Web User Interface Migration through Different Modalities with Dynamic Device Discovery

Renata Bandelloni, Giulio Mori, Fabio Paternò, Carmen Santoro, Antonio Scorcia

ISTI-CNR, Via G.Moruzzi, 1
56124 Pisa, Italy
{renata.bandelloni, giulio.mori, fabio.paterno, carmen.santoro, antonio.scorcia}@isti.cnr.it

Abstract. In this paper we present a new environment for supporting Web user interface migration through different modalities. The goal is to furnish user interfaces that are able to migrate across different devices offering different interaction modalities, in such a way as to support task continuity for the mobile user. This is obtained through a number of transformations that exploit logical descriptions of the user interfaces to be handled. The new migration environment makes use of service discovery both for the automatic discovery of client devices and for the dynamic composition of the software services required to perform a migration request.

Keywords: User Interface Migration, Adaptation to the Interaction Platform, Ubiquitous Environments.

1 Introduction

One important aspect of pervasive environments is the possibility for users to freely move about and continue interacting with the services available 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 users and the environment itself. 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 new solution for supporting migration of Web application interfaces among different types of devices that overcomes the limitations of previous work [2] in many respects. Our solution is able to detect any user interaction performed at the client level. Then, we can 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 Web interfaces with different platforms (fixed and mobile) and modalities (graphical, vocal, and their combination). The new solution also includes a discovery module, which is able to detect the devices that are present in the environment and

collect information on their features. Users can therefore 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 server. The discovery module also monitors the state of the discovered devices, automatically collecting their state-change information in order to understand if there is any need for a server-initiated migration. Moreover, we show how the approach is able to support migration across devices that support various interaction modalities. This has been made possible thanks to the use of a logical language for user interface descriptions 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.
In the following section we discuss related work. Next, we provide an overall introduction of the environment, followed by a discussion on the logical descriptions used by the migration environment and how they are created by a reverse engineering process starting with the source desktop Web pages. Then, we provide the description of the semantic redesign module, explain how the migration environment functionalities have been incorporated, and describe the device discovery module. Lastly, we present an example application describing a migration through desktop, mobile and vocal, and draw some conclusions.

2 Related Work

The increasing availability of various types of electronic interactive devices has raised interest in model-based approaches, mainly because they provide logical descriptions that can be used as a starting point for generating interfaces that adapt to the various devices at hand. In recent years, such interest has been accompanied by the use of XML-based languages, such as UsiXML [5] and TERESA XML [7], in order to represent the aforementioned logical descriptions. The research in this area has mainly focused on how to help designers efficiently obtain different versions that adapt to the various interaction features, but contributions for runtime support have started to be proposed. For example, the Personal Universal Controller [8] automatically generates user interfaces for remote control of domestic appliances. The remote controller device is a mobile device, which is able to download specifications of the functions of appliances and then generate the appropriate user interface to access them. The architecture is based on a bidirectional asynchronous communication between the appliance and the remote controller. However, the process of discovering the device is far from automatic as the user needs to manually enter the device's network address in the remote control application before any other action can be performed. ICrafter [11] is a more general solution for user interaction in ubiquitous environments, which generates adaptive interfaces for accessing services in such environments. In ICrafter, services beacon their presence by periodically sending broadcast messages. A control appliance then requests a user interface for accessing a service or an aggregation of services by sending its own description, consisting of the user interface languages supported (i.e. HTML, VoiceXML) to an entity known as the Interface Manager, which then generates the user interface and sends it back to the appliance. However, ICrafter does not support

the possibility of transferring the user interface from one platform to another, while the user is interacting with it, maintaining the client-side state of the interface.
SUPPLE [4] generates adaptive user interfaces taking functional specifications of the interfaces, a device model and a user model as input. The remote solver server that acts as the user interface generator is discovered at bootstrap by the client devices, and they can thus request rendering of interfaces to it once it is discovered. However, discovery is limited to the setup stage of the system, and it does not monitor the runtime status of the system, thus loosing some of the benefits that could arise from a continuous monitoring activity. SUPPLE does not support the migration of a user interface from one device to another, but only adapts it to different types of platforms.
Luyten and Coninx [6] present a system for supporting distribution of the user interface over a federation or group of devices. Migratability, in their words, is an essential property of an interface and marks it as being continuously redistributable. These authors consider migration and distribution of only graphical user interfaces, while we provide a new solution supporting graphic, vocal and even multimodal user interfaces migration. A general reference model for user interfaces aiming to support migration, distribution and adaptation to the platform is proposed in [1]. Our system, in particular, proposes a more 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.

