Migratory User Interfaces Able to Adapt to Various Interaction Platforms

Renata Bandelloni, Fabio Paternò

ISTI-CNR,Via G.Moruzzi 1, 56124 Pisa, Italy.
{renata.bandelloni , fabio.paterno}@isti.cnr.it

Abstract. The goal of this work is the design of an environment for supporting run time migration of Web interfaces among different platforms. This allows users interacting with a Web application to change device and continue their interaction from the same point. The migration takes into account the runtime state of the interactive application and the different features of the devices involved. We consider Web interfaces developed through a multiple-level approach using: the definition of the tasks to support, the abstract description of the user interface, and the actual code. The runtime migration engine exploits information regarding the application runtime state and higher-level information on the available target platforms. Runtime application data are used to achieve interaction continuity and preserve usability, while information on the different platforms is considered to adapt the application's appearance and behaviour to the specific device. The paper also discusses a sample application in order to provide concrete examples of the results that can be achieved through our approach.

Keywords: Migratory interfaces, Run-time architectural support, Multi-platform applications

1 Introduction

Today we are witnessing a proliferation of new mobile devices offering the possibility of accessing the Internet through different modalities. Everyday life is becoming a multi-platform environment where people are surrounded by devices through which they can get connected in different ways.

Many efforts are currently aimed at supporting users to interact through multiple devices. Users like to move freely and still be able to access their applications through the device at hand. For example, the use of a desktop PC to perform on-line operations can be switched to a wirelessly connected personal device, when the user needs to leave the office and the operations on the way are not yet completed. To this end, a multi-platform migration service is necessary, by which the user can interact with applications while changing devices and still maintain interaction continuity. Migration can satisfy many user needs such as these. There are two main issues concerning this kind of service. Firstly, the diversity in features of the platforms involved in migration, like different screen size, interaction facilities, processing power and energy supply, can make an application developed for a desktop, unsuitable for a PDA and vice versa. Thus, an application cannot migrate *as is* from one device to another, and must be adapted at runtime, taking into account the diversity of the devices involved (see for example, Kaikkonen and Roto, 2003). The second issue concerns interaction continuity. Users who want the application to migrate, do not want to have to restart the application on the new device; they want to continue their interaction from the same point where they left off, without having to re-enter the same data and go through the same long series of links to get to the page they were visiting on the previous device (these issues are introduced in Song et al. 2003). Two main kinds of information are relevant in performing migration: static information referring to the features of the devices, and runtime information that refers to the state of the migrating application, which can be summarised by the history of user interactions with the application, including visited pages, submitted data and results of previous data processing. There are several techniques for migrating user interfaces to different devices, in particular to small screens. Herein we focus on interaction continuity and device adaptation at runtime that takes into account usability principles. We consider different platform-specific versions of the same application, starting with a general task model (Paternò and Santoro, 2003) from which it is possible to generate the actual application by means of the TERESA tool (Mori et al. 2003). We consider the migration of TERESA-generated applications. This tool produces a description of the pages and the interactions that they support at different abstraction levels. Runtime data on the state of the application for which migration is required will be collected locally from the platform requesting migration. This information is transmitted to the server in order to recreate the corresponding state in the application for the target device.

The issues raised by a device-aware runtime migration for Web applications were introduced in (Bandelloni and Paternò, 2003*)*. In this paper we provide a systematic solution to such issues composed of an architecture, a set of algorithms and protocols, and a prototype. This solution uses information created following a model-based design approach in order to obtain a migration service offering user interaction continuity, platform awareness and runtime adaptability.

The paper is organised as follow. In section 2 we present an overview of related work and in section 3 we introduce the TERESA approach to design and develop multi-platform user interfaces. In section 4 we discuss the possible cases that should be considered when migrating interfaces between different interaction platforms and in section 5 we present the solution we adopted to support the runtime migration of Web interfaces. The architecture of the migration service is presented in section 6, while the reason for our architectural choices are discussed in section 7. In section 8 we present a case study to show a sample application of the service. Finally, in section 9, we provide some concluding remarks and an overview of the future extension of the service.

2 Related Work

Application migration is a field undertaken by several research projects, which address it from different points of view. The growing interest in migration in the ubiquitous computing field aims to make an application follow a user from one device to another.

An interesting paper (Bharat and Cardelli, 1995*)* provides an overview of the main matters concerning application migration. They present a programming model and an implementation of a tool for developing migratory applications, placing no restriction on the kind of application that can be built. The approach is agent-based: agents carrying pieces of code and the state of the migratory application are sent from a host to another where a server allows the agent to rebuild the migrating application. This approach can also be used to build migratory interfaces into the agent, including interface code and state. Such an approach is not suitable for our goals, our intent is to build a migration service supporting several kind of platforms, from powerful stationary PCs to PDAs to cell phones. Most of them are mobile platforms, having to cope with power consumption, low storage and elaboration capabilities. The processing load involved with using agents that migrate to a platform hosting an agent server, where the application is rebuilt at runtime, would be too heavy for most of the platforms we have targeted in our migration service.

Our service is designed to support adaptation to interaction platforms while preserving usability. This property has been called plasticity by some authors (Calvary et al., 2001) who proposed a reference model for developing plastic user interfaces. This reference model mainly use concepts developed in the model-based community over the last decade (Einsenstein et al., 2001 and Paternò 2001). As indicated in the paper, plasticity is not only obtained by rearranging the elements in a page: the tasks the user can perform on one platform or another may also vary and this must be taken into account in designing interface plasticity. An ontology for multi-surface interaction is presented in (Coutaz at al, 2003), where the distribution of an interface over diverse kinds of physical surfaces is considered, not only as screens of computational devices, but also as any everyday object that can be treated as a surface, such as a wall or a table. A classification is made on the basis of a set of properties able to describe the surface features and is useful for deciding if a surface can be used to support a certain application interface.

