Automatic Reverse Engineering of Interactive Dynamic Web Applications to Support Adaptation across Platforms

Federico Bellucci, Giuseppe Ghiani, Fabio Paternò, Claudio Porta
CNR-ISTI, HIIS
Via Moruzzi 1, 56124 Pisa, Italy
{federico.bellucci, giuseppe.ghiani, fabio.paterno, claudio.porta}@isti.cnr.it

ABSTRACT
The effort and time required to develop user interface models has been one of the main limitations to the adoption of model-based approaches, which enable intelligent processing of user interface descriptions. In this paper, we present a tool to perform reverse engineering of interactive dynamic Web applications into a model-based framework able to describe them at various abstraction levels. We indicate how information in HTML, HTML 5, CSS, Ajax and JavaScript is transformed into such logical framework, which facilitates adaptation to other types of interactive devices. We also discuss how this reverse engineering tool has been exploited in an environment for run-time adaptation or migration of interactive Web applications to various devices in ubiquitous use cases.

Author Keywords
User interface reverse engineering, Web applications, Model-based user interface descriptions.

ACM Classification Keywords
H.5 Information Interfaces and Presentation; H.5.2 User Interfaces

General Terms
Design, Human Factors.

INTRODUCTION

Model-based approaches have been considered in order to aid user interface design and development by providing abstractions useful to manage the increasing complexity of user interface implementations. They have been used for many purposes (for example in [11] a model-based tool to guide early interface design has been proposed), and are also currently under consideration by the W3C for standardization purposes [4]. One of their main applications is for supporting design and development of multi-device user interfaces, since each device has specific interaction resources and implementation languages to execute such user interfaces. The basic idea is to provide a universal small conceptual vocabulary to support user interface design, which can then be refined into a variety of implementation languages with the support of automatic transformations, without requiring developers to learn all the details of such implementation languages. However, the effort and time required to develop user interface models has been one of the main limitations to the adoption of model-based approaches.

In order to overcome such limitation, we propose a tool that performs reverse engineering of interactive Web applications into MARIA [9], a model-based framework for describing interactive applications at two abstraction levels. In particular, the MARIA framework is composed of one abstract (platform-independent) language and various concrete (platform-dependent) languages. We use the platform concept to indicate groups of devices that share similar interaction resources (desktop, smartphone, vocal, …). The concrete languages add to the abstract language refinements that depend on the class of devices to which they correspond. Thus, they still provide descriptions that are independent of the final implementation languages. The advantage of transforming the implementation into a concrete MARIA description is that in this way it is then easier to adapt it to a different platform, since all the platform-dependent descriptions share the same core set of concepts that are derived from the abstract description. In addition in this way the adaptation rules can be independent of the implementation languages used for the interactive applications.

The reverse engineering tool is able to create logical descriptions of the whole implementation of Web pages into the MARIA language, managing HTML as well as CSS and most JavaScript, and thus preserving the aspect and functionality of the original pages. A logical description of the page is easier to manipulate than the original page itself: the interactive application elements can be more easily transformed according to predefined rules for specific target devices before re-implementing the page starting with the modified logical description. This will result in a more compact user interface that better suits the limited screen of a mobile device.

The Reverse engineering tool supports several steps, each of which manages an aspect of the input page. It is able to analyse the DOM of the current page, thus any effect of dynamic changes can be immediately detected. The analysis includes both the styles defined within the HTML document

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IUI'12, February 14–17, 2012, Lisbon, Portugal.
Copyright 2012 ACM 978-1-4503-1048-2/12/02...$10.00.

and the external CSS files. JavaScript code is also extracted from the document and the referenced external files. The events related to HTML elements (such as *onclick*, *onload*, …) are mapped in the MARIA specification, which provides the possibility to describe events and event handlers as well.

In the paper, after discussing some related work we provide some background information on the MARIA framework in order to allow readers unfamiliar with it to better understand how our reverse tool works. Then, we move on to describe the algorithm and the rules that allow the tool to transform the implementation of Web applications into MARIA specifications. We show an example application of the approach proposed and report on some work to validate it and show its generality. We discuss what results this reverse engineering tool can enable by describing how it has been integrated in a platform supporting interactive application adaptation. Lastly, we provide some concluding remarks and some indications for future work.

RELATED WORK

In recent years, interest in user interface reverse engineering has received strong impetus from the advent of mobile technologies and the need to support multi-device applications. To this end, a good deal of work has been dedicated to user interfaces reverse engineering in order to identify corresponding meaningful abstractions (see for example [1, 2, 13]). Other studies have investigated how to derive the task model of an interactive application starting with the logs generated during user sessions [5]. However, this last approach is limited to building descriptions of the actual past use of the interface, which is described by the logs, but is not able to provide a general description of the tasks supported, which includes even those not considered in the logs. A different approach [3] proposes re-engineering Java graphical desktop applications to mobile devices with limited resources, without considering logical descriptions of the user interface. Hwang et al. [6] have introduced Web transcoding techniques, based on heuristics, that take into account the structure of the page, the main components and their semantics. The page is then automatically rearranged in order to be accessible from handheld devices.

