Migrating the User Interface while Moving in the Home and Outside

G.Mori, F.Paternò, S.Sansone, C.Santoro

ISTI-CNR, Via G.Moruzzi 1, 56124 Pisa
{g.mori, f.paterno, s.sansone, c.santoro}@isti.cnr.it

Abstract.

Our life is becoming a multi-device experience. In order to improve such experience it is important to allow users to freely move, change device, and still be able to continue to perform their tasks without having to start from scratch at each device change. We present a solution able to support migration among digital TVs and mobile devices, thus useful for users freely moving in the home and outside. The proposed environment is able to understand what tasks the users are performing and generate dynamically user interfaces for the target device, with the state updated to the point in which it was left off in the previous device and their features adapted to the different interaction resources of the new device.

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 practise, it is sufficient that only the part of an application that is interacting with the user migrates to different devices.
In this paper, we present a solution for supporting migration of Web application interfaces among different types of devices that overcomes the limitations of previous work [1] and is able to support migration involving the digital TV. Our solution is able to detect any user interaction performed at the client level, get the state resulting from the different user interactions, and associate it to a new user interface version that is activated in 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 with different platforms (digital TV, mobile, desktop) and modalities (graphical, vocal, and their combination). Users can conduct their regular access to the application and then ask for a migration to any device that has already been discovered by the migration server. 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.

General description of the migration environment

In order to support migration involving Digital TV, we have extended a previously developed infrastructure for migratory interfaces [1]. In practise, the environment supports a number of reverse and forward transformations that allow taking an existing Web PC desktop implementation of an application, building the corresponding abstract description and using it as a starting point for creating the logical description and the implementation adapted for the device accessing it. 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 migration infrastructure 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. Then, a version of the interface for the target device is generated, the state detected in the source device version is associated with the target device version so that the selection performed and the data entered are not lost. Lastly, the user interface version for the target device is activated at the point supporting the last basic task performed in the source version.

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 very low) is detected.

In order to perform such transformations the logical descriptions play a key role [3]. We use two levels of user interface logical descriptions: the concrete, which is dependent of the type of platform, and the abstract, which describes interfaces in a completely platform independent way. In practise, the concrete level acts as an intermediate level between the implementation and the abstract, semantic-oriented description. Such logical descriptions are used when deciding how to associate the state of the source version to the target version. In this case, our migration infrastructure looks for the abstract elements corresponding to the implementation elements modified in the source version, identifies what elements in the target version are used for their implementation, and then associates the entered values to such elements. The logical descriptions are also used to identify the point in which the target interface is activated. In particular, the environment identifies the last input entered and the corresponding abstract element and then looks for its implementation in the target version. The presentation including such element will be activated in the target device because it is assumed that the users want to continue from the point in which they left off in the source device.

In order to support transformations from a platform-independent language to an implementation (for example, for the digital TV), we use the intermediate language, the concrete description, which is a refinement of the abstract interface but platform-

dependent. Three types of elements are considered: output-only, interaction and the composition elements (indicating how to put the other elements together). Thus, for example, an abstract navigator element (which indicates the need for an element allowing users to move to another point of the application) can be supported through textual or graphical links or buttons in graphical interfaces.

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, which is an Xlet, an application that is immediately compiled and can be interpreted and executed by the interactive TV decoders. Xlets bear a strong resemblance with common Java applets, with the difference that, instead of the Web browser (which executes the applets) there is the MHP layer of the digital receiver (Set-Top-Box) which interprets them.

User Interfaces for the Digital TV

As already stated, herein we focus on how to support migration among devices, such as the digital TV, which can be easily available for the users. In the case of the digital TV, we analysed the kind of issues that such platform raise in comparison with traditional desktop systems. Indeed, this platform is similar to a graphical desktop but we have to take into account that in this case users have no mouse or keyword to interact with it but just a TV controller. Usability is a fundamental consideration for interactive services delivered through technologies, such as the digital TV, which can be accessed by people with any background, often with limited computer skills. Since this is a new area there is still a lack of standards, but some guidelines have started to emerge [2] based on a number of user tests. If we analyse the main characteristics of digital TV interfaces, we can find that they involve many aspects. Therefore, as for the differences between the concrete user interface for the digital TV and for the graphical desktop, the work led to some interesting results. One of these was deciding to use a specific kind of font –Tiresias– due to its highly suitability to be visualised on TV displays. In addition, in order to guarantee the best readability of the text, quite high font dimensions were selected (range between 24-36pt), avoiding the smaller ones, which do not guarantee sufficient readability.