Kaikkonen and Roto, 2003 prove how the different features of the interaction platforms can influence Web applications usability. Different screen size, interaction facilities, processing power and energy supply, can make a Web application developed for a desktop, unsuitable for a PDA and vice versa. Thus, an application cannot migrate as it is from one device to another, and must be adapted at runtime, taking into account the diversity of the involved platforms.

An overview of the techniques used in adapting user interfaces to different devices, in particular to small screens rely on size reduction and data summarisation as it is shown in (MacKey, 2003*)*. This approach raises the risk to make the application unusable because objects on the page become difficult to recognise. We want to overcome this kind of problem adapting interfaces to different platforms, taking into account the effects on the usability of the application.

In (Song et al. 2002*)* aspects concerning interaction continuity are shown. The runtime state of the migrating application must be preserved on the target device, to allow users to continue the interaction from the exact point where they left. Interaction continuity is one key factor in our migration service, what is new in our approach is that when target and source platforms involved in migration are different, the runtime state will not be sent to the target platform as it is, data are transformed and adapted to the features of the target platform.

3 Model-base Development of Platform-Aware User Interfaces

As we mentioned in the introduction, in this work we present a solution to obtain migration of interfaces developed by the TERESA authoring environment. The reason for this choice is that TERESA supports a model-based development and is able to maintain links between descriptions of the user interface at different abstraction levels. Our solution exploits such information. In this section we recall some basic concepts related to TERESA for the readers who are not familiar with it. More information is available in (Mori et al., 2001). TERESA (publicly available at http://giove.cnuce.cnr.it/teresa.html) is intended to provide a complete semi-automatic environment supporting a number of transformations useful for designers to build and analyse their design at different abstraction levels and consequently generate the concrete user interface for a specific type of platform.

The abstraction levels considered are: the task model, where the logical activities to support are identified; the abstract user interface, in this case the interaction objects (but classified in terms of their semantics, still independent of the actual implementation) are considered, and the concrete user interface (the actual corresponding code). The main transformations supported in TERESA are:

- *Presentation Task Sets and Transitions Generation.* From the specification of a ConcurTaskTrees (Paternò, 1999) task model it is possible to obtain the set of tasks, which are enabled over the same period of time according to the constraints indicated in the model. Such sets, depending on the designer's application of a number of heuristics supported by the tool, might be grouped together into a number of *Presentation Task Sets (PTSs)* and related *Transitions* among the various PTSs.
- *From Task Model-related Information to the Abstract User Interface.* Both the task model specification and PTSs are the input for the transformation generating the associated abstract user interface, which will be described in terms of both its static structure (the "presentation" part, which is the set of interaction techniques perceivable by the user at a given time) and dynamic behaviour (the "dialogue" part, which indicates what interactions trigger a change of presentation and what the next presentation is). The structure of the presentation is defined by elementary interactors characterised in terms of the task they support, and their composition operators. Such operators are classified according to the communication goals to achieve: a) *Grouping*: indicates a set of interface elements logically connected to each other; b) *Relation*: highlights a one-to-many relation among some elements, one element has some effects on a set of elements; c) *Ordering*: some kind of ordering among a set of

elements can be highlighted; d) *Hierarchy*: different levels of importance can be defined among a set of elements. There is also the option to automatically generate the abstract UI for the target platform (instead of going through the two transformations mentioned before), starting with the currently loaded (single-platform) task model, and using a number of default configuration settings related to the user interface generation.

- *From the Abstract User Interface to the User Interface for the specific platform.* This transformation starts with the abstract user interface, it is possible to move into the related concrete user interface for the specific interaction platform selected. A number of parameters related to the customisation of the concrete user interface are made available to the designer in order to obtain the concrete user interface. Lastly, the tool generates the code according to the type of platform selected from the concrete user interface description. Currently it generates code in HTML, XHTML Mobile Profile and VoiceXML.

4 Runtime Migration Cases

Different types of runtime migration can be identified, along with different levels of complexity for each one of them:

- *Total Migration*: the client application migrates totally from a device to the other.
- *Control Migration*: the client application is divided into two parts, one for user interaction (control part) and one for information presentation (presentation part). The control part remains on one device, while the presentation one migrates to the other device, or vice versa (an example is discussed in Nichols et al. 2002).
- *Mixed Migration*: the client application is split into several parts, concerning both control and presentation and different parts are distributed over two or more devices.

In this work, we focus on *Total Migration*, with the goal to support a runtime migration that takes into account the differences between the two platforms involved.

TERESA structures user interfaces into presentations and transitions among them. When we migrate a presentation from a platform to another one the runtime support first identifies the closest presentation in the target platform. The difference between presentations in different platforms is calculated in terms of the number of logical tasks supported. A task can be supported through different interaction techniques. However, the logical meaning of the task is still the same. Taking into account interactive applications developed by means of TERESA we can identify the following situations concerning the runtime migration of a presentation between two platforms (see also Figure 1):

- The migrating presentation corresponds to one target presentation. In this case the target page to be loaded on target device can be immediately identified through a one by one mapping. The two presentations can differ in the number of supported tasks, in particular the target presentation can support:

  - Same tasks. Even if the same tasks are supported by the two presentations, the runtime state data may need to be modified too, because tasks can be implemented by means of different kinds of concrete objects.
  - Lower number of tasks. As in previous case, there can be a mapping of data concerning corresponding tasks. Data concerning tasks not supported by the target presentation are ignored.
  - Higher number of tasks. Source tasks are treated as in previous cases, while target tasks for which information cannot be retrieved from source runtime state data, are untouched and loaded with their default values.
- The migrating presentation corresponds to multiple target presentations, among which the tasks set of the source presentation are spread. In this case the target presentation is identified by computing the one having the highest number of tasks in common with the source one. In case that more than one target presentation has the same similarity degree according to this criterion, it is chosen the one supporting the task associated with the last concrete object through which the user interacted with the application on the source side.
- Multiple presentations in the source platform correspond to one presentation in the target platform. In this case the migrating task set will correspond to part of the task set supported by one of the target presentations.

