Journal of Software Engineering and Applications, **2011, 4, 86-94**
doi:10.4236/jsea.2011.42010 Published Online February 2011 (http://www.SciRP.org/journal/jsea)

Scientific Research

An Agent-Based Framework for Automated Testing of Web-Based Systems

Samad Paydar, Mohsen Kahani

Computer Engineering Department, Ferdowsi University of Mashhad, Mashhad, Iran.
Email: samad.paydar@stu-mail.um.ac.ir, kahani@um.ac.ir

Received November 15th, 2010; revised January 18th, 2011; accepted January 22nd, 2011.

ABSTRACT

Parallel to the considerable growth in applications of web-based systems, there are increasing demands for methods and tools to assure their quality. Testing these systems, due to their inherent complexities and special characteristics, is complex, time-consuming and challenging. In this paper a novel multi-agent framework for automated testing of web-based systems is presented. The main design goals have been to develop an effective and flexible framework that supports different types of tests and utilize different sources of information about the system under test to automate the test process. A prototype of the proposed framework has been implemented and is used to perform some experiments. The results are promising and prove the overall design of the framework.

Keywords*: *Web Application, Web Service, Agent, Framework, Test, Test Script

1. Introduction

In the last few years, web-based systems[1] as a new genre of software systems have found their way into many different domains like education, entertainment, business, communication, and marketing. Parallel to this interest in development of web-based systems, many needs arise due to the importance of assessing the quality of these systems. Software testing is the traditional mechanism for this purpose and it has long been used in the software history. Web-based systems, due to their special characteristics and inherent complexities are more difficult to test, compared to traditional software [1-4]. These complexities increase the cost of testing web-based systems. Test automation is the main solution for reducing these costs. Considerable effort has been dedicated to the development of tools, techniques and methods that automate different tasks in the testing process [1,5], but they are usually limited to one part or activity of the test process (e.g. test case generation, test execution). In addition to these limited solutions, some works have focused on presenting an integrated test framework that can be used to perform the whole test process with as much automation as possible. The complexity of web-based systems dictates that a systematic test framework, which is suitable for their architecture, is needed rather than a set of independent tools [1].

In this paper, an agent-based framework is presented for testing web-based systems and a prototype of this framework is developed. The main design goals have been to develop an effective and flexible system that unlike most of the existing test frameworks is capable of supporting different types of test with as much test automation as possible. The framework is designed to be capable of utilizing different sources of information about the System Under Test (SUT) in order to automate the test process.

To meet these goals, the proposed framework is a multi-agent system consisting of a set of agents. Different agents, collaborating with each other, perform the activities involved in the test process. Therefore, one of the main issues in the design of the framework is the identification and separation of different parties and roles that are involved in the test process. From this point of view, a reasonable design helps to improve the extendibility and flexibility of the framework. As will be discussed in Section 0, these goals are met in the design of the proposed framework.

The paper is organized as follows: Section 2 briefly reviews the most related works. In Section 3, the design of the proposed framework is discussed in detail and its components are introduced. Section 4 briefly discusses the implementation of the prototype system. Section 5

[1]By web-based systems, we mean traditional web applications and also web-service based systems.

discusses the evaluation of the proposed framework. Finally, Section 6 concludes the paper.

2. Related Works

In [6] a framework has been presented to automate testing of operational web-based systems. In this framework, user sessions are extracted from the web server logs during real user interactions. This information is then used to generate a reduced set of test cases. Each test case consists of a set of URLs and name-value pairs. These test cases are replayed against the system and system responses are recorded. Then, a fault-seeded version of the system is created and all test cases are rerun against this version. Results are captured and compared by the oracle with the previously recorded results, *i.e.* expected results. One of the main benefits of this framework is that the test suite evolves automatically as the operational profile of the application changes. This framework, despite its success in automating the test process to some extent, is focused only on user-session based testing of an operational system. Further, it is required that the source code of the system is available and accessible. The white-box strategy clearly limits the applicability of the framework in real situations.

Beside this work, in [7,8] session data have been used for web application testing. However, they have not proposed a new framework.

In [9] a framework is developed focusing on testing web services. The main idea of this work is that a service should be accompanied by a *testing service*. The testing service that tests the original service, called functional service, can be provided either by the same vendor of the functional service, or by a third party. Therefore, the testing framework is itself a service-oriented system. Despite its valuable insights that present a good theoretical framework, this work has only focused on web services and there are many issues to be addressed before it can be considered as a complete framework for automatic testing of web based systems.

