user input*. People make selections, enter text and modify the state of various input controls during a session, and such modifications should not be lost when moving to a new device if we want to maintain

continuity. An associated element is *client-side variables* associated with small functionalities (e.g. Javascript variables).

Another component that can be dynamically modified is the *content* of a Web application. While this can be easily managed with dynamic Web sites using PhP, JSP and similar languages because whenever a new request is performed then a different page is uploaded, with Ajax scripts this aspect becomes more problematic. Indeed, in this case the content of the page can vary without requiring the loading of a new page. Thus, it becomes more complex to detect what is actually composing the currently displayed page.

Cookies are more and more used and they allow an application to provide small pieces of information to the client in such a way that whenever the client accesses the application, then the client identifiers are inserted in the HTTP protocol. It is important that if and when a user changes device, then the current application preserves the same cookies in order to be recognised by the application server. A related technique is the *session*: it is a server-side mechanism, which stores information related to the user session, which is in turn associated with a specific identifier.

Another important aspect is the *history* of user accesses, which is maintained by the Web browser and drives the behaviour of the frequently used browser back button. Since the user is still the same, even if she has changed device, then she would appreciate still being able to easily return to recently accessed pages, even if through a different device. It is clear that the pages accessed through the new device may be adapted to the currently available interaction resources. In some cases (e.g. migration from desktop to mobile), it may even happen that the original desktop page is split into multiple mobile pages, thus accessing all its content may require further navigation.

Bookmarks are another interesting aspect that can be considered part of the user interface state. Users often use them to quickly find and access favourite pages. In migration, the devices change but not the user, who still has the same interests and may appreciate the possibility to find in the current bookmarks including the pages that were bookmarked in the previous device. Another element that has similar characteristics is the browser home page: in some cases users may be interested to migrate it to different platforms as well.

A last element that can be considered part of the state is the *query string* included in a URL after the "?" symbol. It is usually used to specify parameters for a dynamic site, which define some data that are presented in the associated page. By modifying the query string we will access the same Web site but since the parameters vary, then the corresponding page varies in terms of content.

4 An Architecture for Migratory Interfaces

Our architecture for migratory interfaces is based on a migration/proxy server. The advantage of this choice with respect to installing the necessary functionalities on the application servers is that we can concentrate them in a single server without the need for replication in the servers supporting the various possible applications. Indeed, we want to apply the migration support to a wide set of applications, and we do not want to force the application developers to use any specific authoring environment or to

apply specific annotations to ease the migration process. In general, we consider that a wide set of Web applications for desktop systems already exist and they can be the target for a migration infrastructure.

Our migration infrastructure exploits logical descriptions of user interfaces. In such description there is an abstract level, which is platform independent and a concrete level, which refines the previous one by adding concrete elements and attributes. The environment has a service-oriented architecture based on four main functionality:

- *Reverse Engineering*, takes the existing Web pages for desktop systems and builds the corresponding logical descriptions;
- *Semantic Redesign*, this module is in charge to perform the adaptation to the target device. For this purpose it takes the abstract elements identified by the reverse engineering module and maps then into concrete elements more suitable for the target device. It also splits the source presentations into multiple presentation if they are too expensive for the interaction resources of such target device.
- *State Mapper*, once a concrete description for the target device has been obtained then the state resulting from the user interactions in the source use interface is associated with it. The abstract elements are used to identify which concrete element in the source interface correspond to the concrete elements in the target interface.
- *User Interface Generator*, this module generates the user interface in some implementation language. One concrete description for a given platform, for example a graphical form-based interface, can be associated with various implementations languages (such as Java, XHTML, C#). The generated user interface is then uploaded on the target device.

In addition, when the host acting as a migration/proxy server passes the Web pages to the client, it adds Ajax scripts, which are used to communicate to the server the interface state accessed through DOM when the migration is triggered.

All the devices that are involved in the migration should run a migration client, which is used for two purposes: in the device discovery phase, when the devices interested in migration are identified and provide information about themselves, and to trigger migration. Users can trigger migration through an interface separated from the application interface, which shows the list of available devices from which the user can select the target one.

5 Migration Preserving Rich Information State

In this section we discuss how our migration infrastructure has been modified in order to support rich state information, which includes various aspects additional to the changes due to the user input in some controls.

The use of Ajax scripts implies that the content of a Web page changes dynamically without loading an entire new page. This means that when the migration server starts the reverse engineering process to build the logical description of the current page it should work on the page version currently loaded in the client browser and not that in the application server, because they may be different.

