WebUml: Reverse Engineering of Web Applications

Carlo Bellettini*, Alessandro Marchetto, Andrea Trentini
Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
Via Comelico 39, 20135 Milano, Italy

ABSTRACT

Web applications have become complex and crucial for many firms, especially when combined with areas such as CRM (Customer Relationship Management) and BPR (Business Process Reengineering). Since then the scientific community has focused attention to Web application design, development, analysis, testing, by studying and proposing methodologies and tools. This paper describes an automatic tool for the construction of UML models from existing Web applications. This tool, named WebUml, generates class and state diagrams by analysing source code and by interacting with the Web server. This reverse engineering tool is based on source code static analysis and also applies mutational techniques in order to exploit the server side execution engine to accomplish part of the dynamic analysis. This tool will be the core of a testing suite under construction at our laboratory. WebUml generated models (diagrams) will be used as a base for test case generation and coverage analysis.

1. INTRODUCTION

Web applications have become the core business for many companies in several market areas. The development, distribution and control of on-line services (on-line retail, on-line trading and so on) can be the mean to and/or the object of business. The growth of the World Wide Web led to the expansion of application areas for new on-line services. For example, many businesses have at least some Web presence with the relative e-commerce (buy/sell, CRM, products information) functionalities.

Web applications quality, reliability and functionality are important factors because any software glitch could block an entire business and determine strong embarrassments. These factors have increased the need for methodologies, tools and models to improve Web applications (e.g., applications design and development methodologies, automatic documenting tools, and development process and testing tools). One of the main concerns in Web applications is "speed" in technology change, content update and fruition. This speed is at the base of the Web software life cycle. Other important factors are complexity, large dimensions and applications design maturity. Web applications are heterogeneous, distributed, and concurrent: their testing is not an easy task. Conventional methodologies and tools may not be adequate.

This article describes WebUml, an automatic tool to reverse engineer existing Web applications. A taxonomy of approaches for web reverse engineering is described in [24]. WebUml aim is the extraction/reconstruction of a (albeit partial) UML model for an existing Web application. The extracted information is particularly meaningful for the kind of legacy Web applications where the business logic is embedded into the Web pages, instead of more recent and layered Web applications where the business logic is implemented through server-side components. A model is needed for the subsequent test phase (the test phase will be covered by TestUml, currently under development). The use of a model for testing allows the use of verified and powerful testing methodologies, e.g., the white–box ([5]) testing based on Object–Oriented techniques ([6],[18]).

The reverse engineering approach is also important to test all types of existing applications, above all the ones without any type of documentation or design project (i.e., old Web applications, or legacy applications). For example, it is possible to use a procedure based on reverse engineering techniques to create a set of test cases for regression testing. Regression testing allows the introduction of new functionalities keeping continuous coherence verification. The use of an Object–Oriented model, in particular the UML model, to represent existing Web applications allows the definition of a good description level (as stated by [8]), and permits to take advantage of pre-existing knowledge in Object–Oriented software testing techniques.

This paper is organized as follows. Section 2 is a review of existing works in modelling, reverse engineering and testing Web applications area. Section 3 introduces the tool application domain and the goal of WebUml. Section 4 details WebUml Web applications modeling. Section 5 introduces a simple XML Web application. Section 6 analyzes the WebUml reverse engineering tool and describes its architecture. Finally Section 7 describes future works based on WebUml.

2. RELATED WORK

Several Web modeling methodologies are presented in literature. RMM [16] is a method based on Entity-Relationship

*email: carlo.bellettini@unimi.it

SAC'04, March 14-17, 2004, Nicosia, Cyprus

diagrams to design hypermedia applications, it is specialized into applications based on regular data structures or databases. In RMM the application navigation is specified by an associative relation between entities (application domain objects), and information entities can be sliced in order to organize the navigational structure. WSDM [9] defines a user-centered design method for Web sites. HDM [14], OOHDM [28], are important design methodologies. They use models to define the conceptual structure of the applications, the navigational design and the user interface. These methodologies are often extensions of traditional methodologies, such as Object–Oriented ones. Moreover, there are proposals to use UML (Unified Model Language) [12] in Web applications [4],[17]. In particular [21] and [22] analyze specific assets of Web applications, as system navigation or dynamic aspects . [21] proposes a methodology to represent navigational structure through UML statecharts, and [22] proposes UML activity diagrams to represent user interfaces. An important work is [8] in which Conallen extends UML with stereotypes for Web design. WARE [10] and Rational Rose Web Modeler [3] are tools for reverse engineering supporting Conallen's extensions [8]. WARE performs essentially static analyses generating class, use cases and sequence diagrams.

Currently available Web testing tools (e.g., [2],[1]) are usually classifiable as syntax validators, HTML/XML validators, link checkers, load/stress testing tools, and regression testing tools. They are not focused on important functionalities of Web applications that cannot be tested by this type of tools. Some of these tools are often integrated into Web Browsers to capture user gestures and replay them in testing scripts. Tools and methodologies cannot provide structural or behavioral test artifacts of Web application to test. Moreover the capture and replay test represents a good compromise when a formal model is not available and the only implicit model is the user.