3 Overall Description of the Environment

Our migration environment is based on a service-oriented architecture involving multiple clients and servers. We assume that the desktop version of the considered applications already exists in the application servers. In addition, we have a proxy service and the services of the migration platform, which can be hosted by either the same or different systems.

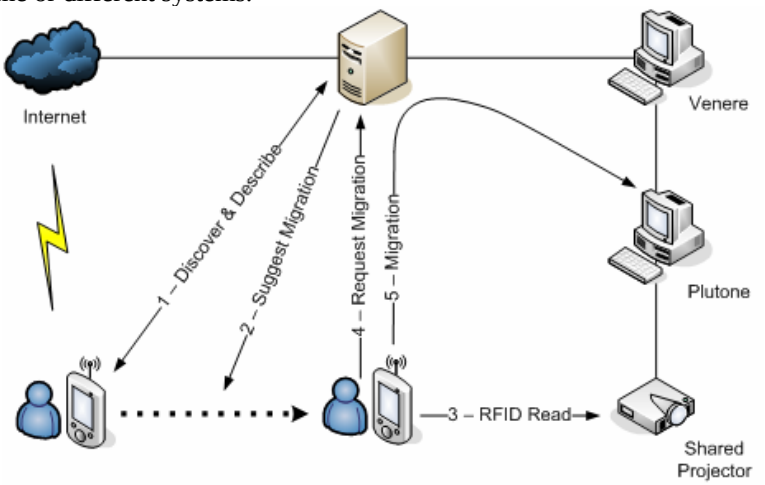

Fig. 1. Migration scenario.

Figure 1 provides an overview of the system through an example from the user viewpoint. First, there is a dynamic discovery of the new user device as it enters the environment. Then, the environment may suggest migration to a nearby device (automatic migration), or the user can explicitly request a specific migration, e.g. by pointing with a RFID reader the projector (user-initiated migration), and the user interface migrates to the target device with the mediation of the migration server. In the example in Figure 1, the projector is associated with a PC, which will be considered the target device.

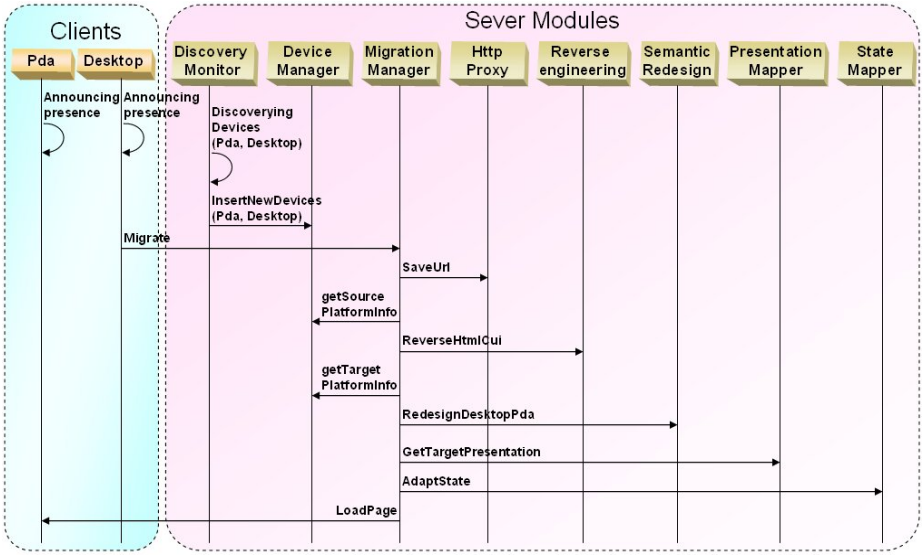

Fig. 2. Main communication among the migration services.

When a migration has to be processed, no matter which agent (the user or the system) starts the process, the *Migration Manager*, which acts as the main server module, retrieves the original version of the Web page that has to be migrated by invoking the *HTTP Proxy* service and retrieving the Web page(s) the user has accessed. Once the Web page(s) are retrieved, the *Migration Manager* builds the corresponding logical descriptions, at a different abstraction level, by invoking the *Reverse Engineering* service of our system. The result of the reverse engineering process is then used as input for the *Semantic Redesigner* service, in order to perform a redesign of the user interface for the target platform, taking into account the logical descriptions it received from the *Migration Manager*. Once the target Web pages have been generated, the *Presentation Mapper* service is used to identify the page which should be uploaded first into the target device.

