OPEN Project

STREP Project FP7-ICT-2007-1 N.216552

Title of Document:	Early infrastructure for migratory interfaces
Editor(s):	Fabio Paternò, Carmen Santoro, Antonio Scorcia
Affiliation(s):	CNR-ISTI
Contributor(s):	
Affiliation(s):	
Date of Document:	February 20
OPEN Document:	D2.1
Distribution:	EU
Keyword List:	Migration, User Interface, Multi-Device Environments, Adaptation, Continuity
Version:	2.0

OPEN Partners:

CNR-ISTI (Italy)
Aalborg University (Denmark)
Arcadia Design (Italy)
NEC (United Kingdom)
SAP AG (Germany)
Vodafone Omnitel NV (Italy)
Clausthal University (Germany)

Title: Early Infrastructure for Migratory Interfaces	**Id Number: D2.1**

Abstract

This deliverable describes the software architecture of the prototype under implementation for supporting user interface migration. It is a middleware aiming to support automatically the main functionalities, adaptation and state persistence, across multiple devices with various interaction resources.

Table of Contents

1 Introduction

This deliverable describes the early infrastructure supporting the migration of the user interface software of an interactive application in order to allow users to continue their activities across various devices with different interaction resources.

2 Prototype Description

Our prototype for migratory interfaces is based on a migration/proxy server and it exploits logical descriptions of user interfaces, which include an abstract level (platform-independent) and a concrete level (which refines the previous one by adding concrete elements and attributes).

The prototype currently implemented at ISTI-CNR is developed using Java version 5 (and higher), and it is based on a Migration Server (which uses Apache Tomcat 5.5) which is aimed at coordinating the various phases of the migration. It also exploits a Migration client for mobile devices implemented in C#. The prototype architecture has a service-oriented architecture based on some functionalities:

- *Device Discovery Manager,* which is the module in charge of discovering the devices currently available for migration
- *Trigger Manager*, which decides when the migration has to be activated
- *Reverse Engineering*, which builds logical descriptions from the desktop Web implementations;
- *Semantic Redesign*, which transforms the logical description of the source user interface into the logical description for the target device;
- *State Mapper*, which associates the state of the source user interface to the logical description for the target device;
- *User Interface Generator*, which generates the user interface implementation for the target device.
- *Migration Client*, which runs on each device notifies its presence/availability and provides information on the state of the user interfaces running on its device;
- *Migration Manager*, which orchestrates the general behaviour of the system

Each client device has a software module running on it, which announces the availability of the device for the migration (1-2 arrows in Figure 1) to the Device Discovery Manager module, which in turn updates the list of available devices (with related information) in order to keep it up-to-date, and then sends this information to the Migration manager (3a). This communication can be used to provide information regarding events that occur in the devices during the user session and can be relevant for migration. When a browser on a client device (e.g. a PDA) tries to access the desktop version of a Web page (let us call it web.html), the request is filtered by a proxy server, which accesses the application server to obtain it and then asks the Migration Manager to start the process for delivering the appropriately modified page version to the PDA. This means performing the reverse engineering of the desktop version, then the semantic redesign of the resulting concrete description for the considered platform (the PDA in this case), and lastly the generator produces the associated implementation. As a result of this process, the version suitable for the PDA is delivered to the client device. This process is iterated for each new page that is requested by the device. The migration can be triggered either by a user request or proactively by the Trigger Manager module, which can identify situations where it would be better for the user to change device. At this point, the Migration Manager asks the Reverse Engineering Module to get the (5a) Web page (Desktop version), in order to produce the corresponding concrete description (5b). Once this logical description is obtained, the Migration Manager asks the Semantic Redesign Module (6a) to perform the semantic redesign of that concrete description for the target platform. Then, the Semantic Redesign module performs such transformation to create a logical description for the target platform (6b), whose state needs to be updated. This is done by the State Mapper module (7a-7b). As soon as the State Mapper has retrieved the logical presentation and has updated it with the latest state information, the State Mapper sends the new concrete user interface for the target device to the Migration Manager. Then, the Migration manager sends this concrete UI to the UIGenerator module (8a-8b), which transforms this logical description into a final user interface. The result of the process is then sent to the target client (9) where it is rendered and made available to the user.