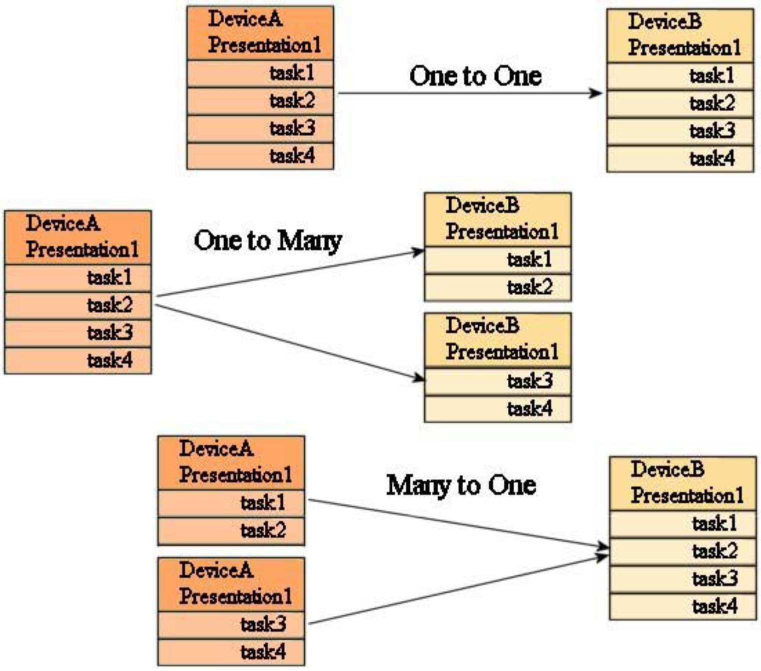


Figure 1: The possible relationships between source and target presentations.

Control Migration is a more challenging issue than Total Migration. It requires a further analysis of the abstract user interface of the migrating page in order to decide at runtime if and how it can be divided into control and presentation parts. When splitting is possible, two new abstract user interfaces have to be generated corresponding to the two parts. From the new abstract user interfaces, the control and presentation pages must be generated and loaded onto the source and target devices. Control Migration requires more runtime processing for the analysis of the set of tasks to be considered. The choice to address only Total Migration herein was dictated by considerations of complexity. Total migration provides the foundation on which we will build to introduce Control Migration.

5 Our Migration Solution

Our migration service is designed to meet two main requirements, device awareness and interaction continuity. To keep interaction continuity it is necessary to collect information concerning the runtime state of the migrating application, to activate the application on the target device, from the same point where it left. Since migration will involve different types of platforms, runtime state will not be migrated as it is. Data concerning the platform type will be used to adapt the runtime data collected on the source platform to the target one. Hence we have implemented a mapping algorithm that makes use of both runtime state and involved platform data, to load on the target the application version fitting its features, and keeping state consistency with the state the application had at migration time.

The first step is to identify the page to be loaded on the target device. Different platform specific version of an application produced by TERESA can have a different number of pages and also support a different set of tasks, hence it is not possible to create a one-to-one correspondence between presentations for different platforms.

When migration is required, the logical description of the page is retrieved. The Abstract User Interface, describing the features of the application for the kind of platform corresponding to the source one, is accessed and the presentation corresponding to the migrating page is retrieved. The set of tasks contained in the presentation, hence supported by the migrating page is extracted and used by the mapping algorithm in order to find the right target page.

The mapping is performed at Abstract User Interface level. First it is retrieved the kind of the target platform in order to select the target Abstract User Interface that is used to identify the target presentation. From the set of tasks previously contained in the source presentation, the tool identifies the most similar abstract presentation in the target Abstract User Interface and then the corresponding page in the application version for the target platform.

Similarity is calculated in terms of tasks supported, the higher number of tasks the source and target presentation share, the more similar the presentations are. This similarity criterion can lead to ambiguity in case more than one target presentations share the same number of tasks with the source one, hence they have the same similarity degree. To resolve this conflict, we identify the target presentation supporting the task associated with the interaction object last modified by the user, since the user is most likely to continue interaction from that point. Once the target page has been identified, it is necessary to calculate the state of the objects contained in the corresponding page, which will be communicated to the target device along with its URL The state contains information regarding the results of previous performed user interactions (elements selected, text entered and so on). For each source object the corresponding target one is identified through the identifier of the corresponding task in the source presentation and its corresponding in the target presentation. If the source concrete object and the target concrete object have the same type, the runtime state of the source object can be applied to the target one as it is, otherwise it will be adapted to match the features of the target object.

One potential issue for migrating interfaces to a target device where the same task is supported by means of different interaction objects is whether the change of user interface can disorient the users. Since our migration service is designed to address TERESA-generated interfaces, this potential problem is taken into account because the tool takes into account the tasks that the application should support and for each of them only the concrete objects suitable for their support are used for implementation. The actual interaction object to be used in generating the user interface will also depend on the kind of platform it is intended for. Hence, interaction objects that can disorient the user will not be proposed to support the task performance.

6 Migration Service Architecture

Our migration service relies on a server machine working as a Web server making accessible the platform-specific application implementations as well as a migration server managing context information to support migration requests. Client platforms use the migration client loaded from the server in order to enable or disable the possibility of receiving incoming applications and migrating Web applications. References to all platforms, which enable the reception of incoming applications, are stored in the server.

Figure 2 shows the user interface for the control service: it is possible to access the list of migratory applications available and the list of target systems that are currently enabled to the migratory service, request a dynamic update of such information and trigger the migration of the current application.