In order to support task continuity throughout the migration process, the state resulting from the user interactions with the source device (filled data fields, selected items, …) is gathered through a dynamic access to the DOM of the pages in the source device. Then, such information is associated to the corresponding elements of the newly generated pages and adapted to the interaction features of the target device

by the *State Mapper* module, in a totally user-transparent way. Figure 2 represents with UML interaction diagrams the main communication among the migration services.

In order to allow for a good choice of the target device, the migration server retrieves and stores information about the devices that are automatically discovered in the environment. The collected information mainly concerns device identification and interaction capabilities. Such information allows, on the one hand, users to choose a target migration device with more accurate and coherent information on the available targets and, on the other hand, the system to suggest or automatically trigger migrations when the conditions for one arise. Thus, both the system and the user have the possibility to trigger the migration process, depending on the surrounding context conditions. A review of the different migration situations is described later on.

Users have two different ways of issuing migration requests. The first one is to graphically select the desired target device in their migration client. Users only have the possibility of choosing those devices that they are allowed to use and are currently available for migration. The second possibility for issuing migration requests occurs when the user is interacting with the system through a device equipped with an RFID reader. In this case, users could move their device near a tagged migration target and keep it close from it for a number of seconds in order to trigger a migration to that device. In this case a time threshold has been defined in order to avoid accidental migration, for example when the user is just passing by a tagged device. This second choice offers users a chance to naturally interact with the system, requesting a migration just by moving their personal device close to the desired migration target, in a straightforward manner.

Migration can also be initiated by the system skipping explicit user intervention in critical situations when the user session could accidentally be interrupted by external factors. For example, we can foresee the likelihood of having a user interacting with a mobile device that is shutting down because its battery power is getting too low. Such situations can be recognised by the system and a migration is automatically started to allow the user to continue the task from a different device, avoiding potential data loss.
Alternatively, the server can provide users with migration suggestions in order to improve the overall user experience. This happens when the system detects that in the current environment there are other devices that can better support the task being performed by the user. For example, if the user is watching a video on a PDA and a wide wall-mounted screen, obtained through connecting a projector to a desktop PC, is detected and available in the same room, the system will prompt the user to migrate to that device, as it could improve his performance. However, the user can continue to work with the current device and refuse the migration. Receiving undesired migration suggestions can be annoying for the user, thus users who want to receive such suggestions when better devices are available must explicitly subscribe to allow for this mixed-initiative migration activation service. In any case, once a migration has taken place, nothing prevents the user or the system from performing a new migration to another available device.

4 The User Interface Logical Descriptions Supported

Our migration service considers different logical views of the user interface, each one giving a different level of abstraction of the users' interactions with the system:

- The task level, where the logical activities are considered.
- The abstract interface level, consisting of a platform-independent description of the user interface, for example at this level we can find elements such as a selection object.
- The concrete interface level, consisting of a platform-dependent but implementation language independent description of the user interface, for example in the case of a graphical user interface, the abstract selection object can become a radio button, a list or a pull-down menu.
- The final user interface, the actual implemented user interface.

The abstract description level represents platform-independent semantics of the user interface and is responsible for how interactors are arranged and composed together (this will also influence the structure of the final presentations).