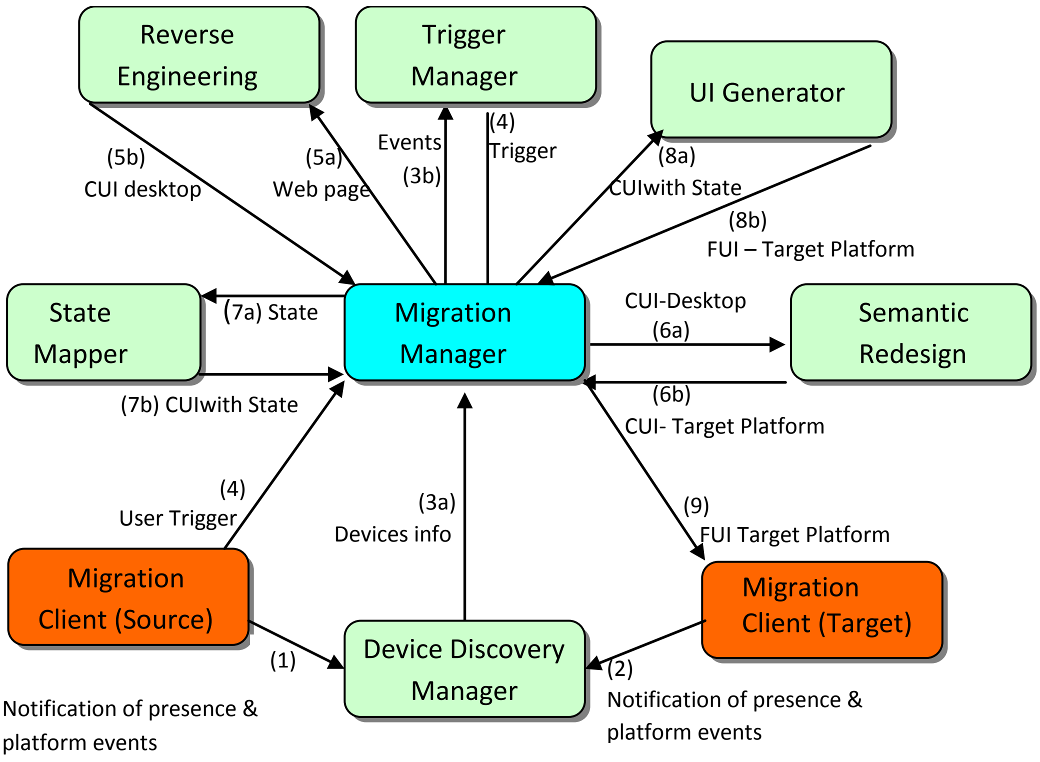


Figure 1: The input/output information that is exchanged between the different modules of the Migration.

Device Discovery Manager/Migration Client(s)

The Device Discovery Manager is a tiny program developed in C#, and, in order to work properly, it needs .NET Framework on desktop platforms and .NET Compact Framework on mobile devices. It has to be activated in every device that is involved in the migration (including the Migration Server), and in the client version the device discovery is integrated with other functionalities that allow the user to select the migration target and trigger the migration (see Fig.2, top left part).

