Journal of Web Engineering, Vol. 7, No.3 (2008) 220-238

A MIGRATION PLATFORM BASED ON WEB SERVICES FOR MIGRATORY WEB APPLICATIONS

FABIO PATERNÒ CARMEN SANTORO ANTONIO SCORCIA
ISTI-CNR, Via G.Moruzzi, 1 Pisa (ITALY)
{fabio.paterno, carmen.santoro, antonio.scorcia}@isti.cnr.it

Received November 11, 2007
Revised March 9, 2008

In this paper we present a new environment for supporting Web user interface migration through different devices. The goal is to furnish user interfaces that are able to migrate across different devices, 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 migration environment supports the automatic discovery of client devices and its architecture is based on the composition of a number of software services required to perform a migration request.

Key words: 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). Many interesting issues are related to such new environments, even for Web applications. Indeed, the W3C Consortium has started a new working group on Ubiquitous Web Applications, with the focus to extend the Web to all kind of devices including effectors and sensors (http://www.w3.org/2007/uwa/). In this area, one important goal is to support continuous task performance, which 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. 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 paper, after discussing related work, we provide an overall introduction of the environment, followed by a discussion on device discovery mechanisms used in the environment. Next, we focus 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. 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] for representing the aforementioned logical descriptions. The research in this area has mainly focused on how to help designers efficiently obtain different versions of an application that adapt to the various interaction features of the different devices, but also 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 signal 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 interface migration. A general reference model for user interfaces aiming to support migration, distribution and adaptation to the platform is proposed in [1], without providing specific architectural solutions for such issues. Our system, in particular, proposes a concrete software architecture that is able to support migration of user interfaces, associated with Web applications hosted by different application servers, among automatically discovered devices.

3 Overall Description of the Environment

The main characteristics of migration are: device change, adaptation, and continuity. The basic idea is that people would like to freely move and still be able to continue to perform their tasks and thus the interactive part of an applications should be able to follow them and adapt to the changing context of use.

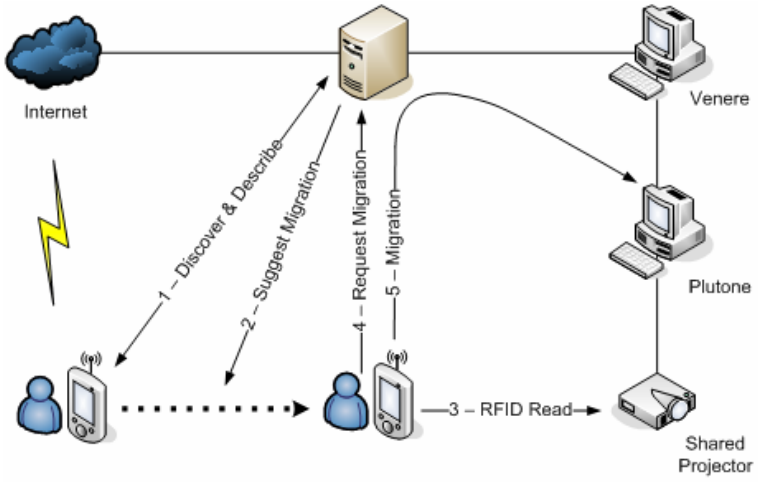

Figure 1 Migration Scenario.

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. This is performed thanks to a small piece of software that has to be loaded onto the devices and through which the clients

are able to announce their presence in the migration environment. Then, the environment may suggest migration to a nearby device (automatic migration), or the user can explicitly request a specific migration. In the latter case, the user can select the target device either from a list of available devices or by pointing it with a RFID reader. Lastly, the user interface migrates to the target device with the mediation of the migration server. In the example in Figure 1, we consider a user-initiated migration, with a user using a PDA equipped with an RFID reader, which detects the tagged projector, which is associated with a PC that will be considered the target device.

Our migration environment is based on a service-oriented architecture involving multiple clients and servers: the architecture aims at providing interoperability between the different services, which can be also combined for delivering composite services, as it happens in the migration support. We assume that the desktop version of the considered applications already exists in the application servers. In addition, we have a migration platform, which is composed of a proxy service and a number of specific services and can be hosted by either the same or different systems.