A different approach, more strictly related to this paper, is based on functional, structural and behavioral testing, such as: G.A.Di Lucca et al. [11]; Ricca and Tonella [27],[25]; and D.C.Kung, C.H.Liu and P.Hsia [20],[19]. G.A.Di Lucca et al. propose Object–Oriented Web testing strategy in order to build unit and integration testing process essentially based on functional criteria. The testing process is based on a high level representation of Web applications derived from a reverse engineering procedure realized in WARE tool [10]. D.C.Kung et al. propose an Object–Oriented Web Test Model that captures the artifacts representing three aspects: object (entities of application); behavior (navigation and state-dependent behaviors); and structure (flow control and data flow information). From described model, structural and behavioral test cases can be derived automatically to support the test process. Ricca and Tonella have developed ReWeb and TestWeb tools. ReWeb for reverse engineering Web applications into UML model, it performs several traditional source code analyses and uses UML class diagrams to represent components and navigational features. ReWeb is a semi-automatic tool, because an important role is played by the user, mainly in the insertion of form input values. TestWeb uses extracted model information to perform white-box testing to validate paths.

We propose a reverse engineering model that is based on static and dynamic analysis of a Web application. Our techniques uses static methods derived from the traditional source code analysis adapted to extract static and *dynamic* information about the composition of the application. Moreover a combined method based on static and dynamic analysis is used to define the navigational structure and the application behavior. We have paid particular attention to the server–side dynamic aspects of Web applications, we analyzed it with a dynamic method based on application execution and on mutational analysis applied to source code [13]. This dynamic analysis is performed with the generation of a set of server side script source code mutants, used into a navigation simulation, then the procedure results are analyzed with a static traditional HTML and client side script source code techniques. The use of mutation allows the lowering of user interactions in the reverse engineering phase and permits defining a more detailed description.

3. WEBUML RATIONALES

Web applications are based on several components (often written in many different languages) and distributed (potentially over the Web). Generally speaking, Web application components are divided into: Web documents (or pages); Web objects (i.e., compiled components that provide services to the rest of application through a defined interface); and server objects and components (database, legacy system, server components). This article focuses on Web documents, categorised as follows: static, active and dynamic.

A *static* document is a simple markup language (HTML) file on a Web server, its content is fixed at creation, and can only be changed by file editing. An *active* document arrives at the Web client usually in a form that must be partially interpreted (before simple rendering). Involved technologies are Javascript, Java Applets, and other scripting languages.

A *dynamic* document is generated by a Web server upon request of Web clients. I.e., the server runs a program to generate the output to be sent to the client. This type of document can be more complex and may involve many types of languages (ASP, PHP, Java server language, CGI, ...).

WebUml is based on the following phases: analysis and information extraction (in this phase WebUml needs server-side write access), model construction, model visualization. The extraction phase is based on source code analyzing engines (e.g., scanners and/or parsers). Web source code analysis is very difficult, because of the mix of languages (HTML, Javascript, VBScript, Java, embedded objects and so on), often poorly defined and implemented. Moreover, lexical and syntactical analysis can also be used to extract *dynamic* information, by means of particular constructs identification (e.g., specific library function call). The difference between static and dynamic information in Web application is a crucial concept. For example, a dynamic page (ASP page), can build a set of client–side pages using run-time information. Static analysis (i.e., traditional source code analysis) of that page cannot reach a good result level, because it does not consider run time elaboration. In fact, server-side generated client pages must still be interpreted or rendered (client–side page may contain scripting code) by the browser, adding one more step of indirection. Dynamic information extraction can be done with static or dynamic analysis, static analysis is often preferred because dynamic analysis is context–based and results are driven by execution cases (potentially infinite), even if it can be more accurate. To be useful, dynamic analysis should try to be exhaustive in defining the application execution paths. The traditional approach would imply

the application of static analysis to every generated client-side page [26]. Dynamic analysis may help to reconstruct the relevant execution paths (i.e., the relevant client–side pages generated by the ASP page) obtaining better results. Examples of traditional approaches used to analyze Web applications are (A) the user sessions recording combined with navigation simulation/replication (e.g., capture and reply techniques [2],[1]) and (B) navigation simulation with user controlled interaction (e.g., [26]). These approaches need many user interactions to define valid analysis levels and to exceed the context–based limitation of dynamic analysis, e.g., a server page building client–side pages based on HTML-form inserted values. The (A) approach forces the user to carry out many navigation sessions in order to raise the validity of the process. The (B) forces the user to analyze pages and the application domain, to choose the range of input values, to classify them and to determine adequate input for application analysis (i.e., input needed to define relevant client–side generated pages). The core of dynamic methodologies consists in sending requests with input values to the Web server and saving the response for later analysis. Our proposal follows the direction shown by [26], but tries to bound and simplify user interactions, lower the analysis computational complexity, increase the accuracy of analysis results and the methodology portability. Our approach is not focused on the input values but on the application, and is essentially based on two steps: dynamic analysis to define the relevant executional paths and static analysis to analyze them. Static analysis is based on a parser that analyzes source code while dynamic analysis is based on mutation analysis and simulation of navigation sessions (i.e., interaction with the Web server hosting the application under analysis). These combined operations can better define relevant execution paths. This technique can be integrated with a set of result validation techniques, such as bad links analysis, error pages detection, pages similarity analysis, and, finally, user validation analysis. Application execution paths are built through the application of mutation techniques combined with random input values, so that no user interaction is needed except at the beginning-end of the analysis.
Web applications can be described in UML with various diagrams: class diagrams for application components; object diagrams for object–components; activity diagrams for control and data-flow structures; state diagrams for component states and state changes; collaboration diagrams for scenarios and interactions among objects; use-case diagrams for application functionalities and interactions with external systems. Moreover, state diagrams can be also used to describe the navigational structure. The WebUml tool presently focuses on the use of UML class diagrams (to describe the type of components composed the application) and state diagrams (to describe behaviors and navigation assets of application).