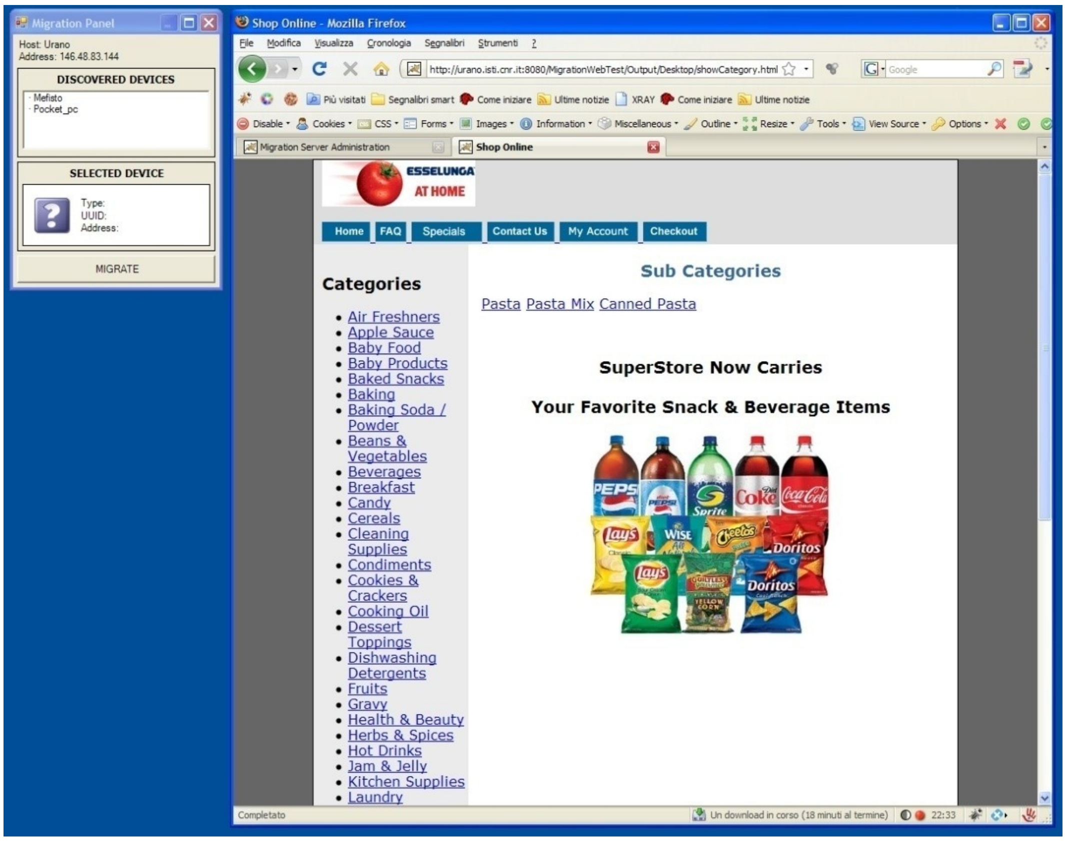


Figure 2: The Migration Client (top-left part) and the Home page on the desktop platform of the Shopping application example (right part)

First, this module has to notify the presence of the associated device to a known multicast group. The list of the devices currently subscribed to such a group defines the list of devices that are available and could be involved in the migration. In order to do carry out this notification, the Device Discovery/Migration Client module use multicast datagrams communications using UDP/IP protocol.

After this, the behavior of the module will be different depending on whether the device on which the software is currently running is the server or one of the other client devices. In order to discovery its role, the software module reads a file in a specific location: if the device is not associated with the Migration Server, the device sends the information about itself to the server, otherwise, it will only receive the information about/from the other devices and stores it in a specific location. In the current implementation the Migration Client, when implemented on mobile devices, has been empowered with a feature of DiscoveryMap [see Del 3.2].

Reverse Engineering

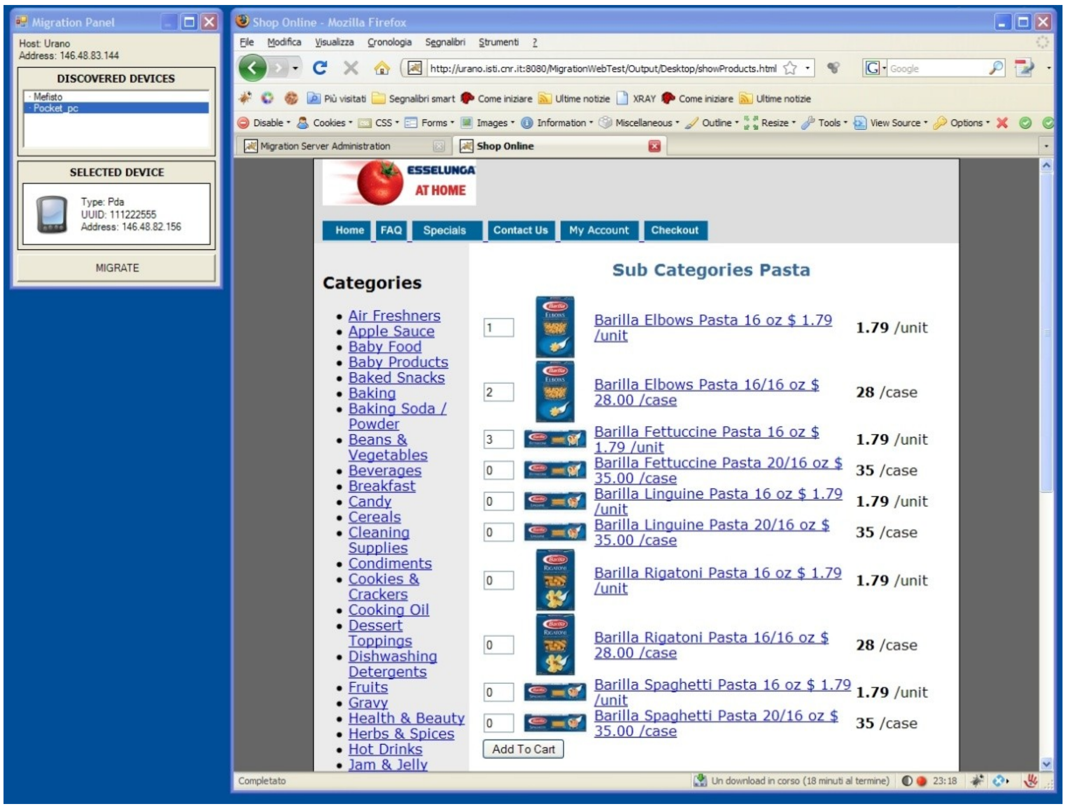
The Reverse Engineering is a software module whose goal is to analyse the implementation of an existing Web application desktop version, capture the logical design of the user interface (in terms of basic tasks supported and the ways to appropriately structure the user interface in order to accomplish them), and then generate a concrete description of a user interface (CUI), described according to the specifications of a language which, in its current state, is an intermediate version between TERESAXML and MariaXML languages. The CUI description will then be used as the starting point for the design and generation of the interface for the target device.

