Migrating Web Applications through Declarative Models

Giulio Mori, Fabio Paternò, Carmen Santoro, Antonio Scorcia

ISTI-CNR
Human Interfaces in Information Systems Laboratory
Via Moruzzi 1
5124 Pisa, Italy
http://www.isti.cnr.it/ResearchUnits/Labs/hiis-lab/

1. Introduction

One important aspect of pervasive environments is to provide users with the possibility to freely move about and continue the interaction with the available services through a variety of interactive devices (i.e. cell phones, PDAs, desktop computers, digital television sets, intelligent watches, and so on). However, continuous task performance implies that applications be able to follow users and adapt to the changing context of use.

In this position paper, we provide an overview of our solution for supporting migration of Web applications among different types of devices, including the digital TV. This opens up the possibility of creating environments supporting users freely moving in the home and outside, which can be interesting for many application domains, including business and entertainment applications.

The motivation for migration is that the world is becoming a multi-device experience and one big potential source of frustration is that people cannot continue to perform their tasks when they move about and change their interaction device. There are many applications that can benefit from migratory interfaces. In general, services that require time to be completed (such as games, booking reservations) or services that have some rigid deadline and thus need to be completed wherever the user is (eg: online auctions).

Our solution is able to detect any user interaction performed at the client level. Thus, our migration environment can get the state resulting from the different user interactions and associate it to a new page version that is activated on the migration target device. Solutions based on maintaining the application state on the server side have been discarded because they are not able to detect several user interactions that can modify the interface state. In particular, we present how the solution proposed has been encapsulated in a service-oriented architecture and supports interfaces for different platforms (digital TV, mobile device, desktop) and modalities (graphical, vocal, and their combination). Users can conduct their regular access to the Web application and then ask for a migration to any device that has already been discovered by the migration environment. Migration among devices supporting different interaction modalities has been made possible thanks to the use of a logical language for user interface description that is independent of the modalities involved, and a number of associated transformations that incorporate design rules and take into account the specific aspects of the target platforms.

Our system, in particular, proposes a concrete software architecture that is able to support migration of user interfaces, associated with Web applications hosted by different application servers, among automatically discovered devices, therefore resulting in a concrete solution to the issue of migration of Web applications for different modalities.

In the position paper, we first introduce our approach, then we provide more detail on the device independent languages used and the reverse and forward transformation supported in order to adapt user interfaces to the device at hand. Lastly, we draw some conclusions along indications for future work.

2. The Approach

In order to support migration involving many types of devices including the Digital TV, we have considered previous experiences in the area of migratory interfaces (Bandelloni *et al.*, 2004) and extended them in several aspects in order to be able to support migration to the digital TV. In practise, our environment supports a number of reverse and forward transformations that are able to transform existing Web desktop applications for various interaction platforms and support task continuity.

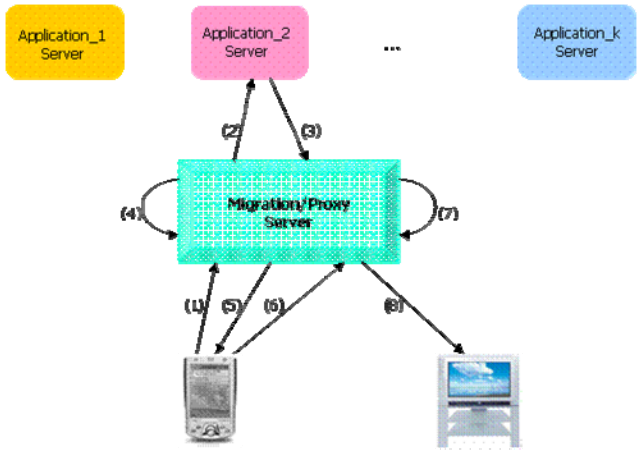

Figure 1: The architecture of the migration approach

As Figure 1 shows, the architecture is based on a proxy/migration server. Users who subscribe to the migration service can then access Web applications through such server (see arrows (1,2,3) in Figure 1). If the interaction platform used is different from the desktop, the server transforms it (arrow (4)) by building the corresponding abstract description and using it as a starting point for geneating the implementation adapted for the device accessing it, which is the delivered (arrow (5)). In addition to interface adaptation, the environment also supports task continuity. To this end, when a request of migration to another device is triggered the environment detects the state of the application modified by the user input (elements selected, data entered, …) and identifies the last element accessed in the source device. The information about the user interaction state is collected into a string formatted following a XML-based syntax and submitted to the server together with the IP of the target device in the migration (arrow (6)). Then, a version of the interface for the target device is generated, the state detected in the source device is associated with the target device version so that the selections performed and the data entered are not lost (arrow (7)). Lastly, the user interface of the target device is activated at the point supporting the last basic task performed in the initial device (arrow (8)).

