Proceedings of the 5th WSEAS Int. Conf. on APPLIED INFORMATICS and COMMUNICATIONS, Malta, September 15-17, 2005 (pp305-312)

A New Method on Automated Web Application Testing

Mohsen Sharifi[1], Shahab Tasharrofi[1], Hamid Mahmoudzadeh[1]
[1] Computer Engineering Department, Iran University of Science and Technology,

*Abstrac*t: Internet and its web contents are extensively used in real trade and the existence of bugs in software can be disastrous. According to the specific properties of web applications, traditional test approaches are not applicable to this area. Though, lots of methods are proposed to assure the quality of web applications, advances are not considerable yet. This is because the specific properties of web applications have not been taken quite seriously. Our proposed method analyzes the web applications from the user's aspect and based on this analysis, extracts the test model. Then, our implemented software agents run the test model to simulate the user's behavior. These agents use both automatically generated and handwritten scenarios to test the web application. In addition to security, performance, and load tests, the method supports functionality test. The proposed method also indicated the necessity of a new phase in the web application development process.

Key-Words: Web Application, Testing, Software Agent

1 Introduction

Web applications usually get their input through HTML forms or XML structures and provide their output based on them. They have become popular within the last decade so that nowadays, lots of businesses have changed their business architecture to web based architectures. The experiences of past decade revealed that one of the most critical sections of web application development process is testing them.

Unique properties of web applications made traditional sets of tools and approaches unsuitable for them. And although researchers have proposed some new methods and accompanying tools to test web applications, these methods and their related tools have usually had restrictions especially in the area of proper functional testing of web applications. The proposed method, on the contrary, mainly focuses on functional testing of web applications. However, it also makes load and security tests more robust.

In the following, we will discuss about unique properties of web applications in section 2. Section 3 describes the different views which web applications could be tested based on. Section 4 defines a sample web based application and introduces some scenarios in which a real user tests the sample web application. Section 5 reviews the related works done in the field of web application testing. Section 6 introduces a new grammar which models web applications. The grammar is designed so that it automates the test process. Section 7 models the sample web application, discussed in section 4, using the grammar introduced in section 6. It also describes how this modeling enables a software agent to test the sample application. Section 8 proposes the architecture used for implementation and also mentions some implementation specific notes.

2 Traditional application testing versus web application testing

The differences of traditional applications and web applications make it hard to use traditional test methods for web applications. Knowing these differences provides us with the opportunity to propose an approach which covers most important aspects of web application testing. Some of these differences are as follows:

Almost all web applications, compared to traditional applications, need shorter maintenance periods [1]. Furthermore, the maintenance process usually includes some little changes to the business rules. To test these little changes, we should have automated test approaches along with flexible test suites [2].

Web application development involves lots of development kits and these kits are continuously changing. Due to these variations, test process should be independent of development kit [3].

Due to the huge amount of people who have the potential of accessing your web site, you never know how much customers you should expect for the next day. So, load test has always been one of the most important tests that an approach should be able to perform.

Web applications are usually developed due to individual requests of customers, unlike traditional applications which are usually developed based on

their necessity to public. While these customers are eager to use third-parties to perform test, they do not like to share source codes of these applications with them. So, independency from source code is much more plausible.

These differences raise the need to have new methods along with their accompanying tools to test the results of development process of web applications. These methods have to satisfy the conditions mentioned above.

3 Different Aspects for Web Application Testing

There are two different aspects, namely "User's View" and "Developer's View", which web applications could be tested based on.

3.1 Testing web applications based on developer's view of the system

Testing web applications based on developer's view of the system includes modeling the application as series of files, source codes, databases, documentations, etc and, then, checking each compartment of the model for existing bugs. It is also common to test different compartments of application along each other. Some of the benefits and shortcomings of this view is discussed here:

Doing the exact tests, based on developer's view along with formal definitions, assures you of the system's correctness.

Accompanying components of application, and thus test scenarios, could be extracted automatically.

Having access to files and data bases enables the test agent to split a long test to smaller tests. Thus, it makes test process much easier to be modeled. For example, you could refer to database to see whether the results of an action done on input are as expected or not?

