Journal of Visual Languages and Computing 24 (2013) 37–52

Contents lists available at SciVerse ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Interactive customization of ubiquitous Web applications ☆

G. Ghiani, F. Paternò *, C. Santoro

CNR-ISTI, HIIS Laboratory, Via Moruzzi 1, 56124 Pisa, Italy

ARTICLE INFO

Article history:
Received 1 August 2010
Received in revised form
31 May 2012
Accepted 25 October 2012
Available online 2 November 2012

Keywords:
Ubiquitous Web user interfaces
End user customization
Migratory interfaces

ABSTRACT

Ubiquitous environments pose new challenges for end users who often need to access their applications from various devices. In this paper we present a solution that allows users to easily customise and migrate interactive web applications starting with an existing desktop version. This is obtained through an intelligent infrastructure that enables users to select the relevant part of an interactive Web application in order to create a mobile version and migrate it.

1. Introduction

Our life is already a multi-device experience. Recent studies [5] have demonstrated that many users have various interaction devices available at home and in their working environment. Such devices, whose number is steadily increasing, can be grouped together according to the interactive resources they share, so as to identify a number of *platforms*, such as desktops/laptops, TV receivers with set-top box, smartphones, interactive walls, devices supporting vocal interaction, personal gaming devices. One main issue that emerges from this multi-device scenario is how users can proficiently exploit it. One possibility is to provide people with the means to seamlessly and opportunistically switch between devices anytime, even while they freely move about, and still continue to accomplish their tasks.

Another clear trend regards Web applications: not only are they frequently and extensively used, but also their interactivity and complexity are steadily increasing. Their content is becoming more and more dynamic, able to take information from a variety of Web services, and their interactivity is ever greater, generally obtained through the use of various types of scripts. Such trends offer many new interesting opportunities to end users, who need novel tools able to address the mentioned issues (large number of devices to manage, increased complexity and interactivity of Web applications), otherwise they risk getting disorientated and frustrated in today's huge technological, multi-platform offering. One example of these novel tools are those supporting migratory user interfaces. They allow users to interact with one application and at some point decide to change to a different device and continue the interaction from the point in which they left off. An example scenario is a user at home or in the office who has to leave a desktop position and would like to continue the interactive session while moving and using the mobile device. A number of solutions have been identified to address such scenarios (see for example [12]). However, they usually require that the application be implemented using specific toolkits in order to make it migratory. Therefore, there is a need for more general solutions that do not impose the use of specific development environments and should be able to support the generality of Web applications. In addition, since Web pages are becoming more and more articulated, when users decide to perform a desktop-to-mobile migration, they should also be able to select the relevant parts of the application to migrate. To some extent, this means that the users should be able to dynamically edit the (mobile)

☆ This paper has been recommended for acceptance by S Levialdi.

* Corresponding author. Tel.: +390503153066.
E-mail address: fabio.paterno@isti.cnr.it (F. Paternò).

1045-926X/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jvlc.2012.10.005

version of the Web application to be activated on the target device. Obviously, such users are unlikely to have programming skills, or know script and mark-up languages for the Web. Thus, they should be enabled to create such mobile versions in an intuitive and interactive manner.

In this paper we describe our solution to this problem: it allows users to interactively select the parts of a Web application to migrate and dynamically compose them into a mobile version, which is then activated on the target device indicated by the user. This work is based on some preliminary results (described in [8]) of a solution that suffered from various usability issues and was able to support migration only towards a limited set of mobile devices. In this paper, we present a novel solution that allows users to select the parts of interest by direct manipulation and to migrate them to any target mobile device. We moreover report on a usability evaluation which confirmed that the improvements achieved received positive comments by end users.

In the paper, after discussing related work, we introduce an example scenario in which we show how users can dynamically create a mobile version of a widely known Web application (Wikipedia) by using the proposed solution. We then describe the architecture of the environment, which makes such scenarios possible, and report on a user test. Lastly, some conclusions along with indications for future work are set out.

2. Related work

In recent years, various types of networked devices have started to proliferate, such as smart phones and set-top boxes. Researchers have observed that this high device variability often represents an issue because of their different interaction resources and the parallel continuous increase in application content complexity.

One of the first approaches to providing seamless access in multi-device environments was Pick-and-Drop by Rekimoto [17] in which the user was enabled to move pieces of information from one device to another through intuitive interactions. However, that approach did not support the possibility of moving interactive functionalities from one device to another.

The migration of interactive applications has been addressed through various types of approaches. One approach is pixel replications, an example is provided by WinCuts [19]. In that tool users can replicate arbitrary regions of existing windows across various devices. That approach has limitations in preserving the state of the underlying interactive applications across multiple devices. Kozuch and Satyanarayanan [10] proposed a migration solution based on the encapsulation of the volatile execution state of a virtual machine, but this worked only for migration of applications among systems with similar resources.

The access to encoded content, such as audio or video files, embedded on a Web page is a typical example of task that can raise some issues when the device has to properly interpret and present the associated information. Adaptation of complex content to mobile devices has been considered since the introduction of the first networked handhelds. In 1996, Fox et al. [7] tackled this issue by studying a proxy-based architecture for transparently providing adapted content to client devices. Indeed, they identified and proposed three design principles fundamental for addressing client variation effectively: (i) datatype-specific lossy compression mechanisms can achieve better results than "generic" compressors: since their decisions about what information to neglect are based on the semantic type of the data, they provide more effective adaptation mechanisms than their typeless counterparts; (ii)it is computationally feasible to perform "on the fly" adaptation rather than relying on a set of precomputed representations, although there is some latency associated with generating the representation; (iii)there are various (e.g. technical and economic) reasons to push complexity away from both clients and servers, by therefore doing adaptation at an intermediate proxy that has access to substantial computing resources and is well-connected to the rest of the Internet. They proposed a proxy architecture based on the above design principles that has the ability to adapt dynamically to changing network conditions. Adaptation is performed by proxies that host a series of dedicated "distillers" (long-lived processes which perform distillation and refinement on behalf of one or more clients).

Our proposal also exploits a proxy server that acts as an intermediate module between the application server and the user client, but in addition our server includes scripts in the accessed Web pages in such a way to enable their migration through various devices. In this way we can make Web applications migratory without posing any constraints on the tools originally used for their development. In addition, as detailed in the following sections, the platform we propose also allows the users to explicitly customise the Web applications when migrating to mobiles, rather than to automatically adapt Web content.

A solution called "Multibrowsing" was presented in [9]. Multibrowsing allows users to move existing pages among multiple devices, including wall displays. Some Multibrowsing functionalities are available through simple hyperlinks that encode requests to a special servlet, and that are accessible with standard browsers. Other functionalities are only available through browsers enhanced with a specific browser plugin called MB2Go. Thus, different devices may have different roles in the multi-device environment. For example, stationary devices enhanced with a MB2Go browser plugin can push Web pages to other clients, while PDAs can only send hyperlinks to other devices. In our architecture, instead, the Web browser is the only requirement for the device to exploit all the platform capabilities. The Migration Client, which is the user control panel, is indeed implemented through standard Web technologies. Thus, in our solution, each client device can act either as a sender (source) or as a receiver (target), independently from its type. As detailed in the following sections, the same device can even be first a migration target and then become a sender in the same interaction session.

The issue of user customization of mobile application was addressed in Collapse-to-Zoom [3] through a tool that allowed users to indicate parts of Web pages to extend or collapse through pen-based gestures. However,

that work did not address the exploitation of customization techniques in multi-device contexts allowing users also to migrate the Web pages across devices.

The subject of migration and in particular the case of *partial* migration raises a number of issues, which have been addressed from different viewpoints in other work by the research community. Partial migration can be related, to some extent, to the issues connected with Distributed User Interfaces (DUI). A toolkit for deploying distributed graphical UIs is presented in [12]. It is based on a widget distributed structure composed of two main parts: one part (the "proxy" of the widget) remains stationary within the process that created the widget; the other part (the renderer) is distributed and migratory and the user can interact with it. The toolkit is based on a peer-to-peer architecture in which a multi-purpose proxy is connected to one or more engines able to render (partially or entirely) a graphical user interface. In contrast to our solution (which is basically a client-server one), this solution is characterised by a peer-to-peer architecture. Moreover, the granularity of distribution is not limited to the level of groups of UI elements but can also extend to the widget level and even down to the different components of the widget itself. In our solution we have instead opted for a distribution down to the granularity of the single interactor but not deeper (we judged such finer granularity unimportant for our goals). In addition, that solution [12] requires that the user interface be implemented using an extension of the Tcl/Tk toolkit, while we are interested in solutions that allow partially migrating any Web application developed with the standard Web languages (XHTML, CSS and JavaScript).