Figure 2 shows the six main web services that have been identified to compose the migration platform:

- the Discovery Manager, which includes the functionalities for discovering the available devices and update the device list accordingly;
- the Migration Manager is the core of the system: it handles the communication with the other modules, and also includes proxy functionalities;
- the Reverse Engineering, is in charge of reversing the desktop implementation into a logical user interface description;
- the Semantic Redesign module, which transforms the logical description of the user interface designed for the source platform into a logical description of the user interface for the target migration platform;
- the State Mapper, which updates the final user interface with the values of the current state, which have been saved at the time the request of migration occurred;
- The UIGenerator, which reifies the logical concrete description into an implementation language for the target platform.

Figure 2 represents with UML interaction diagrams the main communication among the migration services. The process starts with the source and target devices notifying their presence to the Discovery Manager, which is in charge of discovering the available devices and updating the list of devices accordingly. Indeed, in order to allow for a good choice of the target device, information about the devices that are automatically discovered in the environment is saved. Such information mainly concerns device identification and interaction capabilities and, on the one hand, it enables users to choose a target migration device with more accurate and coherent information on the available targets and, on the other hand, it enables 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.

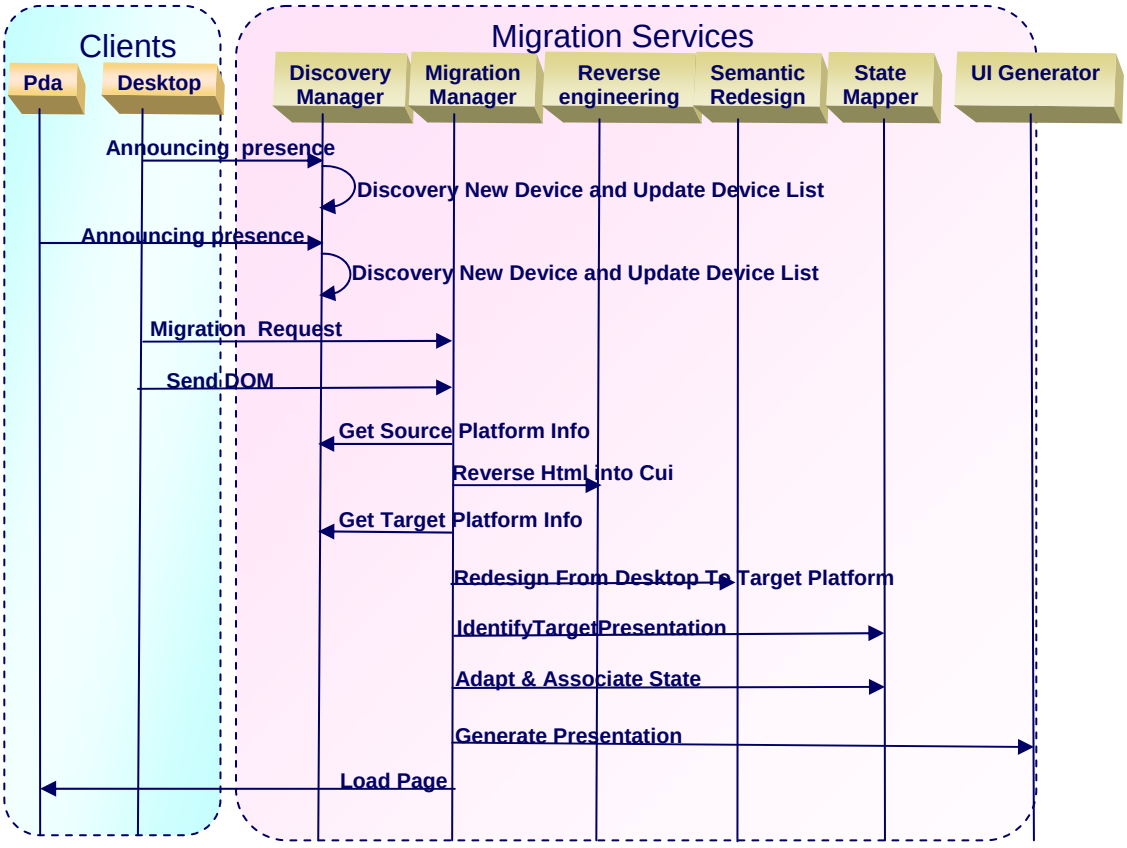

Figure 2 Main communication among the migration services.