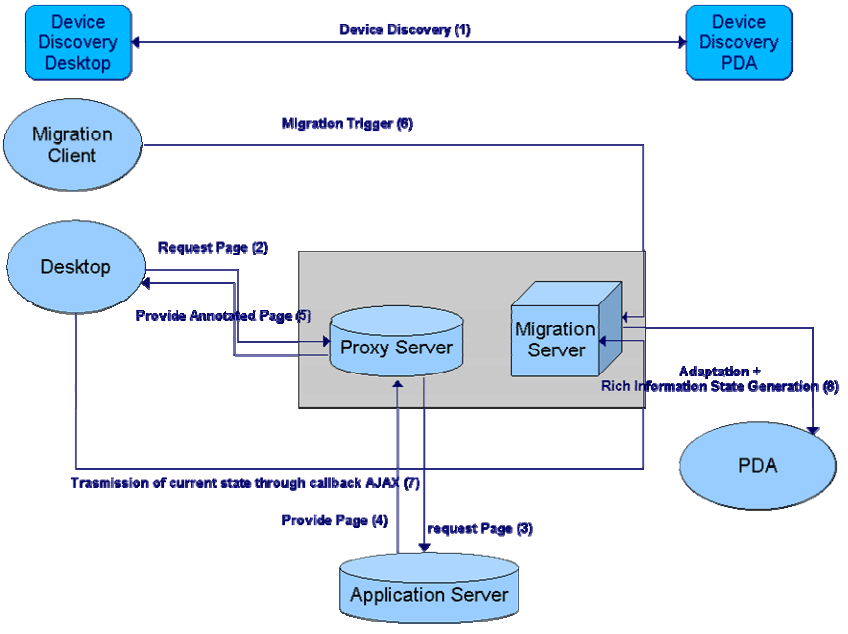

Fig. 1. The Migration Architecture Preserving Rich Information State

Figure 1 describes the current architecture of the environment. After the device discovery phase (1), there is an access to the Web page through the migration/proxy server (2), which downloads the page from the application server (3, 4) and annotates it by adding an Ajax script (5) whose goal is to collect the rich information state of the client device and send its description to the migration/proxy server. Among the functionalities that are included in such an AJAX script, we first mention a functionality performing a continuous monitoring (polling), whose objective is detecting whether or not a migration has been triggered by the migration client. In addition, a piece of invisible script code (since it uses a IFRAME element) is added to the page downloaded by the server with the aims of getting the cookies and the current history, storing their content, and continuously updating such information to maintain the state consistent in an automatic and transparent way. This is managed, again, by AJAX scripts: indeed, both the list of addresses (URLs) representing the history, and the set of cookies generated during the user session can be rewritten in the IFRAME and can be sent to the migration server without the user's awareness. While the user navigates through the source device browser without sending any migration request, the loaded Web page sends an invisible HTTP request to the Migration Server, which is suspended until a migration request is activated through the migration client.

When a migration trigger is generated (6), the AJAX callback function is automatically activated and thus sends (7) the DOM file (containing the state of the current page), together with the portion of the invisible content (IFRAME) previously mentioned. Then, the migration server will first associate the content state of

the page on the source device accessed through the DOM to the concrete description of the version for the target device, and will add a new portion of invisible content (IFRAME) containing the AJAX functions to re-create the rich state on the target client device (8) in the corresponding generated implementation. Such rich state information will be obtained through the functions able to read the cookies and re-create them in the target device, as well as functions that will re-load within the IFRAME the addresses connected with the chronology and create the history accordingly on the target device.