The issue of distributing a user interface onto multiple devices is also analysed in [16], with particular attention to how to leverage legacy applications to obtain the new distributable features easily. Indeed, that work describes an instant messaging (IM)-based architecture aimed at reducing the additional work needed for extending a user experience to span among multiple personal devices (owned by a certain user). In order to support this, the IM-based architecture presented gives to each device an IM account, and affiliates devices to users. The server maintains a list of (personal) devices for each user in order to determine which messages an application can receive (by default, only those from the user's own devices). On each of their own devices, users run a single client that connects to the server. The various applications connect to that client, which routes the messages between them and the server. However, the relations on which this infrastructure is based include a strong limitation: it narrows the set of devices that can be interconnected to each other (only the personal devices of a user). Instead, fully migratory applications should be able to opportunistically exploit all the devices in the environment (even the devices not owned by the users but accessible to them).

Obje [6] is an infrastructure that supports building interoperable systems without having prior knowledge about them. In order to allow devices on the network to interact with each other, the approach uses a set of "meta-interfaces": they define agreements on how to achieve compatibility at runtime, rather than requiring that communication specifics be built in at development time. This approach is called "recombinant computing" (in the sense that devices and services can be arbitrarily combined with each other) and shifts some of the knowledge necessary for communication from development-time to runtime. In order to achieve interoperability in this model, three criteria must be met: (i) the interfaces supported by devices must be fixed, to ensure future compatibility; (ii) the interfaces must be minimal, to facilitate interoperability; (iii) the set of interfaces must be generic. Such fixed agreements become meta-interfaces that specify the ways in which the devices can acquire new behaviour to interact with each other. This new behaviour takes the form of mobile code that is provided by devices on the network to their peers at the time of interaction. While the motivation of the Obje architecture was to provide an infrastructure for opportunistic interoperation in device-rich environments (as in migration) this approach basically addresses problems of interoperation rather than migration.

Huddle [14] is a system that automatically generates task-based interfaces for a household in which there are multiple connected appliances. A key point of Huddle is to generate the interfaces without requiring substantial programming specific to each system of devices, but using models of the content flow within the multi-appliance system. The modelling work is divided among the manufacturers of the appliances. Huddle is implemented on top of the Personal Universal Controller (PUC) system [13], which previously generated remote control interfaces only for individual appliances. Like PUC, Huddle users access handheld devices (e.g. a PDA or a mobile device) to control the home appliances. From each device, Huddle receives an abstract specification of its functions, which also includes a description of the appliance's physical ports and internal content flows. However, Huddle does not support migration across devices, which implies starting an interactive session with one and then continuing with another from the point in which the session was left off.

An original solution to support Web session migration using dynamic 2D-barcodes was presented in [1], however it does not support the possibility for the user to select the interface parts to migrate. An automatic solution for migrating user interfaces and preserving their state has been recently presented in [4]. This approach, called Deep Shot, allows the migration of an interface (or part of it) by simply "shooting" it with a mobile phone camera. The authors claim that Deep Shot is compatible even with applications that are not Web based. However, some extra work of the developer is needed for enabling the deep shooting/posting within an existing application. Our platform currently only supports Web applications, but does not require any modification to existing Web pages or services because it exploits a proxy server that automatically adds the necessary support within the navigated pages. Firecrow [11] aims to extract code responsible for the desired behaviour of the selected Web UI control but it is limited to extraction of the basic standard controls of Web applications.

We can notice that, while some approaches have been put forward for the design of multi-device interfaces, including migration (see for example [15], where migration of only entire interfaces is addressed), none of them has shown a general solution able to work on any Web application implemented according to the W3C standards for supporting end-user driven partial interface migration from desktop to mobile systems.

3. An example application scenario

Our solution is a server-based architecture in which the requested Web pages are downloaded from the application server by a module, which works as a proxy and annotates them through the automatic inclusion of some scripts. Such client-side scripts (which, from the user's point of view, are included transparently) are needed for adding to any Web page migration capabilities, which are the functionalities for: preserving the current interactive state of the page, triggering the migration, enabling the user to interactively select relevant parts of the page for migration. This solution is totally Web based, and no specific add-on or browser plugin is needed: in order to enable the migration of a page, the user has just to open the page through the Migration Client, which is a simple Web application (implemented in HTML+JavaScript) for accessing the Migration Platform functionalities. The Migration Client also allows the user to specify the URL of the interested page, and manages the Proxy-based navigation. The functionalities added by the Proxy to the navigated Web page (i.e. page components selection, state extraction, migration, …) are activated, upon request, by the Migration Client. The Migration Client also manages incoming migration requests by asking confirmation to the user and by opening the target page.

Further information on the architecture of the system will be provided in the next sections. In this section we describe an application scenario in which our platform can be used.

In particular, for this example we consider a widely known Web application, Wikipedia. We assume that the user accesses its home page, which is structured into various sections, (e.g. "Today's featured article", "In the news", "Did you know…", as shown in Fig. 1), some of them are frequently updated. The user opens the Web page through the Migration Client, which creates a new browser window. After enabling the partial migration from the Migration Client, the user directly selects the parts of

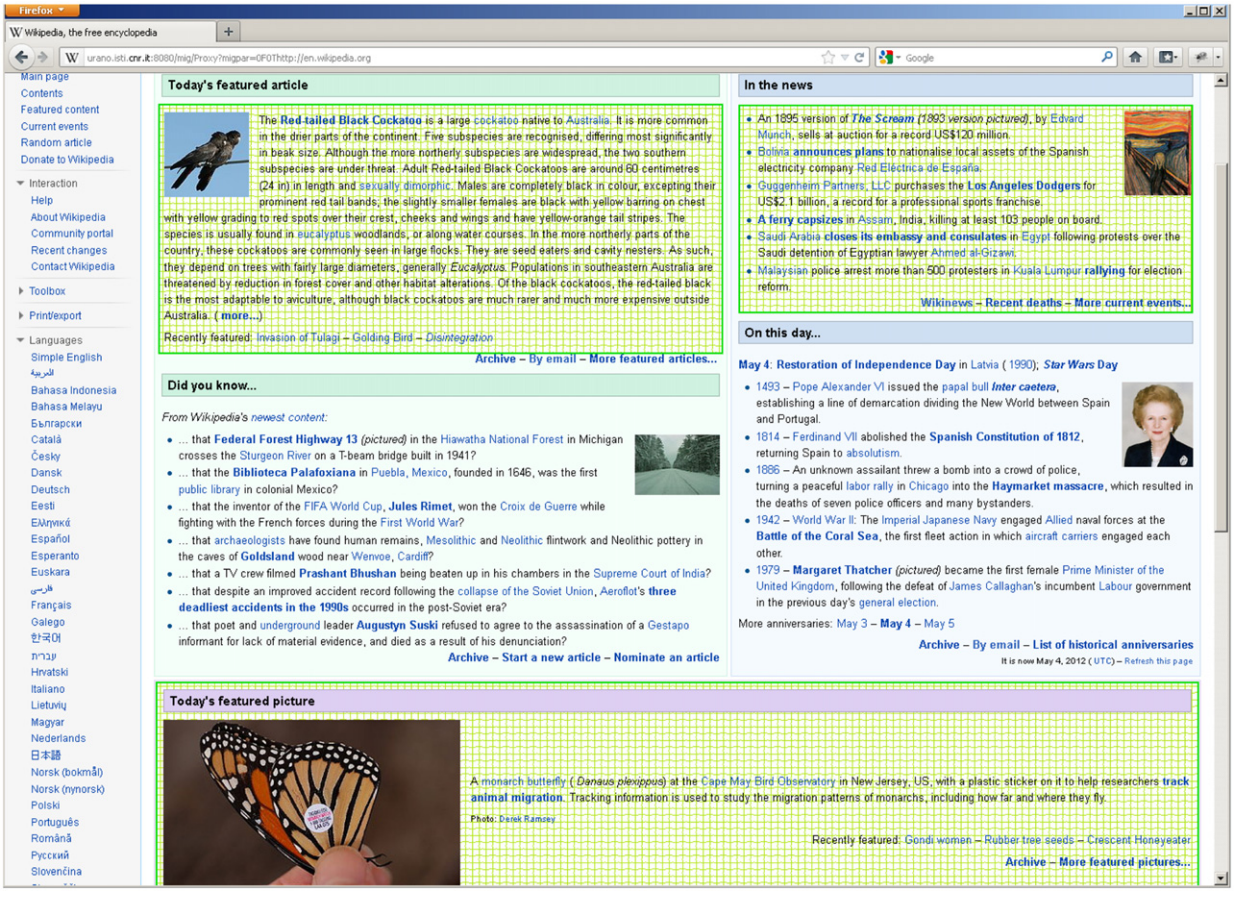

Fig. 1. Web application to be migrated with some components interactively selected. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

interest by clicking on them in the page. One of the effects of the scripts that are automatically included in the page by our migration/proxy server is indeed to highlight the parts currently selected (in Fig. 1 their border colour is green and the background is changed to a green pattern), so that the user is aware of the selections currently performed.