Migration can be triggered either by the user (through either a graphical interface or scanning RFID tags associated to the target device) or by the environment when some specific event (such as battery or connectivity very low) is detected or in a mixed solution in which the environment suggests possible migrations based on the devices available at a short distance and the user decides whether to accept them or not. In the process of creating an interface version suitable for a platform different from the desktop we use a semantic redesign module, which transforms the logical description of the desktop version into the logical description for the new platform, which is then used for generating the new implementation version.
In order to support transformations from an abstract platform-independent language to an implementation (for example, for the digital TV), we use an intermediate specification, the concrete description, which is a platform-dependent refinement of the abstract interface. These two types of user interface logical languages (abstract and concrete) consider two types of elements: basic

elements (output-only and interaction), and composition elements, aiming to indicate how to combine together two or more elements.
Our approach also supports interoperability between various implementation languages. Thus, a Web application can be transformed into a Java application for the Digital TV. In this case the generation of the user interface implementation involves the generation of a file in a Java version for digital TVs representing an Xlet, which is an application that is immediately compiled and can be interpreted and executed by the interactive digital TV decoders.

3. User Interface Logical Languages

In the research community in model-based design of user interfaces there is a general consensus on what the useful logical descriptions are (Calvary and others, 2003; Paternò, 1999):

- The task and object level, which reflects the user view of the interactive system in terms of logical activities and objects manipulated to accomplish them;
- The abstract user interface, which provides a modality independent description of the user interface;
- The concrete user interface, which provides a modality dependent but implementation language independent description of the user interface;
- The final implementation, in an implementation language for user interfaces.

Thus, for example we can consider the task "switch the light on": this implies the need for a selection object at the abstract level, which indicates nothing regarding the platform and modality in which the selection will be performed (it could be through a switch or a vocal command or a graphical interaction). When we move to the concrete description, we have to assume a specific platform, for example the graphical PDA, and indicate a specific modality-dependent interaction technique to support the interaction in question (for example, selection could be through a radio-button or a drop-down menu), but nothing is indicated in terms of a specific implementation language. When we choose an implementation language we are ready to make the last transformation from the concrete description into the syntax of a specific user interface implementation language.

In addition, at the abstract level we also describe how to compose basic elements through some operators which have been defined taking into account the type of communication effects that designers aim to achieve when they create a presentation (Mullet and Sano, 1995). They are: *Grouping*, indicating a set of interface elements logically connected to each other; *Relation*, highlighting a one-to-many relation among some elements, one element has some effects on a set of elements; *Ordering*, some kind of ordering among a set of elements can be highlighted; *Hierarchy*, different levels of importance can be defined among a set of elements. Therefore, an abstract user interface is composed of a number of presentations and connections among them. While each presentation defines a set of interaction techniques perceivable by the user at a given time, the connections define the dynamic behaviour of the user interface, by indicating what interactions trigger a change of presentation and what the next presentation is.

The concrete level is a refinement of the abstract interface: depending on the type of platform considered there are different ways to render the various interactors and composition operators of the abstract user interface. For example, in the case of a graphical desktop platform a navigator can be implemented either through a textlink, or an imagelink or a simple button, a single choice object can be implemented using either a radio button or a list box or a drop-down list and the same holds for the operators. We have defined a set of XML-based languages composing TERESA XML, which also has an associated authoring environment (http://giove.isti.cnr.it/teresa.html), supporting

all the logical levels indicated, thus providing descriptions that can be easily exported/imported in different tools other than those used in our migration infrastructure.