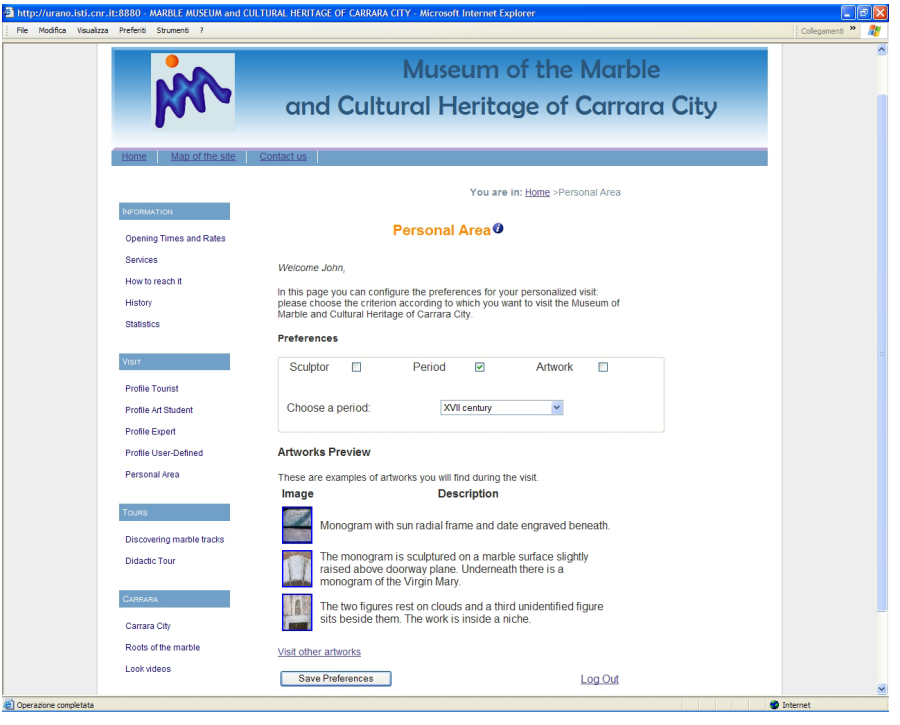

Fig. 2. The Example Ajax Application

6 An Example Application

In this section we describe an example to show how our approach concretely works. The scenario considered regards a user who starts interacting with a digital museum application providing information about artworks. At some point the user accesses a page, which allows him to specify preferences regarding artworks in the museum. While the various preference options are specified, the application using an Ajax script provides a preview of artworks that satisfy them (see Figure 2) without requiring access to a new page. Then, the user asks for migration to a mobile device in order to continue by means of indicating preferences on the move.

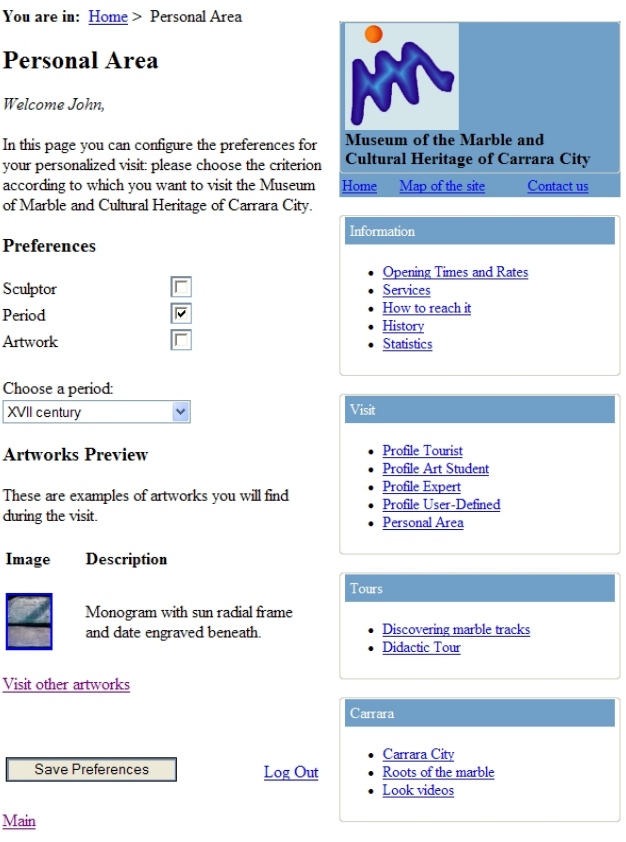

Fig. 3. The Ajax Application after Migration to a Mobile Device

The left side in Figure 3 shows the user interface activated in the mobile device immediately after migration. Since the cookies are also migrated, the application still recognises the user (John). The desktop page is too large for the mobile device and, consequently, the adaptation component of the migration platform splits it into two pages for the mobile device after analysing its logical structure obtained by the reverse engineering functionality. One mobile page is dedicated to the interactive form enriched with the Ajax script for the preview, which is the page immediately uploaded because the user was interacting with it when migration was triggered. The other the page (Figure 3 right) is the main page including the navigational structure of the application. In the uploaded page some other adaptations take place: the check-box is lined up vertically because it would not fit horizontally and the preview of only one artwork is shown at a given time.

7 Conclusions and Future Work

In this paper we have discussed the many aspects that characterise the state of a Web application from the user point of view. Then, we have presented a solution for

migration of Web applications, including those with Ajax scripts, able to preserve this rich state information and have shown an example application.

Future work will be dedicated to testing the usability of the proposed solution and its integration to a more general migration platform.

References

1. Ponnekanti, S.R., Lee, B., Fox, A., Hanrahan, P., Winograd, T.: ICrafter: A service framework for ubiquitous computing environments. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) UbiComp 2001. LNCS, vol. 2201, pp. 56–75. Springer, Heidelberg (2001)
2. Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P.: Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing 21(2), 22–31 (2002)
3. Bharat, K.A., Cardelli., L.: Migratory Applications. In: Proceedings of User Inteface Software and Technology (UIST 1995), Pittsburgh, PA, USA, November 15-17, 1995, pp. 133–142 (1995)
4. Kozuch, M., Satyanarayanan, M.: Internet Suspend/Resume. In: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002). IEEE Press, Los Alamitos (2002)
5. Chung, G., Dewan, P.: A mechanism for Supporting Client Migration in a Shared Window System. In: Proceedings UIST 1996, pp. 11–20. ACM Press, New York (1996)
6. Bandelloni, R., Mori, G., Paternò, F.: Dynamic Generation of Migratory Interfaces. In: Proceedings Mobile HCI 2005, pp. 83–90. ACM Press, Salzburg (2005)