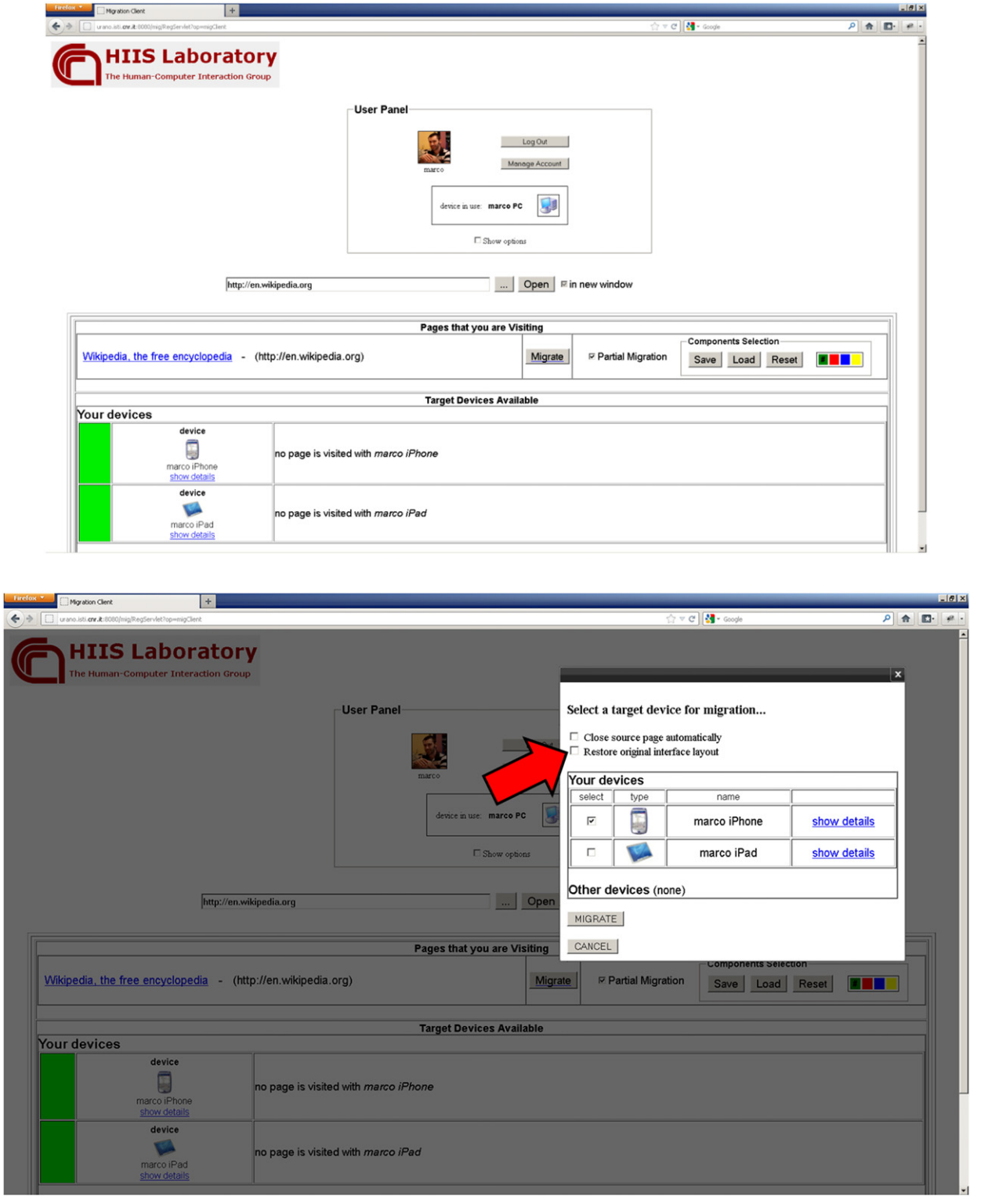

Fig. 2. Migration client interface: the arrow highlights the checkbox for restoring the original user interface layout when migrating back (from a mobile device to a desktop one).

The users can also unselect some areas if they are no longer of interest (by clicking again on them), or can undo the selection from the Migration Client. However, it is worth pointing out that it is possible to select the parts to migrate also on a mobile device, not only on a desktop device.

Fig. 2 shows the interface of the Migration Client, which is displayed in a window separated from the navigated page and is a Web application as well. The example in the figure shows the availability of two target devices, i.e. an iPhone and an iPad.

Migration is triggered by pressing the "Migrate" button (see Fig. 2 top) associated with the navigated page on the Migration Client. This event pops up a window (see Fig. 2 bottom) for selecting the target device and other options. By clicking "Migrate" on the selection window, the migration script of the related page is invoked. The script is aimed to serialise the current Document Object Model (DOM) together with the interaction state of the page, and to forward it to the Migration Platform. The list of selected components identifiers is also forwarded.

At this point, the Migration Platform creates the target version of the page and sends a message to the target Migration Client for notifying the incoming migration request. Before launching the new UI on the mobile device, a dialog box pops up (see Fig. 3) on the Migration Client of the target device and asks the user for incoming migration confirmation (this dialog box, as part of the Migration Client, is also implemented in HTML+JavaScript). The user interface launched on the target device is state-persistent: the content of forms, as well as the values of cookies and JavaScript variables are maintained (Fig. 4).

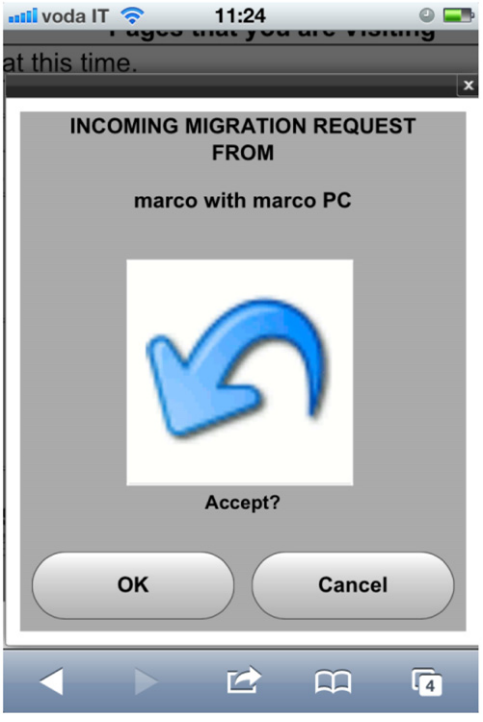

Fig. 3. Confirmation box in mobile device.

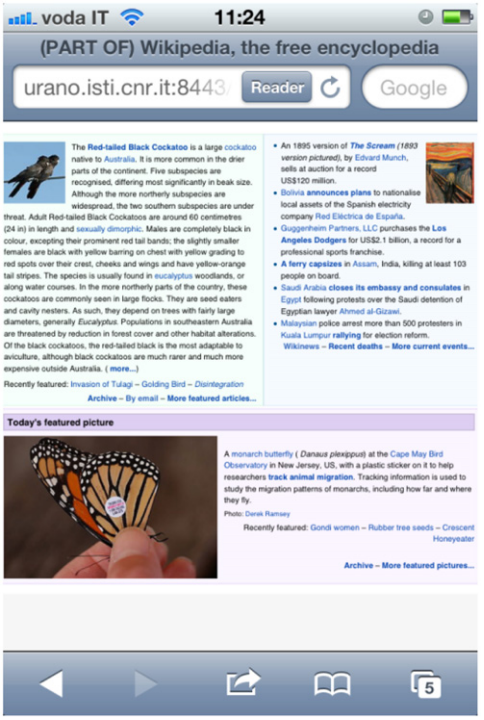

Fig. 4. Dynamically created mobile version on the target device.

4. Architecture

In this section we first provide an overview of the architecture modules that support the dynamic composition of migratory mobile applications and how they interact with each other. We then go on to provide some more details on each of them.

In the proposed architecture, all the devices that can be involved in migration must run an instance of the Migration Client (e.g. in Fig. 5 both Devices A and B run the Migration Client). Among its functionalities, the Migration Client exploits the device discovery service (see (1) in Fig. 5), by periodically announcing the device presence to the Migration Platform and by getting the list of other available devices.