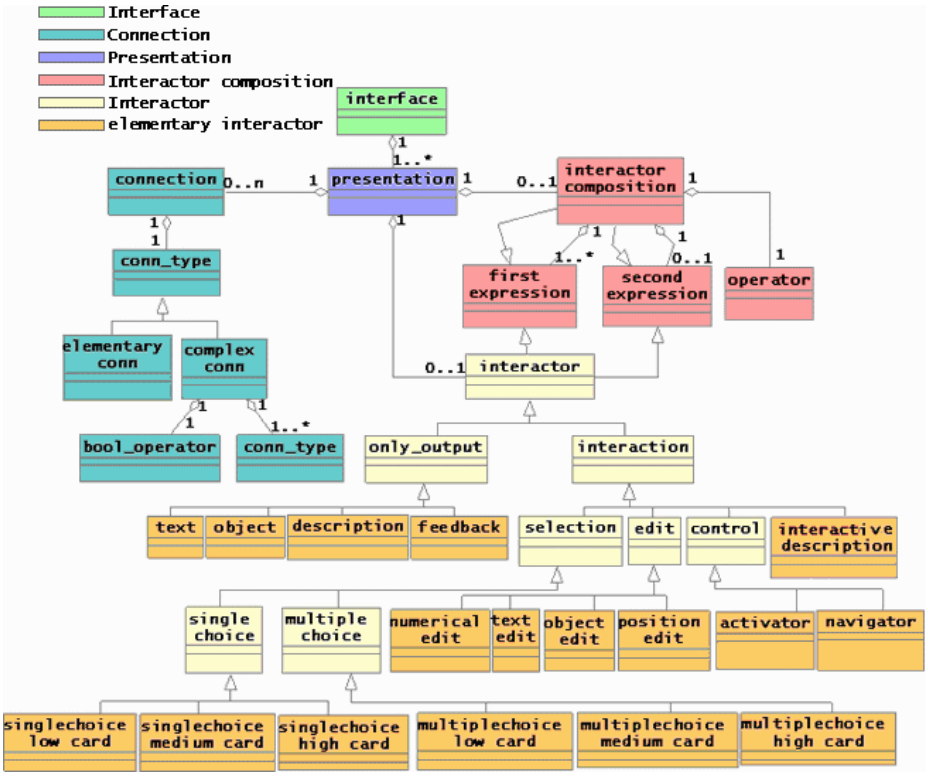

Fig. 3. The specification of the abstract user interface language used in our migration approach

The concrete description represents platform-dependent descriptions of the user interface and is responsible for how interactors and composition operators are refined in the chosen platform with their related content information (text, labels, etc.). The concrete description adds information regarding concrete attributes to the structure provided by the abstract description. The abstract description is used in the interface redesign phase in order to drive the changes in the choice of some interaction object implementations and their features and rearrange their distribution into the redesigned pages. Both task and logical interface descriptions are used in order to find associations between how the task has been supported in the original interface and how the same task should be supported in the redesigned interface, and associate the runtime state of the migrating application. We have used TERESA XML for the logical description of the structure and interface elements [7]. The logical description of the user interface is organised in presentation(s) interconnected by connection elements (see Figure 3). Connections are defined by their source and target presentations, and the particular interactor in the source presentation in charge of triggering the activation of the target presentation. Presentations are made up of logical descriptions of interaction objects called interactor elements. Interactors are composed by means of composition operators. The goal of such composition operators is to identify the designers' communication goals, which determine how the interactor should be arranged in the presentation. Thus, we have a grouping operator indicating that there is a set of elements logically connected to each other, a relation operator indicating that there is one element controlling a set of elements, a hierarchy operator indicating that different elements have different importance for users, and an ordering operator indicating some ordinal relation (such as a temporal relation) among some elements.

5 From Web Pages to Logical Descriptions

As we have introduced before, our environment exploits transformations that are able to move back and forth between various user interface description levels. Thus, reverse engineering support has been introduced. The main purpose of the reverse engineering part is to capture the logical design of the interface (in terms of tasks and ways to structure the user interface) and then use it as a key element to drive the generation of the interface for the target device. Some work in this area has been carried out previously. For example, WebRevEnge [9] automatically builds the task model associated with a Web application, whereas Vaquita [3] and its evolutions build the concrete description associated with a Web page. In order to support the automatic redesign for migration purposes, we need to access all the relevant abstract descriptions (the concrete, abstract and task level). Thus, the reverse engineering module of our migration environment is able to take Web pages and then provide any abstract logical descriptions, as needed.

Each page is reversed into a presentation, its elements associated with interactors or their composition operators. 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 an interactor a specific function analyses the

corresponding node and extracts information to generate the proper interactor or composition operator.
After the first generation step, the logical description is optimised by eliminating some unnecessary grouping operators (mainly groupings composed of one single element) that may result from the first phase. Then, according to the X/HTML DOM node analysed by the recursive function, we have three basic cases:

- The X/HTML element is mapped into a concrete interactor. This is a recursion endpoint. The appropriate interactor element is built and inserted into the XML-based logical description.
- The X/HTML node corresponds to a composition operator. The proper composition element is built and the function is called recursively on the X/HTML node subtrees. The subtree analysis can return both interactor and interactor composition elements. In any case, the resulting concrete nodes are appended to the composition element from which the recursive analysis started.
- The X/HTML node has no direct mapping to any concrete element. If the element has no child nodes, no action is taken and we have a recursion endpoint, otherwise recursion is applied to the element subtrees and each child subtree is reversed and the resulting nodes are collected into a grouping composition.

Each logical presentation can contain both elements that are the description of single interactor objects and composition operator elements. The composition operators can contain both simple interactors and multiple composition operators. Our reverse engineering transformation identifies the corresponding logical tasks [10]. This is useful for two main reasons: the interface on the target device should be activated at a point supporting the last basic task performed on the source device in order to allow continuity, and in the redesign phase it is important to consider whether the type of tasks to support is suitable for the target device.