The migration clients are supposed to access the various applications through the proxy available within the Migration Server. This means that this module is in charge of intercepting the clients' request of accessing a page, retrieving such a page from Internet and saving it locally together with the referred entities (images, CSS files, etc.). Afterwards, the Migration Server receives from the source device the request for migration (which specifies the source device, the target device, and the URL of the page that has to be migrated), and it triggers the sequence of actions needed for fulfilling such a request.

The Migration Server, after receiving the request of migration by the source device, interrogates the target device asking about its availability/willingness for accepting a migrating user interface: if the migration is accepted, the environment detects the state of the application modified by the user input (elements selected, data entered, …) and identifies the last element accessed in the source device. This is obtained by JavaScript functions that are automatically inserted by the proxy server and are in charge of collecting the information that describes the state of the migrating page by accessing its

DOM. The information is collected into a string formatted following a XML-based syntax and submitted to the server together with the IP of the target device. This information is sent to the server through an AJAX script. The reason for this is that only the application running on the client device can access the DOM and the AJAX callback can transmit the data without requiring any additional explicit action from the user.

Then, the Migration Manager gets information about the source device and builds the logical descriptions of the corresponding desktop page by invoking the Reverse Engineering service of our system. At this point, the Migration Manager asks the Discovery Manager information about the target device in order to understand for which platform the redesign process has to be carried out. Indeed, the result of the reverse engineering process, together with information about target platform is used as input for the Semantic Redesign service, in order to perform a redesign of the user interface for the target platform. This part of the migration environment transforms the logical description of the desktop version into the logical description for the new platform. This solution allows the environment to exploit semantic information contained in the logical description and obtain more meaningful results than transformations based only on the analysis of the specific implementation language used for the final UI.

Afterwards, the Migration Manager has to identify on the target device the logical presentation to be activated. In this case the problem to solve is that in some cases there is no one-to-one mapping between the pages in the source and the target device: for example one desktop page can be mapped onto multiple mobile pages in case of a desktop-to-mobile migration. In such cases the page activated on the target platform is the one that supports the last user input in the source device in order to allow the users to continue from the point they left off. Once such a presentation has been identified on the target device, the Migration Manager asks the State Mapper to adapt the state of the concrete user interface with the values that have been saved previously. Then, once the concrete user interface adapted with the new values has been obtained, the reification of such a logical description towards the final user interface for the target platform is delivered by the UIGenerator module and finally, the resulting page is sent to the target device.

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 mobile 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, in addition to a spatial threshold used to indicate when the user is sufficiently close to trigger a migration, 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 by just 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 Device Discovery

Device discovery is another important aspect in migratory user interface environments. It allows the system to identify potential migration-source and migration-target devices. The technology that enables this discovery in our migration architecture is a custom discovery protocol explicitly created at the Internet Protocol (IP) 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 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 message is received by the other network peers, they respond by sending a unicast message to the issuer of the discovery request. In 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 will be sent to all the discovered devices, requesting them to present 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 multicast messages are sent by all parties every 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, since 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.

5 The User Interface Logical Descriptions Supported

Our migration environment 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). In this case the semantics refers to the type of effect that the user interface element is expected to support.

The concrete description represents platform-dependent descriptions of the user interface and it 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. To provide an example of what such abstraction levels represent, we can consider the task of setting the temperature in a room. Such task is an interaction editing task (since it allows for modifying a value), which can be supported in abstract terms by an interactor through which it is possible to edit a quantity within a predefined range (numerical_edit_in_range). At the concrete level, depending on the considered platform, such interactor can be rendered through a vocal command (on a vocal platform) or a graphical gauge (on a graphical platform).