When a Web page is opened through the Migration Client (2), its URL is provided to the proxy (3) which gets the original Web page from the application server (4), annotates it and responds to the client browser (5). Migration triggering is done from the Migration Client by clicking the "Migrate" button associated to the navigated page title (see Fig. 2). Upon migration triggering, the Migration Client invokes the migration procedure of the page (6) defined on the JavaScript block (which was injected by the proxy during navigation). The migration procedure serialises the page DOM, extracts the interaction state and calls the Migration service (7). State Mapping (8) and Partial Extraction (9) are performed according to the interaction state of the page and to the list of selected components. A message notifying the incoming migration request is sent to the target Migration Client (10). The user is then asked for incoming migration

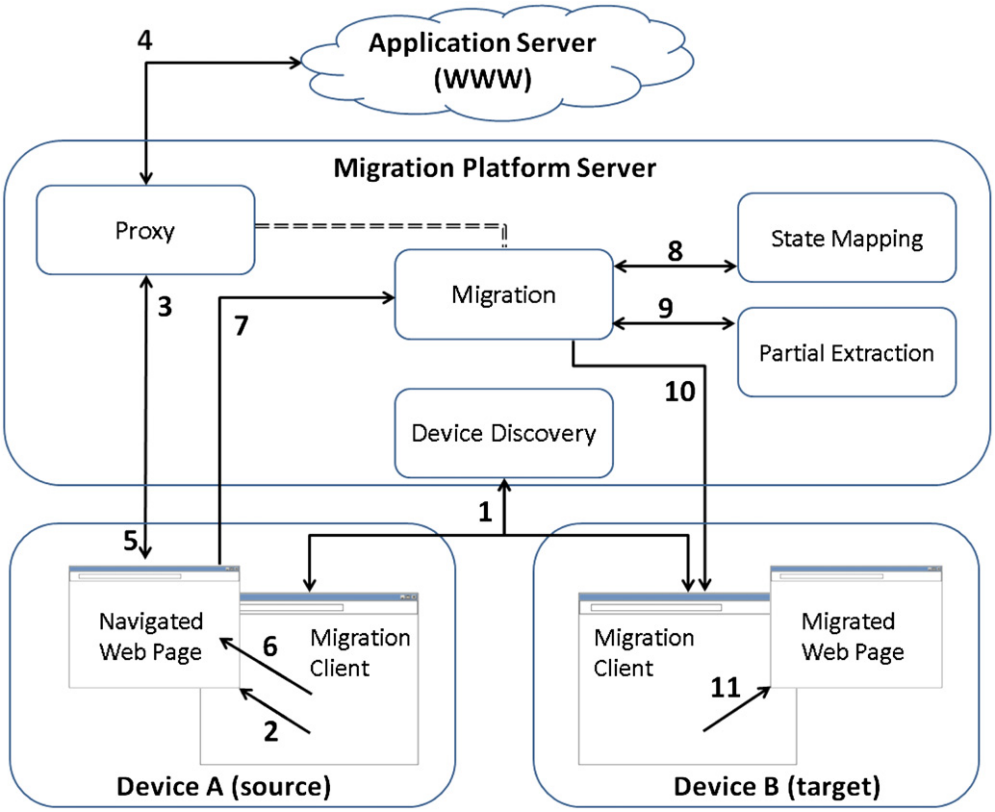

Fig. 5. Architecture of the proposed solution.

confirmation and, in case of acceptance, the Migration Client launches the page with the state resulting from the interactions in the source device in a new browser window in the target device (11).

More detail on the main architecture modules is provided in the following sections.

4.1. Device discovery

The Device Discovery is the service aimed to detect the presence of devices that could be involved in a migration. Through this service, each device on which the Migration Client is currently running, is informed about the presence of the other available devices. This discovery process is done by the Migration Client of each device that can be potentially involved in migration, by periodically and asynchronously invoking this service through an Ajax script. Such script provides the Migration Platform with information about the concerned device (such as the type of device and the list of pages that are accessed by the device through the platform) and gets as response the list of other available devices.

4.2. Proxy

Whenever the Migration Client requests access to a certain Web page, the request is received by this module. The Proxy first accesses the application server to obtain the original page, and then annotates it by including the scripts enabling and managing migration. As will be seen in more detail in the following, such scripts have various objectives: they enable the component selection for partial migration, manage the feedback to the user regarding the portions of the UI currently selected and capture the UI state when migration is triggered. The Proxy also adapts the links in the considered page so that subsequent requests to other pages will in turn be redirected towards this module before reaching the original application server.

The two basic steps performed by the proxy are summarised as follows:

- **HTML Parsing**. The source HTML of the page downloaded from the application server is parsed. A DOM instance is then created in order to access and manipulate the nodes.
- **Page Annotation**. It consists of enhancing the page with additional functionalities needed for supporting further proxy-based navigation and migration. It is performed through a number of sub-steps:

1. *Links modification*. Once the DOM structure has been built, the Proxy navigates the tree structure and searches for elements that have to be modified. Links or references to other resources within the page will be redirected to the Proxy module, which will thus be able to locate any external resource and to manage migration requests from any linked page. More in detail, the links and references (e.g. *src*, *href* attributes) are modified in such a way to pass through the proxy.

Furthermore, since the links contained in the referred resources also have to be modified, this module has to parse such external resources (and suitably modify them when they in turn include other links). The JavaScript code is also parsed in order to identify and adapt any possible link that may be used at run time. From a performance point of view, it is clear that the computational effort required by this module to modify the links is proportional to the number of links contained not only in the page, but also in all the referred resources/files (e.g. external style sheets, script files, …).

2. *Inclusion of unique identifiers*. For each page element that can be potentially relevant for migration (HTML containers such as DIV, TABLE, FORM, etc.), the "*id*" attribute is checked. If it is not defined, a unique id is added to the element.
3. *Inclusion of scripts for handling the migration process*. In this step the Proxy automatically includes the JavaScript excerpts that will be used to capture the state of the interactive UI elements, to serialise it and to forward it to the platform when migration is triggered. The migration-related scripts also include the procedure for restoring the JavaScript variables values on the target device.
4. *Inclusion of events for enabling the selection of components and handling the partial migration*. The main HTML elements of the page (e.g. DIV, TABLE, FORM elements …) are enhanced with "*onmouseover*", "*onmouseout*" and "*onclick*" attributes that refer to JavaScript event handlers. This is done in order to support partial migration. When the partial migration is enabled and an element is hovered/clicked, the corresponding event handler changes the element background colour accordingly. When migration is requested, an additional attribute of the element, that is the selection flag, specifies whether the element has to be migrated or not.

4.3. Migration

In order to trigger migration of an active page, the associated "Migrate" button on the Migration Client has to be pressed (see Fig. 2). The button event calls the script that carries out the migration of the related page. The communication between Migration Client and the page to be migrated is possible because the Migration Client owns a reference to all the document windows opened through it. The migration script, which is injected in the page by the Proxy when the page is opened, extracts the following information:

- *Representation of the current DOM*. A standard JavaScript library function is used to serialise the DOM to a string.
- *Form fields*. The browser JavaScript support is not able to automatically serialise the whole DOM since it lacks the serialisation of form field values. Thus, the JavaScript inserted by the migration Proxy individually accesses each form field in order to get its updated content (e.g. text input strings, list box values, etc.). The form(s) state is then formatted in a JavaScript Object Notation (JSON) string. From a performance point of view, the time required by the State Mapper depends on the number and type of elements to be mapped.
- *JavaScript variables*. Most Web pages use scripts. The JavaScript code injected by the migration Proxy reads the JavaScript variables at migration time and stores their value in a JSON string.
- *The list of ids of the selected elements*. If one or more elements have been selected for partial migration, their ids are needed by the platform in order to identify the elements to be extracted.

The Migration service is invoked by a script included in the page that is going to be migrated. The invocation parameters of this script are: DOM serialisation string, JSON strings encoding the form fields and the JavaScript variables, list of selected elements id, target device id, preferences (e.g. whether to close or not the page on the originating device).

Since such data can be large (the DOM serialisation length depends on the page size), the Migration service is called through a "POST" method. The migration is carried out through the support of Partial Extraction and State Mapping functionalities, which are discussed in the following sections.

4.4. State mapping

The State Mapping module is responsible for preserving, after migration (on the target device), the results of user interaction with the page, which took place on the source device before migrating it.

As discussed in the previous subsections, form fields content and JavaScript variables are forwarded by the page to the Migration service. The State Mapping module enriches the DOM of the migrated page with state information. This is done through the following steps:

- HTML form fields are filled according to the values encoded in the JSON string: in practice, the attributes of the form input elements are set to the value they had on the source device at the time when migration was triggered;
- The JSON string that encodes the JavaScript variables is saved on a text file within the Migration Platform server;
- A JavaScript excerpt for restoring the JavaScript variables is attached to the page. In particular, it is appended to the "onload" event of the document body and, if the event is already defined by the original page, the excerpt is executed after the original code. This script loads, from the Migration Server, the JSON text file containing the JavaScript state and replaces the value of each variable on the page with the value specified in the JSON text file. The technical aspects concerning JavaScript variables access and manipulation are discussed in [2].

It is worth noting that the DOM itself is also part of the general interaction state of the page. Indeed, as already mentioned, it results from the user interactions with the page.

The interaction state persistence copes also with *cookies and user session*. Web applications often exploit cookies for binding information (e.g. user preferences) to a specific client. Most of the time user sessions are tracked by a cookie. Our Migration Platform manages the binding between Web application cookies and multiple devices associated with the same user: when a Web page is migrated, the cookies related to the originating device are automatically associated to the target one. For instance, if login was performed before migrating, when users access through the target device, they are not required to log-in again because the session cookie is maintained.