6. Semantic Redesign for Different Types of Platforms

The redesign transformation aims at changing the design of a user interface. In particular, we propose a redesign for identifying solutions suitable for a different platform, which is performed automatically by exploiting semantic information contained in the logical description of the user interface (created by the reverse process). Given the limited resources in screen size of mobile devices (such as cell phones or PDAs), desktop presentations generally must be split into a number of different presentations for the mobile devices. The logical description provides us with some semantic information that can be useful for identifying meaningful ways to split the desktop presentations along with the user interface state information (the actual implemented elements, such as labels, images, etc.). The redesign module analyses the input from the desktop logical descriptions and generates an abstract and concrete description for the mobile device from which it is possible to automatically obtain the corresponding user interfaces. The redesign module also decides how

abstract interactors and composition operators should be implemented in the target mobile platform. In order to automatically redesign a desktop presentation for a mobile presentation we need to consider semantic information and the limits of the available resources. If we only consider the physical limitations we may end up dividing large pages into small pages which are not meaningful. To avoid this, we also consider the composition operators indicated in the logical descriptions. To this end, our algorithm tries to maintain interactors that are composed through some operator at the conceptual level in the same page, thus preserving the communication goals of the designer. However, this is not always possible because of the limitations of the platform, such as limited screen resolution. In this case, the algorithm aims at equally distributing the interactors into presentations of the mobile device. In addition, the splitting of the pages requires a change in the navigation structure with the need for additional navigator interactors that allow access to the newly created pages. More specifically, the transformation follows the subsequent main criteria:

- The presentation split from desktop to mobile takes into account the composition operators because they indicate semantic relations among the elements that should be preserved in the resulting mobile interface.
- Another aspect considered is the number and cost of interactors. The cost is related to the interaction resources consumed, so it depends on pixels required, size of the fonts and other similar aspects.
- The implementation of the logical interactors may change according to the interaction resources available in the target platform (for example an input desktop text area, could be transformed into an input mobile text edit or also removed, because writing of long text is not a proper activity for a mobile device).
- The connections of the resulting interface should include the original ones and add those derived from the presentation split.
- The images should be resized according to the screen size of the target devices, keeping the same aspect ratio. In some cases they may not be rendered at all because the resulting resized image is too small or the mobile device does not support them.
- Text and labels can be transformed as well because they may be too long for the mobile devices. In converting labels we use tables able to identify shorter synonyms.

In particular, the following rules have been applied for creating the new connections:

- Original connections of desktop presentations are associated to the mobile presentations that contain the interactor triggering the associated transition. The destination for each of these connections is the first mobile presentation obtained by splitting the original desktop destination presentation.
- Composition operators that are allocated to a new mobile presentation are substituted in the original presentation by a link to the new presentation containing the first interactor associated with the composition operators.
- When a set of interactors composed through a specific operator has been split into multiple presentations because they do not fit into a single mobile presentation, then we need to introduce new connections to navigate through the new mobile presentations.

In the transformation process we take into account semantic aspects and the cost in terms of interaction resources of the elements considered. 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. Thus, each interactor and (even each composition operator) has a different cost in terms of interaction resources. The algorithm inserts interactors into a mobile presentation until the sum of individual interactor and composition operator costs reaches the maximum global cost supported. 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. One example of the costs associated with composition operators is the minimum additional space (in pixels) needed to contain all its interactors in a readable layout. This additional value depends on the way the composition operator is implemented (for example, if a grouping is implemented with a fieldset or with bullets). Another example is the minimum and maximum interspace (in pixels) between the composed interactors. After such considerations, it is easy to understand that each mobile presentation could contain a varying number of interactors depending on their consumption of interaction resources.

There are various differences to consider between graphical and vocal interfaces. In vocal interfaces there are several specific features that are important in order to support effective interaction with the user. For example, it is important that the system always provides feedback when it correctly interprets a vocal input and it is also useful to provide meaningful error feedback in the event of poor recognition of the user's vocal input. At any time, users should be able to interrupt the system with vocal keywords (for example "menu") to access other vocal sections/presentations or to activate particular features (such as the system reading a long text). An important aspect to consider is that sometimes users do not have an overall control of the system state, such as in graphic interfaces. In fact, short term memory can be easily disturbed by any kind of distraction. Thus, a useful technique is to provide some indication about the interface state in the application after a period of silence (timeout). Another useful technique for dealing with this problem can be the use of speech titles and welcome or location sentences in each vocal presentation to allow users to understand their position and the subject of the current presentation and what input the system needs at that point. Another important difference between speech and graphic interfaces is that the vocal platform supports only sequential presentations and interactions while the graphical ones allow concurrent interactions. Thus, in vocal interfaces we have to find the right balance between the logical information structure and the length of presentations. The analysis of the result of the reverse engineering provides useful information to understand how to organise the vocal version (for example what elements should be grouped) and then the arrangement is implemented using vocal constructs. When problems with long labels require looking up shorter synonyms then we use specific tables for them.