The Reverse Engineering Module reverse XHTML pages, by recursively analysing the DOM tree of the X/HTML page. Well formed X/HTML files are needed as input in order to properly perform this transformation. If it is not the case, 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. Depending on the type of node analysed, the algorithm follows one of the following branches:

- The X/HTML 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> (images), <a> (anchor) and <select> (selection) cause the generation of concrete objects of type respectively image, navigator and selection.
- The X/HTML node corresponds to a composition operator. In this case, the proper composition element is built and the function is called recursively on the X/HTML node subtrees which can be, in turn either elementary interactors or composition of them. As an example of the latter case, the node corresponding to the tag <ul> (unordered list) is reversed into a Grouping composition operator.
- The node does not require the creation of an instance of an element 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).

The component currently provides as output the CUI for the desktop platform, including the state of the interaction techniques.

In the current implementation, further objects have been handled by this module, which were not handled in previous versions. In addition, this module is also able to save the information regarding the last UI element that got the focus before the migration was activated. Indeed, when a request for migration to another device is triggered, the environment has to extract and collect information regarding the current state of the user interface, which is a precondition to ensure state persistence during migration, and support task continuity across multiple devices.



Migration Panel
Shop Online - Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti ?
http://urano.isti.cnr.it:8080/MigrationWebTest/Output/Desktop/showProducts.html
Migration Server Administration
Shop Online
ESSELUNGA AT HOME
Home FAQ Specials Contact Us My Account Checkout
Add To Cart
Completato
Un download in corso (18 minuti al termine)
23:18

Host: Urano
Address: 146.48.83.144

DISCOVERED DEVICES
- Mefisto
- Pocket_pc

SELECTED DEVICE
Type: Pda
UUID: 111222555
Address: 146.48.82.156

MIGRATE

Categories
- Air Freshners
- Apple Sauce
- Baby Food
- Baby Products
- Baked Snacks
- Baking
- Baking Soda / Powder
- Beans & Vegetables
- Beverages
- Breakfast
- Candy
- Cereals
- Cleaning Supplies
- Condiments
- Cookies & Crackers
- Cooking Oil
- Dessert Toppings
- Dishwashing Detergents
- Fruits
- Gravy
- Health & Beauty
- Herbs & Spices
- Hot Drinks
- Jam & Jelly

Sub Categories Pasta

1	Barilla Elbows Pasta 16 oz $ 1.79 /unit	**1.79** /unit
2	Barilla Elbows Pasta 16/16 oz $ 28.00 /case	**28** /case
3	Barilla Fettuccine Pasta 16 oz $ 1.79 /unit	**1.79** /unit
0	Barilla Fettuccine Pasta 20/16 oz $ 35.00 /case	**35** /case
0	Barilla Linguine Pasta 16 oz $ 1.79 /unit	**1.79** /unit
0	Barilla Linguine Pasta 20/16 oz $ 35.00 /case	**35** /case
0	Barilla Rigatoni Pasta 16 oz $ 1.79 /unit	**1.79** /unit
0	Barilla Rigatoni Pasta 16/16 oz $ 28.00 /case	**28** /case
0	Barilla Spaghetti Pasta 16 oz $ 1.79 /unit	**1.79** /unit
0	Barilla Spaghetti Pasta 20/16 oz $ 35.00 /case	**35** /case

Figure 3: The XHTML-desktop Shopping application Form page with some Pasta item quantities specified (right part) and the Migration Client with the target migration PDA platform selected (top-left part).

The process of state extraction includes the identification of the last element accessed by the user in the source device version of the application, and it basically depends on the user's inputs performed till the time when the migration is triggered. For instance, in Figure 3 (right part) the desktop page of the Shopping Application example allowing the user to specify the pasta items that s/he want to buy is visualised. As you can see, the user has already inserted some quantities in correspondence with the items s/he want to buy. Moreover, it is worth pointing out that, in order to be able to collect such data deriving from e.g. user's interactions, the pages have to be slightly modified before being actually used by the user. Indeed, when the clients access the Web pages, their requests are in reality captured by a proxy server, which downloads the pages from the application servers, and it annotates them with scripts that support capturing the UI state. After having performed this step the page is able to capture (and continuously update) the current state of the UI resulting after the various user interactions and, when the migration is activated, send such collected data to the Migration Server (see next sections).