4. WEB APPLICATION MODELING

WebUml generates class and state diagrams. Class diagrams are used to describe the structure and components of a Web application, e.g., forms, frames, Java applets, HTML input fields, session elements, cookies, scripts, and embedded objects. State diagrams are used to represent the behaviours and the navigational structure according to a model derived from [21]. This navigational structure is composed by client/server pages, navigation links, frames sets, form

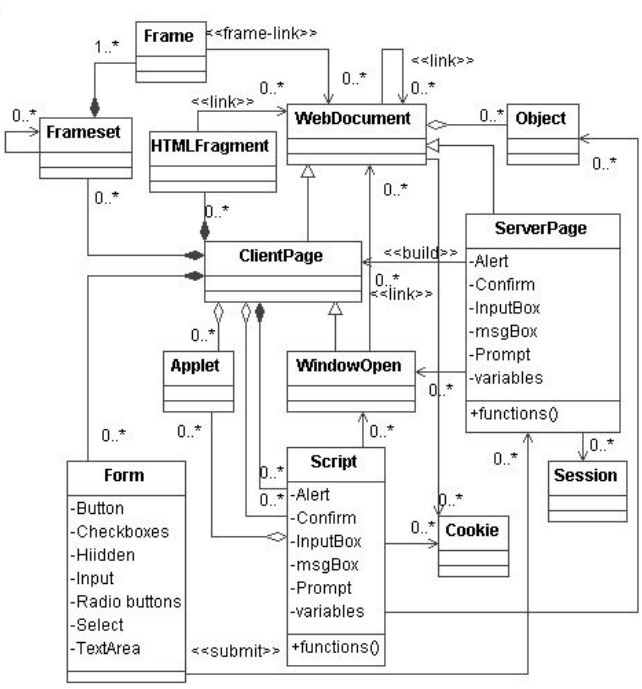

Figure 1: Class diagram meta model

inputs, scripting code flow control, and other static and dynamic contents. The use of state diagrams let us model relevant assets, such as an active document (i.e., composed by HTML and client–side scripting code). In particular, the state diagram of an active document can define the function calls flow of the scripting code, and some relevant behaviours/navigation dynamic information (e.g., dynamic links, frames, and so on).
Fig. 1 shows the meta model (similar to [8], and like [26] model) class diagram used to define a generic Web application structure and components. A WebUml generated model is an instance of the meta model. The core of a Web application is the *WebDocument*, it can be static/active (*ClientPage*) or dynamic (*ServerPage*). A *ClientPage* element representing an HTML client side page may contain some structural elements, such as inputs, buttons, textareas, selections, and so on. Moreover, it can contain object types such as Java applets (*Applet*) or other embedded objects (*Object*) (e.g., ActiveXobject, Microsoft COM object, and so on). The HTML page may contain client side scripting code fragments (*Script*) to support dialogue with the user (such as alerts, inputs boxes, and so on) and functions declarations. The scripting code may use a *Cookie* and may define new windows (*WindowOpen*) with HTML code contents (i.e., client–side pages dynamically built). An HTML page may be inserted in a frame context, or can contain a set of frames grouped in a *Frameset*. An HTML frame (*Frame*) is an area in the HTML client side page where the navigation can take place independently. Moreover, a page can be decomposed into many frames that can interact with each other, and in particular a frame can be used to create menus defining navigation paths in other frames (*frame-link*). A *ServerPage* defines a server side page composed by variables, prompts, alerts, an so on (class attributes),

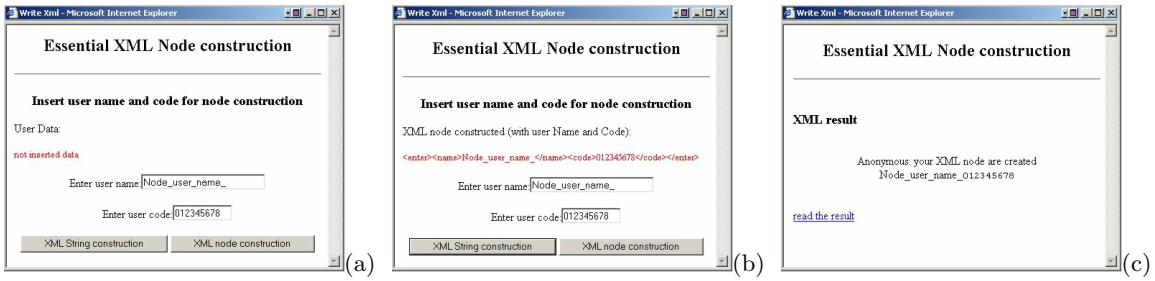

Figure 2: Essential XML node construction application

and function declarations (class methods). A server page can use embedded objects (*Object*) and can define sessions (*Session*). A server page may receive data from (*Form*) elements, it may contain redirection links or may build a client side page based on (*Form*) data. *HTMLFragment* defines HTML code fragments dynamically built by server pages.
In our model, a Web application is associated to a state diagram and Web documents are associated to substates (subdiagrams). A static document is represented by a simple state, while an active document is represented by a composed state that may be concurrent if the page contains client-side scripting code. Dynamic documents are modeled by simple or composed state. If the document does not contain some relevant navigation element, it is described with simple state, with composed state otherwise. E.g., a dynamic page that builds many client side HTML pages is modeled with a composed state with many substates, one for every HTML page generated. In general, the transitions are defined by links, function calls, and various HTML form inputs. An HTML frame set is modeled via composed concurrent state where every frame corresponds to a substate.