4.5. Partial extraction

When a partial migration is performed (therefore, a subset of the elements of the original page has been migrated onto a target device), the Migration Platform preserves as much as possible this selection in the pages that are reached through the links contained in the migrated page. This is possible when navigation occurs between similar pages (i.e. pages with the same structure and/or with the same element identifiers). This is the case of several Web applications whose pages are dynamically created at the time of navigation, such as e-commerce websites. Similar scenarios have been considered for the usability test reported in the next section.

When a partial migration occurs, the list of *id*s of selected elements is bound to the session of the target device. Due to information exchange between the Proxy and Migration services (see the dashed double line in Fig. 5), the Migration Platform is thus able to keep track of the desired elements, even when the user continues navigating.

The strategy for preserving the selection in the navigated pages is based on two steps: definition of the selected elements in the current page (i.e. the migrated page), and extraction of the corresponding elements from the new pages (i.e. the pages that can be accessed through the links in the migrated page).

Elements definition is carried out by analysing the entire version of the migrated page in order to extract the following information about the selected elements (which are identified by their id): Type; Class; Path within the document (from the body node to the parent node) and, for every element in between: id, type, class, and its position within the list of siblings.

Elements extraction is done by getting from the new page only the elements matching the id of the ones selected by the user for partial migration, or satisfying a "similarity" criterion. The aim of the similarity criterion is to find, in the new page, the elements that are more similar to the ones selected in the original entire page. This criterion is only applied if no element on the new page is found to match the id of the selected element. The following steps summarise how the similarity algorithm is applied to each selected element, starting from the new page document body node:

1. The children of the current node in the new page DOM, referred to as level *j* node, (which is the body at the first iteration) are obtained;
2. The node at position *k* on the path of the selected element in the original entire page is considered; its attributes are compared with the attributes of level *j* node children.

a. If, among the children of level *j* node, there is one with the same id as the node at position *k* in the path, then the child becomes the new current node, the current position in the path (i.e. *k*) is incremented and the procedure continues from (3);
b. Otherwise, among all the children of level j node, the procedure selects the one with the highest *similarity score*, which is computed as the weighted sum of matching attributes: e.g. the type attribute has higher weight than the class one. In the case of no type/class matching, the node selection is done according to the sibling index (i.e. the position of the node with respect to its siblings). The selected node becomes the new current node, the current position in the path (i.e. *k*) is incremented and the procedure continues from (3).

3. If the new current node has the same depth as the selected element (i.e. no more nodes are available in the path), then it is considered to correspond to the selected element and the procedure ends.

The element extraction algorithm is run for every selected element. For instance, if *n* elements have been selected from the migrated page, then n elements are likely to be extracted for the new pages. The partial new pages are created by composing the extracted elements in a single document.

With respect to the *j* and *k* indexes, both are incremented by 1 for every loop iteration. Indeed, the algorithm searches for similarities between nodes that have the same depth in the saved path and in the new page DOM. As a consequence, the algorithm works properly only when the new page has the same structure of the previous one.

When the above algorithm does not provide any matches (e.g. in the case the user opens a link to a totally different page), the entire version of the new page is presented to the user.

The search engine of an e-commerce website can be a typical application of the above mentioned strategy. After performing a search, the user migrates only the search bar and the containers of the first results to another device (e.g. a mobile). The Migration Platform stores the current page and the list of selected elements' ids.

When a new page is requested (i.e. when the user clicks on a link to refine the search, or performs a new search), the Proxy detects that a list of desired elements is bound to the current user session. The Partial Extraction module is then provided with the original current page (i.e. the entire page from which the partial one was obtained), the new page and the list of the elements' ids. It performs the elements definition from the current page and the elements extraction from the new one, and returns a partial page.

Although differences in the ids of the selected elements may exist between the pages, thanks to the similarity criterion it is possible to obtain a new page with the same structure of the partially migrated one.

4.6. Sharing of partial compositions and migration "back"

In this section we describe two features that are included in the platform and are both related to partial migration.

The first one is a feature for storing and recalling compositions of Web page parts, and it is available in the Migration Platform so as to facilitate sharing and reusing previously selected sets of components. When the partial migration option is enabled on a navigated page, before migration, it is possible to save the current selection of components in order to be able to quickly re-apply it to a similar page in the future. Thus, the available compositions (including those defined by other subscribers of the platform) can be subsequently loaded from a list that is shown upon request.

Each composition obtained through interactive selection of page components is identified by the following information:

- *Address of the original page*. The URL of the page where the composition is performed, is stored on the Migration server. When a request to open a page is done, only the compositions associated to the URL of that page are presented to the user. This is because in general the user is not interested in compositions associated to other pages or domains.
- *DOM of the page including the selection* (*i.e., with the selected components highlighted*). The DOM of the page from which the composition has been generated is saved on the Migration Server.
- *Screen dump of the page highlighting the selection*. An image capture is automatically generated server-side starting from the DOM of the page with the selection. As a result, the screen dump shows approximately what the user would see in his/her browser if the associated selection is loaded on the currently visited page. *The set of ids of the selected components*. The identifiers that characterise the saved composition are needed in order to find the related components in the page where the composition will be loaded.
- *The name of the author*. This is the *userid* of the subscriber that saves the composition.

An example of selection loading is shown in Fig. 6. The window on the right side lists the previously saved selections for the currently navigated page (left). Each one of the saved selections is described by author, date of creation, list of identifiers of the selected HTML components, and a screen dump of the page highlighting the selection. In order to restore one of the listed selections on the currently navigated page, it is sufficient to click on the preview picture. On the navigated page, the corresponding components are automatically selected. The inter-window interaction between the navigated page and the page for loading the combinations occurs through the Migration Client window (i.e. the Migration Client

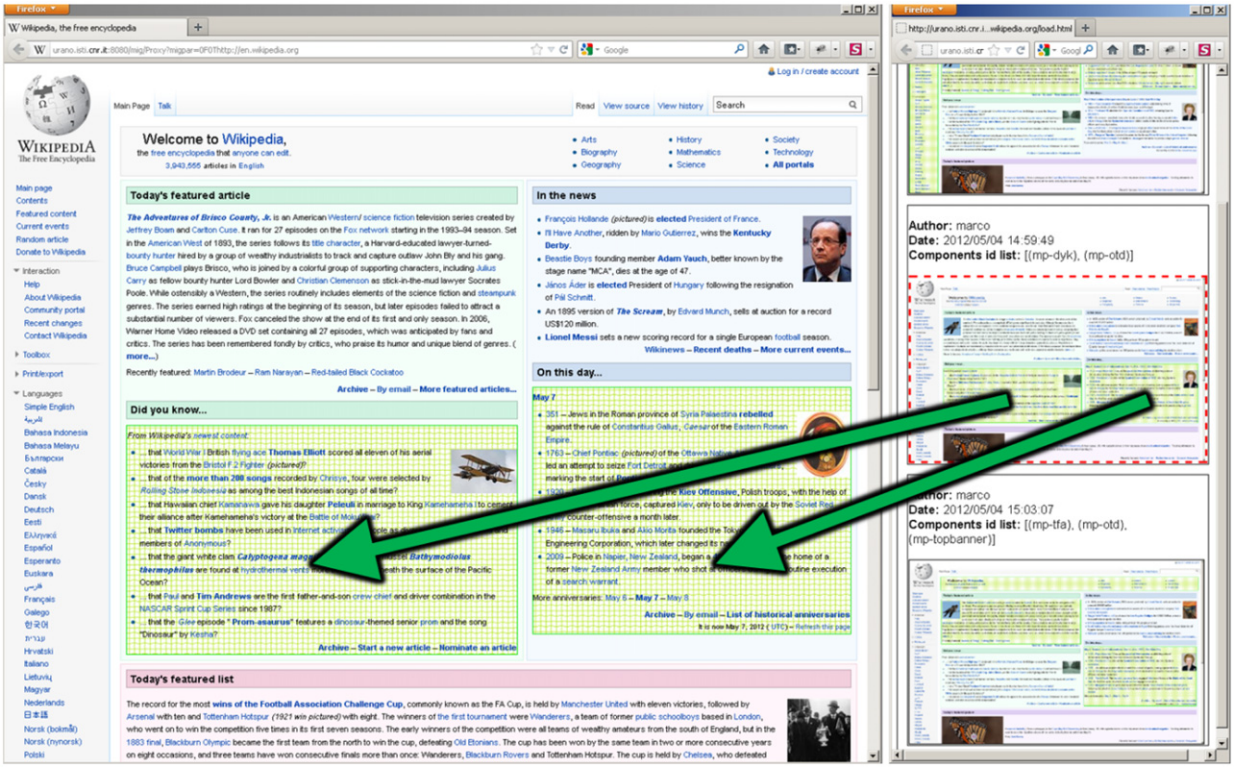

Fig. 6. An example of loading of a set of previously selected components. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

provides the navigated page window reference to the loading window, so as to enable the loading window to select components on the navigated page). The user can then trigger a partial migration (i.e. the migration of the selected components) towards another device, or can further manipulate the selection (e.g. by adding or removing components to/from the selection) before migrating. In Fig. 6, the selection with red dashed border (right) is "copied" on the navigated page (left). The aim of the selection previews listed is to highlight which parts of the Web page were selected, independently from the actual content of the current page. Thus, in general, the contents displayed in the preview list do not exactly match with the currently navigated one. However, the structure of the pages is matching.