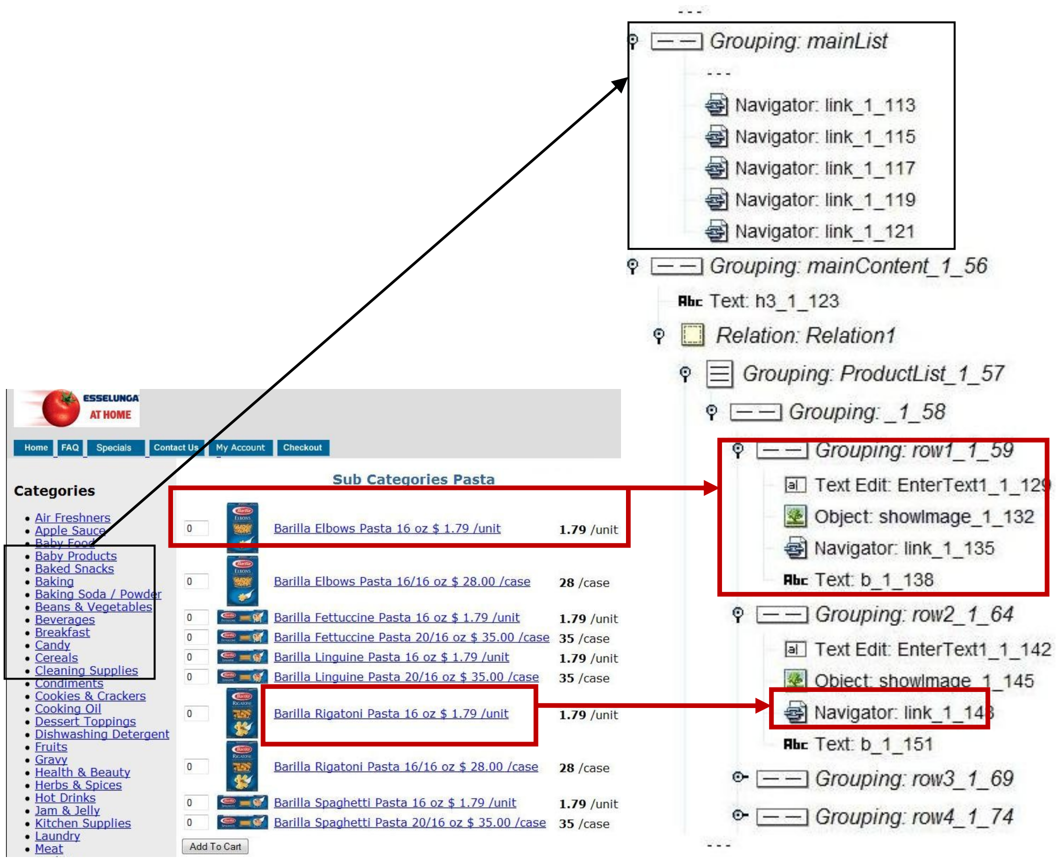


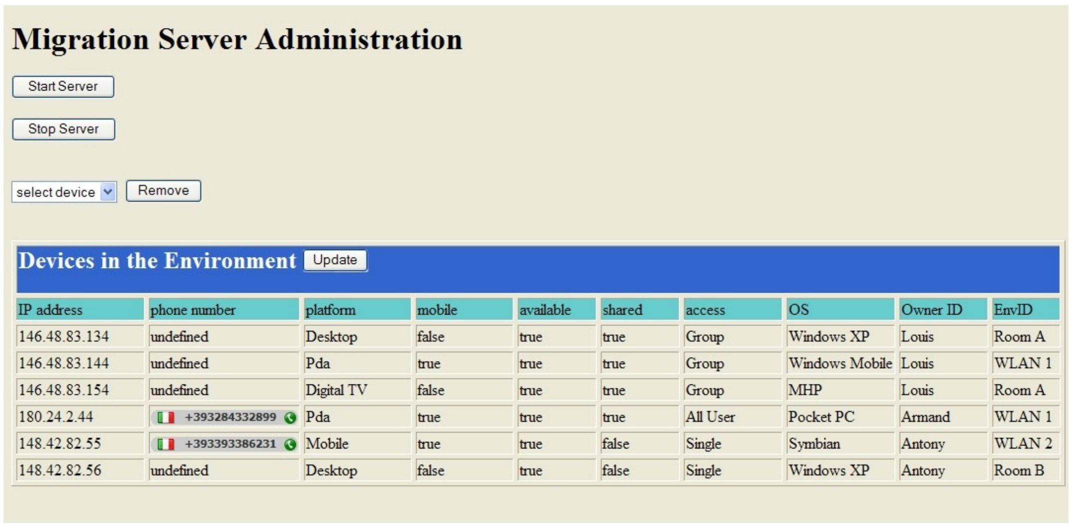
Figure 4: Examples of the transformations carried out by the Reverse Engineering Module applied to the Shopping Example

In Figure 4 some examples of the (parts of the) transformations carried out by the Reverse Engineering Module to the Shopping Application case study have been highlighted. As you can see, a list of bullets has been reversed in a grouping of navigators, while the information regarding each pasta item has been reversed into another grouping (composed of a text_edit, an object element, a navigator and a textual element).

Trigger Manager

Migration can be triggered by the user, who decides when it is the moment to activate a migration. In this case, the user selects the device on which the UI should migrate by interacting with a Migration Client (see Section 1.1).

In Figure 5 there is a screenshot highlighting an example of the list of devices that are detected in the environment, together with their associated properties. Such properties, which characterise each device, are described in a specific file contained in each device, and specified through appropriate XML tags. Such information is sent by the device to the server so as to enable the latter one to analyse and appropriately manage the list of the devices currently discovered in the environment.