Despite this view of test process is thoroughly investigated specially in the field of traditional application testing, there are some major drawbacks to this view which are discussed below.

Complexity of algorithms needed for an exact test makes it unrealistic. They could not actually be implemented with current knowledge of human. Thus, the most important benefit of this view is almost completely out of reach.

The need for source codes threatens the independency of test process and development process. It is also noticeable that source code security is of high importance to lots of businesses. And so, it's almost impossible to use third-parties to test applications while insisting on this view.

Unlike test scenarios, which could be extracted automatically, it's absolutely difficult to extract test steps autonomously.

While the correctness tests are easier to be defined here, the quality tests do not seem to be that much easy.

The hidden relations of test scenarios, which are not clearly coded, are hard to be defined here.

3.2 Testing web applications based on user's view of the system

Testing web applications based on user's view of the system includes a series of actions in which every action sends an HTTP request to server and server responds to it using HTML syntax.

Based on the fact that, in both web and traditional application domains, it's finally the end user which determines whether the application works properly or not; this method's result turns out to be more useful than the previous method. There are also some benefits and shortcomings to this view which are discussed below.

There's no need for reverse engineering process in this view. So, different development platforms, which are common these days, make no difficulty in test process. Even the newest platform, which you've published only yesterday, is capable of benefiting from existing tools using this view.

Systems that pass such a test may not be bug free; however, there is no need to be fearful of publishing these systems. Because the end user, hopefully, will not experience any inconvenience using these applications. And it's totally because of the nature of these tests which signify the bugs a typical user could come across with.

"Load Test" and "Fault Tolerant Test" are more meaningful here than their equivalents in tests based on developer's view of the system.

Because of independency from source codes, it's quite easy to expand a methodology to benefit from this type of test.

Typical users' behaviors could be easily applied to these kinds of test to model different user behaviors such as malicious users, novices, etc. Therefore, it's easy to perform security tests, usability tests, etc.

On the other hand, there are also some shortcomings which worth to be discussed here, such as ones that follow:

Some tests, which are related to human intelligence, are hard to be defined here.

Despite the possibility of providing scenarios as input of the system, they will usually be generated

by selecting random paths. So, it is likely to find types of scenarios which have not been tested yet.

Although it's possible to generate some parts of test model automatically, a large portion of it should be extracted manually. And, for sure, it's a time consuming process which makes the modeling of an enterprise system extremely tiring. And, perhaps, the opportunity of expanding a methodology, which was described before, becomes a necessity.

Having all advantages and disadvantages of these two aspects in mind made us to test web applications based on user's view. Next section describes some sample scenarios in which a user tests a sample web application. After that, in section 7, we will discuss how these sample scenarios will be converted to automatically generated scenarios.

4 Case Study: Web Application Testing based on User's View

Consider a sample web based application which is designed to act as a phone book. As shown in navigation diagram in **Fig**.1, it includes an index page, a page to add new contacts, a page to search for a name, a page to show the search results and a page to show errors.

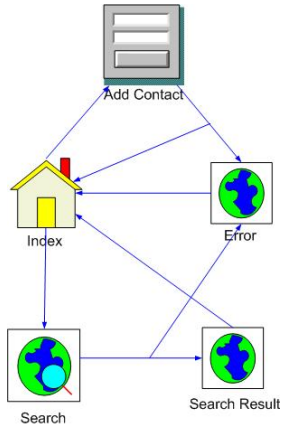

Fig. 1. Navigation diagram for sample web application

As it's shown in **Fig**.1, some of the links between pages have branches (like the link which connects 'Search' page to 'Search Result' page and 'Error' page). These are the links which could have more than one result based on the state of the server and the query of the user. For example, when you search a name in the 'Search' page you would come across the 'Search Result' page giving phone number of person you searched for or it may lead you to 'Error' page that contains a message indicating that this name does not exist in the database.

Although this sample is so simple, it's capable of demonstrating both complicated and non-complicated tests.