Another feature that we included in the Migration Client was to provide users with the possibility to select, after performing a partial migration (generally carried out from a desktop source to a mobile target device), a migration "back" from the target device to the source device (namely: from mobile to desktop). Indeed, a partially migrated page (or a page obtained by navigating from a partially migrated one) can be, in turn, migrated. For example, a user could partially migrate a page from the desktop to the smartphone. After browsing from the smartphone, the user might want to migrate "back" the page to the desktop device. Such a task can be performed in two different ways: either preserving the current customised layout (i.e. the partial page) and losing the original layout or restoring the original layout and losing the user-driven customization for the mobile device. Among the benefits of the preserved migration there is the full navigational persistence, since only the currently displayed components are migrated. The aim of the restored migration is instead to provide a representation more suitable for the target device. In the mobile-to-desktop back migration it can be reasonable to restore the original desktop page layout because originally designed for that type of device. The state persistence is provided by mapping the result of the interaction (e.g. cookies, form fields content,…) from the partial page into the entire page.

In the Migration Client, this has been supported by providing a checkbox (see Fig. 2) "Restore original interface layout": when checked, the "restored" option will be applied to the back migration. Therefore, the choice of whether restoring or not the original layout is performed during the selection of the target device.

5. Evaluation

A user test has been conducted in order to evaluate the usability of the proposed platform and the quality of the results provided by it, and to collect relevant user feedback, also for future improvements. In particular, one of the main goals of the study was to analyse how both partial migration and "back" migration were assessed by users, also verifying which option for back migration was the preferred one for users. Moreover, we were also interested to understand whether there was any correlation between the usability of the results reported by the users and the user's experience/familiarity with mobile browsing/devices.

5.1. Participants

Sixteen volunteers (7 females) aged 19–45 ($M=31.7$, $SD=5.5$) participated in the study. All subjects had moderate-to-high experience using Web browsing on mobile devices: 13 people access the Web through mobile devices daily, the others monthly. People also provided separate scores for their experience in *mobile browsing* the two web sites considered for the tests. For eBay, 7 users declared not to have ever used it, 5 used yearly, the remaining 4 use it on a weekly basis. For Amazon, 7 users declared not to have ever used it, 3 users use Amazon yearly, the remaining six ones use it weekly. All users owned at least two personal devices, while the maximum number of devices owned was five ($M=3.6$, $SD=0.9$). In particular, each one of the participants owned at least one desktop or laptop and used it daily. The vast majority of users (15 out of 16) owned at least one mobile device (smartphone, touch-based smartphone or tablet), which 14 people used daily.

Subjects were recruited using several channels. We involved personnel of our institute (researchers as well as technicians and administration personnel), but not of our laboratory. Some relatives of the institute staff also participated in the evaluation.

The educational level of the users was varied: 4 held a PhD, 5 a Master Degree, 5 a Bachelor and 2 a High school degree. All users had already visited electronic commerce websites, such as Amazon and/or eBay. All subjects were novices with respect to usage and knowledge of our Migration Platform (before this test, they had never used it). Volunteers received a small gadget or a lunch ticket for their participation.

5.2. Tasks

Users were asked to use our migration platform to first perform a partial migration (from a desktop PC to an iPhone device) and then a back migration. Both options of back migration were analysed by users, who exploited a different web site—one between Amazon or eBay—to analyse each option/condition.

In particular, users were asked to make a search for an item *at their choice* through one of the two considered web sites (Amazon or eBay), and to perform a partial migration from a desktop PC to an iPhone. Then, on the iPhone, they had to refine the list of results obtained from their search and then carry out a migration back, (from iPhone to desktop again), by using one of the two options for the migration back. After, the users had to perform again the same activities, using the other web site and the other option for migration back.

We selected tasks with low complexity, chosen to mimic activities typically done on this kind of e-commerce websites. This was done to get higher external validity, while at the same time leaving some amount of freedom to users.

	User1	User2	User3	User4	User5	User6	User7	User8	User9	User10	User11	User12	User13	User14	User15	User16
1st Phase	Eb. V1	Eb. V1	Eb. V1	Eb. V2	Eb. V2	Eb. V2	Eb. V2	Am. V1	Am. V1	Am. V1	Am. V2	Am. V2	Am. V2	Eb. V1	Am. V2	Am. V1
2nd Phase	Am. V2	Am. V2	Am. V2	Am. V1	Am. V1	Am. V1	Am. V1	Eb. V2	Eb. V2	Eb. V2	Eb. V1	Eb. V1	Eb. V1	Am. V2	Eb. V1	Eb. V2

Fig. 7. Order of presentation of the two versions (V1 and V2) and the two web sites (eBay/Amazon) during the user test.

More in detail, the tasks were:

1. **Task1.** On the desktop device, through the Migration Client, open the considered Web site, search for an item/brand of your choice and obtain a first list of results
2. **Task2.** In the Migration Client, enable the option for partial migration, select some components in the page to migrate, and trigger their migration towards the iPhone.
3. **Task3.** On the iPhone, check whether all (and only) the previously selected components belong to the migrated page; look at how they have been composed in the resulting page *P*.
4. **Task4.** In this page P, further refine the search (you might want to restrict the list of results) according to a criterion of your choice. In the obtained page *P'*, check whether the layout is consistent with the one provided in the page resulting after the partial migration.
5. **Task5.** Trigger the migration of this refined page *P'*, "back" towards the desktop device by selecting one option for the migration back and check again the layout obtained.

In the following, we refer to the two versions of the back migration as Version 1 (or V1, the "preserved" version) and Version 2 (or V2, the "restored" one).

5.3. Equipment

The test was carried out through a Desktop PC and an iPhone 3Gs within the same local network. The Desktop was connected via cable while the iPhone was connected wirelessly through a 54 Mbps 802.11 g wireless router. The Desktop PC also hosted the Migration Platform server, and was equipped with a 2.80 Pentium D processor, 2 GB of RAM and Windows 7 OS. The resolution of the desktop PC was 1680×1050, the Phone's one was 480×320.

5.4. Procedure and design

Each participant carried out the test on his/her own. A moderator was also available during each user session. The study was initiated with a short presentation in which general information and instructions were given to each subject by the experiment moderator: during it, the moderator also explained to the user how the two different versions V1/V2 worked.

All users were novices with the migration platform. Thus, in order to familiarise with the platform, each subject performed a set of practice tasks equivalent to those that would be presented to them in the main part of the experiment (navigate a page, select one or more components in it, trigger a migration). Then, a demographic questionnaire was applied: each participant answered to a set of general questions about his/her education, device usage experience and habits.

The study was a within-subject one (each user had to analyse two versions). During the main part of the experiment, participants performed *twice* the five tasks described before, each time using a different version V1/V2 for back-migration and a different web site. In order to diminish the possibility of carryover effects, we counterbalanced the presentation order of the web site (Amazon/eBay) and of the version (V1/V2), as it is possible to see from the Fig. 7. Thus, half of the users performed first (namely: in the first phase) the *preserved* V1 migration and afterwards the V2 *restored* one, while for the other half the order was the opposite. Also, among the users of each of the two groups, half of them executed the preserved migration on eBay and then the restored migration on Amazon, while for the other half the order was the opposite.

After each of the task series using one Web site and one back migration solution, the participants were asked to fill in a brief questionnaire about the solution they just experienced. After analysing the second back migration option, participants ranked the two types of back migration by specifying their preferred version. They also commented on other migration features (which were kept constant among the two versions), also providing general suggestions/recommendations. The sessions lasted between 19 and 45minutes ($M=32$, $SD=6$).

Fig. 8 shows a timeline summarising each user session and indicating the exact times when the various events occurred (a legend is also provided within the picture to properly understand each step). As you can see, after a familiarisation phase, each user experienced two phases: in each phase s/he tested one version for the back migration (among V1/V2) by using one of the web sites considered (eBay/Amazon). Since in each of the two phases the sequencing of events occurring was the same, we can focus on just describing the structure of the first phase (interval [B–G] in Fig. 8), since the same structure holds for the second one (interval [G–L] in Fig. 8). In the first phase, you can see that each user had an interaction with the desktop ([B–C] interval in Fig. 8) and an interaction with the mobile device ([D–E] interval in Fig. 8). Please note that during the interaction on the desktop users had to select in the concerned desktop page the elements to consider for partial migration, while during the interaction on the mobile device, users had to interact with the mobile customised page to refine their search. In both cases (desktop interaction and mobile interaction), users also had to use the Migration Client (this can be identified by e.g. migration triggers in Fig. 8).