Solutions for adaptation of Web interfaces to specific target devices also directly involving end users have being investigated as well. For example, a support for crowdsourced adaptation is presented in [14]: the end user is seen both as a consumer and as a contributor, and is provided with a visual environment to easily adapt the application to different contexts (e.g. mobile devices).

In the area of user interface reverse engineering, a model-based architecture to reverse event handlers of applications developed with RAD (Rapid Application Development) environments is presented in [15]. The aim is to convert existing legacy applications into Web-based ones by exploiting platform-independent representations for the original code, in particular for the event handlers. In particular, the authors of this work have focused on the case of Oracle Forms converted into Web applications with Ajax.

Memon et al. [8] describe an application called GUI Ripping, which consists in a dynamic process that transverses a GUI by opening all its windows and extracting all the widgets (GUI objects) and their information. GUIsurfer [13] defines a framework for tools capable of analysing the source code of applications written in Java/Swing, and generating behavioural models of their user interfaces, which can be useful for supporting formal reasoning. Another tool, Swing2Script [12], supports conversion of Java-Swing applications to Web applications implemented in XUL.

One of the main areas of interest has been how to recover semantic relations from Web pages. An approach based on visual cues is presented in [16], in which semantic relations usually apply to neighbouring rectangle blocks and define larger logical rectangle blocks. However, previous work in the reverse engineering of interactive Web applications (such as ReversiXML [2] for UsiXML, ReverseAll [1] for TERESA, …) has mainly addressed the possibility of reversing HTML tags. Unfortunately, the way to write Web applications has changed considerably in recent years and the HTML part is only a small fraction of Web applications. Usually a Web application is a set of resources (style sheets, scripts, …), which can be dynamically loaded, linked by the HTML core, and the corresponding DOM tree can be modified at runtime within the browser. In general, there is a lack of approaches able to address all such resources at runtime and build the corresponding logical descriptions, which can be exploited in multi-device and multimodal environments. This also implies the ability to analyse the JavaScript parts of Web applications. This is a difficult task given the wide variety of ways in which JavaScript can be used. An example JavaScript analysis was done with Feedlack [7], a tool that explores Web applications' behaviours to identify missing feedback. This processing is performed by enumerating control flow paths originating from user input, identifying paths that lack output-affecting code. FeedLack was applied to 330 applications; of the 129 that contained input handlers and did not contain syntax errors, 115 were successfully analysed (which is about one third of the original set of applications considered, indicating how complex this type of analysis is).

MARIA FRAMEWORK

As mentioned before, MARIA is composed of one abstract language and various concrete languages that refine it. The Abstract User Interface (AUI) level describes a UI only through the semantics of the interaction, without referring to a particular device capability, interaction modality or implementation technology.

An AUI is composed of various Presentations that contain model elements, which are presented to the user simultaneously. There are two types of model elements:

Interactor or InteractorComposition. The former represents every type of user interaction object, the latter groups together elements that are logically associated. According to its semantics an interactor belongs to one of the following subtypes:

- Selection. Allows the user to select one or more values among the elements of a predefined list. It contains the selected value and the information about the list cardinality. According to the number of values that can be selected, the interactor can be a Single Choice or a Multiple Choice.
- Edit. Allows the user to manually edit the object represented by the interactor, which can be text (TextEdit), a number (NumericalEdit), a position (PositionEdit) or a generic object (ObjectEdit).
- Control. Allows the user to switch between presentations (Navigator) or to activate UI functionalities (Activator).
- Onlyoutput. Represents information that is submitted to the user, not affected by user actions. It can be a text, a Description that represents different types of media, an Alarm, a Feedback or a generic Object.

The different types of interactor compositions are:

- Grouping: a generic group of interactor elements.
- Relation: a group where two or more elements are related to each other.
- CompositeDescription: represents a group aimed to present contents through a mixture of Description and Navigator elements.
- Repeater: used to repeat the content according to data retrieved from a generic data source

MARIA allows describing not only the presentation aspects but also the interactive behaviour. For this purpose it has various features:

- Data Model. The user interface definition contains descriptions of the data types that are manipulated by the user interface. The interactors can be bound with elements of the data model, which means that, at runtime, modifying the state of an interactor will also change the value of the bound data element and vice-versa. This mechanism allows the modelling of correlations between UI elements, conditional layout, conditional connections between presentations, input values format. The data model is defined using the standard XML Schema Definition constructs.
- Generic Back End. The interface definition contains a set of ExternalFunctions declarations, which represents functionalities exploited by the UI but implemented by a generic application back-end support (e.g. web services, code libraries, databases etc.). One declaration contains the signature of the external function that specifies its name and its input/output parameters.
- Event Model. Each interactor definition has a number of associated events that allow the specification of UI reaction triggered by the user interaction. Two different classes of events have been identified: the Property Change Events that specify the value change of a property in the UI or in the data model (with an optional precondition), and the Activation Events that can be raised by activators and are intended to specify the execution of some application functionalities (e.g. invoking an external function).
- Dialog Model. The dialog model contains constructs for specifying the dynamic behaviour of a presentation, specifying what events can be triggered at a given time. The dialog expressions are connected using CTT operators in order to define their temporal relationships.
- Continuous update of fields. It is possible to specify that a given field should be periodically updated by invoking an external function.
- Dynamic Set of User Interface Elements. The language contains constructs for specifying partial presentation updates (dynamically changing the content of entire groupings) and the possibility to specify conditional navigation between presentations.

This set of new features provides, already at the abstract level, a model of the user interface that is not tied to implementation layout details, but it is sufficiently complete for reasoning on how the UI supports both the user interaction and the application back end.

A Concrete User Interface (CUI) in MARIA provides platform-dependent but implementation language-independent details of a UI. A platform is a set of software and hardware interaction resources that characterize a given set of devices. MARIA currently supports the following platforms:

- Desktop CUIs model graphical interfaces for desktop computers.
- Mobile CUIs model graphical interfaces for mobile devices.
- Multimodal Desktop CUIs model interfaces that combine the graphical and vocal modalities for desktop computers.
- Multimodal Mobile CUIs model interfaces that combine the graphical and vocal modalities for mobile devices.
- Vocal CUIs interfaces with vocal message rendering and speech recognition.

Each platform-dependent meta-model is a refinement of the AUI specifying how a given abstract interactor can be represented in the current platform. For instance, if we consider the abstract Single Choice interactor, it can be implemented (on a graphical desktop platform) with a radio button, a drop down list or a list box, while on the vocal platform it can be rendered with a list of vocal messages, one for each option associated to a given keyword.

The same applies to the interactor compositions: in a desktop platform a grouping can be implemented using background colours, borders, etc., while in a vocal platform it is possible to use sounds at the beginning and the end of a group of elements. The model definition can be exploited for creating (or deriving with a code generator) final implementations in different target languages. Indeed, it is possible to exploit the same mobile concrete user interface description for representing an App for the iPhone or an Android device.

THE OVERALL ALGORITHM IN THE REVERSE PROCESS

As already stated in the introduction, our reverse tool takes an HTML page, and the associated files (with stylesheets and scripts) as input and produces a CUI as output. The tool is able to directly access the DOM (Document Object Model) of the page to be reversed if it was previously created (e.g., for preprocessing purposes). Otherwise, the HTML file is passed to the Reverse tool, which exploits an HTML parser, named Tidy[1], to build the DOM. We have extended the Tidy parser in order to also manage the new tags introduced by HTML5. This parser also allows us to manage Web pages that are not well-formed according to the (X)HTML or HTML5 languages by correcting tags that are missing, improperly positioned or misspelled.

The Reverse main procedure consists of translating the HTML document, i.e. the actual page implementation, into a MARIA concrete, implementation-independent representation. The first check is to see whether JavaScript nodes exist in the DOM, in this case all their content is stored in an external file with extension ".js". The next check is whether there are CSS nodes in the DOM, in this case their information is stored in a cache memory. Then, the document is analysed through a deep first visit of the DOM tree. Each node is analysed and its CSS properties are managed by looking into the cache memory to see whether there are any CSS selectors referring to the currently analysed HTML node; in this case its content is used to specify attributes of the corresponding MARIA elements.

As mentioned before, CSS properties are reversed for all the elements present in the page and stored within their respective CUI representations. So some CSS rules may be lost in this process if there are no elements referring to those rules in the version of the page considered when the reverse is performed. This can be a problem when reversing dynamic Web pages that can add new elements at runtime or simply change the style properties of existing elements. To deal with this issue we make a static analysis of the JavaScript code to spot assignments to class attributes, which are largely responsible for the changes in style elements in dynamic pages. Then, the CSS content is analysed to see whether there are attributes referring to the classes identified. If they exist their content is included in the CUI.

[1] http://tidy.sourceforge.net/

In a next step, the algorithm checks whether there are events associated with the current DOM node. There are two possibilities: one is that an event is indicated (e.g. onclick, onmouseover, onfocus, onload, …) along with the corresponding function call; in the other case, instead of a function call there is some code including functions definitions, declarations, variable instantiations. In the former case we add an event handler in the MARIA specification to indicate the JavaScript function that should be called. In the latter case, a function is created and is associated with the code contained in the event handler. Such code is included in the list of external functions and it is called by the corresponding event handler. Indeed, in MARIA a user interface can be associated with a list of external functions.