IP address	phone number	platform	mobile	available	shared	access	OS	Owner ID	EnvID
146.48.83.134	undefined	Desktop	false	true	true	Group	Windows XP	Louis	Room A
146.48.83.144	undefined	Pda	true	true	true	Group	Windows Mobile	Louis	WLAN 1
146.48.83.154	undefined	Digital TV	false	true	true	Group	MHP	Louis	Room A
180.24.2.44	+393284332899	Pda	true	true	true	All User	Pocket PC	Armand	WLAN 1
148.42.82.55	+393393386231	Mobile	true	true	false	Single	Symbian	Antony	WLAN 2
148.42.82.56	undefined	Desktop	false	true	false	Single	Windows XP	Antony	Room B

Figure 5: The list of devices discovered in the environment

Semantic Redesign

This module is in charge to perform the adaptation to the target device. For this purpose it takes the abstract elements identified by the Reverse Engineering module and maps them into concrete elements more suitable for the target device, (while supporting a similar set of tasks and communication goals), from which it is possible to automatically generate the corresponding implementation. Figure 6 shows the phases of semantic redesign for a desktop-to-mobile transformation.

- *Transforming the desktop logical interface into a mobile logical interface*: the concrete elements of the desktop description are substituted by concrete elements supported by the mobile platform (for example, a radio-button with several elements can be replaced with a pull-down menu, which occupies less screen space, or images originally displayed in the source (desktop) platform are resized according to the screen size of the target (mobile) device, while keeping the same aspect ratio.

- *Calculating the resulting cost in terms of resources*: this transformation is based on the number and cost of interactors and their compositions (example of "cost" are: the font size (in pixels), number of characters in a text, image size (in pixels),etc.). Once the cost of the presentation is calculated, if it is under the maximum allowed cost, then the process terminates, otherwise, the presentation is split into two or more pages and the cost of each composition of elements is calculated: the one with the highest cost is associated to a newly generated presentation, which is replaced in the original presentation with a link to the new presentation.

- *Splitting the logical interface into presentations that fit the cost sustainable by the target device*. Page splitting requires a change in the navigation structure with the need for additional navigator interactors for accessing the newly created pages.

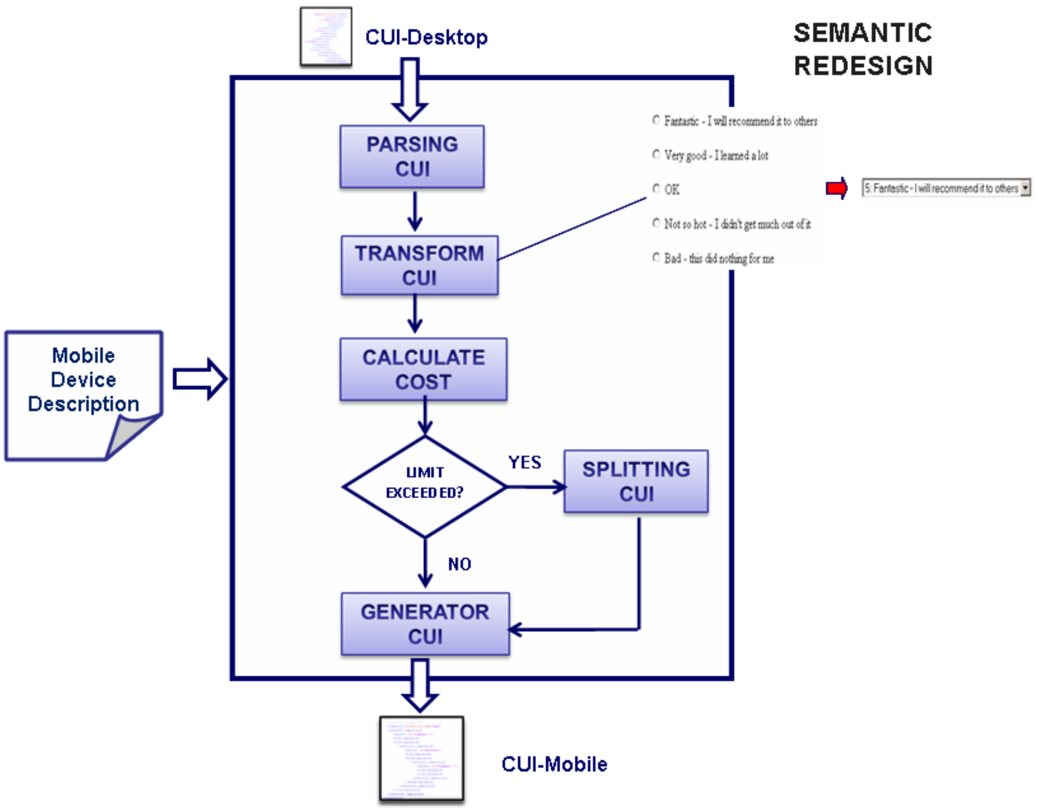

CUI-Desktop
SEMANTIC REDESIGN
PARSING CUI
TRANSFORM CUI
CALCULATE COST
LIMIT EXCEEDED?
YES
SPLITTING CUI
NO
GENERATOR CUI
Mobile Device Description
CUI-Mobile
Fantastic - I will recommend it to others
Very good - I learned a lot
OK
Not so hot - I didn't get much out of it
Bad - this did nothing for me
5: Fantastic - I will recommend it to others

Figure 6: Desktop-to-Mobile Semantic Redesign.