First of all, a real user has the ability of defining what page he/she is viewing and whether this page contains any mistakes or not? For example, 'Index' page is supposed to have two links to 'Search' page and 'Add Contact' page. A real user could define whether the page has these two properties or not?

Second, a real user could define whether the links on pages work correctly or not? For example, he/she could check if the only link from 'Error' page directs him to 'Index' page?

Third, he/she can determine if the whole system works properly or not? For example, if he/she gets an error message, while adding a name, indicating that the name already exists in the database, he/she is capable of searching through the search page to know whether the message was correct or not?

Fourth, he/she has the memory which tells him/her what names has been given to the system before. So, he/she would detect the error if system accepted a name that he/she knows it has previously been given to it.

These four example scenarios of testing web applications define four different levels of complexity in test process, ordered by their complicatedness. As we will discuss in the next sections, it's easy to automate the process of generating these scenarios having a well-defined model for web application.

5 Related Works

There is a vast literature on testing applications. However, when it comes to web application testing, it could be considered as a quite new topic. Further, there are only a few published works which have thoroughly investigated the critical aspects of web application testing. Brief descriptions of some of these works are as follows:

Hieatt and Mee [4] introduce an approach called "Acceptance Test". Acceptance test allows managers to describe and perform tests in scenario level. They've also developed a tool box to describe the scenarios and expected results in XML structure.

Elbaum [5] uses user session data to test web application. This approach concentrates on two types of techniques. First technique generates test cases based on the whole session while the second technique generates them based on combinations of sessions. Technology independency, low test cost and use of real users in test process are some of the reasons that favor for this method.

Jia [6] proposes an automated test approach based on formal definitions of web applications. In this approach, web applications are tested from the

Test-model	=>	{ **Element** }
Element	=>	Name URL Method **Constant-params Input-params Output-params** { **Validation** } { **Navigation** }
Constant-params	=>	**Parameter-definition**
Input-params	=>	**Parameter-definition**
Output-params	=>	**Parameter-definition**
Validation	=>	Name **Input-params Regular-expression**
Navigation	=>	Target-element Name **Fill-form-data Output**
Params-definition	=>	{ **Parameter-definition** }
Regular-expression	=>	Regular-expression
Fill-form-data	=>	**Params-definition**
Output	=>	{ **Page-descriptor** }
Parameter-definition	=>	Name Type Value
Page-descriptor	=>	**Expected-result** { **Post-condition** } { **Memorize** }
Expected-result	=>	Element-name Validation-name **Validation-input**
Post-condition	=>	Initial-element-name { **Step** }
Memorize	=>	**Parameter-definition**
Step	=>	**Use-navigation Expected-result**
Use-navigation	=>	Name **Navigation-input**
Validation-input	=>	**Parameter-definition**
Navigation-input	=>	**Parameter-definition**

Fig. 2. Proposed language grammar. Bold text indicates variable of the language while ordinary text indicates terminal of the language

aspects of functionality, web page structure, security and performance. Based on formal definitions, in XML syntax [7], each test contains one or more test suites in which each of them contains some test cases. Test cases, also following formal definitions, contain one or more of test steps. Having these formal definitions, a test engine is used to execute test suites automatically. The approach which Jia proposes is based on user's view of the system. And so, our approach is the same as Jia's approach in the way that we both test the system based on what user views.

Di Lucca [8] uses an object oriented test model to propose the definition of web application test units. Based on this model, an approach for "Unit Test" and "Integrated Test" is proposed. So, the approach includes two important phases for functional testing. The first one is the "Unit Test" which deals with the test process of individual client and server pages and the second one is "Integrated Test" which tests the pages that are involved in specific use cases. Authors have also introduced the WAT (Web Application Testing) tool to support their approach.

Kung [9] proposes a model to describe web sites using the graph theory. First of all, he defines different kinds of web page navigations using graph theory. And then, he introduces "Intra-object" test which tests selected paths for variables based on their "definition-usage" chain in their own object. He also uses "inter-object" test to test the selected paths of variables which their "definition-usage" chain is spread among all objects. Further, the "inter-client" test, computed based on reachability graph, is used to map the data interaction between the clients.