REVERSING HTML CODE

This phase of the Reverse process considers each single occurrence of HTML tags within the document and converts them into CUI elements, according to the specifications of the MARIA language. An excerpt from the conversion rules used for this purpose is shown in Table 1.

The document analysis starts from the *body* element, and the CUI is built while the visit proceeds in depth down to the leaves of the HTML tree.

For example, the table shows that the HTML INPUT tag is mapped onto various concrete MARIA elements depending on its type attribute, which substantially determines it semantics.

The Reverse primary output is an XML file containing the logical description of the input Web page in the MARIA language and a file containing the associated JavaScripts that can thus be reused in case of generation of an adapted version for another platform.

The Reverse is able to manage HTML 5 tags as well. Some conversion rules are listed in Table 2. In the case of TIME and METER tags the Reverse creates a MARIA element and in addition a corresponding data type in the data model. Various HTML 5 tags mainly provide an indication of the type of content associated with some presentation techniques (e.g. ARTICLE, HEADER, NAV, FIGCAPTION). This distinction is mainly captured through the role attribute in the MARIA specification.

REVERSING CSS

Two modules are involved in the CSS reverse phase: *CSS Handler* and *CSS Cache*. The CSS Handler is in charge of parsing the CSS information of the input page, extracting it and storing it in the CSS Cache. It is worth noting that any style information is considered in the CSS processing phase, including:

- External CSS files referred by *link* tags
- Internally defined *style* tags containing CSS code
- Content of style attributes within HTML tags

X(HTML) element		MARIA CUI element (desktop)
Tag name	*Attribute(s)*	
A	-	Navigator
	any event attribute	Activator
P	-	CompositeDescription (description | navigator | activator)
DIV	-	Grouping
IMG	-	Image
FORM	-	Relation
INPUT	type = null | text | password	TexEdit
	type = checkbox	MultipleChoiceType(Checkbox, Choiceelements++)
	type = radio	SingleChoiceType(RadioButton, ChoiceElements++)
	type = hidden	Object
	type = reset | submit	ActivatorButton
	type = submit	SubmitButton
BUTTON	type = reset | submit	Activator(Button)
	type != reset, type != submit	Navigator(Button)
TEXTAREA	-	TextEdit
SELECT	multiple = null	SingleChoice(DropDownList, ChoiceElements++)
	multiple != null	SingleChoice(ListBox, ChoiceElements++)
OL, UL, LI	-	Grouping
IFRAME	-	Grouping
COMMAND	-	Grouping

Table 1. HTML to MARIA conversion rules.

The first two cases are managed by a CSS parser (CSSOMparser) that builds an object Java structure, which facilitates its processing. The third case does not require any parser and is managed directly within the HTML analysis phase (because the content of a style attribute is already in HTML).

The CSS Cache is a data structure implemented as a Hash Table, with CSS selectors as keys and CSS properties as values. The data structure hosts in memory any CSS information until CUI elements creation begins. Thus, the CSS information is not immediately used because it defines properties that depend on the document structure. Indeed, the binding between attributes and elements is done according to the CSS selectors.

HTML5 element		MARIA CUI	
Tag name	*Distinctive element*	*Element*	*Attribute*
TIME	-	Description(text) + DataType containing the data	
METER	-	Description(text) + DataType containing the values max, min, val	
ARTICLE	If only output children or links	Composite description (description|navigator|activator)	role = article
	not only output or link children	Grouping	
FIGURE	only output children or link	Composite description (description|navigator|activator)	role = figure
	not only output or link children	Grouping	
COMMAND	-	Activator(button)	
MARK	-	Composite description (description|navigator|activator)	
ASIDE	-	Grouping	role = aside
FOOTER	-	Grouping	role = footer
HEADER	-	Grouping	role = header
HGROUP	-	Grouping	role = hgroup
NAV	-	Grouping	role = nav
SECTION	-	Grouping	role = section
MENU	-	Grouping	role = menu
DETAILS	-	textType	role = details
FIGCAPTION	-	textType	role = figcaption
OUTPUT	-	text	role = output

Table 2. HTML 5 to MARIA conversion rules.

INTEGRATION OF JAVASCRIPT IN MARIA

Scripting languages are becoming more and more common, not only within Web pages, but also in servers (e.g. the NodeJS JavaScript library) and in standalone applications (e.g. ActionScript-AIR applications). When the MARIA language was designed it was provided with some basic

constructs to represent generic programming statements such as assignments, conditions, loops, etc. When we started to extend the Reverse module to support reverse engineering of dynamic Web applications we realized that it was not worth reimplementing a new abstract programming language within MARIA. So we decided to add the possibility to include scripts in MARIA using JavaScript itself. This type of solution mainly works for Web languages, but within them it could work with different interaction modalities (graphical, vocal, multimodal, …).

