International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:4, 2007

Validation of Reverse Engineered Web Application Models

Carlo Bellettini, Alessandro Marchetto, and Andrea Trentini

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). The scientific community has focused attention to Web application design, development, analysis, testing, by studying and proposing methodologies and tools.

Static and dynamic techniques may be used to analyze existing Web applications. The use of traditional static source code analysis may be very difficult, for the presence of dynamically generated code, and for the multi-language nature of the Web. Dynamic analysis may be useful, but it has an intrinsic limitation, the low number of program executions used to extract information. Our reverse engineering analysis, used into our WAAT (Web Applications Analysis and Testing) project, applies mutational techniques in order to exploit server side execution engines to accomplish part of the dynamic analysis.

This paper studies the effects of mutation source code analysis applied to Web software to build application models. Mutation-based generated models may contain more information then necessary, so we need a pruning mechanism.

Keywords**—**Validation, Dynamic Analysis, Mutation Analysis, Reverse Engineering, Web Applications

I. Introduction

WEB applications quality, reliability and functionality are important factors because software glitches could block entire businesses and determine strong embarrassments. These factors have increased the need for methodologies, tools and models to improve Web applications (design, analysis, testing, and so on).

Important factors for Web applications are "speed" (in technology change, content update and fruition), complexity, large dimensions and design/use maturity. Web applications are heterogeneous, distributed, and concurrent: their analysis, understanding, reengineering, and testing are not easy task. Conventional methodologies and tools may not be adequate.

This paper focuses on analysis of legacy Web applications where business logic is embedded into Web pages. Analyzed applications are composed by Web documents (static, active or dynamic) and Web objects. In particular we focus in server side components that dynamically generate Web documents in response to some user inputs and gestures. The structure of these documents is not given *a priori*, but is dynamically constructed based on user interactions.

Web software is often developed without a formalized process, and Web documents and objects are directly coded in incremental way. Often, new documents and objects are obtained by duplicating ("copy & paste inheritance") and modifying existing ones. Web software life-cycle is very compressed, in the range of tree to six months. Techniques for application analysis, understanding, reusing and testing are badly needed.

The analysis of existing Web applications through traditional static source code examination of "highly" dynamic applications is a very difficult task. The most complex problem is to define the structure of the Web documents produced by the server-side component. This problem is traditionally known as an undecidable problem, because it is related to the program execution paths analysis, and in particular to determine if a given execution path is feasible.

Moreover, Web application may be distributed, of very large dimension, and composed by various programming languages. For example, on the client-side, every document may be a mix of HTML, Java, Javascript. While in server-side some components, written in other languages (Java, PHP, ASP.NET, Perl, SQL, XML etc.), may interact to build Web documents. This languages mix is the power of the Web, but it also contributes to make application analysis a difficult task.

Performing static analysis may be difficult and not much effective. Dynamic analysis may be used to integrate static analysis to simplify it, to increase analysis effectiveness, and it may be more "Web-adequate", because it let us perform analysis with some degree of language abstraction.

Dynamic analysis can be: profiling, debugging, user-gestures capture and reply, log files analysis, event-trace recording, and so on. Some of the existing dynamic techniques may define a partial application model. Information is extracted through a limited number of program executions. Dynamic analysis may produce an application model without covering all relevant application behaviors. This is an intrinsic limitation of all types of dynamic analysis.

Static analysis is often preferred, because dynamic analysis is context–based and results are driven by execution cases (potentially infinite). To be useful, dynamic analysis should try to be exhaustive in defining application execution paths.

In this paper we present the validation process for UML

Manuscript received January 27, 2005.

Carlo Bellettini, Alessandro Marchetto, and Andrea Trentini are with Information and Comunication Department, University of Milan. Via Comelico 39, 20135 Milan, Italy.

{Carlo.Bellettini, Alessandro.Marchetto, Andrea.Trentini}@unimi.it

models generated by our WAAT system via our reverse engineering technique, which is a mix of static and dynamic techniques. Our approach differs from traditional dynamic ones because it is not focused on user gestures replication but on application evolution. The approach is essentially based on two steps: dynamic analysis to define relevant execution paths and static analysis to analyze them. Static analysis is based on a scanner/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. Mutation analysis tries to bound and simplify user interactions, to lower analysis computational complexity, to increase the accuracy of analysis results and analysis methodology portability. Our proposed 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 mutation techniques combined with random input values, so that no user interaction is needed.