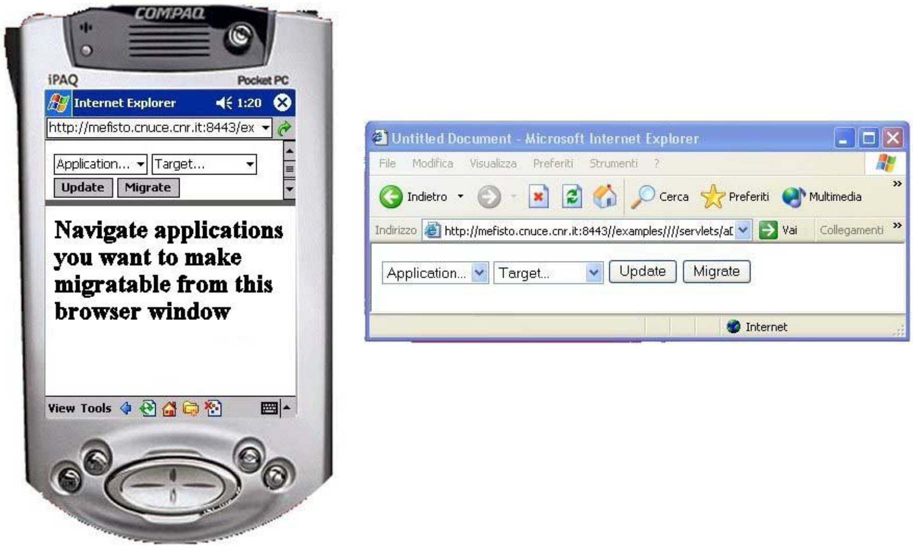

Navigate applications you want to make migratable from this browser window

Figure 2: The interface for the migration service in the PDA and desktop environments.

When a platform asks for migration, the request sent by the locally running migration client reaches the migration server, which will exploit both runtime and static context data to perform the presentation mapping process as described in Section 5.
The corresponding page and its runtime context for the target device will be finally sent to the migration client in the target platform that will open a local browser window allowing users to continue their interaction (the sequence of functionalities to perform is indicated in Figure 3).

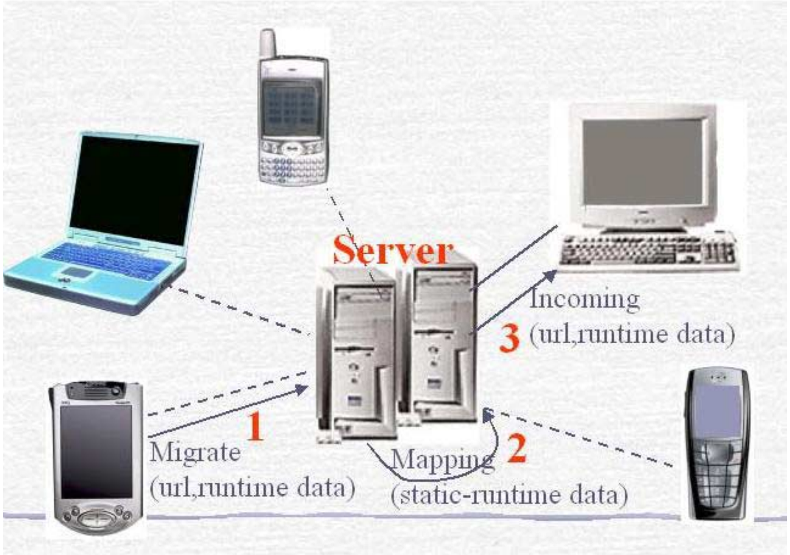


Figure 3: The Migration Process.

A number of modules and algorithms are used both for starting up the migration server and performing migration requests.

6.1 Web Server Startup. No specific algorithms have been implemented for the Web server.

6.2 Migration Server Startup. The migration server must be started in order to support migration requests. In this phase an internal data structure is filled in with static data concerning all applications supported by the migration service. For each application and for each platform, information regarding associations between different abstraction levels is built to allow the matching of the presentations to supported tasks and the Abstract User Interface elements to the corresponding user interface elements. Hence, for each platform-specific version of the application, the XML file defining the corresponding abstract user interface is analysed to retrieve the set of presentations making up the specific version of the application and the tasks supported by each presentation. Finally, for each presentation the static data are completed by adding the concrete type, name and identifier of the concrete elements implementing the supported tasks. The concrete data are retrieved by analysing the file implementing each presentation.

6.3 Migration Service Loading. Users who want to access the migration service have to request it by loading the client service manager, which depends on the actual platform. In any case, the client migration service will store information on the platform accessed by the user communicating it to the server, and a graphic interface is activated. A further client module is started in order to allow incoming application migration acceptance.

6.4 Client Migration Request Sending. On the source client side, JavaScript functions collect runtime data concerning the URL of the loaded application page and the state of the interaction objects contained in the page when the user requests migration. A migration request, including the IP address of the target device and all runtime data collected into a single string, is sent to the server by submitting a form, which causes the migration server run-time module activation.

6.5 Server Migration Request Elaboration. Once the migration request is received, the server checks whether the target IP address matches a currently connected platform, and the corresponding platform type is retrieved, hence the URL string of the migrating page is analysed in order to extract the name of the migrating application. Also the set of corresponding tasks are retrieved and matched against the whole set of presentation data corresponding to the target version of the application in order to retrieve the most similar presentation. The most similar target presentation is the one sharing the highest number of supported tasks with the source presentation. In case this criterion identifies multiple presentations, the runtime state data is used to identify the one containing the task corresponding to the last interaction performed by the user, as described in section 5. At this point it is possible to build the URL string that must be loaded on the target device. Once the target URL has been built, the service has to map the state of source interaction objects to the target ones. Information related to tasks that are not supported by the target presentation is eliminated. Tasks implemented by the same kind of interaction objects maintain the runtime state, and their concrete name and ID must be updated. Tasks implemented by means of different interaction objects also need information concerning such objects to be adapted.
Once the mapping of both the URL and runtime state is completed, a connection is opened with the target platform, which will receive the target URL string followed by the target runtime state data.

6.6 Client Migration Request Acceptance. On the target client side, the migration application keeps listening for connection requests coming from the server. Once a connection is received, the URL of the page to be loaded and runtime state data are read from it. The incoming URL is loaded in a new browser window in case of a desktop system, and into the browser window that is already opened in case of PDAs. Hence, runtime state data are applied to the concrete object of the loaded page. The application of state information to the interaction objects is performed by means of Javascript functions implemented on client side.