5. RUNNING EXAMPLE

In this section we introduce a simple ad-hoc Web application that will be used as an example to show the mechanisms used by WebUml. *Essential XML node construction* (the main screen-shots are in Fig. 2) is a Web application composed by static and dynamic pages. Code fragments are shown in Fig. 3 (a HTML page with client side scripting) and Fig.4 (an ASP page with server side scripting). The application builds XML nodes using two values inserted by the user, see Fig. 2(a). The user inserts two values in the HTML form, one representing user name and one representing a numerical user code. The user may activate one of the two buttons, and the application builds an XML node in one of two possible ways. *XML String construction* button activates the first way, calling a local (client–side) Javascript elaboration that builds an XML string with the user inserted data. Fig. 2(b) shows the page result for this client-side XML construction. *XML Node construction* button activates the second way, it sends the values to a server page that builds an XML node with a Microsoft XML ActiveX object and then visualizes the node building an HTML client page (see Fig. 2(c)). In both cases, the values are checked for coherency, in particular the numeric value. Finally, the application uses a cookie in order to personalize the dialogue with the user.

6. WEBUML

WebUml [23] is a Web applications reverse-engineering

```
<HTML>
 <HEAD><TITLE>Write Xml</TITLE>
 <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
 <!--function WriteCookie(){
 //control the cookies
 //.......}
 function data_error(typeMsg){
  //control the user messages
  //.......}
 function form_data() {
  //.......
  //control if code contains only number
  var cont=document.InsertForm.code.value;
  //.......
  for( count=0; count<len; count++ ){
   if ((cont.charAt(count)<'0')||(cont.charAt(count)>'9'))
   {validity=false;
   //.......}
  }
  if (validity==false) {
   //code not contains only number
   //.......}
  else {
   //code contains only number,
   //build an XML node with string
   //.......
   document.all.UserDataTag.innerText="<enter><name>"
     + document.all.name.value + "</name><code>"
     + document.all.code.value + "</code></enter>";
  //.......}
  }
 function validate() {
  //control the user input insert values
  //.......}
 //--></SCRIPT>
 </HEAD>
 <BODY onload="WriteCookie()">
 <center><H2>XML Node construction</H2></center>
 <HR>
 <center>.....</center>
 <P ID=UserDataView>.....</P></FONT>
 <P ID=UserDataTag>.....</P></FONT>
 <center>
 <FORM NAME="InsertForm" action="aB.asp" method="POST"
    onSubmit="return validate()">
 User name:<INPUT TYPE="Text" NAME="name" Size=25>
 User code:<INPUT TYPE="Text" NAME="code" Size=10>
 <INPUT TYPE="Button" onClick="form_data();"
    VALUE="XML String construction">
 <INPUT TYPE="Submit" VALUE="XML node construction">
 </FORM>
 </center>
 </BODY>
 </HTML>
```

Figure 3: Fragment of aB.html page

prototype tool based on the general architecture described in Sec. 3. Generated diagrams are saved in XMI (XML Metadata Interchange) format to allow their visualization through common UML tools such as ArgoUML, Together, MagicDraw, and so on. The WebUml architecture (Fig. 5) is based on two software modules: the class diagram constructor and the state diagram constructor. They share an information module that contains common information about the Web application under analysis.

6.1 Class Diagram

The WebUml class diagram constructor needs to work on the whole application source code. WebUml transfers the source code locally and, for every linked page, builds a rela-

```
//.......
<%
//.......
//retrieve data of name and code
name=Request.Form("name")
code=Request.Form("code")
//.......
//proceed with the code data control
for count=0 to len(code)-1 step 1
//.......
 ch=Asc(MID(code,cont,1))
 //if code contains not only numbers,
 //set validity to false
 if ch < 48 OR ch > 57 then
   validity=false
   //.......
 end if
next
if validity = true then
 //code contains only numbers, create the XML node
 Set objXML = Server.CreateObject("Microsoft.XMLDOM")
 objXML.loadXML("<enter>"&"<name>" & name &"</name>"&
   "<code>" & code & "</code>" & "</enter>")
 //build a result client page with a link
 Response.Write("<H2>XML Result</H2>")
 //.......
 Response.Write("<a href='result.html'>result</a>")
else
 //code not contains only numbers
 //build client page with a specific error link
 Response.Write("<center>XML Result</center><HR>")
 //.......
 Response.Write("<a href='erroResult.html'>Error</a>")
end if
%>
//.......
```