To make the original JavaScript compliant with the CUI generated by the reverse process, the JavaScript code is parsed and modified. For instance, since the MARIA language does not implement a *name* attribute for its elements, the Reverse concatenates the *name* to the *identifier* (eventually generated), and modifies every related *identifier* in the JavaScript. The modifications are applied to the Abstract Syntax Tree (AST), which is created by a specific procedure starting from the JavaScript code. The modified AST is then stored in XML format along with the CUI.

In the case of an adaptation of the HTML page to a mobile platform, the XML representation of the JavaScript can be used to regenerate the original JavaScript, along with an adapted version of the original HTML, disabling or modifying any undesired part of the JavaScript code. An even more challenging scenario is the adaptation towards a vocal platform, such as a VoiceXML voice browser, which usually integrates a limited JavaScript interpreter that cannot access the DOM of the VoiceXML document. An example of a tool supporting desktop-to-vocal adaptation is reported in [9], which produces a simplified VoiceXML adaptation of the original Web page. In order to improve it, the JavaScript XML representation produced by the Reverse module can be translated into native VoiceXML constructs (such as VAR, ASSIGN, IF) and integrated into the VoiceXML adapted version of the Web application. Thereby, it reproduces part of the dynamic behavior of the source Web application in the generated vocal application.

VALIDATION

The Reverse tool has been successfully applied to various existing Web sites. An excerpt from a validation exercise is shown in Table 3.

The input was a set of pages belonging to the 100 most highly ranked well-known international Web sites (according to http://www.alexa.com).

Before triggering the Reverse, each page was downloaded and annotated through our proxy server. The annotation consisted of two steps:

- converting every URL on the page into an absolute address including the proxy server, in order to enable the Reverse to find every resource referred to by the page (e.g., external CSS and JavaScript)
- insertion of the scripts for automatically forwarding the currently visited document to the Reverse Web service.

The URL conversion is needed because the Reverse procedure runs locally and is not able to resolve relative addresses; thus, any reference within the original HTML page has to be extended as an absolute URL.

The input pages were classified by HTML depth and size, as well as the size of externally referred CSS. Each input page was also validated through the W3C Markup Validator Service (http://validator.w3.org/) in order to detect the number of errors and/or warnings. The validation was carried out on the (X)HTML version of the page, which was automatically detected by the W3C validator.

All the trials were successful (i.e. there were no failures in building the corresponding CUI), and the output obtained from the Reverse was classified by size and number of lines of the generated CUI file, and by the time taken to perform the Reverse.

The results do not reveal any correlation between the Reverse performance and a particular aspect of the page. For example, the three pages that took the longest time, which are #2, #4 and #10 in Table 3) are among the ones with the greatest amount of HTML and CSS. Nevertheless, #9 also has quite a large amount of HTML and CSS but took less than 2 seconds to be reversed. However, it is worth noting that, while #2, #4 and #8 contain more than 100 validation errors, #7 was totally correct. Thus, a combination of parameters, such as page size and errors, seems to affect the Reverse time.

The more elements the page has, the more time is needed by the Reverse to translate the page into the MARIA language, because every element has to be analyzed before being converted. Unfortunately, most of the Web pages are affected by validation errors (see the column "Errors/Warnings" on Table 3). Validation errors also impact negatively on the Reverse time, because every inconsistency, missing attribute or invalid character may raise an exception on the parsing procedure, which is performed according to W3C HTML specifications. All such issues have to be fixed in some way by the parser, thus having a negative impact on the overall Reverse time.

A quality indicator for the pages back-generated starting with the MARIA concrete description obtained by the reverse tool is reported in Table 3. The values listed in the "Generation" column refer to the level of consistency with the layout and the functionalities of the original page. The generation of the new pages was obtained from MARIA concrete descriptions adapted to a mobile device (i.e., to a device with limited screen size). We have defined the following three quality levels:

- A: The layout of the page as well as the functionalities are maintained.
- B: The layout is mostly maintained, while a few functionalities are not fully accessible, but can be easily corrected.

- C: The layout is mostly maintained and it would enable the user to interact with the page, but the regenerated JavaScript does not support all of the original functionalities.

Most of the generated Web pages received the intermediate score "B" because of some layout inconsistencies and/or some small lack in the original functionalities. In the "B" level, lacking functionalities are typically caused by issues in the dynamic query formulation, when the actual URL of the page is different from the one expected by the JavaScript. This type of issue can be solved by a minimal manual intervention.

The "C" score was given to those pages that, even while maintaining an acceptable layout to potentially preserve the interaction, had some underlying functional inconsistencies. For instance, in pages with dynamic JQuery management of the event handlers, the environment was unable to correctly restore to the adapted implementation the event handlers previously associated to the interactive elements.