This paper focuses on validation of WAAT generated models. This phase is very important because the use of source code mutation may solve the intrinsic limitations of traditional dynamic analysis, but it may define a super-set of behaviors, that must be pruned. In the WAAT project we propose two alternative ways for models validation. The first one is integrated in the applications testing phase. We present here the second one: a pruning technique based on log files and partial user intervention.

This paper is organized as follows. Section II introduces a review of existing works in Web applications modeling, reverse engineering and testing. Section III details reverse engineering used techniques and implications. Section IV defines proposed validation model algorithm. Section V defines a simple case study of Web application validation-model analysis. Finally Section VI concludes the presented work and describes future work ways.

II. Related Works

Several Web modeling methodologies are available in literature. RMM [8] is a method based on Entity-Relationship diagrams. WebML [2] enables the description of a Web site under distinct orthogonal dimensions. [14] introduces a Web application simulation model framework. OOHDM [18] for OO application modeling. Moreover, some of these are UML based. WARE [4] and Rational Rose Web Modeler [15] are tools for reverse engineering supporting Conallen's extensions [3]. Our WebUml [1] is tool to reverse engineering Web application through dynamic analysis. Taxonomy of reverse engineering approaches is described in [12] and [6]. Reverse engineering road-map is in [11] and [9]. Static and dynamic software analysis techniques are described in [20] and [19]. [20] discusses dynamic reverse engineering techniques for Java software. [19] gives an overview of currently used dynamic analysis. Ricca and Tonella have developed a semi-automatic tool named ReWeb [16], 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. Other approaches are statistically based, such as [17] uses reverse engineering techniques to extract UML models, then it uses log files to create a usage model to be analyzed with Markov Models. [10] defines statistic testing, it creates an application model from log files and then analyzes it with Unified Markov Models. Finally [13] describes a specific Web model based on control flow statement.

III. Model Recovery

The WAAT analysis core is composed by: application **behavior analysis** and application **model building**.

Application **behavior analysis** [1] is performed through static and dynamic analysis. Static and dynamic analyses treat static and dynamic application components using source code and on-line interactions with the Web server. For example, for static pages, we use traditional source code analysis based on a language parser. While, for a single server page generating multiple client pages, we apply dynamic analysis to try to determine a meaningful number of client pages (through mutation analysis and application executions). Then, the dynamically generated client–side pages are analyzed (with traditional source code analysis) to build diagrams. More generally, for every dynamic Web document, we use mutation analysis to define mutants (for example changing the control–flow structure of original source code page) to be fed into session navigation simulations, in which every mutant replaces the original source code and the simulation performs generated interactions. This simulation is used to send input values and page requests to the Web server, and saving responses that are analyzed later.

Mutation [5] analysis is based on mutant operators applied to source code, and in particular to control–flow source code fragments (e.g., "if-then else", "while", etc.), such as 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, to cover as many navigation paths as possible.

This approach does not need knowledge about the language, only a simple map of mutant operators, deployable with easy to program parsers and with low computational complexity.

Model building; with the information extracted by the previous phase we build an application OO model (such as described in [16], [3], [4]) using UML class and state diagrams. We have defined a UML meta-model usable to describe applications [1]. Class diagrams are used to describe structure and components of a Web application (e.g., forms, frames, Java applets, input fields, cookies, scripts, and so on), while state diagrams are used to represent behavior and navigational structures (client-server pages, navigation links, frames sets, inputs, scripting code flow control, and so on).