7 Architectural Choices and System Performance Considerations.

The core of our system is the server. The decision to use a server, rather than relying on a peer-to-peer service, stems from considerations on the computational load involved with supporting migration. The mapping algorithm needs to access the abstract description and the concrete implementation of each possible platform to be supported. Using a peer-to-peer service would mean replicating such information on each supported device and the migration engine would have to be installed on each platform. This raises the risk of data inconsistency and system maintenance would become hard to handle, requiring installation of the engine on each platform every time the service is updated.

Moreover, it must be borne in mind that power consumption is a limiting factor when mobile devices are used. By adopting the chosen solution, we intend to delegate most of the computational load to the server, which is assumed not to have power problems, and require the clients to perform only the two basic operations of sending migration requests and loading the data sent by the server after processing.

At migration server start-up, for each application the Abstract User Interface of each platform-specific version is analysed as well as its corresponding implementation. This is a large amount of data to be processed and it increases with the number of supported application and platforms. The building of mapping tables used by the migration server actually requires a considerable amount of time. In any case, such computations need to be performed only once, off-line at the first initialisation of the server. Once all the data have been loaded, the runtime processing for serving migration requests are performed by accessing data through hash keys, hence in a very short time. This makes it possible to avoid long latency times for the user request to be served.

8 A Case Study

In this section we discuss a case study that provides concrete examples of the results that can be achieved through our approach. We first introduce a couple of scenarios that outline the main features covered by the migration service. On the basis of the scenarios, we will show the user interface of a sample migrating, multi-platform application, focusing on the operations that the migration service must perform in order to keep interaction continuity and to adapt the migrating application to the different kind of platforms involved.

8.1 The scenarios

Louis is walking down the street on his way to work. He is thinking about vacations and decides to check his bank account to see how much he can afford. Louis turns on his PDA and accesses the bank's web site. He had previously registered with the bank's web service, hence the bank application automatically identifies the PDA as Louis' personal device. From the main page Louis chooses to access his bank account data. He does not need to enter any personal information or the account number, the application has already retrieved all data after having identified the PDA and his balance is promptly displayed. Louis cannot remember the previous situation and would like to access information on the last operations performed on his account. He requests that the application display the 10 last operations, and the data appear on the

PDA. Only the amounts of money added or subtracted at each operation are shown. To access more details concerning each of the operations, a new page must be loaded. Moreover, operation data are split into two pages, and are not displayed together. Too tedious, it would be much more comfortable to have a full overview of operations and their details. However, such a feature is not supported by the PDA application version: it would be useless to try and show a large table on a small-sized PDA screen.

Meanwhile, Louis has reached his office and turned on his Desktop PC. Accessing the migration function on the PDA, he requests to have the bank application migrate to the Desktop PC. The application is automatically shut down on the PDA and is activated on the Desktop in the same runtime state it was when the migration was initiated. Bank account related operations are immediately shown on the desktop without Louis having to re-enter any data or make any requests, as he would have had to do if he had accessed the application directly from the desktop. The Desktop screen displays a table showing detailed information for each of the latest 10 operations performed on the bank account. Louis can easily see that the last operation is a payment by the company he works for, a reimbursement for travel expenses incurred during the conference he attended four months ago. On the other hand his salary has not yet been credited to his account.

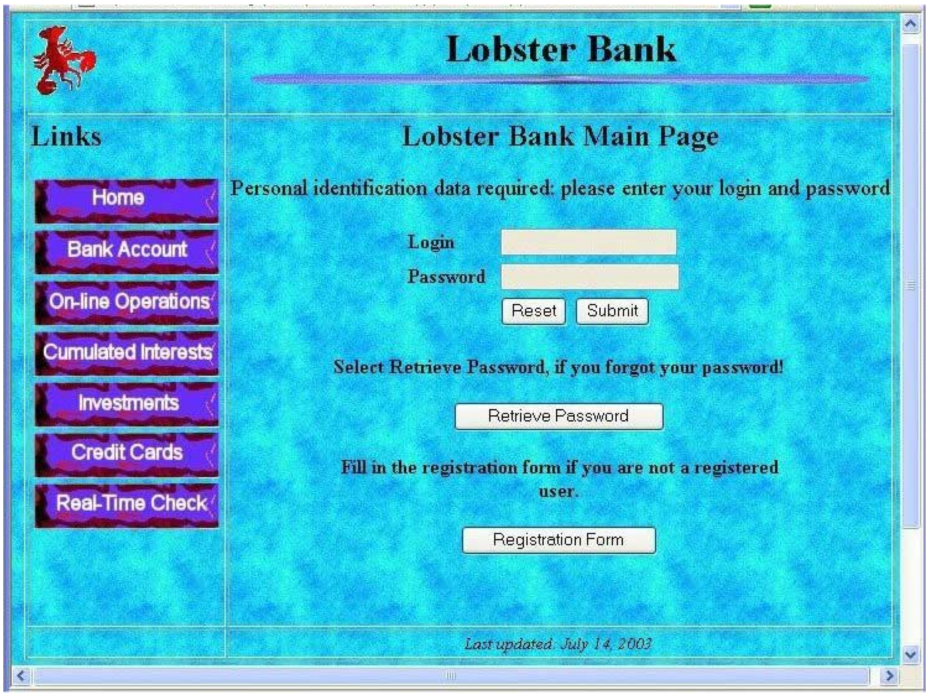
Louis decides to go to the bank's branch office personally, in order to withdraw the money he plans to spend during his vacations. Through the desktop bank application he checks to see how many people are currently waiting to be served. He sees that the estimated waiting time is one hour and a quarter, and decides to pick up a reservation ticket. He gets ticket number 40 and uses the migration service to have the bank application migrate back to the PDA where he can see his reservation number and the real-time situation in the branch office. He can also activate an alarm feature on the PDA, which will alert him when his turn is getting close.