The main criteria of the redesign algorithm for the vocal platform are:

- Before redesign for vocal interaction, elements regarding tasks unsuitable for the vocal platform (for example, long text inputs) are removed and labels which are too long are modified (with the help of a database of terms suitable for vocal activities).

- Semantic relations among interactors in the original platform are maintained in the vocal platform, keeping interactors composed through the same composition operators in the same vocal presentation, and implementing them with techniques more suitable for the vocal device.
- Before redesign, images are removed and substituted by alternative descriptions (ALT tag).
- Implementation of interactors and composition operators will change according to the resources of the new vocal devices.
- The algorithm aims at providing a logical structure to vocal presentations avoiding too deep navigation levels because they may disorient users. To this end, only the highest level composition operators (in the case of nested operators) are used to split desktop presentations into vocal presentations.
- Composition operators that are allocated to new vocal presentations are substituted in the main vocal presentation that cannot contain them by a vocal link to the new presentation, which contains the first interactor associated with the composition operator.

7. Device Discovery

Device discovery is another important aspect in migratory user interface environments. It allows the system to notice potential migration-source and migration-target devices. The technology that enables this discovery in our migration architecture is a custom discovery protocol explicitly created to handle service discovery, service description, and service state monitoring tasks at Internet Protocol level. The protocol is implemented as a module in the server, and as a client application on each of the devices. The design of this protocol provides multicast mechanisms for peer-to-peer device and service discovery, using well-known UDP/IP mechanisms. Once the module and the user devices have discovered each other, they make use of reliable unicast TCP/IP connections to provide service description and service monitoring primitives to the system. The implementation and use of the description capabilities of our discovery protocol provides means for the system to gather a rich set of information from the devices that are present in the environment, regarding both their interaction and communication capabilities as well as their general computational ones.

The device discovery of our migration infrastructure is based on multicast datagrams using UDP/IP. When one device enters the network, it issues a discovery (DIS) message to a well-known multicast address to which all existing devices must subscribe. Subscription to multicast groups is handled by the network equipment and usually limited to the current subnet. In any event, when this discovery (DIS) message is received by the other network peers, they respond by sending a unicast message (RES) to the issuer of the discovery (DIS) request. This way, all active migration clients and servers found in the network discover each other via a minimal exchange of network packets. At this point, the discovery algorithm changes depending on the nature of the migration application running on that particular device. If the device is to act as a server, then a unicast description request (DES) will be sent to all the

discovered devices, requesting them to discover themselves by sending an XML description file to the server device. This description will be saved in the server for future reference. If, on the other hand, the device is to act as a client to the migration infrastructure, then it will wait until a server is found and a description file is requested by it. Once this state is reached, the system is configured and fully functional. In order to guarantee consistency, keep-alive (DIS) multicast messages are sent by all parties with a periodicity of 1 second. When no keep-alive is received from a given device for a configurable amount of time, the device is deemed as having departed the network and no further communications are allowed with it until proof of its re-activation is gathered, in the manner of new multicast keep-alive messages. The periodicity of the keep-alive datagrams is low enough to ensure no considerable network traffic will be generated.

In order to supply the migration server with information about the devices that are present in the environment, XML-based device description files have been used. These files include all the relevant information the migration server needs in order to identify the device and find out its capabilities and features. The description files also provide an efficient way to monitor the state of the devices available in the environment by allowing the migration server and other interested parties to subscribe to certain events in order to receive a notification message each time a device state-change occurs. This has improved the support for automatic migration through richer and more accurate monitoring of the environment and the user interactions with it. The use of our custom discovery protocol in combination with these XML description files has proven to be successful and addresses our objectives and goals. In our new discovery-enabled prototype, users do not need to manually specify the IP address of the migration server, the middleware automatically discovers it for them. Neither do they need to login their personal interaction device into the migration environment, as their devices are automatically detected by the system both when connecting to it and when disconnecting from it. Thus, the new migration architecture offers an increased robustness and better consistency over the previous versions of our migration prototype, without increasing the prototype's complexity from the development point of view, and keeping things transparent and simple for the end user.