Figure 4: Fragment of aB.asp page

tive tag–tree (in Fig. 6 tag–tree for client page of *Essential XML node construction* application; see [7]). A tag–tree is a tree-like logic structure extracted from HTML/XML pages, where every node is mapped to a tag. The Document Object Model (DOM) defines a set of API (Application Programming Interfaces) for navigating and manipulating content and structure of XML and HTML documents. A DOM HTML/XML parser analyzes an HTML/XML page and builds a tree mimicking the page structure, this tree represents the basis of WebUml information extraction. Every tag–tree is analyzed in order to find nodes matching elements in a dictionary derived from the Web application meta model (see Sec. 4). For example, the tag <`script`> is a class, the tag <`P`> is not a class, the tag <`Form`> is a class while the tag <`Input`> of a form is an attribute of the class representing the form tag. For the tag–tree in Fig. 6 the nodes filled with clear and dark gray are the matching ones. When the selection is complete, a set of nodes (the ones that identified as classes, attributes and methods) is analyzed again, in order to identify "composite" nodes. A node is defined as "composite" when it contains scripting code. This type of node needs specific analysis to search for pages, links, components and all static and dynamic information. In Fig. 6 the dark gray node is a composite one because it contains Javascript code. After the second analysis, it is possible to use all recognized elements to build the set of candidate classes and elements. Fig. 7 shows an example of candidate classes for our XML application example. A *parser manager* analyzes source code and dispatches code fragments to specialized parsers. These ad hoc written scripting code parsers are based on regular expressions. For static information extraction these parsers perform simple matching searches. Instead, for dynamic information, regular expressions are used not only to search information, but also to describe control-flow structures in application source code. The WebUml class diagram constructor produces: an XMI file with the class diagram; an XML file with

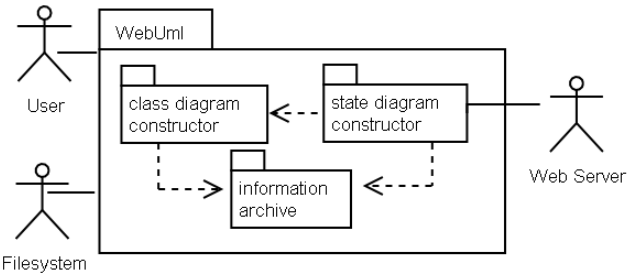

Figure 5: WebUml tool, general architecture

hierarchically organized links; and several structured information used by state diagram constructor. Fig. 8 shows the class diagram generated by WebUml for the *Essential XML node construction* application described in Sec. 5. An initial HTML page (*ClientPage ... /aB.html*) is composed by a script with functions, variables (*Script_1 ... /aB.html*), one cookie object and by a form (*Form_InsertForm ... /aB.html to ... /aB.asp*) with four input fields. The *Form_InsertForm .../aB.html to .../aB.asp* element submits input values to the server page (*ServerPage ... /aB.asp*) that uses a Microsoft object and builds a client side page (*ClientPage ... /aB.asp*). Moreover the client-side page built by the server *ClientPage ... /aB.asp* is composed by a fragment of code dependent on the input values (*HTMLFragment_1/2 ...*). Code fragments contain links to other HTML pages (*ClientPage ... /error.asp* and *ClientPage ... /erroResult.asp*). WebUml recognizes if server side pages generate HTML client side pages, such as in this running example.
Class names are generated according to this pattern:

<Object type>_<*Object ID*>_<*URL trunc.*>

Object type: meta-model belonging class; **Object ID**: (optional) e.g. "name" field value; **URL trunc.**: (optional) the complete URL of object (truncated in the figure for visualization).

6.2 State Diagram

In order to generate this type of diagram a combined use of static and dynamic analysis is performed. WebUml state diagram constructor requires as input: the output data from the class diagram constructor; the application source code; and user settings information. It also needs a live connection with the Web server hosting the application. The first operation consists in class diagram analysis to define application components that have fundamental states for Web application evolution. The next phase examines *components*, to search for the events that change the state of the component itself, such as inputs inserted values, function scripting call, mouse clicks, and so on. The integration of static and dynamic analysis performed by WebUml involves different treatment for server and client pages. WebUml analyzes client side pages by elaborating the information extracted in the class diagram constructor modules, and if necessary (e.g., for active Web pages) applies specific static code analysis in order to define details about the client–side dynamic characteristics (scripting code, dynamic links, and so on). Server pages analysis (Sec. 6.2.1) is a combination of: source code mutation, application executions (WebUml dialogue with Web server) and traditional source code analysis. For example, for a single server page generating multiple client pages, WebUml attempts to determine a mean-

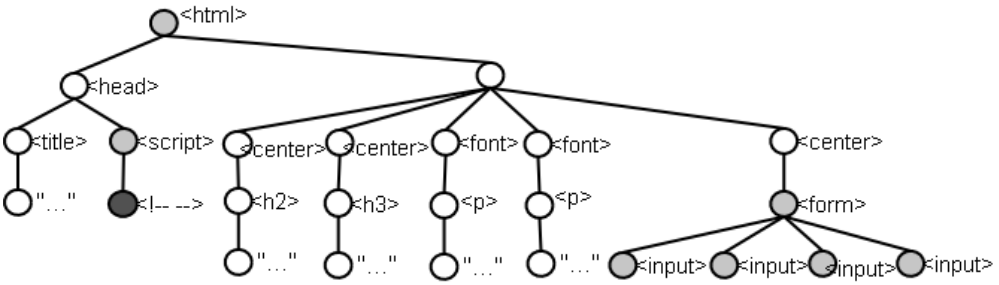

Figure 6: Tag-Tree

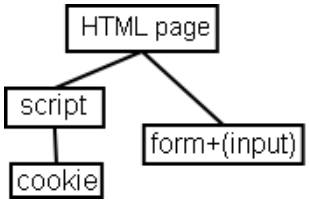

Figure 7: Candidates classes

ingful number of these client pages with mutation analysis and application executions. Then, the dynamically generated client–side pages are analyzed (with traditional source code analysis) and their assets are inserted into the diagram. More generally, for every Web document WebUml uses mutation analysis to define mutants (changing the control–flow structure of original source code page), then the mutants are used into session navigation simulations, in which every mutant replaces the original source code and the simulation performs generated interactions. In the navigation simulation, WebUml sends input values and page requests to the Web server, and saves responses that are analyzed to define elements to complete the model.