IV. Model Validation

The “mutation” generated model may contain more information than what is needed. In particular, it may contain “*Not-Valid*” information, such as not valid dynamically-generated client-side pages. A client-page is “*Valid*” if it is reachable in the original application (without mutants) via an execution path. Since mutation may define a model with a super-set of behaviors we need a pruning technique. Our validation technique on follows these steps:

Model analysis: we extract a list of dynamic server-pages and their related client-side dynamically-generated pages

First-reduction: we prune the list by applying content validation analysis tools (such as HTML Tidy by W3C)

Page analysis, subdivided in:

- we analyze client-side dynamically generated pages to extract information, such as: inputs (e.g., GET request parameters sending to call pages); structure, which may be all pages source code (with text), or only the source code structural properties related (such as: form tags, script information, links, and so on). These information are used in the next steps to define page similarity, so a minimal set of information may be composed by navigational system information [7]
- we compute unique hash function with structural extracted information
- finally, we associate a tuple of dynamic-specifics (DS) to every dynamic page. DS={<input page>, <inputs parameters>, <output pages>, <output hash related> }

Second-reduction: pruning using **page analysis** phase results [7]

“**Test cases**” generation; we compute a set of “test cases” with sub-steps:

- for every dynamic page we define specifics such as: TcT={<inputs parameters>, <output pages hash>}
- for every page we define: TcS={<input pages>, <output pages>}

Log files analysis; sub-steps are:

- for every dynamic page we extract requests (URL and related parameter inputs and values)
- for every request in log file we repeat navigation, saving Web response, and re-apply page analysis
- we reduce the set of output pages through structural similarity analysis
- then we fill a table of Tlog tuples: {<dynamic page>, <inputs parameters>, <generated pages>, <output hash related>}

Log based validation sub-steps are:

- we map every TcT with Tlog, by URLs matching. A match between Tct and Tlof validates the generated client-side page. The set of non-matched pages is labeled “Not-Verified”

Visual navigation; every “Not-Verified” TcT must now be validated with user intervention. We mark “Valid” a path, if the page is reachable in the original application (without mutants). To define page-related path, we use the TcS table.

Model update; finally, we update the model by deleting the remaining "Not Valid" pages

V. Validation Sample

MiniLogin is a simple Web application we used as validation approach sample. This application is composed by PHP and HTML files. We generated UML models and now we apply our validation technique.

Now we compute DS specifics. For example, the dynamic-specifics for “member.php” dynamic page are: DS={<input page>, <inputs parameters>, <output pages>, <output hash related>}, where:

-*inputs pages*: index.html (via form);

-*inputs parameters*: $login & $pwd (that are: username & password input fields in index.html form);

-*output pages*: defines list of currently dynamic page client-side generated pages (Table I-column “output pages”)

-*output related hash key*: defines a list of hashes related to output pages (Table I-column “Hash key”). Hashes are computed on the output of page structure information extraction analysis (“pages analysis” in previously section).

From this DS table we compute TcT and TcS tables, defining couples of “<inputs parameters>, <output related hash key>” (e.g., TcT row may be: $login,$pwd & hash(u6.htm)) and defining couples of “<inputs pages>, <output pages>” (e.g.,TcS row may be: index.html & u6.htm).

TABLE I
DS –*member.php* page

output pages	Hash key
u1.htm	f9c895c7ad2921648ef7d23a8c80ca1bc4d659e1
u2.htm	3b863c3352ea817a51ea5aa12ad7b57eae06fa18
u3.htm	1e193c2e43e9ecc41cf99ed03626379d0a89b5ec
u4.htm	bb5bbcc6589ecd340196c8868e0c14643c8125ae
u5.htm	d5651edc1e9957b20e3e6b4c2f2730e47fdc8f14
u6.htm	c393fb5807177597fc43e184cf2c9d8df2266c0c
u7.htm	3ea74fa0635f49128c95a0fc8624ef55efe2cec1
u8.htm	e64953d5ad93a42db1a115678183aa3da7257dbe
u9.htm	993918b27c87f95124d02f3da7e9a7f7daef3af3

When specifics are computed, we analyze Web server log files to build the Tlog table. For example, for the “member.php” page we take all Web server requests, replaying navigation and saving the response. In the MiniLogin case we used Apache Web server log- files, so we are able to analyze the GET requests (composed by: URL, parameters and parameter values), while other requests (e.g., POST) can not be used through log-validation.

Then we analyze every saved server response to extract structural information and compute hashes on the defined information. Table II shows a fragment of Tlog for MiniLogin (inputs detected are the same as in DS).