APPLICATION TO ADAPTATION AND MIGRATION

We have integrated our tool for reverse engineering in an environment for adaptation or migration of interactive applications. This adaptation environment is server-based (see Figure 1). The adaptation server includes a proxy so that when users navigate Web pages, the links therein are modified to force them to pass through the server. The accessed Web pages are downloaded in the server, including the external resources that they refer to (JavaScripts, CSS files, …). These are passed to the reverse engineering tool, which performs the processing described in the previous sections. The resulting MARIA specifications can be passed to one of the possible adaptation engines. The choice of which adaptation transformation to apply depends on the device currently used by the user. So far, we have considered the possibility to adapt to mobile devices, characterised by smaller screens and lower processing capabilities, and to vocal devices, in which case there is a complete change of interaction modality. These adaptation tools are complex and their description is beyond the scope of this paper, whose focus is on the reverse engineering tool that enables such adaptation possibilities. However, it is worth pointing out that they are based on the features of the MARIA language, since they exploit concrete languages for the target platforms (vocal devices, mobile devices). Since such languages share the same core structure (the abstract MARIA language) it is easier to define and implement the adaptation rules. Indeed, when such transformations find a graphical element (e.g. a radio button), then they look at the corresponding abstract semantic effect (in this case a single selection), and lastly inspect how they are supported in the target concrete language (for example in the vocal concrete language there is a vocal choice).

In addition to adaptation, another feature that we have found useful in ubiquitous environments is migration. It allows mobile users to dynamically change the interaction device and migrate the interactive application from the source to the target device while preserving its state. In a Web application this means that users can find on the target device the interactive application at the point in which it was left off and also still find the input they entered in the forms, the JavaScript in a consistent state, the same cookies and session variables. This allows users to freely move about, change device and still be able to continue their activities in a seamless manner.

Input								*Output MARIA CUI*		*Reverse Time (s)*	*Generation*
#	*Domain*	*Page*	*Size (B)*		*DOM Depth*	*Version*	*Errors / Warnings*	*Size (B)*	*Lines*		
			HTML	**CSS**							
1	www.ebay.com	Home	211.015	69.120	18	HTML 4.01 transitional	359 / 16	569.045	9.307	3,736	**A**
2	motors.shop.ebay.com	25 items search result	456.398	123.767	30	HTML 4.01 transitional	175 / 1	978.707	18.144	12,703	**C**
3	www.google.com	Home	91.595	13.045	22	HTML 5	35 / 2	55.460	940	4,390	**B**
4	www.yahoo.com	Home	391.897	209.311	18	HTML 5	130 / 8	281.694	3.517	12,172	**A**
5	www.youtube.com	Home	374.195	232.526	18	HTML 5	130 / 2	357.385	4.801	10,892	**B**
6	www.youtube.com	24 videos search result	268.423	232.518	16	HTML 5	97 / 3	544.135	7.189	10,453	**B**
7	www.msn.com	Home	226.549	177.212	19	XHTML 1.0 strict	0 / 0	220.129	5.532	1,797	**B**
8	www.linkedin.com	User Home	388.518	192.542	20	HTML 5	117 / 118	1.007.023	15.828	11,920	**C**
9	www.bbc.co.uk	Business News	198.135	327.400	20	XHTML + RDFa	0 / 0	543.858	9.016	6,484	**B**
10	en.wikipedia.org	Technology	246.725	98.459	15	XHTML 1.0 transitional	2 / 0	541.514	10.731	4,188	**B**
11	www.amazon.com	Kindle eBooks	462.143	104.257	31	XHTML 4.01 transitional	433 / 157	463.086	8.147	8,875	**C**
12	wordpress.org	Forum	52.514	66.871	18	XHTML 1.0 transitional	1 / 0	180.140	3.524	2,563	**A**
13	www.bing.com	10 items search result	119.816	4.310	20	XTHML 1.0 transitional	6 / 0	147.909	2.511	1,297	**B**

Table 3. Validation test results.

The output of the adaptation components are MARIA specifications in the concrete language of the target platform. Then, there are generators for various corresponding implementation languages. For example, for vocal interfaces we have generators into VoiceXML, for Web applications we have generators in HTL+CSS+JavaScripts or JSP, if Web service access is required.

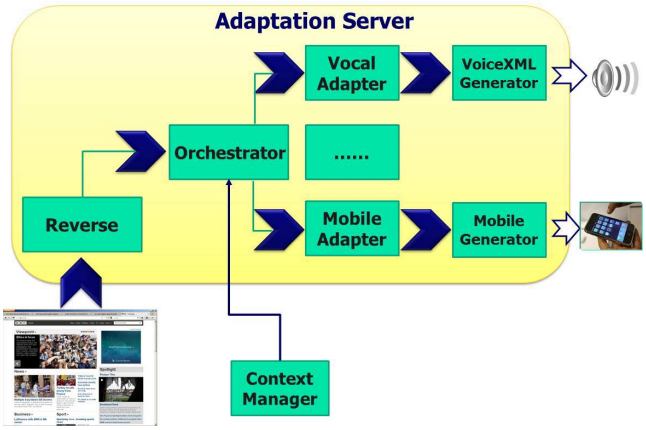

Figure 1. Adaptation Platform Exploiting our Reverse Engineering Tool.