The generated state diagram for the *Essential XML node construction* application 5 is shown in Fig. 9. There are two main states, for HTML initial client page (*ClientPage ... /aB.html*) and for the server ASP page (*ServerPage ... /aB.asp*), both composed by substates. The client page is composed by a scripting code and a HTML fragment code represented by an AND–substate, in which the first shows the evolution of the function calls and the second the use of scripting functions. In the state representing the ASP server page is important to notice that the boolean value of the variable *validity* causes this server page to build two client–side pages (*ClientPage_1/2 ... asp*), differing in the set of navigation links. The state diagram is completed by two other simple states showing the presence of two HTML pages (*ClientPage ... /erroResult.html* and *ClientPage ... /error.html*). State names follow one of these patterns:

all:<**Object type**>_<***Object ID***>_<**URL trunc.**>

substates only:[<**Object ID**>]_<**URL trunc.**>

Object type: the meta-model belonging class; **Object ID**: (optional) e.g., the name of a field or function or an automatically generated numeric identifier; **URL trunc.**: the complete URL of object (truncated in the figure for visualization).

6.2.1 Server Page Analysis in detail

This phase combines server–side document analysis with Web server execution. The state diagram constructor module depending of the kinds of documents is able to accomplish four functions: (server–side) source code mutation, session navigation simulations, analysis and documents validation. If it finds a *static* document it uses exclusively the data extracted in the previous phase from the class diagram constructor module, such as links, frames, inputs forms, and so on. If it finds an *active* document it performs a custom analysis by activating an ad hoc parser to search client–side dynamic information even inside scripts

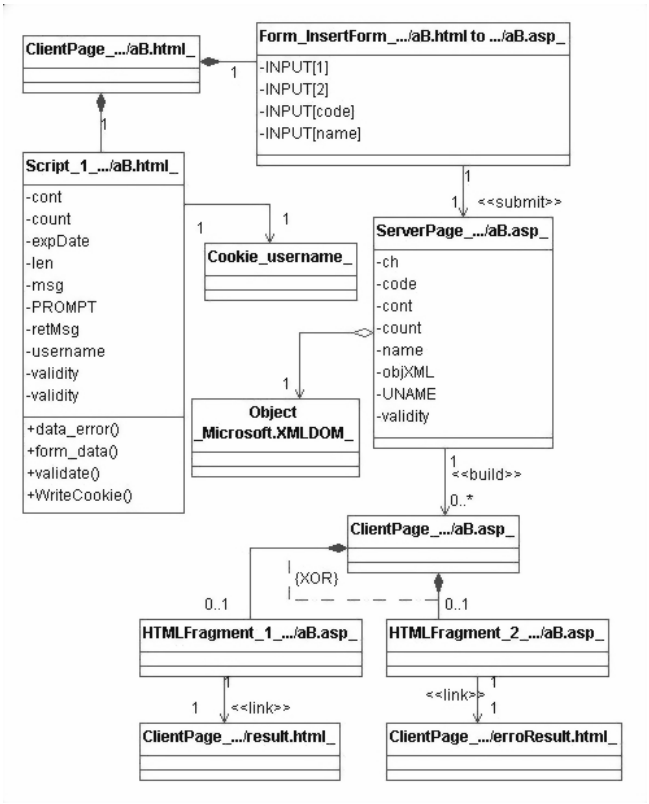

Figure 8: XML application class diagram

(i.e., in particular navigational information). Finally, if it finds a *dynamic* document, it activates a parser performing server–side mutation–based dynamic analysis with navigation simulation. The iterative *dynamic* analysis phase can be subdivided into:

Initial settings: WebUml sets navigation environment and conditions. Analyzing (server–side) code and using information taken from the class diagram constructor, it identifies input fields and generates random tuples for input values.

Mutation + Navigation Simulation: WebUml generates a set of source code mutants. For every mutant it performs navigation simulations, i.e., it saves the new (mutated) page on the server filesystem and requests it (using HTTP) from the web server. More than one single request is performed on a mutated page: random input values are generated and used in the HTTP requests. Results are saved for later analysis.