In [10] authors present a model for developing software testing frameworks using agent-oriented software engineering paradigm. The layout of this framework consists of three sets of components: distributor agents, testing agents, and cloning agents; each of which runs locally on different machines in a network. The distributor agents, at the topmost layer, are responsible for coordination activities and resource allocation of lower layers. Testing agents receive their assignments from their respective distributor agents, and are responsible for providing different testing environments. Testing agents, based on the load of their task, select one or more cloning agent to delegate the task to. The authors have concluded that although their experimented goals have been quite narrow, but it is quite powerful in reducing the test case generation time and effort, and also testing effort and fault detection cost.

In [11] a discussion is presented with some supporting examples on how a test specification based approach using a language, such as TTCN-3, can be used to define the test cases at different levels of abstraction [12]. It has concluded that test specification at an abstract level is less volatile in the face of presentation and implementation complexities. The main drawback of this approach is the high level of sophistication needed in terms of understanding TTCN-3 specification and also required effort to analyze and generate these specifications. Also, as these specifications are usually lengthy and they are created manually not automatically, they can become a source of failures, as well.

In [13] a framework is discussed for testing security of the web-based systems. The framework is developed to address two types of security attacks; SQL injection and XSS attacks. The main drawback of this work is its limited scope, as it is only useful for security testing.

Kung introduces an agent-based framework for web application testing [14]. The agents are designed based on the Belief-Desire-Intention (BDI) model of rational agents. Each agent has belief, desire, and intention. The framework has been discussed at an abstract level. A set of abstract classes (e.g. Belief, Goal, Agent) are defined and it is said that in order to test a web application, specific test agent classes must be subclassed from the appropriate abstract classes. Therefore, the proposed frame-work provides just a high-level sketch of an agent-based framework for web application testing. No discussion is presented from the implementation point of view. For instance, it does not discuss how the action plans are generated, how test cases are defined.

In [15] a multi-agent framework is introduced for data-flow testing of web applications. The proposed framework is based on the idea presented in [14] and it realizes the framework discussed there. A set of testing agents are developed for performing method level, object level, and object-cluster level data-flow tests on web applications. This work is focused only on data-flow testing and uses white-box strategy. The framework is not fully automatic and test cases are generated manually or by the use of some automatic test case generation technique, but no special technique has been presented in [15].

A test harness is proposed in [16] for web applications. Here, test scripts are presented in an abstract test script language. Test harness analyzes the source code of the web application to determine which technologies are used in its development. It then selects a testing tool capable of testing features specific to the detected technologies, converts the abstract test script to the required technology-

specific version which can be run by the tool. It then uses the testing tool to execute the test and stores the results. The proposed test harness does not provide any facility to automate generation of abstract test scripts.

Webtest [17] is another test framework which is based on the notion of hybrid testing, *i.e.* using both static and dynamic testing methods. Record-replay style is used as a static testing method, while model-based testing is used for dynamic testing.

3. The Proposed Framework

The existing web application test frameworks have two main characteristics in common. First, all of them are somehow limited both in terms of the test strategy they use (white-box, black-box, gray-box) and the types of tests they are designed for. For instance, [6] addresses only white-box strategy and session-based test case generation, while [13] is addresses only security tests. The second point is that, despite their differences, the way they finally execute a test is almost similar. In other words, regardless of whether a security test is being executed, or a functional test generated from TTCN-3 specifications, in both cases the test execution is performed by a set of HTTP interactions with the target system. Therefore, it can be concluded that it is possible to have a framework that supports different types of tests. The reason is that a test, whether a security test or a load test, finally is executed in terms of a set of HTTP interactions with the SUT. So, if there is a formal format for test specification, then it is possible to develop different modules, each of which generates the specification of a special type of test. In addition, a single module can be developed for execution of all types of tests. All that is needed is that the tests are represented in a format that the executer module understands, and the executer module is able to behave like a web browser and perform HTTP-based interactions. The proposed framework relies on this point to support different test types.