Then we map Tlog and TcT entries to search pages with hash matches. In our case, we find the maps:

TcT(u6.htm)=Tlog(g16.htm);
TcT(u1.htm)=Tlog(g1.htm);
TcT(u2.htm)=Tlog(g5.htm).

Now we may mark {u6, u1, u2}.htm as “Verified”, but also other uX.htm pages as “Not-Verified”.

TABLE II
TLOG, fragment *–member.php* page

output pages	Hash key
g1.htm	`f9c895c7ad2921648ef7d23a8c80ca1bc4d659e1`
g5.htm	`3b863c3352ea817a51ea5aa12ad7b57eae06fa18`
g16.htm	`c393fb5807177597fc43e184cf2c9d8df2266c0c`
….	….

Then we may perform a user based validation asking the user (application developer) to mark “Not-Verified” pages as “Valid” or “Not-Valid”. For every dynamic page we define a set of paths in the form of (TcS derived) :

<input pages>→<dynamic page>→<output pages>

In our case one of the paths can be the following:

“ index.html → member.php ($login,$pwd) → u3.htm“

After path definition, we ask the user to identify valid paths. When every output page is marked, we may update our model by deleting “Not-Valid” pages.

The process complexity may be summarized by the following: two HTTP-requests for every mutation-generated client-side page, (in particular one GET for every output-page defined in DS and Tlog tables); one hash values computed for every received HTTP-response; and the matching search between HTTP-responses from DS and Tlog.

VI. Conclusion

We propose a validation process for our technique for reverse engineering of Web applications. This process reduces the super-set of information extracted with mutation analysis.

The combination of mutation analysis and validation process represents a dynamic reverse engineering technique that bounds and simplifies user interactions.

“Bounds” because it reduces the number of pages that must be examined by a user. The super-set of pages automatically generated by the mutation analysis technique is reduced through cross linking with log files.

“Simplifies” because the user does not need any knowledge about the application language, he only need to choose between reachable and not-reachable paths.

We are currently working on a statistical comparison between our technique and other approaches.

References

[1] C. Bellettini, A. Marchetto, and A. Trentini. *WebUml: Reverse Engineering of Web Applications*. 19th ACM Symposium on Applied Computing (SAC 2004), Nicosia, Cyprus. March 2004.

[2] S. Ceri, P. Fraternali, and A. Bongio. *Web Modeling Language (WebML): a modeling language for designing Web sites*. WWW9, Amsterdam, Netherlands. May, 2000.

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

[4] 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 CSMR 2002, Budapest, Hungary. March 2002.

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

[6] V.C. Garcia, D. Lucrédio, A.F. do Prado, A.Alvaro and E.S. de Almeida. *Towards an effective approach for Reverse Engineering*. 11th IEEE Working Conference on Reverse Engineering (WCRE’04) 2004

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

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

[9] D. Jackson and M.Rinard. *Software Analysis: A Roadmap*. ICSE (Future of Software Engineering Track), ACM Press, 2000.

[10] C. Kallepalli and J. Tian. *Measuring and Modeling Usage and Reliability for Statistical Web Testing*. IEEE Transactions on Software Engineering, November 2001.

[11] H.A. Müller, J.H. Jahnke, D.B. Smith, M.-A. Storey, S.R. Tilley, K. Wong. *Reverse Engineering: A Roadmap*. A. Finkelstein (ed.) “The Future of Software Engineering”. New York. ACM Press. 2000

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

[13] J. Offutt, Y. Wu, and X. Du. Modeling *and Testing of Dynamic Aspects of Web Applications*. Submitted for journal publication, January 2004.

[14] P. Peixoto, K. Fung, and D. Lowe. *A Framework for the Simulation of Web Applications*. Fourth International Conference on Web Engineering (ICWE 2004), München, Germany. July 2004.

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

[16] 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.

[17] 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.

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

[19] E. Stroulia and T.Systä. *Dynamic Analysis For Reverse Engineering and Program Understanding*. Applied Computing Review, ACM 2002

[20] T. Systä. *Understanding the Behavior of Java Program.* 7th Working Conference on Reverse Engineering (WCRE 2000), Brisbane, Australia, November 2000.