5.5. Method

Apart from collecting feedback from users on the usability of the migration process and its results, in the

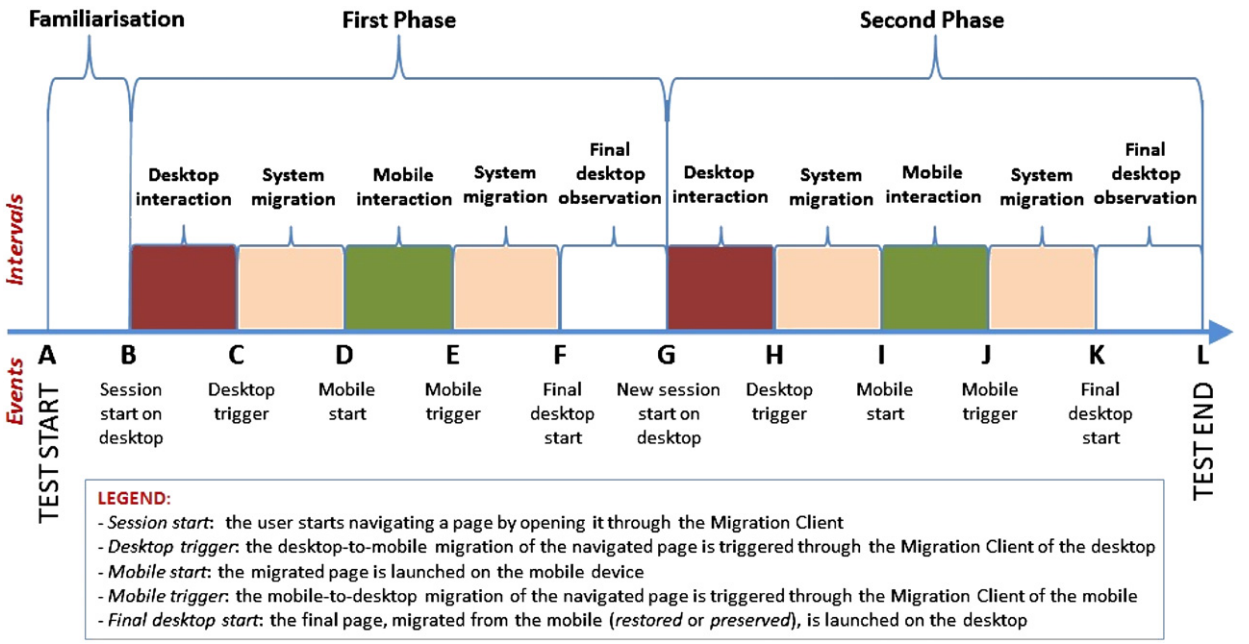

Fig. 8. Structure of each user test.

test we wanted to investigate whether there was a significant difference between the usability of Version1 and the usability of Version2. In addition, we were also interested to know whether there was a significant difference between the times needed on the desktop (resp.: on the mobile) device to carry out the tasks during the 1st and the 2nd phase. Moreover, we also wanted to verify whether there was a significant correlation between the user experience in using mobile devices and the usability of the two versions (V1 and V2).

Owing to the small sample size ($n<20$) of the data and/or the ordinal nature of most of the data (usability scores, user's experience in mobile browsing and users' experience in using mobile devices) we applied non-parametric tests [18]. In particular, we used the Wilcoxon matched-pairs signed-ranks test (as it was a within-subjects test with 2 paired groups of data) to verify: (a) the differences in the usability scores provided by the users between Version1 and Version2; (b) the differences between: (i) the interaction times needed on the desktop to perform the tasks during the 1st and the 2nd phase; and (ii) the interaction times needed on the mobile device to perform the tasks during the 1st and the 2nd phase. Also, we used the Spearman correlation to verify if there was any correlation between the "user's experience" and the usability of the version (V1 or V2). For "user's experience" we considered two separate cases: (i) the user's experience with mobile browsing: and (ii) the user's experience in using touch-based mobile devices. In particular, please note that the experience of the user with mobile browsing was dependent on the site tested with the specific version (V1 or V2) considered. For instance, for the users who tested Version 1 using eBay web site, we took their experience in using the mobile eBay web site as the referring value for the mobile browsing experience variable. Both (i) and (ii) were derived by the answers provided by users within the questionnaire. The Bonferroni correction of the significance level ($\alpha/2$) was applied due to a double correlation involving the same variable (the usability of the version). All the statistical analysis tests were performed by exploiting the SPSS 16.0 statistical package.

5.6. Results

In this section we summarise the results gathered through statistical analysis of the data collected during the test, mainly from the filled questionnaires and the interaction logs of the user sessions. In addition, we also report on additional feedback/remarks/suggestions provided by users.

In the questionnaires, a 1–5 semantic differential scale was used by the participants to provide their ratings on various aspects of migration. In that scale, 5 was the most positive score, and 1 was the most negative one. In particular, the meaning of the scores was the following: $1=$*very bad*; $2=$*bad*; $3=$*neutral/ambivalent*; $4=$*good*; $5=$*very good*.

Regarding the differences in the usability scores between Version1 and Version2, although both were well perceived by users, we found a significant difference (Wilcoxon test: $N=16$, $Z=-2.333$, $p=0.02$) in the perceived usability between V1 and V2, with V2 having a higher usability than V1 (Version1 mean $\pm$ SD: $4.063 \pm 0{,}68$; Version2 mean $\pm$ SD: $4{,}5 \pm 0{,}516$, where the time is expressed in seconds). This was also confirmed by the answers provided in the filled questionnaires. Among the sustainers of the restored version, eleven people motivated their choice citing the possibility of accessing the original desktop Web page: six clearly stated that, when

re-opening on the desktop the migrated page (which was previously partially migrated), they prefer to handle its original version since the desktop device has enough interaction capabilities to support usable interactions with the "complete" version of the page. Among those who preferred the preserved version, two liked keeping just the selected elements when migrating back to the desktop device.

Regarding the differences between the interaction times on the desktop (resp.: the mobile) device between the two phases, we found a significant difference (Wilcoxon test: $N=16, Z=-2{,}379, p=0.017$) in the time taken between the 1st and the 2nd interaction *on the mobile device*, where the time taken for the 2nd interaction was smaller than the time taken for the 1st interaction (*Interaction1_time_mobile*: mean $\pm$ SD: $252{,}563 \pm 64{,}44$; *Interaction2_time_mobile*: mean $\pm$ SD: $188{,}188 \pm 123{,}284$; where the time is expressed in seconds). We explained this with the fact that, during the 1st interaction on the mobile device, users had the possibility to get familiarity with the customised layout provided on the mobile device as result of partial migration and therefore during the 2nd interaction they had quicker performances in carrying out the requested tasks on the mobile device. Therefore, it appeared that users became soon quite familiar with the customised layout provided by our system on the mobile device (as resulting from partial migration). On the desktop device, there was no significant difference between the first and the second interaction (*Interaction1_time_desktop*: mean $\pm$ SD: $345{,}25 \pm 188{,}016$; *Interaction2_time_desktop*: mean $\pm$ SD: $265{,}625 \pm 127{,}789$; where the time is expressed in seconds).

As for the correlations, the only correlation we found was for Version 1: for this version V1 there was a highly significant, moderate and positive correlation (Spearman's test: $N=16$, $r=0.696$, $p=0.003$) between the user's mobile browsing experience and the perceived usability score. For Version 2, instead, there was no particular correlation between the user's mobile browsing experience and the perceived usability score. Taking into account that Version2 was found as more usable than Version1, the reported lack of correlation—between the user's browsing experience and the usability of Version 2—can be explained by the fact that the Version 2 had usability high enough to be independent on the familiarity/experience of the user with mobile browsing.

We also evaluated other aspects connected with our platform: for each of them in the following we provide some descriptive statistical information (range, mean, median, and standard deviation), as well as the gathered users' remarks/suggestions.

System migration time—Users were asked to rate the adequacy of the time taken by the Migration Platform to carry out a page migration (Range (R)=2, Mean(M)=3.7; Median(MD)=4, Standard Deviation(SD)=0.8). The system time for desktop-to-mobile (partial) migration varied between 4 and 22seconds ($M=10.3$, MD=8.5, SD=4.3), and for mobile-to-desktop migration was between 4 and 28seconds ($M=10.2$, MD=8, SD=7.1). Even if none of the users reported serious delay issues, two subjects stated that a shorter migration time would be appreciated.

Usefulness of the partial migration—Users ranked the usefulness of the partial migration ($R=3$, $M=4.1$, MD=4.5, SD=1) in a satisfactory way. The only user that rated the usefulness with a score less than 3 said that they would prefer to migrate the entire view in any case. Among the others, three users highlighted the benefits of state persistence when migrating between devices, eight reported that partial migration is particularly useful to improve usability of Web pages on small devices thanks to the reduction of page complexity. Two users would have liked the possibility of migrating a page towards the device of another user: one, in particular, stated that a partial migration could even allow hiding specific parts of the page from certain target users.