Our goal was to design a test framework for testing web applications. The main design goals were effectiveness and flexibility. By effectiveness we mean that the framework is useful for automated execution of different types of tests, such as functional, load, stress, security or regression test. By flexible we mean that the framework should be designed in a way that adding new functionalities can be achieved with some reasonable level of effort, *i.e.* the architecture of the framework is open to future changes and improvements. To meet these goals, it was decided to design a multi-agent architecture for the framework. By analyzing the system from a more abstract point of view, different concepts (e.g. test script, test code) and roles (e.g. test script generator, test executer) involved in the test process were identified.

In the proposed framework, different kinds of agents responsible for performing different tasks and playing different roles are defined. This separation of concerns is helpful in achieving the desired goals. As each agent is responsible for performing almost a single task, it reduces the complexities of implementing the agents and also enables new agents to be added in the future. Another benefit of using multi agent architecture is that different agents can be distributed across a network and provide a distributed framework for testing web-based systems that are themselves inherently distributed. This distributed architecture can increase the effectiveness of the framework because it facilitates some tests to be performed in a more actual style. The main drawback of using a multi-agent architecture for the framework is that it imposes some communication overhead because of the messages that must be transferred between different agents to perform their activities. In addition to the communication overhead, the definition of interfaces through which different agents collaborate with each other is important.

The overall architecture of the framework and its parts are illustrated in **Figure 1**. Different parts of the system are discussed in the following sections.

3.1. Basic Terms

In this section, some of the basic terms that are frequently used in the description of the framework are introduced.

Test Script: A test is composed of a set of actions or steps (e.g. opening a web page, entering some values in the fields of the page, submitting the page…). Each action has a type (e.g. open, submit, fill, assertTitle) and it may require some parameters (e.g. the URL of the page to be opened). Therefore, having an appropriate set of actions defined, a test can be specified in a text file which we call it test script. It is worth mentioning that a test script contains test criteria and information needed to judge about the test result. In other words, there is no separate part as a test oracle.

Test Code: Test code is a piece of program written in a programming language which is logically equivalent to a test script. A test code is generated by performing some transformations on a test script

Test Case: Test cases are data items used in performing different steps of the test. For instance in the login scenario presented earlier, the values used as the username and password are some test cases.

3.2. Test Runtime Environment Agent

Test Runtime Environment (TRE) agent is the central part of the system. It communicates with other agents in order to manage the setup and execution of different activities of the test process. TRE is also responsible for providing suitable interfaces for the user. TRE uses Test

Script Generator (TSG) agent for creating test scripts. When TSG has created the test script, it sends it to TRE. Receiving the test script from TSG, TRE passes it to a Test Code Generator (TCG) agent, which creates the test code from the test script, compiles it and returns the compiled test code back to TRE. Then, TRE allocates some Test Executer (TE) agents for executing the test, and sends the compiled test code to them to be executed. TRE is also responsible for allocating a Dashboard agent and introducing it to the TE agents executing the test. TE agents communicate with the Dashboard agent to provide real-time information about the test process.

3.3. Test Script Generator Agent

TSG agent is responsible for providing facilities through which the user can create a test script. Using TSG, the user can select how the test script is generated. There are two possible choices in the framework: using a Recorder agent, or using a Modeler agent. Based on the user's choice, TSG calls the recorder agent or the modeler agent to create a test script. These agents, after generating the test script, return it back to TSG. TSG enables the user to view the test script and to edit it if required. After all, TSG sends the test script to TRE and TRE continues the test process.

3.4. Test Code Generator Agent

Test Code Generator (TCG) agent generating a test code from a test script, compiles it and sends the compiled code to TRE.

3.5. Test Executer Agent

A TE agent receives the executable code (generated by TCG agent) from the TRE. It then executes the received code. In addition, during the test execution, it is in communication with a Dashboard agent and sends the partial results to it. After the execution of the test is completed, the TE agent sends the total result to the TRE. It is important to note that it is possible (and even sometimes required) that multiple TE agents be involved in running a test. For instance, in case of load test, multiple agents can be created on different machines and execute the test from that machine to simulate concurrent users of the system.