8. Example Application

This section presents an example application of our migration environment. In the example, John is planning to go on vacation and would like to buy a new camera. He decides to search for a bargain on an online auction website and accesses the "e-Bid" website through his desktop PC. He checks the information about the available cameras by looking at item descriptions and prices. He finds an interesting offer and accesses the page containing information about the selected camera. He then decides to bid on this item, but discovers that he has to register first, and thus starts filling out the long registration form required by the website. Suddenly, the alarm on the desktop reminds him about a meeting which is going to take place this afternoon at his office, so he has to leave. The form is too long to be completed in time, thus he quickly

migrates the application to his PDA and goes out walking towards his car, while he continues filling in the form.

Figure 4 shows the desktop form partially filled in and how it is transformed for the mobile device if migration is triggered at that time. After the reverse engineering phase, the original desktop interface is transformed into a composition of concrete/abstract objects. Then, the composition operators (indicating a semantic relationship among the objects involved) and the number and cost of the interactors of the various presentations are considered in order to redesign the original desktop page for the new device. As a result of this process, the long desktop form is split into two pages or presentations for the PDA. Additional connections are inserted for handling the page splitting and allowing the user to navigate from/to the two pages. After completing the registration, John, with his PDA, places a bid on the camera before the auction ends in a matter of a few minutes, and then he is redirected to the page containing the camera description, where he can monitor the status of his bid.

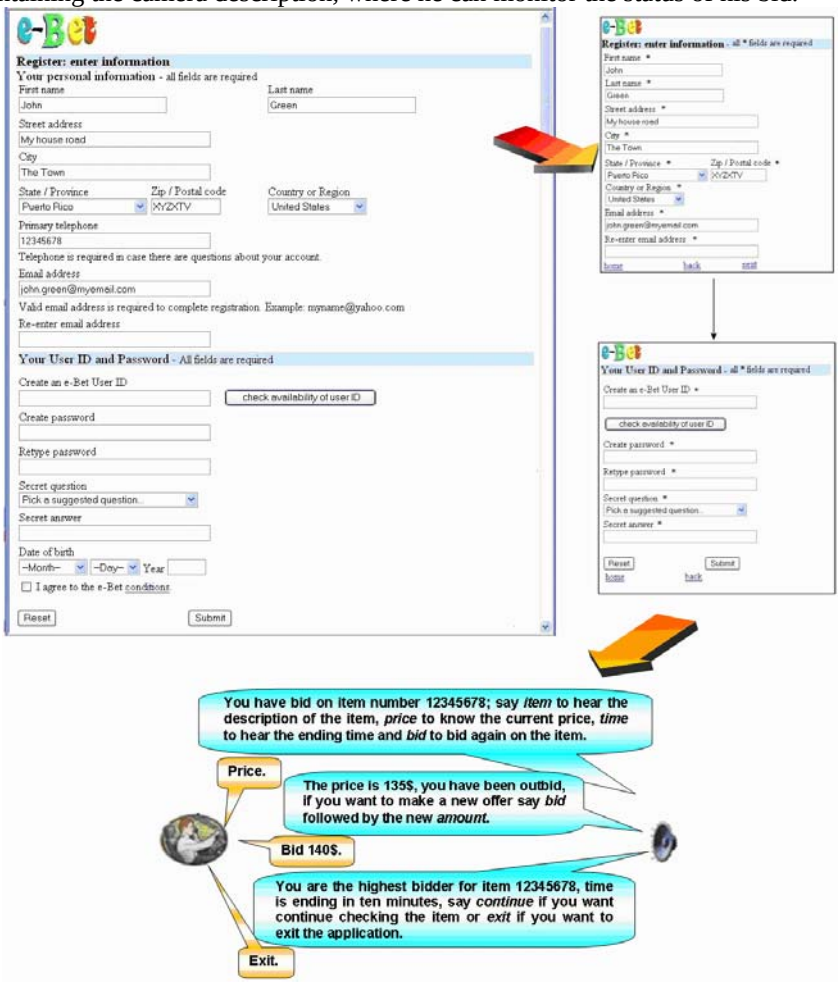

Fig. 4. Example of migration through different devices.

While he is keeping an eye on the bidding, he enters his car and the application automatically migrates from the PDA to his mobile phone and can now be accessed through the vocal interface thanks to the wireless connection to the car voice system. Indeed, the environment carries out a redesign of the application for the new platform (vocal) and therefore identifies how the user interface design should be adapted for the new platform. Moreover, by identifying the point where the user was before the migration was activated, the environment is also able to update the new user interface with the data gathered from user so far, allowing the user not to start from scratch but continuing the interaction from the point where it was left off. Indeed, the speaker says "you have bid on item number 12345678 say *item* to hear the description of the item, *price* to know the current price, *time* to hear the ending time and bid to *bid* again on the item". John says "*price*". The car voice system replies "the *price* is 135 $, you have been outbid, if you want to make a new offer say "*bid* followed by the new *amount*", "*bid 140 $*", "you are the highest bidder for item 12345678, time is ending in ten minutes, say *continue* if you want continue checking the item or *exit* if you want to exit the application". John has reached the maximum amount of money he is willing to spend for the camera and thus says "*exit*" and continues driving towards his office. He will access the e-Bid website later on to check if he won the auction.

