A THREE POLED MODEL FOR WEB APPLICATIONS

Hamzeh K. Al Shaar and Ramzi A. Haraty
Division of Computer Science and Mathematics
Lebanese American University
P.O. Box 13-5053 Chouran
Beirut, Lebanon 1102-2801
Email: h_alshaar@yahoo.com, rharaty@lau.edu.lb

Abstract

Web applications are nowadays at the heart of the business world. All corporate companies and big institutions have very busy e-commerce web sites that host a major part of their businesses. With this great emergence of web applications, techniques for maintaining their high quality attributes should be developed and exercised. Moreover, the quick evolution of web technology and the high user demands made web applications subject to rapid maintenance and change, which require the development of efficient regression testing techniques. The current testing efforts documented in research deal with a specific part of a web application. While some papers model and test the server side programs of the application, others model and analyze the navigation between pages as seen by the user and yet others deal with analyzing the architectural environment of the web application. Motivated by the fact that there is no single model to represent the entire web application, and to model it from different perspectives at the same time, we propose a single analysis model which models the three poles of the web application: the client side pages navigated by the user, the server side programs executed at runtime, and the architectural environment hosting the application.

KEY WORDS: Three poled model for web applications and web applications modeling.

1. INTRODUCTION

As the web is growing and invading our world, and as using the Internet became a normal habit to the new generation, web applications started to take a major part in the software industry and began to invade the business market playing an important role in facilitating the business flow of most companies.

With this emerging importance of web applications in the commercial sector, and with the new and challenging quality requirements of web software, techniques to test and to control their quality attributes became a must. However, developing such testing techniques for web applications is much more complicated than that of classical software and this is due to the nature of the web applications.

Unfortunately there is no well-developed and mature model to analyze and test web applications yet. All the previous work dealt with certain aspects of the web applications while neglecting the rest. Even worse, very few testing and regression testing techniques have been exploited to be used by web applications.

Motivated by the fact that there is no single model to represent the entire web application, and to model it from different perspectives at the same time, we propose a single analysis model which models the three poles of the web application: the client side pages navigated by the user, the server side programs executed at runtime, and the architectural environment hosting the application. Based on this model, we also propose testing and regression testing techniques for each of the three parts of our model. The rest of the paper is organized as follows: section 2 presents the literature review. Section 3 presents our three poled model. And section 4 concludes the paper.

2. LITERATURE REVIEW

In this section, we present at some of the previous work done in the field of web application modeling and testing and in the field of regression testing.

Wu and Offutt [14] presented an analyses model for modeling the server side components and the client server interactions. Wu et al furthered their work by extending their model to cover inter-component connections between different server side components [15]. The new model also covers the current state representation of a web application and the current state of variables. Ricca and Tonnela [1] modeled a web application as a sequence of web pages and they modeled the interaction between them in a UML diagram. Ricca and Tonnela also tackled web application slicing [2]. In their work, Ricc and Tonnela extended the concept of application slicing and applied it to web applications.

Sebatien, Karre, and Rotherm [12] presented a technique for automatically testing a web application based on the user session data collected from the user's navigation of the website. A similar work was proposed by Wen [11]. In his paper, he proposes a technique for generating test cases that

are based on URLs to be automatically exercised. Sneed [4] wrote a paper that focuses on the web application architecture and which recommends that this architectural environment be tested independently of the web application itself.

There are numerous papers that discussed regression testing of classical software [5][6][10][13]. These papers include work done by Xu, Yan, and Li [7], and the work done by Granja and Jino [8] and the work by Xu, Chen, and Yang [9] which also discusses regression testing of web applications using slicing.

3. THE MODEL

Our work models a web application from three different perspectives: The architectural environment, the client side navigation and flow of execution, and the server side programs and their dynamic output. Each of those parts has its own sub model and its own testing techniques.

3.1 The Architectural Environment Model

In this section, we propose a graphical analytical model to describe the architectural environment and also propose a set of testing and analysis techniques based on this model. Testing of this model is more oriented towards analysis and evaluation of the environment rather than checking for correctness, since it is possible to have different architectures for one application but each with certain advantages and disadvantages.

Modeling the architectural environment of the web application includes representing:

- All physical servers,
- All software installed on physical servers (e.g., IIS, Apache, SQL Server, etc.),
- The communication protocols between connected nodes of servers,
- The type of messages exchanged, and
- The clustering and redundancy of servers.

The model represents the architectural layer in a diagram similar to a UML diagram. The diagram represents nodes of physical servers as rectangles named with the computer name of the server. We represent the servers by a set of triplets S, where each triplet contains the server name, the operating system installed on it, and the processor type of the machine (e.g., Intelx86, SPARC, etc.). The rectangular box includes one or multiple squares, with each square representing the software server installed on the machine such as IIS, Oracle Database Server or IBM Websphere.