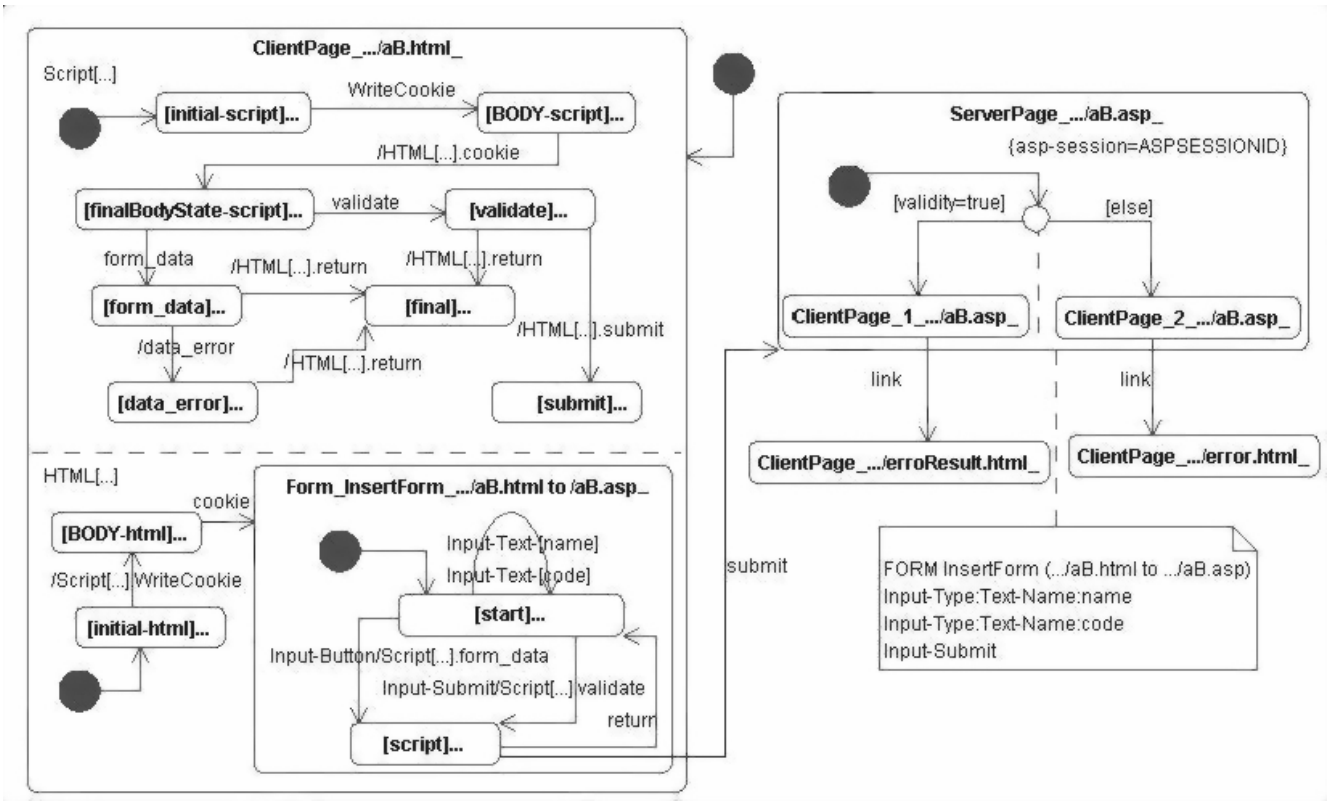

Figure 9: XML simple application UML model

Classification: WebUml analyzes server responses trying to eliminate redundant (client–side) pages. When two pages are similar they are collapsed into one, i.e., one of them is discarded and the other is used as representative of the two. A simple (reduction of [15] approach) definition of similarity was used here: two pages are considered similar if they contain the same set of elements, i.e., mainly the same set of links, are considered similar. This is consistent with the use of the state diagram as navigational abstraction of the application

Validation: WebUml reduces the set of generated (post mutation - client–side) pages with a simple analysis, such as HTML checker (such as HTML Tidy by W3C), bad link and error pages analysis and then asks the user to mark the remaining pages as "valid" or "not–valid". A client page should be considered "valid" by the user if reachable in the original application (without mutants) in an execution path.

Final analysis and Diagram update: then, for every "valid" dynamically generated (client–side) page, WebUml extracts information with a source code analysis to update the application state diagram.

6.2.2 Mutation phase

WebUml allows the definition of mutant operators via an XML configuration file. WebUml mutant operators are applied to control–flow source code fragments (e.g., "if-then-else", "while", etc.). In particular, WebUml mutant operators are defined on language operators, i.e., logic or boolean operators, conditions or check operators, and so on. For example, the "=" operator can be mutated into "<>", the ">" operator can be mutated into "≥" or "<", the "AND" operator can be mutated into "OR", etc.

The aim of mutation is to automatically follow relevant execution paths in the Web application. It is important to follow these paths to cover as many navigation paths as possible. By changing the application control–flow structure and by using the generated mutants in navigation simulation sessions it is possible to automatically cross the application execution paths. A Web application can be thought as a function: fw: (S, V) → (C)

Where: **S** is a finite set of server–side pages (ASP, JSP, etc. but also static HTML pages); **V** is a infinite (Web page parameters are unbounded character strings) set of input values for pages (where applicable); **C** is a (potentially infinite, depends on fw) set of client–side pages.

For example, a static site has $|S| = |C|$ and an empty V.

A Web application analysis tool should examine as many pages ∈ C as possible. To perform such an operation, the C set should be generated. But such exhaustive generation, except in very simple and limited cases is impossible, e.g., when C has ∞ cardinality. In this cases, a subset of C is usually analysed [26]. The construction of the C subset is often done by generating representative values ∈ V. This approach chooses the representative values by examining pages ∈ S and searching for expressions in control flow fragments. For example, in a "if(a>5)" fragment, the randomly chosen representative values may be 1, 5, and 8. A great disadvantage of this approach is that it requires deep knowledge on the generating language (VBScript, Java, etc.).