Yang et. al. [10] integrate traditional test approaches to propose a general architecture for web application testing while focusing on 3 layer models. They add a new sub system, called "Source Documentation Analysis Sub-system (SDAS)", to the architecture proposed at [11] to solve the problems of different programming styles of web application development. To use this architecture, test tool development process is divided to the phases of design, selection, specification, integration and implementation.

Wu [12] analyzes static and dynamic aspects of web applications and then defines a general analysis model which specifies typical web application behaviors in a technology independent manner. This model is based on specifying atomic particles which are generated dynamically using web pages which have static structure but dynamic content. Then analysis model is made based on order, selection and union operators of atomic particles. Based on this model, a family of test techniques is introduced to assure us of the different levels of web application quality.

The proposed method benefits from the user's view of system like what Jia [6] does. But, unlike what Jia does in [6], test suites, test cases and test steps will all be generated automatically. Furthermore, the result of these tests will be determined automatically based on what test model defines. This way, the proposed method also covers the benefits of "definition-usage" which Kung proposes in [9]. The method is also capable of using user session data like what Elbaum does in [5], although, it is not used yet.

6 Web Application Modeling Grammar

The key feature of the proposed method is its language grammar, based on XML, which is capable of describing web application test models from the

user's view. It is considerable that the HTML files, generated by server, are the only requirement for modeling the whole web application based on this grammar.

Fig.2 demonstrates the proposed language grammar. The grammar is designed to model everything like what a real user does. For example, an element models is equivalent to a web page if user considers it as one web page. But, if user considers a web page with one URL as more than one page, it should be modeled with more than one element. This usually happens when a web page is designed to accept some input, like 'action' or 'tab', and generate completely different outputs based on them.

'Test-model,' as shown in **Fig.**2, is the root element of the model and contains any other element.

'Element,' as discussed before, represents web page from users' view. Each web page in the system should be represented in the model by either one or more than one elements. However, it's not possible to represent different web pages by one element because of their different URLs. The value of parameter 'Name' in element 'Element' defines the name with which the model refers to this element. Parameter 'Method,' as its name says, defines the method with which this element should be called. It has either the 'Post' or the 'Get' values. 'Constant-params' are defined to access a web page that is modeled by different 'Element's, like the one which accepts a 'tab' parameter in order to define what page it should generate in output.

'Validation' defines a validation procedure with which we could distinguish different pages from each other. It's important to note that 'Element's could have more than one validation procedures. For example, based on the sample defined in **Fig**.1, the 'Error' page should have two validation procedures each of which validating one of the error messages of the system.

'Navigation' defines a method to navigate from one page to another and pass data from the source page to destination page. It's noticeable that, although, 'Navigation's usually represent a one-to-one relation with 'Navigation's of the real page; they're not restricted to that. Specially, when 'Navigation' expects some input which could be extracted from agent's memory it's common to use two different 'Navigation's. First type generates inputs randomly and second type extracts these inputs from agent's memory.

'Navigation' also defines at least one 'Page-descriptor' within its 'Output' part. 'Page-descriptor's are the elements which define what pages are expected after 'Navigation' is done. For example, navigation from 'Index' page to 'Search' page only expects one page in its output and that's the 'Search' page itself. So, it should be modeled with only one 'Page-descriptor'. While, on the other hand, navigation from 'Search' page expects more than one page in its output, namely 'Error' page and 'Search Result' page. So, this navigation has two 'Page-descriptor's in its 'Output' part.

'Page-descriptor's also define more complicated features of web application testing. They describe some 'Post-condition's within themselves. 'Post-condition' checks state of system after doing the specified navigation. So, they automate functionality test process for web applications.

'Memorize' tags, which are contained in 'Page-descriptor' tags, fill agent's memory with data which it has once came across with. In our example, this memory contains the names and their related phone numbers which an agent has entered the system. While, in other web applications, it could contain quite different pieces of information such as usernames and passwords or the information related to an item that it has placed in an auction.

It's also important to note that the memory of an agent is simply an XML structure which provides agent with the ability of having nested structures.