The set of software servers are represented by a set SV of triplet where each triplet contains the software server name (we name it for ease of reference), the software server installed, and the physical server name.

If two software servers are clustered for redundancy then we represent this as two parallel dashed lines connecting the two boxes (not arrows). If the two software servers are load balanced, then the two boxes are connected with two parallel non-dashed lines (not arrows). Normally if two servers are load balanced then they are automatically clustered for redundancy, so we present them as load balanced servers only and not as both.

Clusters are represented as a set C of pairs where each pair contains the names of the software servers being clustered. In case we have more than two servers in the cluster, then we have the first and the second in the first pair, the second and the third in the second pair and so on. In other words those pairs are transitive. Similarly, we will have a set LB of pairs containing the load balanced servers.

The communication between servers is represented in the diagram by arrows between software servers (squares). If a server sends data to another server then we have an arrow from the first server to the second. If the second server returns data then we will have another arrow in the opposite direction.

The communication links between the software servers are presented as a set of quadruples Cmi where each quadruple contains the source software server name (member of set SV), the destination software server (as defined by set SV), the underlying protocol being used (such as FTP, HTTP, or SSL), and the type of messages sent from the source server to the destination servers. If the source or destination servers are members of a grid or a cluster we name any of the cluster members instead. The type of exchanged messages is predefined and can be one of the following:

- http_rq (http request)
- http_rs (http response)
- db_q (database query)
- db_rs (database result set)
- f (file transfer)
- xml (XML file or XML messages)
- SL (packets sent over a direct socket layer opened from the application)

Finally, the software libraries and application extensions are represented as set LR of pairs where each pair has the server name (as represented in S) and the name of the communication library, driver, or extension installed.

Example

Let us take an example where we have two web servers running Microsoft IIS on Windows Server 2003 and those two servers are clustered for redundancy. Moreover, the servers run an ASP application that reads and writes data from an Oracle database. The Oracle database is running on a grid of three windows servers load balanced and redundant. Moreover, let us assume that the ASP application processes the data read from the application and then formats it in a certain file and sends it to an FTP server. The FTP server is remote (not on the LAN) and it is a normal non- clustered server. For sure we have on both web servers the Oracle Drivers, and we have on each of the web servers a library which supports FTP commands and integrated within IIS. The diagram of the architecture will look as follows (Fig. 1):

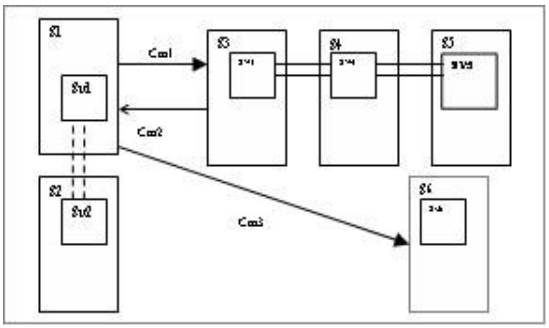

Fig. 1 - Example of an Architectural Environment Model

S= {(S1; Windows 2003, intel_x86), (S2; Windows 2003, intel_x86), (S3; Windows 2003, intel_x86), (S4; Windows 2003, intel_x86), (S5; Windows 2003, intel_x86), (S6; Windows 2003, intel_x86)}
SV= {(sv1; IIS; S1), (sv2; IIS; S2), (sv3; Oracle Db; S3), (sv4; Oracle Db; S4), (sv5; Oracle Db; S5), (sv6; MS FTP; S6)}
C= {(Sv1; Sv2)}
LB= {(Sv3; Sv4), (Sv4; Sv5)}
Cm1= {Sv1, Sv3, TCP/IP, db_q}
Cm2= {Sv3, Sv1, TCP/IP, db_rs}
Cm3= {Sv1, Sv6, FTP, file}
L1= {(S1; Oracle_driver), (S2; Oracle_driver), (S1; FTP_Library), (S2; FTP_Library), (S6; FTP_Listener).

For more details regarding the figure, refer to [3].

3.2 The Client Side Navigational Model

Client side modeling models the web application from the client perspective or as the application is viewed from the client browser. For a normal web surfer browsing the web application at a client browser, the web application consists of a set of web pages residing at the web server and they are navigated by loading them into the browser one after the other by a certain sequence. This sequence is decided by the logic of the web application and is done via HTML hyperlinks.