Our approach uses a random set of values ∈ V, and applies them to a superset S_1 of S, where S_1 is the set of mutated S pages. This approach does not need knowledge about the

language, only a simple map of mutant operators, deployable with easy to program parsers and with a low computational complexity. For example, a "if(a>5)" fragment could be mutated page in "if(a<5)" or "if(a≥5)". This way, with a random value and many generating pages we can cover the representative elements of C.

7. CONCLUSIONS

This paper described the architecture of WebUml, a Web applications reverse-engineering tool used to extract UML models (in particular class and state diagrams) with minimal user interactions. WebUml uses a mix of techniques based on source code static and dynamic analysis. Static analysis is performed through simple parsers based on a pattern matching scanner. Dynamic analysis is performed through source code mutational techniques combined with simulated Web application execution. This technique avoids heavy language analysis, but needs only the implementation of a simple map of mutant operators. The generated UML models may seem *crowded*, but they are also very rich of information, therefore particularly well-suited for Web application testing (structural and behavioral), i.e., the object of WebUml sequel: TestUml, an automatic structural test tool for Web applications that uses WebUml-generated UML models as input and that is currently under development.

8. REFERENCES

[1] Bitmechanic. *http://www.bitmechanic.com*.

[2] Mercury interactive. *http://www.merc–int.com*.

[3] Rational Rose Web Modeler. *http://www.rational.com*.

[4] L. Baresi, F. Garzotto, and P. Paolini. Extending UML for Web Applications. *34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii. January 2001.

[5] B. Beizer. *Software Testing Techniques, 2nd edition*. International Thomson Computer Press, 1990.

[6] R. Binder. *Testing Object-Oriented Systems*. Addison-Wesley, 1999.

[7] D. Buttler, L. Liu, and C. Pu. A Fully Automated Object Oriented Extraction for World Wide Web. *20th International Conference on Distributed Computing Systems (ICDCS'01)*, Phoenix, Arizona. May 2001.

[8] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2000.

[9] O. M. F. De Troyer and C. J. Leune. WSDM: A User Centered Design Method for Web Sites. *Computer Networks and ISDN systems, 7th International WWW Conference*, Elsevier. 1998.

[10] G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, and U. De Carlini. WARE: A Tool for the Reverse Engineering of Web Applications. *6th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, Budapest, Hungary. March 2002.

[11] G. A. Di Lucca, A.R. Fasolino, F. Faralli, and U. De Carlini. Testing web applications. *International Conference on Software Maintenance (ICSM'02)*, Montreal, Canada. October 2002.

[12] M. Fowler and K. Scott. *UML Distilled: Applying the Standard Object Modeling Technique*. Addison-Wesley, 1997.

[13] M. A. Friedman and J. M. Voas. *Software Assessment: Reliability, Safety, Testability*. John Wiley & Sons, 1995.

[14] F. Garzotto, P. Paolini, and D. Schwabe. HDM: A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems*, January 1993.

[15] T.H. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating Strategies for Similarity Search on the Web Similarity search. *WWW 2002*.

[16] T. Isakowitz, E. A. Stohr, and P. Balasubranian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, August 1995.

[17] N. Koch and L. Mandel. Using UML to Design Hypermedia Applications. *Technical report, Institut für Informatik, Ludwig-Maximilians-Universität München*, March 1999.

[18] D. C. Kung, P. Hsia, and J. Gao. *Testing Object-Oriented Software*. Wiley-IEEE Press, 1998.

[19] D. C. Kung, C. H. Liu, and P. Hsia. Object Based Data Flow Testing of Web Applications. *The First Asia-Pacific Conference on Quality Software (APAQS'00)*, Hong Kong, China. October 2000.

[20] D. C. Kung, C. H. Liu, and P. Hsia. An Object Oriented Web Test Model for Testing Web Applications. *24th International Computer Software and Applications Conference (COMPSAC 2000)*, Taipei, Taiwan. October 2000.

[21] K. R. P. H. Leung, L. C. K. Hui, S.M. Yiu, and R. W. M. Tang. Modelling Web Navigation by Statechart. *24th International Computer Software and Applications Conference (COMPSAC 2000)*, Taipei, Taiwan. October 2000.

[22] B. Lieberman. UML Activity Diagrams: Detailing User Interface Navigation. *http://www.therationaledge.com/content/oct_01/toc.html*.

[23] A. Marchetto. Testing di Applicazioni Web. *Laurea Degree Thesis. Università degli Studi di Milano*, October 2002 (in italian).

[24] H. A. Muller and H. Kienle. Leveraging Program Analysis for Web Site Reverse Engineering. *3rd International Workshop on Web Site Evolution (WSE 2001)*, Florence, Italy. November 2001.

[25] F. Ricca and P. Tonella. Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'200)*, Genova, Italy. April 2001.

[26] F. Ricca and P. Tonella. Dynamic Model Extraction and Statistical Analysis of Web Applications. *4th International Workshop on Web Site Evolution (WSE 2002)*, Montreal, Canada. October 2002.

[27] F. Ricca and P. Tonella. Analysis and Testing of Web Applications. *23th International Conference on Software Engineering (ICSE'2001)*, Toronto, Canada. May 2001.

[28] D. Schwabe, R.A. Pontes, and I. Moura. OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW. *SigWEB Newsletter*, 8, June 1999.