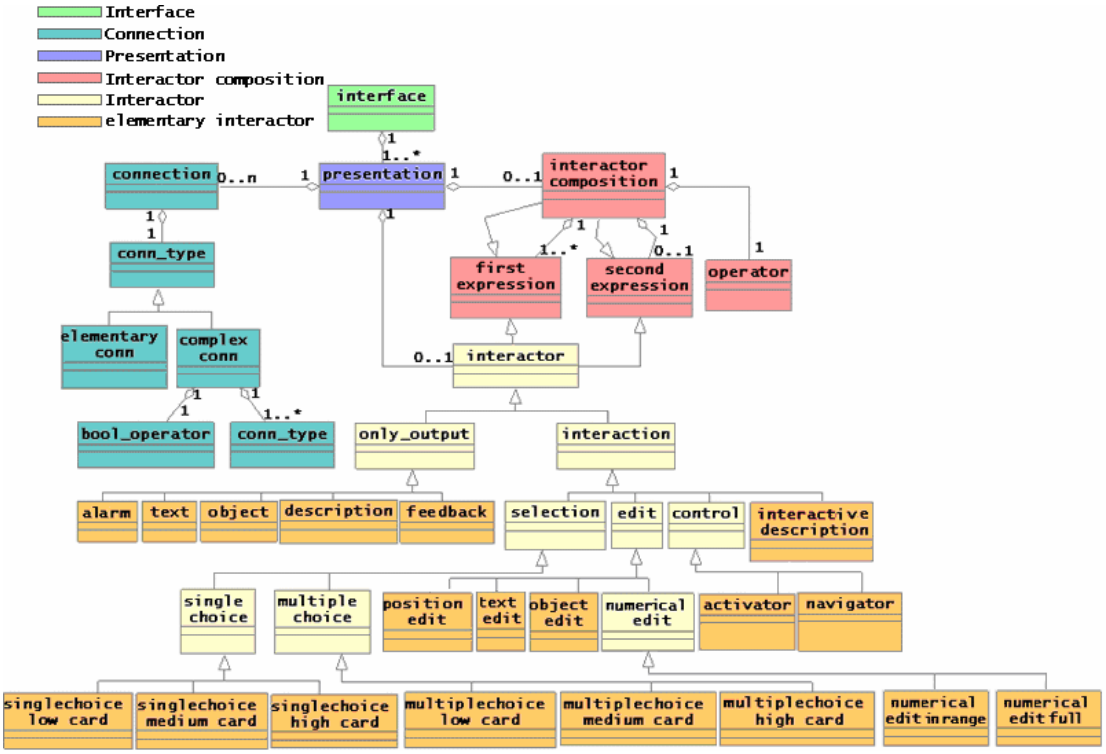

Figure 3 The specification of the abstract user interface language used in our migration approach

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. Thus, connections are used to define the navigation model. 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.

6 From Web Pages to Logical Descriptions

The main purpose of the reverse engineering service is to capture the logical design of the interface (in terms of basic tasks and ways to structure the user interface), which is then used to drive the generation of the interface for the target device. In order to support the automatic redesign for migration purposes, we need to access the relevant abstract descriptions. Thus, the reverse engineering module of our migration environment is able to take Web pages and then provide the necessary corresponding abstract logical descriptions.

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. The reverse transformation can reverse Web sites implemented in (X)HTML, including those which have associated CSS stylesheets. A Web site is reversed considering one page at a time and reversing it into a concrete presentation. When a page is reversed into a presentation, its elements are reversed into different types of concrete interactors and combination of them by recursively analysing the DOM tree of the X/HTML 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.

In order to get the DOM tree, well formed X/HTML files are needed. However, since many of the pages available on the Web do not satisfy this requirement, before reversing the page, the W3C Tidy parser is used for correcting features like missing and mismatching tags and returns the DOM tree of the corrected page, which is analysed recursively starting with the body element and going in depth. 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. Each logical presentation can contain both elementary interactor objects and composition operator elements, allowing for combining objects in structured expressions. The composition operators can contain both simple interactors and multiple composition operators. Our reverse engineering transformation identifies the corresponding logical basic 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.

Three basic cases can be identified (the algorithm is summarised in Figure 4) depending on the node analysed:

- The XHTML element is mapped onto a concrete interactor. This is a recursion endpoint. The appropriate interactor element is built and inserted into the XML-based logical description. For example, DOM nodes corresponding to the tags <img>, <a> and <select> cause the

generation of concrete objects of type respectively image, navigator and selection. The properties of the objects in the source page considered are also used to fill in the attributes of the corresponding concrete user interface elements, out of the peculiarities used in the specific final language used for implementing the page of the source device. For instance, the italic attribute of a text concrete element is set to true although in the HTML implementation it might appear as either <i> or .