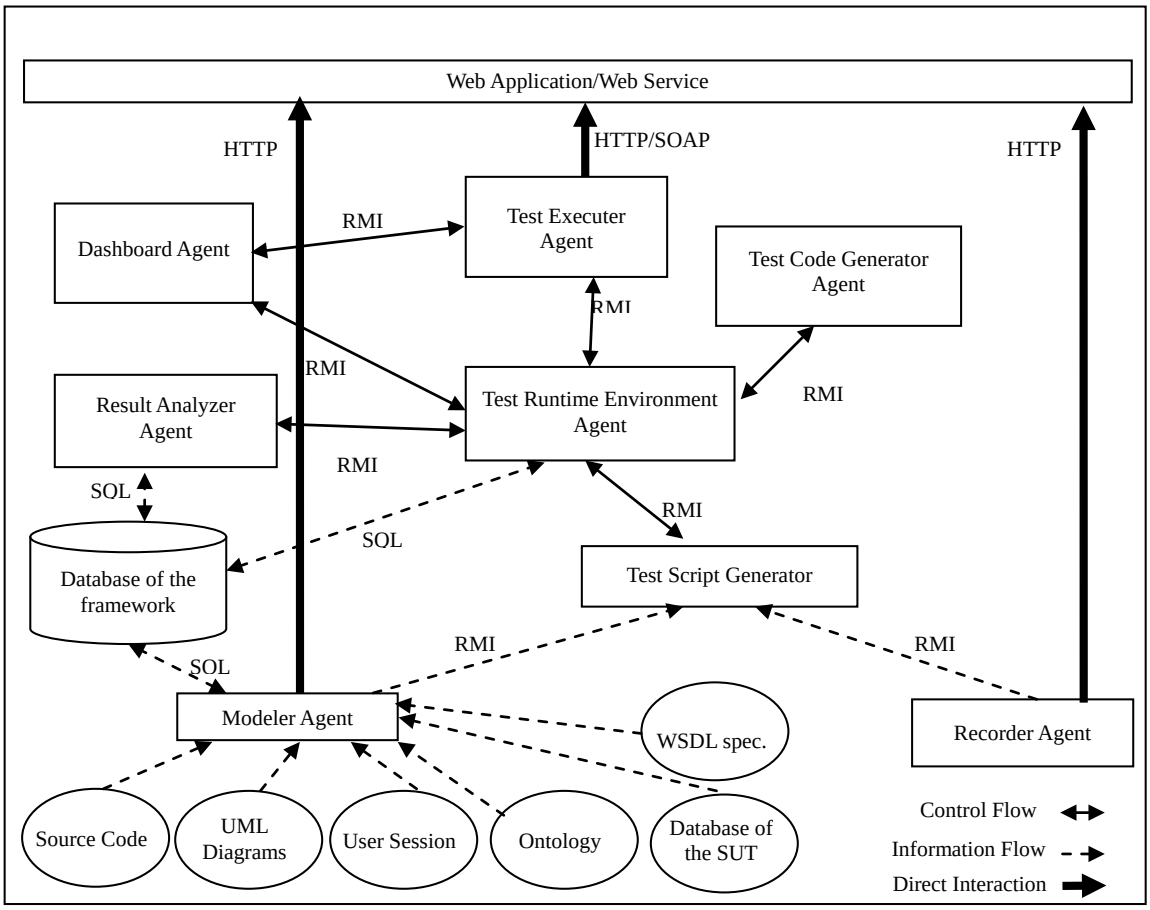

Figure 1. The architecture of the proposed framework.

3.6. Dashboard Agent

When a TE agent is executing a test, it sends the partial results of the test to a Dashboard agent. Dashboard agent uses such data to provide a real-time display of the test execution status and test results.

3.7. Result Analyzer Agent

When the TRE receives the test results from TE agents, it sends them to a Result Analyzer agent to perform user-specified analysis on them. It is possible that different types of Result Analyzer agents, with different capabilities, exist in the system. Such agents can create reports in different formats and generate different kinds of graphs and tables presenting the test results in more comprehensible forms.

3.8. Recorder Agent

Recorder agent is responsible for generating test scripts by recording the user interactions with the SUT. It provides a browser-like facility for the user to perform some interactions with the SUT and it captures these interactions as a test script.

3.9. Modeler Agent

Modeler agent, which enables model-based testing, is used to generate a test script based on some formal or semi-formal model of the SUT. If such models are available, they can be utilized to generate test scripts.