To model the application from a client side, we have to model what this web user sees in the client browser. Even when considering dynamic pages and server side scripts, we only deal with their HTML output as seen by the client regardless of the other logic running at the server. From a user's point of view, s/he is reading HTML pages, and interacting with HTML controls, mainly links and forms.

As with all three parts of our model, we use graphs in order to build our analysis model for this part. This model presents the web application in a graph similar to UML.

For the sake of simplicity and in order to make our web application similar to standard graphs, we will assume that the web application starts at one start page and ends in one end page. If there are more than one start page, we can create one start page with branches to each one of them. Similarly, if we have different exit pages, we can link them all to one exit page.

Web pages have different types and behaviors, and since we cannot model each and every case, it is important to differentiate between different types, which cover most cases of HTML pages:

Static HTML pages: we chose to model those pages as squares where each square is tagged by a pair of the page name and the server name where this page resides.

Dynamic HTML pages are those pages that are generated by a server side script such as CGI, JSP, or ASP. Dynamic pages are modeled as rhombuses tagged by a pair containing the page name and the server name they reside on.

Pages with frames: our aim here is to model the application as seen by the web user, and we model frames as they appear in the client browser, so we model the page with frames as a box that is divided into sub boxes where each internal box refers to a frame of that page. The holding box should be tagged by the name of the main page holding the frameset. Inside each sub box we should write the page name that is originally loaded into that frame. Frame behavior is modeled by replicating the main page as long as navigation is done within one of its frames until the main page (containing the frames) is changed

Page transitions and links: Roughly speaking, pages are called and loaded into the browser using three ways: Hyperlinks, Form Submitting, and Server redirects.

- A *page* with a server redirection is represented as a normal dynamic page with an arrow arriving to it from the main page and another dotted arrow leaving it to the final output page regardless if it involves other intermediate redirects.

- *Links:* Web pages are connected to each other by hyperlinks or simply links. A link is modeled by drawing a single headed arrow from the page containing the clickable link to the page which the link refers to. The arrow is tagged by the list of parameters passed by the link.

- *Forms:* Forms are the most important components that provide the web user with an interface to submit data into the web page. We model a form on a page by a small circle tagged with the form name. The form submission s is modeled by a double headed arrow which is tagged by a list of the parameters that are passed.

Example

Consider a simple web application which allows a user to logon to his email inbox. For more details regarding the example, refer to [3]. This application consists of the following pages:

- *Logon page* (logon.html): This page contains two forms. The first a form for entering username and password and this form submits its content to the logon validation page (loginvalidate.asp). The validation page can redirect to the email page (mail.asp) if the logon information is correct or it redirects back to the logon page if they are wrong. After three unsuccessful logon attempts, the validation page redirects to an error page (sorry.html). The second form in the logon page is a form that requests the password to be sent by email. It submits its content to the page emailmypass.asp, which emails the passwords and then redirects back to the logon screen.

- *The inbox page* has three frames. The first frame has the page links.html loaded into it. This page has one link to sign out from the email which leads to the sign out page (signout.asp), which redirects in terms to the logon page. The second frame a page listing all the available folders in the email and it has the page folders.asp loaded into it. Clicking on any folder name in this frame will load the folder contents in the third frame. The third frame can have any of the following pages loaded into it: inbox.asp, sentitems.aso, or trash.asp.

Figure 2 depicts the example of the client side model.

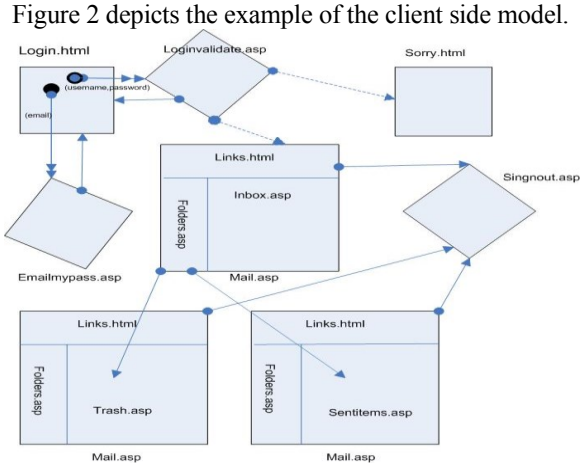

Fig. 2 - Example of Client Side Model

3.3 The Server Side Programs Model

In modeling the client side components, we looked at the web application as a set of static and dynamic HTML pages connected by hyperlinks. We did not analyze the different HTML sections and the way they were generated but rather we considered the page as a whole and analyzed the flow of execution between pages, assuming that the server side code is correct and supports the generation of pages and the flow of execution as per the requirements of the application. In this section we look into the details of the server side code that is simply the engine that generates the HTML output. We decided to adopt Wu and Offut's model which models web applications with emphasis on the server side programs [15]. While Wu and Offut [15] modeled the entire web application in one graph, we adopted his technique to represent the internal structure of an individual web component. We follow the steps below to create our server side model:

1- Atomic sections (AS): identify the atomic sections,
2- Composite sections (CS): derive the composite sections from the AS,
3- Transitions: identify transitions and interactions between different CS and AS,
4- Transition rules: identify transitions rules, and
5- Model the web component from ATS, CS, and transitions.