4. Reverse Engineering

Reverse engineering techniques aim to obtain transformations able to analyse implementations and derive the corresponding logical descriptions. Thus, they can be a useful step towards obtaining new versions of an implementation more suitable for different platforms.
Solutions based on syntactical transcoders (for example from HTML to WML) usually provide results with poor usability because they tend to fit the same design to platforms with substantial differences in terms of interaction resources. One possible solution to this problem is to develop reverse engineering techniques able to take the user interface of existing applications and then build the corresponding logical descriptions that can be manipulated in order to obtain user interfaces for different platforms that share the original communication goal, but are implemented taking into account the interaction resources available in the target platforms.
The reverse tool can reverse both single XHTML pages and whole Web sites. A Web site is reversed considering one page at a time and reversing it into a concrete presentation. When a single page is reversed into a presentation, its elements are reversed into different types of concrete interactors and combination of them. The reversing algorithm recursively analyses the DOM tree of the X/HTML page starting with the body element and going in depth. For each tag that can be directly mapped onto a concrete element, a specific function analyses the corresponding node and extracts information to generate the proper interactor or composition operator. In the event that a CSS file is associated to the analysed page, for each tag that could be affected by a style definition (such as background colour, text style, text font) the tool checks possible property definitions in the CSS file and retrieves such information to make a complete description of the corresponding concrete interactor. Thus, the reverse engineering module of our migration environment takes a desktop Web page and recursively analyses its DOM tree and then can provide any of the three levels of logical descriptions supported (concrete, abstract and task level), as needed. More detail on our reverse engineering solution is available in (Bandelloni, Paternò, Santoro, 2007).

5. Semantic Redesign for Dynamic Page Generation

In order to automatically redesign a desktop presentation for a different device we need to consider semantic information and the limits of the available resources in the target device (Mori and Paternò, 2005). To this end, we have a semantic redesign module able to transform the logical description of an interactive application for one platform (obtained through the reverse engineering module) into the logical description for another platform taking into account the interaction resources available. The underlying algorithm tries to maintain interactors that are composed through some operator at the conceptual level in the source description also in the same presentation of the target description, changing the corresponding concrete elements in such a way to take into account the interaction resources available but still preserving the communication goals of the designer. In the transformation process we also take into account the cost in terms of interaction resources of the elements considered. For example, we have defined for each mobile device class identified (large, medium or small) a maximum acceptable overall cost in terms of the interaction resources utilizable in a single presentation. Examples of elements that determine the cost of interactors are the font size (in pixels) and number of characters in a text, and image size (in pixels), if present. Example of the costs associated with composition operators is the minimum additional space (in pixels) needed to contain all their interactors in a readable layout.

If we consider the case of semantic redesign of desktop presentations to the limited resources of mobile devices, it must split one source presentation into a number of different presentations for the mobile devices. To avoid division of large pages into small ones which are not meaningful, this transformation considers both abstract and concrete descriptions. The abstract description (based

on semantic classification of interactors and composition operators) is important because it identifies the original communication goals of the designer that should be preserved in the newly created mobile presentations. Concrete descriptions are important as well because with their information (such as pixel-based dimensions and the indication of the implementation techniques) they allow for more precisely assessing how many interactors can be inserted into a newly created mobile presentation. For example, a single selection can be represented as a list box in a desktop presentation, but this (depending also on the number of choices) may not be suitable for a PDA presentation and so it has to be transformed into a pull down menu. Dividing presentations also requires a change in the navigation structure, with the consequent need for additional navigator interactors.

6. Conclusions and Future Work

We have presented an environment able to support migration of user interfaces for various platforms including digital TV, mobile devices and desktop systems. Since our architecture is based on logical descriptions independent of the specific interaction devices and corresponding implementation languages, it has a wide spectrum of application. So, it can be applied to any device and modality as long as there are transformations able to take the logical descriptions and generate the implementation in an implementation language supported by the device at hand. Our environment already supports automatic generation of applications in XHTML, XHTML Mobile Profile, VoiceXML, X+V, and Java for Digital TV. A video showing a demo of such environment can be accessed at http://giove.isti.cnr.it/MIgration/videos/Mig-Pda-Tv-noaudio.avi

This solution can be useful for supporting various types of interactive services able to follow and support the user moving through different devices in the home and outside for common tasks such as games, shopping, bids for auction on line, making reservations.

References

Bandelloni, R., Berti, S., Paternò, F. (2004). Mixed-Initiative, Trans-Modal Interface Migration. In Proceedings of *Mobile HCI'04*, pp 216-227, LNCS 3160. Springer-Verlag, 216-227.

R.Bandelloni, F. Paternò, C. Santoro, Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments, Proceedings EIS'07, Salamanca, LNCS Springer Verlag, March 2007.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers. Vol. 15, No. 3, June 2003, pp. 289-308.

G.Mori, F.Paternò, Automatic semantic platform-dependent redesign, Proceedings Smart Objects and Ambient Intelligence 2005, pp.177-182, Grenoble, October 2005.

Mullet, K. & Sano, D. (1995), *Designing Visual Interfaces*. Prentice Hall.

Paternò, F. (1999). *Model-based Design and Evaluation of Interactive Applications*. Springer Verlag, ISBN 1-85233-155-0.