The cost that can be supported by the target mobile device is calculated by identifying the capabilities of the device through the *user agent* information in the HTTP protocol, which can be used to access more detailed information in a local XML repository obtained through WURFL (wurfl.sourceforge.net/), a device description repository containing a catalogue of mobile device information.

State Mapper

The objective of the State Mapper Module is to update the CUI for the target device (which has been produced by the Semantic Redesign module) with latest information regarding the state of the user interface contained in the DOM of the source page just before migration. Therefore, the State Mapper is the module that, once a concrete description for the target device has been obtained, it has to associate in it the state resulting from the user interactions in the source user interface. The corresponding elements in the two files are identified thanks to the fact that each object of the CUI has a unique identification label (ID), which is the same of the corresponding XHTML/DOM element from which that CUI element was generated by the reverse engineering process.

One possible complicating factor is when the semantic redesign has transformed a specific concrete object C1 (for a specific platform) into a different concrete object for the target platform, C2. In this case, since the same ID is maintained among the two concrete objects C1 and C2, the association between the concrete object and the corresponding DOM element is still straightforward (the same ID is maintained). Nevertheless the State Mapper may require a further step, that is, adapting the value of the DOM element to specify the new concrete object. For instance, it might happen that, as a result of the semantic redesign process, a radiobutton element was translated into a pulldown menu element. Therefore, the values included in the specification of the radiobutton element (e.g.: the different items of the radiobutton) have to be appropriately adapted and used to fill in the specification of the pulldown menu element.

User Interface Generator

This module is in charge of building the Final User Interface (FUI) in an implementation language suitable for the target device, starting with a concrete description of the user interface for the platform considered (the so-called CUI or Concrete User Interface): depending on the considered interaction platform and the specific implementation language, a particular transformation is selected by the UI Generator. It is worth noting that, from the earlier version of migration infrastructure this module has been radically improved since now it is able to dynamically generate the presentations for the target device (while in the previous case only pre-computed presentations were selected on the target device).

The UI Generator starts with parsing (in a top-down manner) the target Concrete User Interface description (which has a tree-like form), starting from the *root* node and depending on the type of node it follows different strategies:

- In case of *elementary CUI object*, the UI generator will transform it in the corresponding UI element of the implementation language considered. For instance if we consider as target implementation language XHTML, a CUI listbox element can be mapped onto a <select> element, which is composed of a number of <option> elements defining the various items within the CUI listbox. However, if we consider another implementation language (e.g. Java, or C), for the same CUI listbox element we will produce a different implementation.
- In case of *composition CUI object* (e.g. multiple elements combined by some concrete techniques for grouping them like fieldsets, lists, ..), the associated compositional techniques will be implemented in the final implementation language used. For instance, if at the implementation level we consider XHTML language, in this case when passing from the

concrete level to the implementation one, the concrete level primitives are mapped into XHTML constructs, for instance concrete grouping techniques can be mapped onto tag <fieldset> (group of fields in a form) and <div> (divisions or sections in the XHTML page) (with various attributes), which can be used as composition techniques at the XHTML implementation level.