Different types of Modeler agents can be implemented, each of which uses different source of information as a model to create the test script. We have identified these types of models or information sources:

- Navigation model: The simplest case for a Modeler agent is to create a test script from the navigation model of the SUT. Navigation model represents a web application in terms of its composing pages and navigation links [18]. As an example, [18,19] are based on extracting the navigation model and using this model for testing a web application.
- UML Diagrams: a modeler agent can use the UML diagrams of the SUT to create test scripts. Such test scripts can be used for functional tests for instance. Especially if OCL (Object Constraint Language) is used in the UML diagrams to specify restrictions on concepts of the system, they can improve the performance.
- Session Data: session data can be used by a modeler agent to generate test scripts.
- Ontology: Ontologies can also be used as a source of information to generate test cases required for a test script. For instance, in [13] a knowledge base is used to generate appropriate data for automated form filling. Ontologies can be used in a similar way. Some ideas about ontology-based web application testing are presented in [20].
- Source code: in case that the source code of the system is available, it can be utilized to generate test scripts, for instance test scripts that cover all the execution paths. Techniques like Java annotations can be used to add useful metadata to the source code to ease such test script or test case generation [21].
- Database of the SUT: Although it is not a model of the system, but the database can contain useful information about the concepts and entities present in the SUT.
- Security: A modeler agent for security testing generates a model for the system from the perspective of evaluating its security. The result of this test provides some useful information about the degree of security of the system based on ASVS standard [22].

4. Implementation

A prototype of the proposed framework was implemented in Java. In this section some issues about the implementation of this prototype are briefly discussed, since a comprehensive discussion of the implementation details is beyond the scope of this paper. JADE[2] is used as the underlying infrastructure of the framework. It provides the essential services for developing a multi-agent system and hides many low level complexities and implementation details. TRE, TE, Dashboard, Result Analyzer, TSG, TCG and Modeler agents have been developed. A Recorder agent is developed which uses Selenium[3]. Selenium is an open source tool that provides the recording functionality through a plugin for Fire-fox browser.

In the prototype system, the format of the test scripts was chosen to be the same as that of Selenium. A Selenium test script is a simple XHTML file, containing a table. Each row of this table (except the first one) indicates one step of the test. Each step represents one action. The first row indicates the title of the test script (*i.e.* its name). Other rows have three columns. The first column includes the name of the action. Other two columns are used for the parameters of that action (e.g. the URL to be opened, the field name to be filled with the input value, the expected title).

Test scripts can be created manually or automatically by the framework. Since test scripts are simple text files, they later can be edited easily by human testers. In the current implementation of the framework, there are different possibilities for creating a test script: Using recorder agent, and using modeler agent. Different types of modeler agents are implemented: based on navigation model, based on session data, based on both the navigation mod-

[2]http://jade.tilab.com/
[3]http://seleniumhq.org/

el and the database of the SUT and also the notion of ontologies. In addition, another modeler agent is developed for web service testing.

A TCG agent is implemented which translates the test scripts into Java source code. The generated source code uses SeleniumHTMLUnit class (from the Selenium tool API) to simulate behavior of the browser. TCG compiles the generated Java class and sends the created .class file to the TRE. When a TE agent receives this compiled test code from the TRE agent. Then it creates a new object from the received .class file (using Java reflections). We call this object the test code object. Then is starts execution of the test by calling the action methods on the test code object. Each action method executes one of the test steps. Dashboard agent receives the test results from TE agents during the test execution and generates diagrams representing number of failed and passed action. A simple result analyzer agent is developed in the framework. Currently, in addition to computing the average number of failed steps among all executers, the result analyzer agent computes 'functional adequacy' and 'accuracy to expectation' defined based on ISO/IEC 9126 standard.

In order to perform security tests, a modeler agent was implemented that focuses on generating test scripts for security tests. This Security agent uses w3af[4], which is a Python-based tool. Based on the user configurations, Security Agent creates a simple test script. This test script is defined using a set of new actions we added to actions defined by Selenium. These actions are specific to w3af, it means that TCG agent translates theses actions to specific Java code which enables running w3af plugins from Java. Therefore, the generated test script, is finally translated to a piece of Java code that when run, calls w3af with appropriate parameters to perform the desired test. However, the details of this translation phase are beyond the scope of this paper, and will be discussed in another paper.

5. Evaluation

A comparison of the proposed framework with similar

Table 1. Comparison of different frameworks.