The other important part of this grammar which needs special attention is its regular expressions. Although, the name 'Regular-expression' describes what we expect from that, but the matter of completeness made us to do some extensions to it so that it could use the value of another variable within itself and also it's able to use XPath to extract data from either the XML structure of its memory or the XHTML structure of server's response.

Having all these facilities for modeling a web application enables us to model quite complicated applications and test them automatically based on this model. Next section contains some notes on modeling our sample web application and how this model could be used to generate test scenarios.

7 Case Study: Modeling and Testing of Sample Web Application

As mentioned before, we will describe important notes on how our sample application could be modeled using our grammar. And more, how different scenarios of section 4, which indicate different complexity levels of scenarios, could be generated automatically based on this model. First of all, we will define the model as it seems in the navigation diagram of section 4. After that, some

parts that are needed to complete the model will be introduced. Then, we would discuss how this model generates those specific scenarios.

Based on the definition of our grammar and description of our sample phone book, we should model our application using 5 elements each of which representing a page in our sample application. We could also assume that the name of each element is identical to its page name, i.e. “Search Result” element indicates “Search Result” page.

Except “Error” element which needs two “Validation” procedures, other elements need only one “Validation” procedure. Besides, “Search Result” element is the only element which its “Validation” procedure needs input.

Other than “Index” element which contains two navigations, which we will call “Index-Search” navigation and “Index-Add” navigation; other elements only need one “Navigation” tag which we will call them using their element name, i.e. “Add Contact” navigation for the only navigation from “Add Contact” element.

Between these elements; there are two elements, namely “Search” and “Add Contact” elements, which their navigation needs “Fill-form-data” tag.

These two elements are also the only elements that contain two “Page-descriptor” tags in the “Output” part of their navigations.

The “Page-descriptor” tags of these two elements’ navigation are also the only “Page-descriptor” tags which contain “Post-condition” tags. Each of these “Post-condition” tags contains steps to check the correctness of results from server. For example; the “Page-descriptor” of “Add Contact” navigation, which describes the branch to “Error” page, contains steps to check whether passing the same name to “Search” navigation supports the idea of existence of such name or not? If it doesn’t support this idea, i.e. no result is found for that specific name, a bug is found in the system.

And the only “Page-descriptor” which contains “Memorize” tags is that of “Add Contact” navigation which describes its branch to “Index” page. This “Page-descriptor” saves name and phone number which were passed to

element we’re working on, we would be able to define whether this element is correct or not? This would be achieved by applying all different “Validation” procedures of that element on given HTML for that element.

Second scenario could be tested using “Page-descriptor” tags in “Output” part of navigations. When we do some navigation, we expect its result to be described using one of navigation’s “Page-descriptor” tags. And, as shown in **Fig**.2, each “Page-descriptor” tag contains an “Expected-result” tag, which indicates a pair of “Element-name” and “Validation” procedure. So, we could check navigation’s HTML result with every “Validation” that is defined in its “Expected-result” tags. If it was not validated with none of the “Validation” procedures, we would suppose that a bug is found in the system. While, if it was validated with one of “Add Contact” navigation in agent’s memory. Using this piece of information in agent’s memory, agent is able to check, in future, that whether the phone book gives that special phone number for this name or not?

Now that we have modeled the navigation diagram of **Fig**.1, we’re able to complete our model to use agent’s memory and do more complicated tests. So, we will add three more parts to our model as follows.

First of all, a “Navigation” tag will be added to “Add Contact” element, called “Memory Add,” to check that whether the phone book allows to add a name which was previously added or not? It’s important to note that this navigation has only one “Page-descriptor”. Because, by adding a name which was added before, the system would only navigate to “Error” page if it worked properly.

Second, a “Navigation” tag will be added to “Search” page, called “Memory Search,” to check if the phone book has memorized specific pieces of information, that has previously been given to it, or not? It’s obvious that, like previous navigation, this navigation also has only one “Page-descriptor” tag. Because, by searching for a name which was added before, the system would only navigate to “Search Result” page if it worked properly.