- The XHTML node corresponds to a composition operator. In this case, after creating the proper composition element, the function is called recursively on the XHTML node subtrees. The subtree analysis can return both elementary interactors and composition of them. In both cases the resulting nodes are appended to the composition element from which the analysis started. For example, the node corresponding to the tag <form> is reversed into a Relation composition operator, <ol> into an Ordering, <ul> into a Grouping. Depending on the considered node to be reversed, appropriate attributes are also stored in the resulting element at the concrete level (e.g. typical HTML desktop <ul> lists will be mapped at the concrete level in a grouping expression using bullets listed following a vertical positioning).
- The node does not require the creation of an instance of an interactor in the concrete specification (for example, if in the Web page there is the definition of a new font, no new element is added in the concrete description). On the one hand, if the node has no children, no action is taken and we have a recursion endpoint (this can happen for example with line separators like
 tags). On the other hand, if the node has children, each child subtree is recursively reversed and the resulting nodes are collected into a grouping composition which is in turn added to the result.

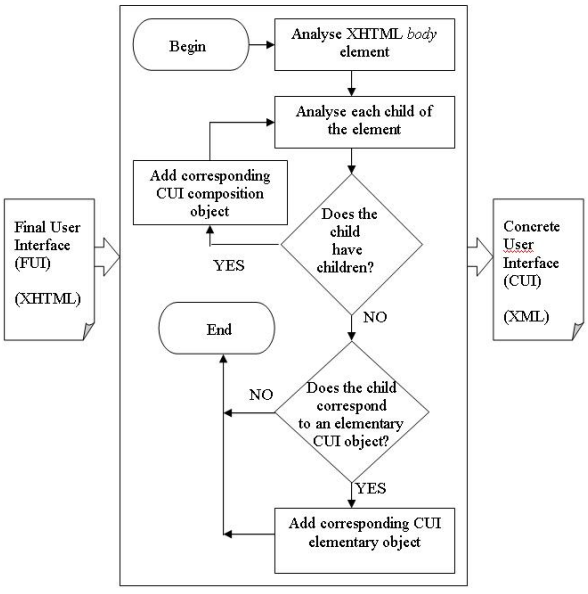

Figure 4: The Reverse Engineering Algorithm.

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 developed 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 aims 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 can map it into a e.g. textual string whose label is derived from the title attribute within the specification of the flash/applet code.

7 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 Service analyses the input from the desktop logical descriptions and generates an abstract and concrete description for the target platform 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 different platform we need to consider semantic information and the limits of the available resources. Indeed, if we only consider the physical limitations we may end up dividing large pages into small pages which are not meaningful. Previous approaches to semantic redesign [2] were not able to dynamically calculate the cost sustainable by the target device and that of the Web pages under consideration, thus providing rather limited results in terms of adaptation.

Figure 5 shows the various phases of semantic redesign in the case of desktop-to-mobile transformations. There are three main phases: transforming the desktop logical interface into a mobile logical interface, calculating the cost of such a new user interface in terms of resources, and splitting the logical interface into presentations that fit the cost sustainable by the target device. The first phase mainly changes the concrete elements of the desktop description into concrete elements that are supported by the mobile platform (for example a radio-button with several elements can be replaced with a pull-down menu that occupies less screen space). In this phase the images are 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 would be too small or the mobile device does not support them. Text and labels can be transformed as well, since they may be too long for the mobile devices. In converting labels we use tables able to identify shorter synonyms.

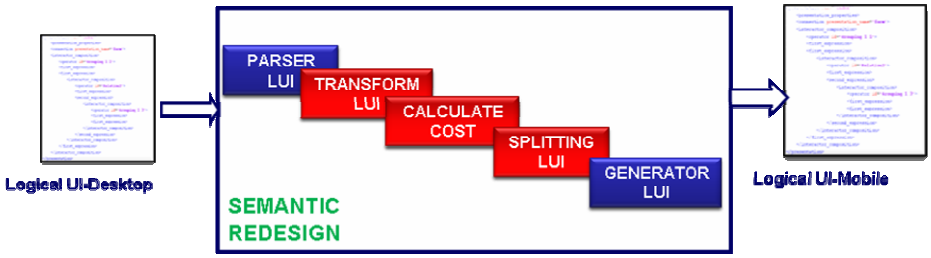

Figure 5 Desktop-to-Mobile Semantic Redesign

In order to automatically redesign a desktop presentation for a mobile device 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 smaller ones that 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 together through some composition operator in the same final presentation, thus preserving the communication goals of the designer and obtaining consistent interfaces. In addition, the page splitting requires a change in the navigation structure with the need for additional navigator interactors that allow for accessing the newly created pages.