EXAMPLE

To provide an example of what our Reverse produces when applied to a dynamic Web page, and how the generated elements relate to the original HTML, we can consider the reverse home page of altavista.com. The input page is shown in Figure 2, highlighting some of the components of the original page. An example of dynamic feature is the possibility to show suggestions of possible terms depending on what characters the user types. The management of such events is not specified in the HTML code but is dynamically added through JavaScrips that modify the DOM. With respect to Figure 2, the component of the page circled in yellow (5) is converted (see Figure 2) into a *Grouping*; the one circled with blue dots (3) becomes an *Activator*; the one highlighted with red lines/dots (2) turns into a *Text Edit*; the components circled with green dashed lines (1, 4) turn into *Navigators*.

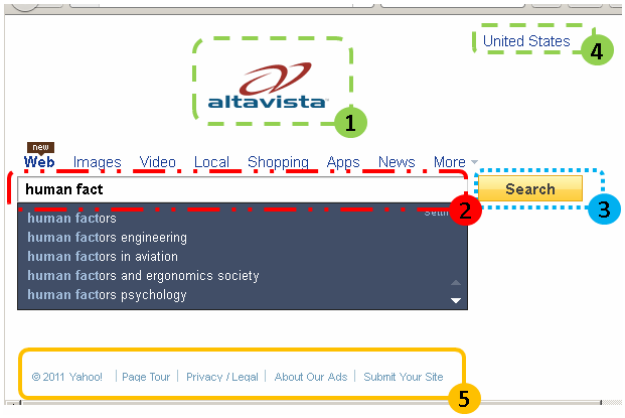

Figure 2. An example of input page highlighting some components.

For the sake of clarity, only some components are considered in this example because the actual Reverse deals with an input HTML page of about 24 kB. Figure 3 shows the sections of the output CUI related to the highlighted parts of Figure 2. Note that most of the CUI MARIA specification has been cut or collapsed, in order to make it more legible. Such CUI can be adapted for mobile access, and then from the adapted CUI it is possible to generate the corresponding implementation.

```
...
<composite_description id="Obj_2">
    <navigator focus="false" id="link_4">
1       <link><label><image alt="Image: " source="http://a.l.yimg.com/a/i/us
    </navigator>
</composite_description>
...
<relation id="sf" padding-right="0" padding-left="0" padding-bottom="0" padd
        ...
            <text_edit id="yschsp.p">
2               <text_field password="false" length="8" text=""/>
            </text_edit>

            <activator focus="false" id="yschbt">
3               <button margin-left="6px">
                    <label><text><string>Search</string></text></label>
                </button>
            </activator>
        ...
    <relation_settings/>
</relation>
...
<navigator focus="false" id="link_43">
4   <link>
        <label><text role="normal"><string>United States</string> <font_sett
    </link>
</navigator>
...
<grouping padding-right="0" padding-left="0" padding-bottom="0" padding-top=
5   <grouping margin-right="10px" id="Obj_67">
        <description focus="false" id="ShowText_68">
            <text role="normal"><string>© 2011 Yahoo!</string></text>
        </description>
        <properties position="column"/>
    </grouping>
        ...
            <navigator focus="false" id="link_69">
                <link><label><text role="normal"><string>Page Tour</string><
            </navigator>
        ...
    <properties position="row"><font_settings decoration="none" weight="norm
</grouping>
...
```

Figure 3. An excerpt of the CUI produced by the Reverser.

In Figure 4 the Web page version resulting from the adaptation to a smartphone is shown. The adaptation support receives as input the MARIA CUI of the page and a set of constraints (device height/width, interactors transformation rules, tolerance to scrolling, ...). The output of the adaptation support is an adapted MARIA CUI, which describes a new version of the page optimized for the destination device (smartphone): the components have been rearranged in order to better exploit the screen space of the mobile device, and the page width has been reduced so that the need for horizontal scrolling is minimized. The actual implementation of the page is produced by a MARIA-to-HTML + JavaScript generator.

The Web page implementation generated from the CUI adapted for smartphones devices (Figure 4), with a slightly different layout, maintains the initial interactive functionalities. Indeed, the generated search form has the original attributes values (action, name and id of the input

elements), and the page generated for mobile access is able to perform a Web search through the submit button.

It is worth noting that almost all the page elements have maintained the original id, and that the JavaScript support is still able to access those elements. Thus, thanks to the ability to reverse and restore the JavaScript functions, the generated page for smartphones also provides the real-time dynamic suggestion of the terms while the user is typing in the text field as in the original version of the page.