In the second scenario Louis is at home and accesses the bank application through his Desktop PC. After having entered identification data, he is allowed to access the bank services. He needs some money transfer in order to pay the fee for a conference registration and accesses the page reserved to on-line operations. He starts filling in the form to perform the transfer, meantime he realises that it is late and it is time to go to work. Through the migrating service, he can migrate the page he was interacting with on his own PDA. On the PDA screen, only the part of the form Louis was filling in when asking for migration is shown and he can complete the form adding the missing data through his PDA, while he catches the metro to go to his office. Before submitting data, he wants to be sure of their correctness and accessing the previous page, he can control the data previously inserted through the Desktop PC. The form is correct and he can submit it. The registration to the conference is now completed.

8.2 The Migrating Web Bank Application.

In this section we introduce the *Web Bank Application*, a sample application built on the basis of the scenarios described in section 8.1. The application is a typical bank application that allows registered users to access their own private data and perform on-line operations. We are presenting here only the most relevant application features outlining how they can benefit of the migration service.

When a user accesses the *Web Bank Application* from a Desktop PC, he is always inquired about identification data, in order to be recognised by the application and being allowed to access the Bank Application Services. A user accessing the application for the first time, can complete a registration form in order to get a login and a password. Once the registration is completed, a user accessing from a PDA will automatically gain access to the application without having to enter identification data. Because of security matters, such a task is not enabled on the Desktop version.



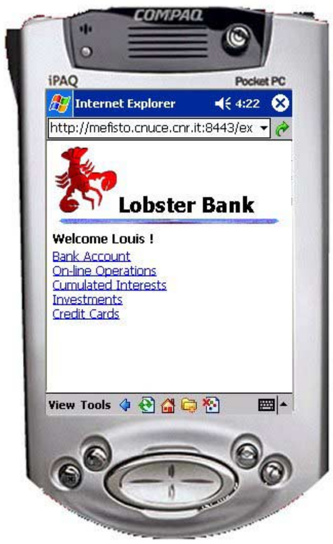
Personal identification data required: please enter your login and password

Select Retrieve Password, if you forgot your password!

Fill in the registration form if you are not a registered user.

Last updated: July 14, 2003

Figure 4. The Desktop Main Page



Lobster Bank

Welcome Louis !

Bank Account
On-line Operations
Cumulated Interests
Investments
Credit Cards

Figure 5. The PDA Main Page

In Figure 4, it is shown how the application appears to a user loading it from a Desktop PC. If migration to PDA is required at this point, and the user previously registered to the Bank service, the PDA version will not show the same accessing form, the automatic access task will be activated and the user will be identified (Figure 5). Selecting access to information on his personal bank account, the user, Louis in this example, will see immediately the information required, without having to enter any identification data.

In this migration case, after the identification of the target page, the migration service will recognise that the automatic access from PDA is possible, since the user previously registered to the bank service. In addition, there is a set of tasks supported by both the corresponding presentations, such tasks are: *BankAccoutAccess*, *OperationsAccess*, *CumulatedIterestsAccess*, *InvestmentsAccess* and *CreditCardAccess*. On the Desktop version, the above mentioned tasks are implemented through clickable button images, while the PDA version implements them by means of simple links. The migration service will have to recognise the different implementation of the tasks and adapt the runtime state retrieved from the page loaded on the Desktop to the one to be loaded on the PDA.

Let us suppose that Louis has accessed his bank account data and selected to have information on the last 10 money movements performed. To see all the operations selected he will have to access two pages and one more page has to be accessed to have more detailed information concerning each operation (Figure 6).

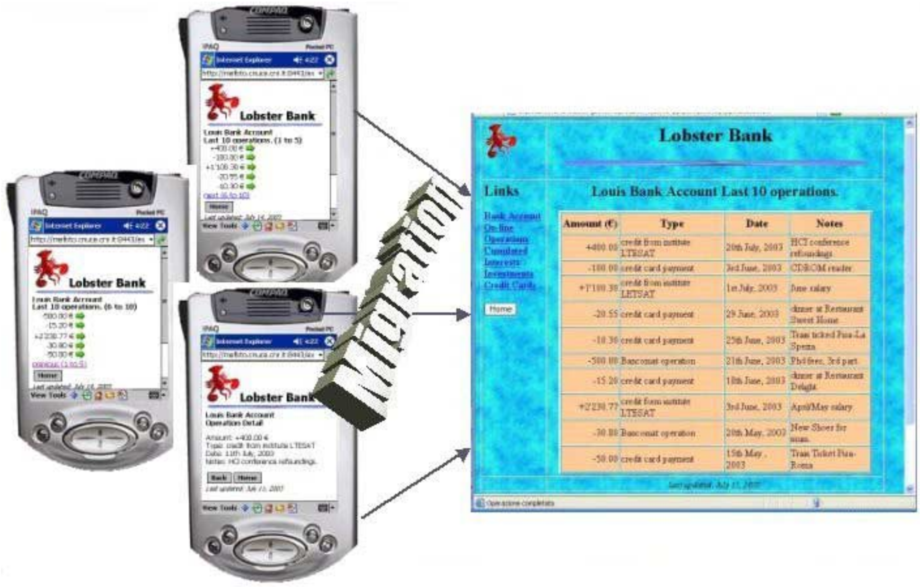
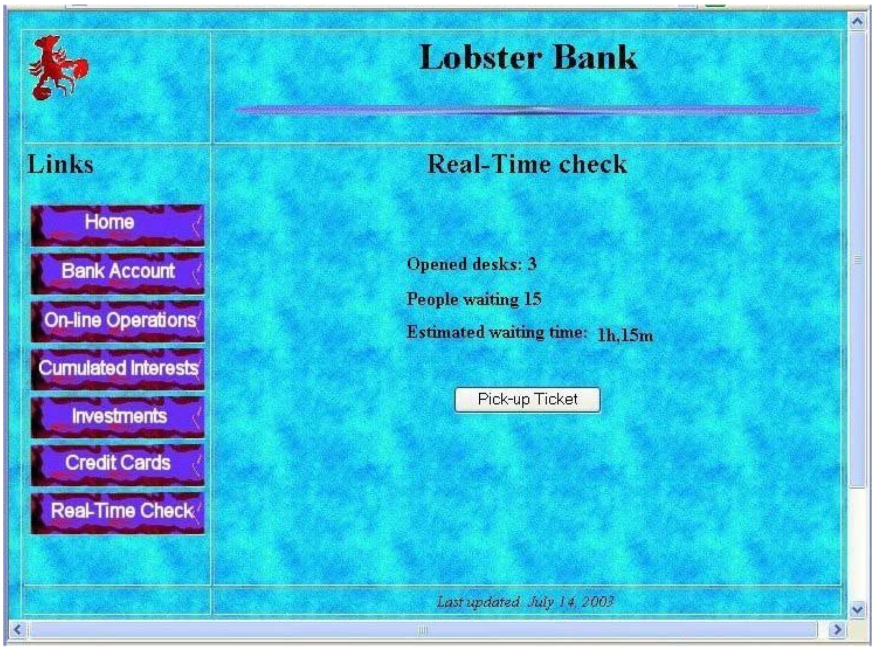