The algorithm for calculating the costs and splitting the presentations accordingly is based on the number and cost of interactors and their compositions. The cost is related to the interaction resources consumed, e.g.: number of pixels of images, font sizes. After the initial transformation, which replaces the desktop concrete elements with mobile concrete elements (for example, a text area for the desktop platform could be transformed into a simpler text edit on the mobile platform), the cost of each presentation is calculated. If it fits the cost sustainable by the target device, then no other processing is applied. Otherwise, the presentation is split into two or more pages following this approach: the cost of each composition of elements is calculated. The one with the highest cost is associated to a newly generated presentation and it is replaced in the original presentation with a link to such a new presentation. Thus, if the cost of the original presentation after this modification is less or equal the maximum cost that can be supported by a single mobile presentation then the process terminates, otherwise the algorithm is recursively applied to the remaining composition of elements. In case of a complex composition of interface elements that cannot be entirely included in a single presentation because of its high cost for the target device, the algorithm aims to equally distribute the interactors amongst two corresponding presentations of the mobile device.

More specifically, the transformation follows the subsequent main criteria:

- 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 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.
- 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 connections of the resulting interface should include the original ones and add those derived from the presentation split.

With reference to the last bullet (connections that might be added after a presentation split), 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.

As we said before, in the transformation process we take into account semantic aspects and the cost in terms of interaction resources of the elements considered. The cost that can be supported by the target mobile device is calculated by identifying the characteristics of the device through the user agent information in the HTTP protocol, which can be used to access more detailed information in its UAProfile, when available. 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.

If we consider redesigning for the vocal platform, there are some specific issues to be considered. For example, in vocal platforms, 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.

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

- During 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. For example, grouping of elements in the case of vocal interaction can be achieved by insert a delimiting sound or pause or using some keywords or a specific volume of the voice synthesizer.
- During the redesign phase, images are removed and substituted by alternative descriptions (ALT tag).
- 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.