Framework	Supported Test Types	Test Strategy	Information Sources	Manual Intervention	Test Applicability Time	Target Type	Framework Architecture
[6]	Regression, unit, functionality	White-box (because fault injection is Performed on the code	User session information	Medium (manual fault injection)	Operational phase	Web applications	Non-Distributed
[9]	Functional	White-box (in presence of a trusted third party), gray-box	WSDL description or source code of the web service	Low (if test services are generated from WSDL descriptions), average (if test services are written manually)	After web services are published in the registry	Web services	Non-Distributed
[11]	Functional, unit, integration	White-box	Requirement documents and information about internal structure of the system	High (writing test procedures in TTCN-3 is manual, time-consuming and complex)	Whole lifecycle	Web applications	Non-Distributed
[13]	Security	Black-box	public web interface	Very low	Operational phase	Web applications	Non-Distributed
[15]	Data-flow	White-box	Source code	medium (manual test case generation)	Whole lifecycle	Web applications	Distributed
[16]	Functional, regression	white-box	source-code	medium (generating abstract test script and test cases)	Operational phase	Web applications	Non-Distributed
[17]	Functional, regression	Black-box	public web interface	low	Operational phase	Web applications	Non-Distributed
Proposed framework	Functional, security, load, stress, performance	All strategies (depending on the available sources)	Source code, ontology, UML models, session data, public web interface	low	Operational phase	Web applications (web services are potentially supported)	Distributed

[4]http://w3af.sourceforge.net/.

works discussed in this paper is presented in **Table 1**. This comparison is performed based on these factors:

- **Supported Test Types**: The more test types are supported by a test framework, the more powerful is that framework.
- **Test Strategy**: Generally there are three test strategies. Black-box testing imposes the least requirements for the test to be performed. It does not require the source code or internal information about the SUT. White-box strategy is on the other end. It requires that the source code of the system to be available. Gray-box strategy resides in the middle. It requires some information about the internal structure of the system or its details, for instance the database structure, but not the source code. A framework that is limited to white-box strategy has less applicability than one that uses black-box strategy, because it may not be possible to ask the providers of a system to make the source code of the system accessible in order to test the functionality of public interface of the system.
- **Information Sources**: This item indicates the types of information sources that are utilized by the framework to automate the test process. A framework that is able to use different sources (e.g. UML models, session information, source code…) is clearly more effective than a framework that works only in the presence of a single source.
- **Human Manual Intervention**: The less human intervention is needed in the execution of a test process, the more effective is the underlying framework.
- **Test Applicability Time**: In which phases of SDLC the framework can be used? Is the framework applicable only when the system is deployed or it can be used during the whole development cycle?
- **Framework Architecture**: As mentioned before, a distributed framework is more powerful and flexible in the testing web applications, because it copes better with the characteristics of these systems.
- **Target Type**: What type of systems can be tested using the framework? Does it support web services or only traditional web application?

Here, we concentrate on discussing the proposed framework with regards to these factors. The framework supports functional, load, stress, security, and performance tests. All of these tests are possible through appropriate test scripts. For instance if a test script for assessing SQL injection is available (for instance using the idea presented in [13]) then this test script can be used to perform a security test. Therefore, the main issue is how to represent the logic of a test in a test script. After such a test script is available, it is executable. Fortunately, all of the test types mentioned above, can be represented in a Selenium test script, because they do not need anything more than a sequence of HTML interactions with the SUT. SeleniumHTMLUnit API declares methods for handling dynamic behavior of web pages, but these methods are not yet completely implemented. Our point of view is that our framework will be capable of testing dynamic aspects of web pages (e.g. Ajax) if required such functionality is provided by Selenium.

Currently the framework is used for performing some load, stress, functional, performance, and security tests. If a valid test script representing the logic of the test is available, the test process can be performed automatically. Therefore, the main issue is the way a test script is generated. As mentioned before, the framework provides facilities for automatic test script generation based on the user session logs and navigation model of the SUT. It also provides semi-automatic test script generation using the recorder agent.

The framework supports all three test strategies. Based on the presence or absence of different information resources, different functionalities of the system might be available or unavailable. At least, the black-box strategy is available and the system requires no access to the internals of the SUT. But if some sources like user sessions or system models are available, the frame can well utilize them.

The manual intervention in the framework is at an acceptable low level. The framework provides automatic and semi-automatic facilities for creating test scripts. After a test script is created, it can be run automatically with little human intervention (e.g. specifying some parameters). Also as mentioned in section 3, the level of automation gained by the framework is much more in case of distributed tests.