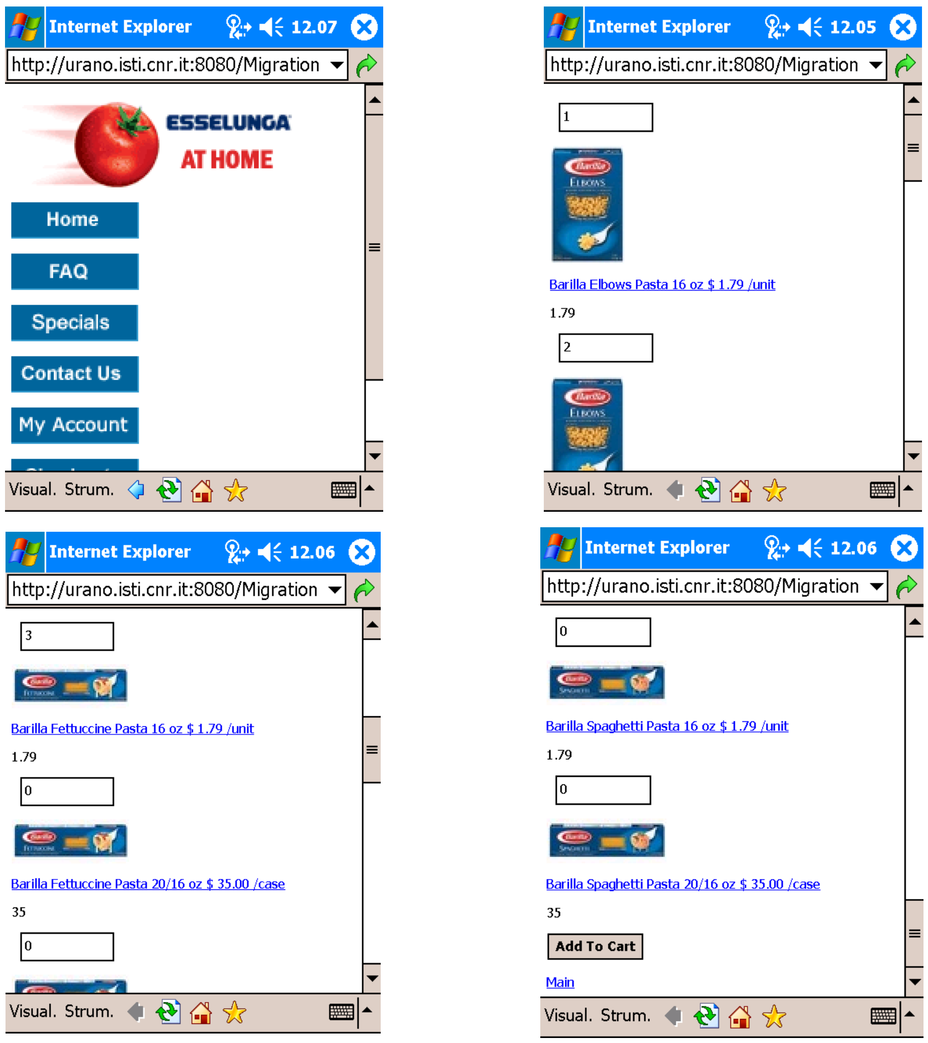

Internet Explorer 12.07
http://urano.isti.cnr.it:8080/Migration
ESSELUNGA AT HOME
Home
FAQ
Specials
Contact Us
My Account
Visual. Strum.
Internet Explorer 12.05
http://urano.isti.cnr.it:8080/Migration
1
Barilla Elbows Pasta 16 oz $ 1.79 /unit
1.79
2
Visual. Strum.
Internet Explorer 12.06
http://urano.isti.cnr.it:8080/Migration
3
Barilla Fettuccine Pasta 16 oz $ 1.79 /unit
1.79
0
Barilla Fettuccine Pasta 20/16 oz $ 35.00 /case
35
0
Visual. Strum.
Internet Explorer 12.06
http://urano.isti.cnr.it:8080/Migration
0
Barilla Spaghetti Pasta 16 oz $ 1.79 /unit
1.79
0
Barilla Spaghetti Pasta 20/16 oz $ 35.00 /case
35
Add To Cart
Main
Visual. Strum.

Figure 7: The pages of the Shopping Application example redesigned for PDA platform after migration.

In Figure 7 you can see an example of how the UI Generator delivers the pages for the PDA platform after the migration of the desktop version of the Shopping application (visualised in Figure 3) has been requested. As you can note, the initial single desktop page has been redesigned and a splitting was necessary in order to produce an usable result on the target platform. As a consequence, four pages have been produced and some adaptation has been carried out (e.g. the links in the navigation bar now are aligned vertically rather than horizontally as happened in the desktop version). In addition, the state has been preserved during migration, as you can see from the item unit values that are still visualised in the PDA version.

Migration Manager

As it is shown in Figure 1, the Migration Manager is the general orchestrator within the prototype architecture. It works as proxy server, since when clients access the Web pages, their requests pass through it, which downloads the pages from the application servers, and also annotates them with scripts able to capture the UI state.

In particular, when a request for migration to another device is triggered, the environment detects the state of the user interface as modified by the user input (elements selected, data entered, …), and identifies the last element accessed in the source device. For this purpose, when clients access the Web pages, their requests pass through a proxy server, which downloads the pages from the application servers, and also adds to them the scripts able to capture the UI state and communicate it to the server. Such scripts through a polling-based monitoring mechanism, implemented through an Ajax script, determine whether or not a migration was triggered by the migration client. When the user sends a migration request an AJAX callback function is automatically activated, which sends the DOM (containing the state of the current page) collected through a specific script. The information is collected in a string formatted following an XML-based syntax and sent to the server. This mechanism was chosen because only an application running on the browser in the client device can access the application DOM, and the AJAX Script can transmit the data without requiring any explicit action from the user.

3 Conclusions

This deliverable describes the current state of the prototype supporting the migration of the user interface software.