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

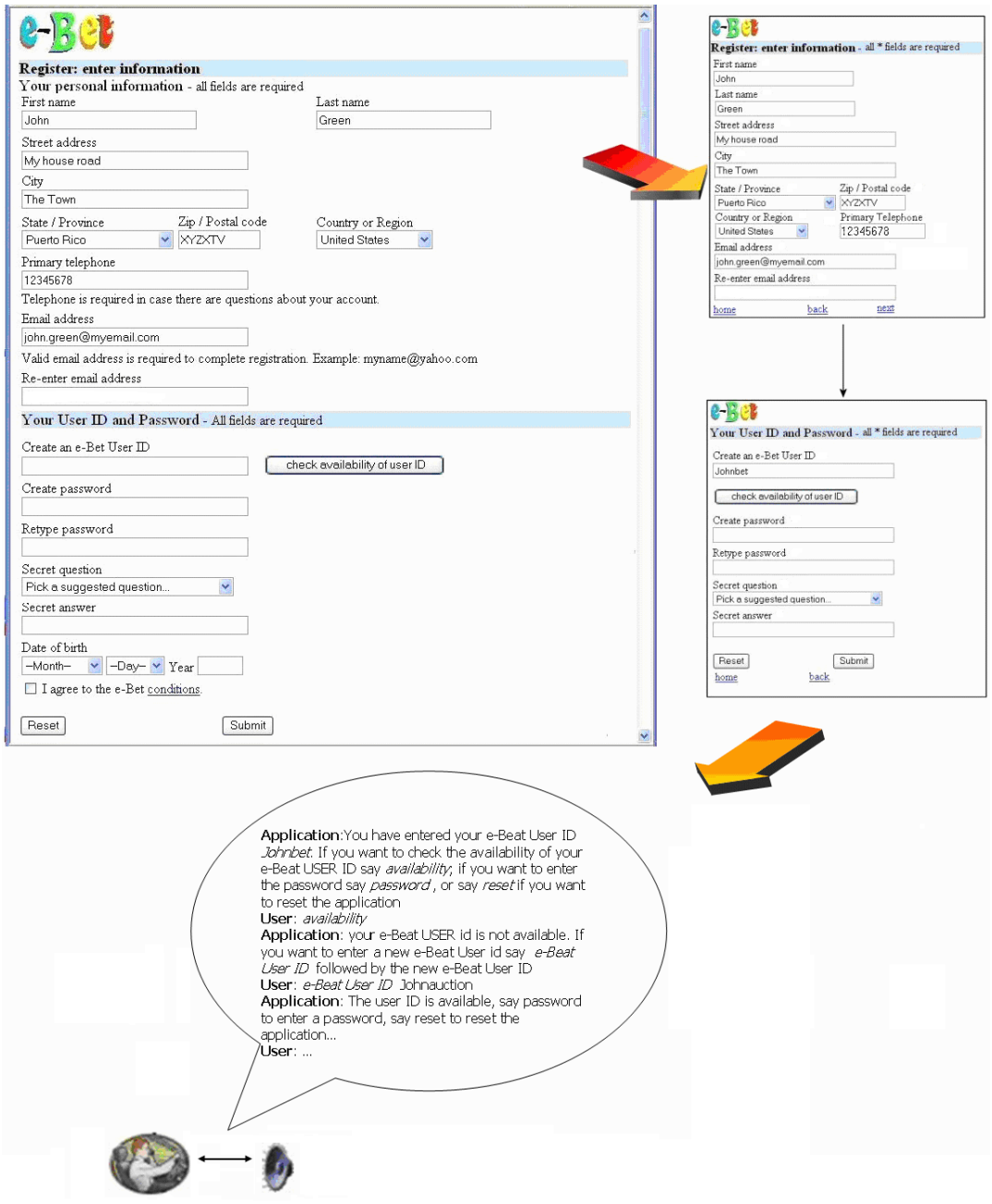

Figure 6 Example of migration through different devices.

Figure 6 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 allowing the user to navigate from/to the two pages. Since the last input in the desktop interface was the email address then the mobile page supporting entering this piece of information is the one activated first in the mobile device. 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.

In the example shown in Figure 6, two migrations occur: in the first migration (from desktop to PDA), the user partially completes the form, by providing only some information. The last information provided when the first migration is requested is the email address. As soon as the migration is activated, the migration server finds where the form filling was interrupted and then shows to the user the PDA presentation with the form partially filled (see PDA presentation on the top). Afterwards, the user continues interacting with the application: he re-enters the email address, as requested, and moves on the next PDA presentation (see PDA presentation on the bottom), by providing further information (e-Beat User ID). Still interacting with the PDA the user provides the user id (Johnbet, see bottom PDA presentation), and then a migration is requested towards to a vocal device. Indeed, 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. Thus, 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, after having updated the new user interface with the data gathered from the user so far, the environment identifies the point where the user was before the migration and allows not starting from scratch but continuing the interaction from the point where it was left off. Indeed, the speaker says “You have entered your e-Beat User ID Johnbet. If you want to check the availability of your e-Beat USER ID say *availability*; if you want to enter the password say *password* , or say *reset* if you want to reset the application”. The user replies saying “availability” and then the application makes aware the user of the fact that the user id is not available. Then the user is asked to provide a new ID. This time the ID is available and then John can continue his interaction with the auction system while driving towards his office.

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. Our prototype of the migration platform has been implemented in Java, with Web services used to communicate among the various parts.

The migration platform supports migration of existing Web applications, which can be developed with any authoring environments, and dynamically generates the user interfaces for different platforms by transforming and adapting the existing content. In addition, support for preserving the state of the user interface resulting from the user input is provided.

The user interfaces that can be generated by the system are implemented using XHTML, XHTML Mobile Profile, VoiceXML and Java for the Digital TV. We are now working on a new version of our environment which is able to generate also -based user interface implementations in other languages (for example Microsoft C#), even supporting different modalities.

The limitation of the current solution is that it is able to reverse engineer only Web pages implemented in (X)HTML. It can maintain JavaScripts, or functionality implemented in other languages, across the various versions automatically generated but it is not yet able to transform these parts depending on the target device capabilities. Another potential limitation is that our environment is not able to correct pages that were wrongly designed. Thus, when the input original interface (as it may happens nowadays) is quite complex and logically or semantically pertaining elements are not structurally connected, the splitting on different pages elements may not be optimal.

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.

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.

Acknowledgements

We thank Renata Bandelloni and Zigor Salvador for their help in the implementation of an early version of the migration infrastructure.

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

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.