The framework is useful in testing operational systems. Therefore, it does not support tests like unit test and integration test. Although some of these tests can be performed by functional tests.

The framework is a distributed one, consisting of different agents collaborating with each other.

As an example of how the framework possesses good flexibility, it is worth to mention an actual experience we made during the implementation of the prototype. First we had used TestGen4Web[5] tool as the recorder facility in the framework. Therefore, the format of the test scripts was based on what TestGen4Web uses for its test scripts, *i.e.* XML format. We had developed TCG agents for this kind of test script, and also we had developed the TE agents. After that, due to shortcoming we found in TestGen4Web, we decided to replace it with another tool, *i.e.* Selenium. This change would result in changing the format of test scripts (which are an important entity in the

[5]http://developer.spikesource.com/wiki/index.php/Projects:TestGen4Web

framework), but fortunately this change was easy to handle. The TCG and TSG agents needed to be modified, but the other agents like TRE, and TE agents did not. Therefore some functionality of the framework was modified with reasonable effort.

Another example that proves the flexibility of the framework is the capability of the system to test web services. In this case we used an open source product named soapUI[6]. This tool takes a WSDL file as input and generates a test script for testing the web service described by that WSDL file. Although this test script is different from the test scripts we had used in the system, but it has the same role and meaning. We created a special Modeler agent that takes a WSDL file as its information source and using soapUI API generates a test script. In fact, it is soapUI that generates a test script from the WSDL file. Then, the Modeler transforms this test script into the format of our test scripts. This test script is then used like simple test scripts in the system.

Adding this functionality (*i.e.* the capability of testing web services) to the framework required developing a new kind of Modeler. This Modeler was then attached to the framework with no extra complexity. Of course it must be noted that it also required some new actions to be added to the set of actions that were possible in definition of a test script. Therefore, it required to add some new functionality in the TCG agent to implement these new actions.

6. Conclusions

In this paper, a multi-agent framework was introduced for testing web-based systems. Different agents are designed with specific roles and they collaborate with each other to perform the test. The main design goals have been to develop an effective and flexible framework that supports different types of tests and utilize different sources of information about the system under test to automate the test process.

One of the novelties of this work is the use of test code which is based on the idea of mobile code. It provides benefits like increasing the performance, and decreasing the complexity of test executer agents. Another novelty of the work is the modeler agents that use different information sources for automatic test script generation. A prototype of the proposed framework has been implemented and is used to perform some experiments. The results are promising and verify the overall design of the framework.

7. Acknowledgements

This work has been supported by a grant by Iran's Telecommunication Research Center (ITRC), which is hereby acknowledged. We would like to thank members of the Web Technology Laboratory (WTLab) of the Ferdowsi University of Mashhad: Saeid Abrishami, Behshid Behkamal, Razieh Rezaee, Soheila Dehghanzadeh, Mahboubeh Dadkhah and Hamideh Hajiabadi.

[6]http://www.soapui.org/

REFERENCES

[1] A. G. Lucca and A. R. Fasolino, "Testing Web-Based Applications: The State of the Art and Future Trends," *Information and Software Technology*, Vol. 48, No. 12, 2006, pp. 1172-1186.

[2] A. G. Lucca and A. R. Fasolino, "Web Application Testing," *Web Engineering*, Springer, Berlin, Chapter 7, 2006, pp. 219-260. doi:10.1007/3-540-28218-1_7

[3] S. Murugesan, "Web Application Development: Challenges and the Role of Web Engineering," J. Karat and J. Vanderdonckt, Eds., *Web Engineering, Modelling and Implementing Web Applications*, Springer, Berlin, 2008, pp. 7-32.

[4] A. G. Lucca and M. Penta, "Considering Browser Interaction in Web Application Testing," *Proceedings of the* 5*th IEEE International Workshop on Web Site Evolution*, IEEE Computer Society Press, Los Alamitos, 2003, pp. 74-83.

[5] F. Ricca and P. Tonella, "Web Testing: A Roadmap for the Empirical Research," *Proceedings of the Seventh IEEE International Symposium on Web Site Evolution*, Budapest, 26 September 2005, pp. 63-70.