Figure 6: The PDA interface for checking the bank account.

To have a complete single view of the operation selected and their detail, Louis decide to migrate to the Desktop PC. Following the criterion of the most similar page, migration from any of the PDA page presented, the resulting page on the Desktop is the one showed in Figure 6.

Another significant migration case, is when a task performed on the migrating page enables a further task on the target platform, that was not supported by the source platform. The *Web Bank Application* allows a user to reserve a ticket for accessing the real bank office and also the monitoring of the real-time office situation. Only on the PDA version, the user can ask the application to keep checking the real-time situation and alert the user when his turn is getting closer. Such a task is not present on the Desktop version, because it is supposed to be useful only in case the user is close to the bank office and waiting for his turn. The alerting service is enabled only after a ticket has been reserved. Figure 7 and 8 show the steps Louis has to perform in order to check the office situation and reserve his ticket.



Lobster Bank

Real-Time check

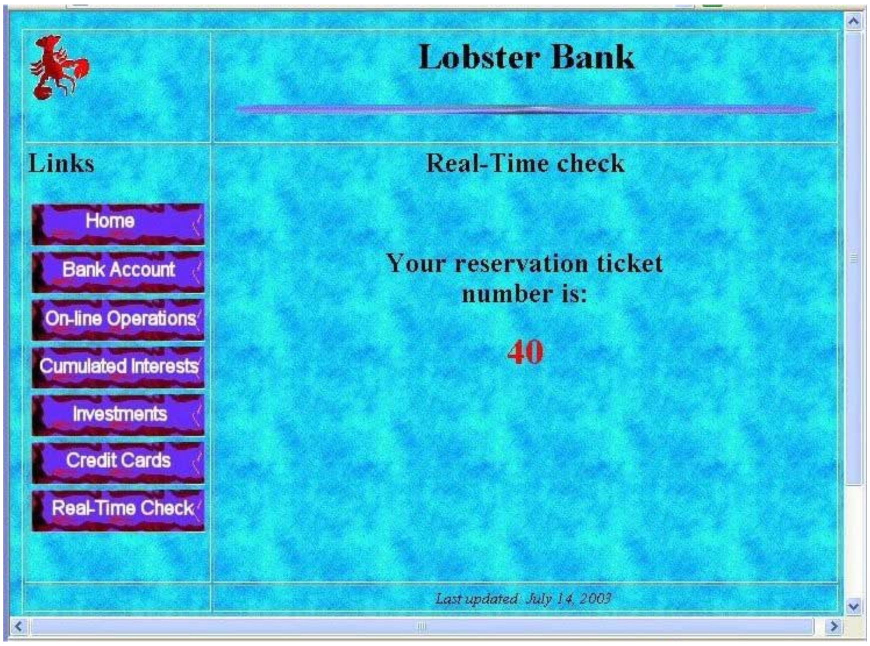
Opened desks: 3

People waiting 15

Estimated waiting time: 1h,15m

Last updated: July 14, 2003

Figure 7. Desktop Real-time Check.



Lobster Bank

Real-Time check

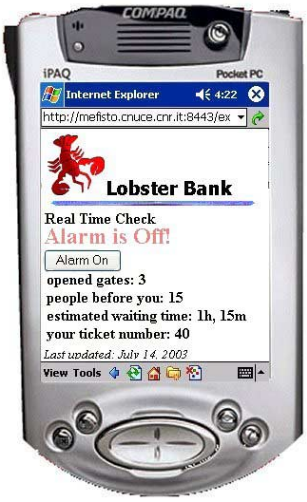
Your reservation ticket number is:

40

Last updated: July 14, 2003

Figure 8. Desktop Ticket Reserved.

Migration required for one of both Desktop pages shown in Figure 7 and 8 will identify the PDA page shown in Figure 9, where the user can decide to activate the alert service.



Lobster Bank

Real Time Check

Alarm is Off!