4. CONCLUSION AND FUTURE WORK

In this paper, we presented a complete theoretical analysis model for modeling web applications, which is divided into three sub models; the architectural environment model representing operational environment hosting the web application, the client side model representing the web pages as seen by the user and the

navigation between them, and the server side programs models that presents the server programs, which execute at run time and that produce dynamic HTML to the user.

Future work includes covering new architectures like .Net. Future work should also be focused on adding a model for representing and testing server side logic and external content sources.

REFERENCES

[1] Filippo Ricca and Paolo Tonella. Analysis and Testing of Web Applications. In *Proc. of the 23rd International Conference on Software Engineering (ICSE'01)*, 2001.

[2] Filippo Ricca and Paolo Tonella. Web Application Slicing. In *Proc. of International Conference on Software Maintenance (ICSM'2001)*, 2001.

[3] Hamzeh K. Al Shaar. Modeling, Testing, and Regression Testing of Web Applications. *Thesis submitted for MS Computer Science, Lebanese American University*, April 2006.

[4] Harry M. Sneed. Testing a Web Application. In *Proc. of the Sixth IEEE International Workshop on Web Site Evolution (WSE'04),* 2004.

[5] Hiroshi Suganuma, Kinya Nakamura, and Tsutomu Syomura. Test operation-driven approach on building regression testing environment. In *Proc. of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01),* 2001.

[6] Hong Zhu. Software Unit Test Coverage and Adequacy. *ACM Computing Surveys, Vol. 29, No. 4*, December 1997.

[7] Hui Xu, Jianhua Yan, Bo Huang, Liqun Li, and Zhen Tan. Regression Testing Techniques and Applications. *Technical paper from Concordia University, Canada, department of computer Science*, March 2003.

[8] Ivan Granja and Mario Jino. Techniques for Regression Testing: Selecting Test Case Sets Taylored to Possibly Modified Functionalities. *The 3rd European Conference on Software Maintenance and Reengineering CSMR'99*, March 1999.

[9] Lei Xu, Baowen Xu, Zhenqiang Chen, Jixiang Jiang,, and Huowang Chen. Regression Testing for Web Applications Based on Slicing. In *Proc. of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03)*, 2003.

[10] Martina Marre´ and Antonia Bertolino. Using Spanning Sets for Coverage Testing. *IEEE Transactions on Software Engineering, VOL. 29, NO. 11*, November 2003.

[11] Robert B. Wen. URL-Driven Automated Testing. In *Proc. of the Second Asia-Pacific Conference On Quality Software (APAQS'01)*, 2001.

[12] Sebastian Elbaum, Srikanth Karre, and GreggRotherme. Improving Web Application Testing with User Session Data. In *Proc. of the 25th International Conference on Software Engineering (ICSE'03),* 2003.

[13] Simeon c. Ntafos. A Comparison of Some Structural Testing Strategies. *868 IEEE Transactions on Software Engineering, VOL 14, NO 6*, June 1988.

[14] Ye Wu and Jeff Offutt. Modeling and Testing Web-based Applications. *GMU ISE Technical ISE-TR-02-08*, November 2002.

[15] Ye Wu, Jeff Offutt, Member, IEEE Computer Society, and Xiaochen Duz. Modeling and Testing of Dynamic Aspects of Web Applications. *GMU ISE Technical ISE-TR-04-01*, March 2004.

Acknowledgement: This work was funded by the Lebanese American University.

Ramzi A. Haraty is an associate professor and the assistant dean of the School of Arts and Sciences at the Lebanese American University in Beirut, Lebanon. He is also the Chief Financial Officer of the Arab Computer Society. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University - Mankato, Minnesota, and his Ph.D. in Computer Science from North Dakota State University - Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 100 journal and conference paper publications. He is a member of Association of Computing Machinery, Arab Computer Society and International Society for Computers and Their Applications.

Hamzeh Al Shaar received his B.S. and M.S. degrees in Computer Science from the Lebanese American University – Beirut, Lebanon. His research interests include software engineering and testing techniques.