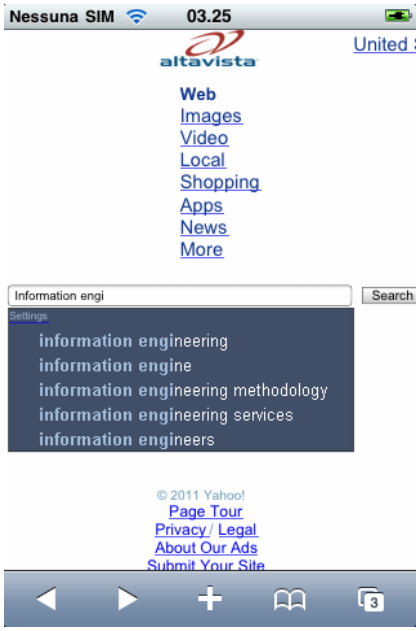

Figure 4. The resulting UI adapted to the smartphone.

CONCLUSIONS AND FUTURE WORK

In this paper we have presented a reverse engineering tool for interactive dynamic Web applications able to transform them into model-based specifications. Such model-based descriptions can be used for various purposes (adaptation tools for multi-device environments, documentation, support to usability evaluation, …). In particular, we have discussed how the Reverse tool has been integrated into a run-time platform where it allows the dynamic adaptation of the pages that users can access through various devices.

The desktop version of the tool is available for download at http://giove.isti.cnr.it/tools/ReverseMARIA/download

Future work will be dedicated to further improving the reverse engineering tool in order to address some cases that are not yet addressed related to JQuery and dynamic modifications, such as when JQuery scripts dynamically add event handlers to the DOM. Another possible extension is to provide the possibility to derive higher-level abstraction descriptions from the MARIA specification, such as task models that describe the activities performed by users to reach their goals, and that allow reasoning about their performance in intelligent environments.

ACKNOLEDGEMENTS

This work has been partly supported by the EU SERENOA STREP project (EU ICT FP7-ICT N.258030).

REFERENCES

1. Bandelloni, R, Paternò, F., Santoro, C. Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments, Proceedings EIS'07, Salamanca, LNCS 4940, 285-302, Springer Verlag, March 2007.
2. Bouillon, L., Limbourg, Q., Vanderdonckt, J., and Michotte, B. Reverse engineering of web pages based on derivations and transformations. In In: Proceedings of third Latin American web congress LA-Web'05. IEEE Computer Society Press, 2005.
3. Canfora, G., Di Santo, G., Zimeo, E. Toward Seamless Migration of Java AWT-Based Applications to Personal Wireless Devices, Proceedings WCRE'04, 1-9.
4. Fonseca, J.M.C. (ed.), W3C Model-Based UI XG Final Report, May 2010, available at http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/.
5. Hudson, S., John, B., Knudsen, K., Byrne, M. A Tool for Creating Predictive Performance Models from User Interface Demonstrations. Proceedings UIST'99, 93-102, ACM Press, 1999.
6. Hwang, Y., Kim, J., and Seo, E. Structure-Aware Web Transcoding for Mobile Devices. Proc. IEEE Internet Computing, pp. 14-21.
7. Ko, A. J., Zhang, X. Feedlack detects missing feedback in web applications. Proceedings CHI 2011, 2177-2186, ACM Press, Vancouver, 2011.
8. Memon, A., Banerjee, I., and Nagarajan, A.. Gui ripping: Reverse engineering of graphical user interfaces for testing. In WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering, page 260, Washington, DC, USA, 2003. IEEE Computer Society.
9. Paternò, F., Santoro, C., Spano, L.D. MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment. ACM Transactions on Computer-Human Interaction, Vol.16, N.4, November 2009, 19:1-19:30, ACM Press.
10. Paternò, F., Sisti, C. Model-Based Customizable Adaptation of Web Applications for Vocal Browsing, ACM SIGDOC, pp.83-90, ACM Press, Pisa, October 2011.
11. Puerta, A. R., Micheletti, M., Mak, A. The UI pilot: a model-based tool to guide early interface design. IUI 2005, 215-222.

12. Samir, H., Stroulia, E., Kamel, A. Swing2Script: Migration of Java-Swing Applications to Ajax Web Applications. WCRE 2007, 179-188.
13. Silva, J. C., Silva, C. E., Gonçalo, R- D., Saraiva, J., Campos, J. C.. The GUISurfer tool: towards a language independent approach to reverse engineering GUI code. EICS 2010, 181-186, ACM Press.
14. Nebeling, M., and C. Norrie, M. Tools and architectural support for crowdsourced adaptation of web interfaces. In Proceedings of the 11th international conference on Web engineering (ICWE'11). Springer-Verlag, Berlin, Heidelberg, 243-257.
15. Sánchez Ramón, O., Sánchez Cuadrado, J., García Molina, J. Reverse Engineering of Event Handlers of RAD-Based Applications. Proc. Of WCRE 2011. IEEE, 293-302.
16. Xiang, P., Shi, Y. Recovering semantic relations from web pages based on visual cues, Proceedings of the 11th international conference on Intelligent user interfaces, January 29-February 01, 2006, Sydney, Australia, 342-344.