9. Conclusions

This paper has presented an environment supporting migration of user interfaces, even with different modalities. The implementation of the system, from the architectural point of view, follows a service-oriented architecture, with its corresponding benefits, both for the end user and for the developers of the system. The user interfaces that can be generated by the system are implemented using XHTML, XHTML Mobile Profile, VoiceXML and Java for the Digital TV. 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.
Our environment is able to reverse engineer, redesign, and migrate Web sites implemented with XHTML and CSS. All the tags of these standards can be considered and manipulated.

An algorithm has been identified for handling code in Web pages that is implemented in different languages, for instance applets and Flash applications, which are generally identified by <object> tags with further attributes in their specification (e.g. title, etc.). The algorithm tries to map applets/flash elements to concrete (simpler) elements, taking into account the provided specification of such elements and also the capability of the target platform considered. For instance, if an applet/flash element has no siblings and there is no further data in its specification, the algorithm simply removes the corresponding node, otherwise it might map it into a e.g. textual string whose label is derived from the title attribute within the specification of the flash/applet code.

We are now working on a new version of our environment which is able to generate Microsoft C# -based user interface implementations, even supporting different modalities, such as gestural interaction.

We have conducted a number of preliminary user studies with the objective of analyzing the user perception of interface migration and we have found the first results to be encouraging. However, we plan to carry out a further user study in the nearby future to better measure the effectiveness of the resulting interfaces.

Acknowledgments

We thank Zigor Salvador for his help in the implementation of the dynamic device discovery part.

10 References

1. Balme, L. Demeure, A., Barralon, N., Coutaz, J., Calvary, G.: CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable and Plastic User Interfaces. In: *Proceedings EUSAI* '04, LNCS 3295, Springer-Verlag, 2004, 291-302.
2. Bandelloni, R., Berti, S., Paternò, F.: Mixed-Initiative, Trans-Modal Interface Migration. *Proceedings Mobile HCI'04,* Glasgow, September 2004, LNCS 3160. Springer-Verlag 216-227.
3. Bouillon, L., and Vanderdonckt, J.: Retargeting Web Pages to other Computing Platforms. In: *Proceedings of IEEE 9th Working Conference on Reverse Engineering* (WCRE'2002) Richmond, Virginia, 2002, IEEE Computer Society Press, 339-348.
4. Gajos K., Christianson D., Hoffmann R., Shaked T., Henning K., Long J., Weld D. S.: Fast and robust interface generation for ubiquitous applications. In: Proceedings UBICOMP'05, pages 37–55. Springer Verlag, September (2005), LNCS 3660.
5. Limbourg, Q., Vanderdonckt, J.: UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence, in Matera, M., Comai, S. (Eds.), Engineering Advanced Web Applications, Rinton Press, Paramus (2004).
6. Luyten, K., Coninx, K. Distributed User Interface Elements to support Smart Interaction Spaces. In: IEEE Symposium on multimedia. Irvine, USA, December 12-14, (2005).
7. Mori G., Paternò F., Santoro C.: Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions. In: IEEE Transactions on Software Engineering August (2004), Vol 30, No 8, IEEE Press, 507-520.
8. Nichols, J. Myers B. A., Higgins M., Hughes J., Harris T. K., Rosenfeld R., Pignol M.: Generating remote control interfaces for complex appliances. In: Proceedings ACM UIST'02 (2002) 161-170.
9. Paganelli, L., and Paternò, F.: A Tool for Creating Design Models from Web Site Code. In: International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing 13(2), (2003), 169-189.
10. Paternò, F.: Model-based Design and Evaluation of Interactive Applications. Springer Verlag, ISBN 1-85233-155-0, 1999.
11. Ponnekanti, S. R. Lee, B. Fox, A. Hanrahan, P. and Winograd T. ICrafter: A service framework for ubiquitous computing environments. In: Proceedings of UBICOMP 2001. (Atlanta, USA, 2001). LNCS 2201, ISBN:3-540-42614-0, Springer Verlag, pp. 56-75.