opened gates: 3

people before you: 15

estimated waiting time: 1h, 15m

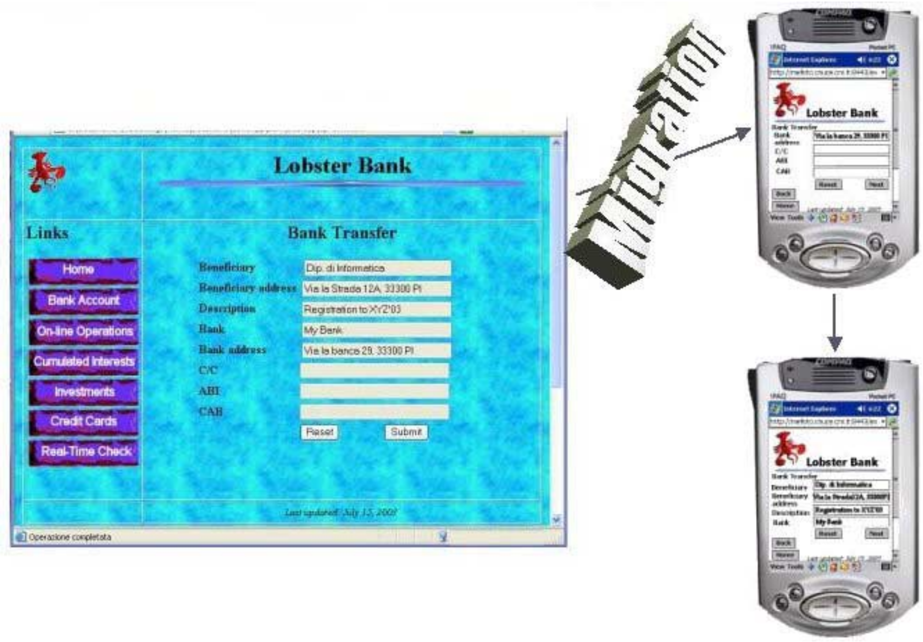
your ticket number: 40

Last updated: July 14, 2003

Figure 9. Real-Time Monitoring on PDA platform.

The above mentioned migration case does not show only the activation of a task from one platform to the other, it is also a good example of a migrating page containing a task performed accessing to a different set of domain objects. On the desktop real-time monitoring page (Figure 7), the task *ShowRealTimeState* is performed accessing the information composed of the objects: opened gates, people waiting, estimated waiting time, while set of information accessed by the corresponding task is the PDA version (Figure 9) is composed of the elements: opened gates, people before you, estimated time and your ticket number.

The following migration case is based on scenario 2 (section 8.1). Figure 10 shows the page Louis is interacting with before asking for migration. Let us imagine that he has filled in the first five fields and now wants to continue on his PDA. The first step is performed by the migration service in order to retrieve the most similar PDA page. It thus identifies two presentations containing the same number of tasks. The second step makes the migration server select the page that contains the object implementing on the target platform the task last performed by the user. As a result, the page shown in Figure 10 will be sent to the PDA and loaded. Any data previously inserted in the form by the user are not lost, and can be visualised by accessing the previous page on the PDA.



Beneficiary: Dip. di Informatica
Beneficiary address: Via la Strada 12A, 33300 PI
Description: Registration to XYZ'03
Bank: My Bank
Bank address: Via la banca 29, 33300 PI
C/C:
ABI:
CAB:

Figure 10. Desktop Bank Transfer Form.

9 Conclusions and Future Work.

We have discussed how to provide migratory interfaces able to support interaction continuity and usability. The user interfaces are developed following a model-based approach. A first prototype for total migration addressing applications obtained through the TERESA tool has been implemented, and we show an example of application. This approach opens up the possibility of intelligent environments able to support users through various platforms and allowing them to move from one platform to another one without having to restart the session from scratch. Currently, we are working on improving the collection of data on the run-time state in order to make the system more thorough and improve the support for interaction continuity. Future work will be dedicated to extending this approach in order to support the *Control Migration* as introduced in the section discussing run-time migration cases.

Acknowledgments

This work has been supported by the IST EU R&D CAMELEON project (http://giove.cnuce.cnr.it/cameleon.html). We would like to express our appreciation to the colleagues in the project and the HIIS laboratory (http://www.isti.cnr.it/ResearchUnits/Labs/hiis-lab/) for useful discussions.

References

R. Bandelloni and F.Paternò, Platform Awareness in Dynamic Web User Interfaces Migration, Proceedings Mobile HCI 2003, pp.440-445, LNCS 2795, Springer Verlag, 2003.

K. A. Bharat and L. Cardelli. Migratory *Applications*. In proceedings of User Inteface Software and Technology (UIST '95). Pittsburgh PA USA. November 15-17, 1995. pp. 133-142.

G. Calvary, J. Coutaz., D. Thevenin. A Unifying Reference Framework for the Development of Plastic User Interfaces. *IFIP WG2.7 (13.2) Working Conference, EHCI01*, Toronto, May 2001, Springer Verlag Publ., LNCS 2254, M. Reed Little, L. Nigay Eds, 2001, pp.173-192.

J. Coutaz, C. Lachenal, S. Dupuy-Chessa. *Ontology for Multisurface Interaction*. Proceedings INTERACT 2003. IOS Press. Zurich, pp.447-453, September 2003.

J. Einsenstein, J.Vanderdonckt, A. Puerta, *Applying Model-Based Techniques to the Development of UIs for Mobile Computers,* Proceedings IUI'01: International Conference on Intelligent User Interfaces, pp 69-76, ACM Press, 2001.

A. Kaikkonen and V.Roto. *Navigating in a Mobile XHTML application*. In Proceedings ACM CHI 2003. Ft. Lauderdale, Florida, April 5-10, 2003. Vol.5, pp. 329-336.

B. MacKey. *The gateway: A Navigation Technique for Migrating to Small Screens.* Doctoral Consortium, CHI 2003. Ft. Lauderdale, Florida, April 5-10, 2003. pp. 684-685.

G. Mori, F. Paternò, and C. Santoro. *Tool support for designing nomadic applications*. In Proceedings of IUI 2003 . ACM Press, 2003. pp. 141–148.

J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, M. Pignol. *Generating remote control interfaces for complex appliances*. Proceedings ACM UIST'02. October 27 – 30. Paris, France. Vol.4, pp.161-170.

F. Paternò, *Model-Based Design and Evaluation of Interactive Application*. Springer Verlag, ISBN 1-85233-155-0, 1999.

F. Paternò, C.Santoro, *A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms, Interacting with Computers*, Vol.15, N.3, pp 347-364, Elsevier, 2003.

H. Song, H. Chu, S. Kurakake. *Browser Session Preservation and Migration*. In Poster Session of WWW 2002, Hawai, USA. 7-11. May, 2002. pp. 2.