[6] S. Sampath, V. Mihaylov, A. Souter and L. Pollock, "Composing a Framework to Automate Testing of Operational Web-Based Software," 20*th IEEE Conference on Software Maintenance*, Chicago, 11-14 September 2004, pp. 104-113. doi:10.1109/ICSM.2004.1357795

[7] S. Elbaum, S. Karre and G. Rothermel, "Improving Web Application Testing with User Session Data," *Proceedings of the* 25*th International Conference on Software Engineering*, Portland, 3-10 May 2003, pp. 49-59. doi:10.1109/ICSE.2003.1201187

[8] S. Elbaum, G. Rothermel, S. Karre and M. Fisher, "Leveraging User-Session Data to Support Web Application Testing," *IEEE Transactions on Software Engineering*, Vol. 31, No. 3, 2005, pp. 187-202. doi:10.1109/TSE.2005.36

[9] H. Zhu, "A Framework for Service-Oriented Testing of Web Services," 30*th International Computer Software and Applications Conference*, Chicago, Vol. 2, 17-21 September 2006.

[10] P. Dhavachelvan, G. V. Uma and V. Venkatachalapathy, "A New Approach in Development of Distributed Framework for Automated Software Testing Using Agents," *Knowledge-Based Systems*, Vol. 19, No. 4, 2006, pp. 235-247. doi:10.1016/j.knosys.2005.12.002

[11] B. Stepien, L. Peyton and P. Xiong, "Framework Testing of Web Applications Using TTCN-3," *International Journal on Software Tools for Technology Transfer* (*STTT*), Vol. 10, No. 4, 2008, pp. 371-381.

doi:10.1007/s10009-008-0082-1

[12] ETSIES 203 873-1, "The Testing and Test Control Notation Version 3: TTCN-3 Core Notation," V3.2.1, February 2007.

[13] Y. Huang, C. Tsai, T. Lin, S. Huang, D. Lee and S. Kuo, "A Testing Framework for Web Application Security Assessment," *Computer Networks*, Vol. 48, No. 5, 2005, pp. 739-761. doi:10.1016/j.comnet.2005.01.003

[14] D. Kung, "An Agent-Based Framework for Testing Web Applications," *Proceedings of the* 1*st International Workshop on Quality Assurance and Testing Web-Based Applications*, Hong Kong, 28-30 September 2004, pp. 174-177.

[15] Y. Qi, D. Kung and E. Wong, "An Agent-Based Data-Flow Testing Approach for Web Applications," *Journal of Information and Software Technology*, Vol. 48, No. 12, 2006, pp. 1159-1171. doi:10.1016/j.infsof.2006.06.005

[16] J. Pava, C. Enoex and Y. Hernandez, "A Self-Configuring Test Harness for Web Applications," *Proceedings of the* 47*th Annual Southeast Regional Conference*, South Carolina, 2009, pp. 1-6. doi:10.1145/1566445.1566533

[17] H. Raffelt, T. Margaria, B. Steffen and M. Merten, "Hybrid Test of Web Applications with Webtest," *Proceedings of the* 2008 *Workshop on Testing, Analysis, and Verification of Web Services and Applications*, Seattle, 2008, pp. 1-7. doi:10.1145/1390832.1390833

[18] P. Tonella and F. Ricca, "A 2-Layer Model for the White-Box Testing of Web Applicaions," *Proceedings of the* 6*th IEEE International Workshop on the Web Site Evolution*, Chicago, 11 September 2004, pp. 11-19.

[19] P. Tonella and F. Ricca, "Statistical Testing of Web Applications," *Journal of Software Maintenance and Evolution*: *Research and Practices*, Vol. 16, No. 1-2, 2004, pp. 103-127. doi:10.1002/smr.284

[20] S. Paydar and M. Kahani, "Ontology-Based Web Application Testing," In: T. Sobh, K. Elleithy and A. Mahmood, Eds., *Novel Algorithms and Techniques in Telecommunications and Networking*, Springer, Berlin, 2010, pp. 23-27.

[21] W. C. Richardson, D. Avondolio, S. Schrager, M. W. Mitchell and J. Scanlon, "Professional Java JDK," 6th Edition, Wiley Publishing, Hoboken, 2007, p. 32.

[22] OWASP Application Security Verification Standard (ASVS), January 2011. URL http://www.owasp.org/index. php/ASVS