Usability of the selection technique—Regarding the usability of the support enabling the selection of components in the page ($R=3$, $M=3.6$, MD=4, SD=0.9), most comments regarded aspects to be improved. For instance, one user declared that the selection mechanism for partial migration was not intuitive; another noted that too many clicks are sometimes needed to perform the selection; one user had troubles selecting several components in the same page; three users complained about the fact that it could be hard to select some areas of the page; three users pointed out the difficulty of selecting links.

Intuitiveness/usability of the arrangement of the migrated page—The way in which the various components are arranged in the page after partial migration appears to be well received by users ($R=2$, $M=4.4$, MD=4.5, SD=0.6). One user commented that the usability of the resulting UI depends on the type of selection performed. Another user, while interacting with the partially migrated page, noticed that the layout (i.e. position and alignment of the elements) was pretty much maintained with respect to the original layout. However, the same user also noticed that, due to the reduction on the number of components, he would have liked to see a more efficient use of the available screen space.

Further Comments—What people liked. People appreciated both the usefulness of the partial migration and the power of the selection mechanism. Among the strong points, seven participants listed the possibility to extract parts from a Web page, four mentioned the clear benefits of the partial migration support when switching from a desktop to a mobile, and three stated that the support can be useful in any scenario in which there is a device change.

Further Comments—What should be improved in users' view. One issue regarded the *selection mechanism for partial migration*. Ten users stated that the selection strategy could be improved. In particular, two users had trouble selecting some specific parts of the page, especially links, without clicking and activating the links themselves. Another aspect that users noted as in need of improvements were the *techniques supporting migration*. Six users were concerned about the Migration Client access. Among them, 4 users pointed out that switching between the navigated Web application and the Migration Client (to access the platform functionalities) can be confusing and time consuming. One user highlighted that the *number of clicks* needed to carry out a migration is too

high. Another user noticed that passing through the Proxy can potentially slow down the navigation, which could in turn affect the user experience.

The selection strategy also revealed possible semantic inconsistencies: some users tried to include a form on the selection by taking only the input elements, in order to reduce the space needed. As a consequence, the migrated page contained the input elements but not the "form" element. The "action" attribute of the form was thus lost, and the form did not work on the target page. In order to enable the form, the participants had to redo the migration by including a slightly larger part around the input elements (i.e. the whole search form). The test supervisor noted that some participants encountered this issue and recorded it, although nobody mentioned this specific problem on the questionnaire. Thus, among the improvements made after the test to support partial migration, automatic detection of the form elements has been implemented. When one or more input elements are selected, the support automatically checks whether the ancestor form is included in the selection. If the form is not among the selected elements, then it is automatically included.

Recommendations—Several users recommended enabling the selection of UI elements in the same way as parts of text are selected. One user was interested in keyboard short-cut combinations to speed-up the selection. Another one would have liked to perform a graphical selection in an area of variable size around the cursor, in order to select all and only the elements within that selection area. One participant suggested adding, on the Migration Client, the possibility of automatically obtaining the original full version of a migrated page. This is because, after partially migrating a page to a device, the user may want to see the full layout version, possibly directly recalling it from the Migration Client on the mobile device. Such functionality would be based on a sort of navigation/migration chronology stored on the Migration Client.

5.7. Discussion

The results of this test suggest that users appreciated the usefulness of the partial migration and acknowledged its potentialities, although a few remarks were reported on the overall usability of the platform. The main issue seems to be the way that UI elements are selected for partial migration. Several users pointed out the advisability of enabling the users to more effectively select the UI objects to migrate.

Apart from this, all the users were able to quickly understand and appreciate the aim of the migration, and the differences between the two versions for supporting back-migration (which, in the scenario considered, was a migration from a mobile device to a desktop one). This was very positive, as none of them had ever used either our Migration Platform or another similar system. Among the two versions tested, the *restored* version was the preferred one. This can be easily understood because when moving back to the desktop platform there is no further need to access a simplified version of the Web page.

6. Conclusions and future work

In this paper, we have presented our solution for supporting dynamic composition and migration of Web applications from desktop to mobile systems. An example taken from real Web applications has been described to show how we support the possibility of migrating a subset of the elements displayed and running in the source device.

Users can directly select what parts should be migrated from the original Web pages in an easy and intuitive manner, thus creating a version of the Web application adapted to the target mobile device.

According to the outcomes of the usability evaluation, the users fully understood the meaning of partial migration (i.e. reduction in complexity of the page when migrating from desktop to small devices). Indeed, even while coping with "back migration", the users noticed the differences between the two migration options (with or without restoring the original user interface) and clearly indicated the preferred one. Although most users preferred the restored migration, we still keep both the preserved and the restored options in the platform prototype, so that every user can choose the favourite one.

Further investigation will deal with some chronology support of the Migration Client for the migrated pages, in order to provide users with the "migration history" of each page. Future work will be dedicated to further empirical validation as well, even comparing our solutions with other ones, and to extending the current architecture to migration involving types of interaction modalities other than only graphical ones, in order to enhance the applicability and generality of the proposed approach. One example would be to have partial migration from a desktop to a vocal device.

References

[1] A. Alapetite, Dynamic 2D-barcodes for multi-device, Web session migration including mobile phones, Personal and Ubiquitous Computing, online 2 April 2009, doi:10.1007/s00779-009-0228-5.
[2] F. Bellucci, G. Ghiani, F. Paternò, C. Santoro, Engineering JavaScript state persistence of web applications migrating across multiple devices, in: Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing System, EICS 2011, Pisa, Italy, June 13–16, 2011, pp. 105–110.
[3] P. Baudisch, X. Xie, C. Wang, W.-Y. Ma, Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content, in: Proceedings of the UIST 2004, ACM, 2004, pp. 91–94.
[4] Chang, T.H., and Li,Y., Deep shot: A framework for migrating tasks across devices using mobile phone cameras, in: Proceedings of the CHI '11, ACM Press, 2011, pp. 2163–2172.
[5] D. Dearman, J.S. Pierce, It's on my other computer!. Computing with Multiple Devices, CHI (2008) 1144–1153.
[6] W.K. Edwards, M.W. Newman, J.Z. Sedivy, T.S. Smith, Experiences with recombinant computing: exploring ad hoc interoperability in evolving digital networks, ACM Transactions on Computer–Human Interaction 16 (1) (2009) 3. Article.
[7] A. Fox, S.D. Gribble, E.A. Brewer, E. Amir, Adapting to network and client variability via on-demand dynamic distillation. in: Proceedings of the ASPLOS-VII, ACM, New York, 1996.
[8] G. Ghiani, F. Paternò, C. Santoro, On-demand cross-device interface components migration, in: Proceedings of the Mobile HCI 2010, Lisboa, September 2010, ACM Press.
[9] B. Johanson, S. Ponnekanti, C. Sengupta, A. Fox, Multibrowsing: moving web content across multiple displays, in: Proceedings of the UbiComp '01. LNCS 2201, Springer-Verlag, 2001, pp. 346–353.

[10] M. Kozuch, M. Satyanarayanan, Internet suspend/resume, in: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02) IEEE Press, 2002.
[11] J. Maras, M. Stula, J. Carlson, Extracting client-side web user interface controls, in: Proceedings of the ICWE, 2010, Springer Verlag, pp. 502–505.
[12] J. Melchior, D. Grolaux,J. Vanderdonckt,P. Van Roy, A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications, pp. 69.78, EICS'09, July 15–17, 2009, Pittsburgh, Pennsylvania, USA.
[13] J. Nichols, B.A. Myers, M. Higgins, J. Hughes, T.K. Harris, R. Rosenfeld, K. Litwack, Personal universal controllers: controlling complex appliances with GUIs and speech, CHI Extended Abstracts (2003) 624–625.
[14] J. Nichols,B. Rothrock,D.H. Chau B.A. Myers, Huddle: automatically generating interfaces for systems of multiple connected appliances, in: Proceedings of the UIST 2006, pp. 279–288.
[15] F. Paternò, C. Santoro, A. Scorcia, Ambient Intelligence for Supporting Task Continuity across Multiple Devices and Implementation Languages, the Computer Journal, the British Computer Society (2009).
[16] J.S. Pierce, J. Nichols, An infrastructure for extending applications' user experiences across multiple personal devices, UIST (2008) 101–110.
[17] J. Rekimoto, Pick-and-drop: a direct manipulation technique for multiple computer environments, in: Proceedings of the UIST'97, ACM, 1997, pp. 31–39.
[18] S. Siegel, N.J. Castellan, Nonparametric Statistics for the Behavioral Sciences, McGraw-Hill Book Company, New York, 1988.
[19] D.S. Tan, B. Meyers, M. Czerwinski, WinCuts: manipulating arbitrary window regions for more effective use of screen space. CHI Extended Abstracts, in: Proceedings of the CHI '04, ACM, 2004, pp. 1525–1528.