Third, adding the second “Navigation” makes us to add a “Validation” procedure to “Search Result” element. Because, the second “Navigation” not only knows the name but also knows its accompanying phone number. Therefore, it needs a “Validation” procedure that takes both these arguments together and checks both of them in the server’s results.

Having modeled the web application completely, it’s easy to describe how different scenarios could be generated based on this model. Therefore, the following paragraphs describe the process of generating test scenarios discussed in section 4 automatically.

First scenario, defined in section 4, could be tested using “Validation” procedures. By examining all different “Validation” procedures of elements that have no input, we could define what element we’re working on. Furthermore, if we knew what

these "Validation" procedures, we would know the next element which we should work on. In our example, in section 4, if the only "Page-descriptor" validated the HTML result of navigation from one "Error" page, we would know that we're in "Index" page. But, if it didn't validate the HTML result, we would know that we've found a bug in phone book.

Third scenario, defined in section 4, could be tested using "Post-condition" tags of "Page-descriptor" tags. Once we know what "Page-descriptor" validates the HTML result of server, we could apply its "Post-condition" tags to check if the system, as a whole, works properly or not?

Fourth scenario, defined in section 4, could be tested using navigations that use information stored in agent's memory instead of randomly generated inputs. As we discussed earlier in this section, it would be useful to create two new "Navigation" tags, "Memory Add" and "Memory Search" navigations, which use information stored in memory and have only one "Page-descriptor" in their "Output" part. Using these two navigations, we could easily simulate the fourth scenario defined in section 4.

8 Implementation notes

Although it's possible to propose lots of different architectures to implement our grammar, **Fig**.3 shows our proposed architecture.

Although lots of subsystems in **Fig**.3 are clear, some of them need more implementation specific information which is included below.

"Test Scenario Generator (TSG)" generates scenarios randomly. While, it's designed so that it would benefit from user session data in future. Using user session data provides this subsystem with the opportunity to generate more realistic test scenarios. It also makes the results of bug reports more clear than they were before it.

Though lots of scenarios could be generated automatically, it's possible to generate scenarios manually. For this purpose, we've developed a scenario description language based on what is shown in **Fig**.4.

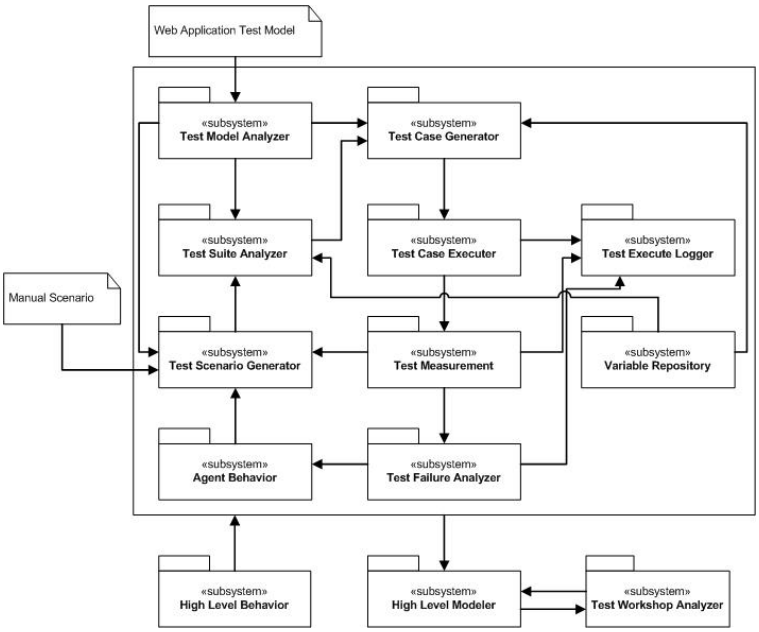

Fig. 3. Proposed architecture to implement an agent which uses a model based on our grammar as its core. Subsystems that are surrounded by rectangle build the agent's internal architecture. The design insists on the independency of each agent from its environment.