As for the generation of the user interface implementation, it involves the generation of a file in a Java version for digital TVs representing an Xlet. Such Xlets bear a strong resemblance with common java applets, with the difference that, instead of the Web browser (which executes the applets) there is the MHP (http://www.mhp.org/) layer of the digital receiver (Set-Top-Box) which interprets them. Once we have the xlet, in real settings (which usually means if a digital phone line is available), the file generated with our transformations is supposed to be downloaded on the Set-Top-Box. If such a line is not available, an emulator can be used to execute the interactive Xlet. For our examples, we used the XletView emulator (http://xletview.sourceforge.net), we will show an example of its use in next sections.

The Xlets are similar to the applets, However, there is a big difference regarding the user interface part: Xlets do not include the AWT package, which contains several widgets for Java programs, such as radio button, check box, button, etc. Thus, we also developed a library that provides such techniques and simplify the target for the user interface generation in the digital TV platform. In the library there is a class that provides the methods to include the widgets in the application without having to specify all the details every time. Then, there is a class for each interactor implementing its appearance and behaviour. The implementation of the appearance is not trivial because the basic Xlets just provide primitives for drawing rectangles and showing images. The semantics of the interactors' implementation is managed through specific event handlers that allow them to update the state and then their appearance accordingly.

An Example Application

The example application considered in this case is a shopping application. This application allows users to edit and send a shopping list while freely moving. For example, users returning home from work can start to prepare the shopping list through a mobile device while they are on the bus or train. Thus, they can access the application (see Figure 1.a) and start to select what they want to buy and the associated amounts (see Figure 1.b).

Figure 1. The user interface for the mobile device

When they arrive home they may realise that something is still missing and thus decide to migrate to the digital TV so that they can continue the editing of the shopping list through a larger screen. After migration, they can still find the shop elements that were specified before (see Figure 2) and edit them or add new ones until lastly they send the request.

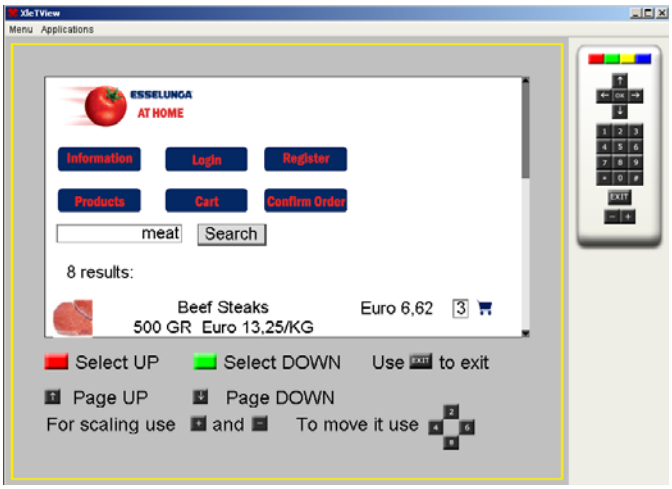

Figure 2. The Digital-TV application interface after migration.

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

We plan to investigate the application of this solution for providing continuous and adaptive support for the elderly.

References

[1] Bandelloni, R., Mori, G., Paternò, F., Dynamic Generation of Migratory Interfaces, Proceedings Mobile HCI 2005, ACM Press, pp.83-90, Salzburg, September 2005.

[2] Fondazione Ugo Bordoni – "Raccomandazioni per le interfacce dei servizi interattivi della televisione digitale", http://www.fub.it/ambientedigitale/repository/Generale/Raccomandazioni.pdf

[3] Mori, G. Paternò, F. Santoro C., Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, August 2004, Vol.30, N.8, pp.507-520, IEEE Press.