Test-Scenario	=>	{ **Step** }
Step	=>	**Use-Navigation** **Expected-Result**
Use-Navigation	=>	Name { **Set-Input** }
Expected-Result	=>	Element-Name Validation-Name **Set-Input**
Set-Input	=>	Parameter-Definition

Fig. 4. Scenario Description Language

To generate scenarios automatically, based on test model; we have to execute a series of actions. First of all, a destination and a path between current location and destination are chosen such that the selected navigations, other than last navigation, contain only one page descriptor. Then, the chosen set of navigations will be executed. Therefore, the new location will be the result of final navigation.

Then, this process will be repeated for the new location.

It should be noted that, in Test Suite Analyzer subsystem, post conditions will be executed in different session, from which we run main scenario on, to assure us that the internal parameters of server, for the session that is running main scenario, will remain unchanged.

Currently, the "Test Failure Analyzer" subsystem only causes a restart in scenario generation process. But it could be used to help TSG to examine sequences near the current sequence of actions to find more samples of this kind of failure. Having more samples helps future reviewers of test results in finding the main cause of test failures.

The "Agent Behavior" subsystem defines the routine behaviors which an agent should show. This is an empty subsystem currently. But there's a hope that this subsystem could be useful in future to simulate different types of users like experts, novices, hackers and etc.

The subsystems of "High Level Behavior," "High Level Modeler" and "Test Workshop Analyzer" are not implemented yet and so, they will not be investigated here. They will generally be used to simulate inter-agent behaviors. They also define models which all agents should follow in their behavior.

Future Works and Conclusion

As mentioned before, a number of subsystems in our proposition are not implemented yet or incomplete until now. Their full implementation will cover more complicated features of web applications.

In addition, user session data could be used to generate scenarios more realistically. User session data could also be used in test case generation. This way, the test cases would be more readable for future reviewers. It is also possible to add special behaviors to agents to simulate different types of users like experts, novices and hackers as well as amateurs.

There's also a hope to generate templates of test models automatically to help the modeling of enterprise web applications.

Given the favorable expected status of web applications, it is important to test them as accurately as possible. The method proposed in this paper presented a language to model web applications for test purposes. The language is designed to generate test scenarios automatically while considering the unique properties of web applications. It simulates users' actions, generates and sends requests to server, validates the server's responses and finds some bugs of web applications. The proposed method supports functionality tests, load tests, performance tests and also security tests.

References:

[1] Kirda, E.; Jazayeri, M.; Kerer,C.; Schranz, M., Experiences in engineering flexible web services, *IEEE Multi-Media*, Jan 2001, Page(s): 58-65.

[2] Offutt, J., Quality attributes of web software applications, *IEEE Software: Special Issue on Software Engineering of Internet Software*, March/April 2002, Page(s): 25–32.

[3] Wu, Y.; Offutt, J., Modeling and testing web-based application, *IEEE Software: Special Issue on Software Engineering of Internet Software*, March/April 2001.

[4] Hieatt, E.; Mee, R.; Going faster: testing the web application, *IEEE Software*, Volume: 19 Issue: 2, March-April 2002, Page(s): 60 –65

[5] Elbaum, S.; Karre, S.; Rothermel, G.; Improving web application testing with user session data, *Proceedings. 25th International Conference on Software Engineering*, 3-10 May 2003 Page(s): 49 –59

[6] Jia, X.; Liu, H., Rigorous and automatic testing of web application

[7] *Extensible Markup Language (XML) 1.0*, Second Edition, W3c Recommendation, 6 October 2000

[8] Lucca, D.; Fasolino, R.; Carlini, F., Testing web application

[9] Kung, D.; Liu, C.; Hsia, P., An object- oriented web test model for testing Web Application

[10] Tzay, J.; Huang, J.; Wang, F.; Chu, W., Constructing an object-oriented architecture for web application testing, *Journal of Information Science and Engineering*, 18(1):59- 84, Jan.2002

[11] Richardson, D., TAOS: testing with analysis and oracle support, *in proceedings of ASM SIGSOFT International Symposium on Software Testing and Analysis(ISSTA 94)*, 1994, pp. 138-153

[12] Wu, Y.; Offutt, J., Modeling and testing web-based application, *George Mason University*