A Framework of Model-Driven Web Application Testing

Nuo Li, Qin-qin Ma, Ji Wu, Mao-zhong Jin, Chao Liu
Software Engineering Institute, School of Computer Science and Engineering,
Beihang University, China
Seraphicln, maqinqinmail @hotmail.com wuji, jmz, liuchao@buaa.edu.cn

Abstract

Web applications have become complex and crucial in many fields. In order to assure their quality, a high demand for systematic methodologies of Web application testing is emerging.

In this paper, a methodology of Model-Driven Testing (MDT) for Web application is presented. Model is the core of this method. Web application model is built to describe the system under testing. Test case models are generated based-on the web application model. Test deployment model and test control model are designed to describe the environment and process of test execution. The test engine executes test cases based-on the test deployment and control model automatically. After that, testing results are reflected to test case models.

A framework is designed for supporting this methodology. In order to get better extensibility and flexibility, it is loose-coupled by a modeler and a tester. The modeler enables developers to design meta-models, and is responsible for creating, visualizing and saving models. The tester takes in charge of recovering the tested Web application model, generating test cases, and executing test.

1. Introduction

Web Applications (WA) have become the core business for many companies in major market areas. Correspondingly, the quality of WA is becoming crucial. Therefore, more than 300 commercial or open source WA testing tools are developed. They focus on different aspects of WA respectively [1]. But, because WAs are complex, distributed and rapidly updated software systems, the testing of them is still an arduous task, and traditional testing methodologies may not be adequate.

MDT is a promising approach for the visualization and automation of software testing. It offers a suite of visualized means to define, execute and analyze testing, and reduces the testing time by reusing common test functions. Furthermore, it separates the testing logic from the test implementation. This enables developers to focus on designing good tests specific to applications while relying on the toolset's test execution environment to solve problems related test execution. [2]

In this paper, a framework, named MDWATP, is proposed for applying MDT process to WAs. After providing a brief review of existing work in model-driven WA testing in section 2, section 3 defines our test models around different views of MDT. And then, section 4 shows details of the MDWATP framework. Lastly, section 5 summarizes and discusses future work.

2. Related work

Many researchers study how to assure the quality of WAs by model based methods. The major of them focus on development modeling [3,4,5], while some others are studying test models especially [6,7,9]. Test model and development model are different, but they

associate with each other. Since they orient the same codes, sometimes test models could be translated from development models. However, because WAs are usually directly coded and no, or poor, documentation is produced during development, it is better to build test models for WAs by reverse engineering.

Di Lucca et al. [6,8] exploits an UML model of a WA as a test model, and proposes a definition of the unit level for testing the WA. Based on the model, a method to test the single units of a WA and for the integration testing is proposed.

Ricca and Tonella [9] also propose a UML model of WAs for their high level representation. Such a model drives WA testing, in which it can be exploited to define white box testing criteria. Compared with the model defined by Di Lucca et al., their model aims at explicitly representing user navigations.

Besides UML models, there are other models being adopted to describe the tested WA. Liu et al. [7] propose an object-oriented WA testing model. The test model captures both structural and behavioral test artifacts of WA and represents the artifacts form the object, behavior, and structure perspectives.

Most of the previous studies focused on the definition of WA test model and the strategy of test case generation [6,7,9]. Although some of them offered a framework to create models and execute tests, it is not enough to arrange testing on demand. The testing of WAs usually is distributed. Therefore test environment and execution process should be modeled.

In order to deal with this problem, MDT proposes a methodology to support modeling the process of testing, and automatically create test execution environment. And the UML2 Testing Profile (U2TP) is proposed as a language to design, visualize, specify, analyze, construct and document the artifacts of test systems [10]. The Eclipse Test and Performance Tools Platform (TPTP) [11] contains an open-source implementation of U2TP. And the IBM Haifa Research lab developed a MDT package which supports the behavioral modeling of distributed applications, test generation, test execution, and test analysis [14]. But both TPTP and Haifa's MDT package do not focus on WAs testing.

3. Model-Driven Web Application Testing

Test model is the core conception of MDT; it appears as different views in different phases of testing (see figure 1). Firstly, the System Under Test View (*SUT View*) presents the model of the system being tested. Based-on the *SUT View*, test cases are automatically or semi-automatically generated. The generated test cases are described in the *Test Case View*. After that, the process and environment of test execution are modeled in the *Test Execution View*. Then the MDT execution engine automatically executes test cases based-on models described in the *Test Execution View*. Finally, test results are represented in the *Test Result View*. In addition to generate report forms of test results, the system can modify models in previous phases for visual presentation of test results. The iterative process is a great help for regression testing.

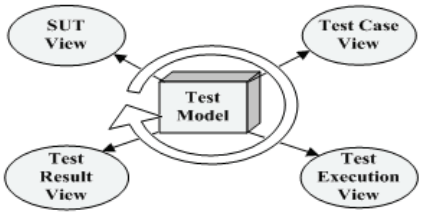

Figure 1. Different views of MDT

By referring to previous studies about WA models, we propose a lightweight model of tested WA. According to the MDT different views, test case model, test deployment model, test control model and test result model are also defined. All of these models extend UML2.0 based-on Meta-Object Facility (MOF) [12] mechanism. This section introduces the definition of these models; the implementations of them are depicted in section 4.

3.1 System under test view

Navigation is a basic characteristic of WA; WAs provide users services and obtain user input through navigation and other HTML components. We also noticed that large numbers of HTML pages are dynamically generated in the server side. Because of the flexibility of programming, it is hard to predict what the result will be before program execution. So

we adopt navigation model as test model of WAs, and attempt to get it from client side by analyzing HTML pages.

By referring to the UML model proposed by Di Lucca et al. [6,8] and Ricca and Tonella [9], Class diagram of UML is extended to define our Web Application Navigation Model (WANM). WANM depicts the navigation relations among the client pages, and hyperlinks and forms in each client page. HTML elements are modeled as classes generalizing the *Class* class from UML2. And relations between the elements generalize the *Association* class from UML2. [13]

Compared with the other WA test models, WANM is easier to be implemented in projects, satisfies the requirements of building WA test model from client side, and contains relative information of functional testing.

3.2 Test case view

Test case model extends UML sequence diagram. Each test case is a sequence of client pages to be accessed. A client page to be accessed is an instance of a client page class in WANM with an arbiter, where the expected contents for accessing the next page are stored.

In order to satisfy the continuously changing needs and requirements deriving from the evolving application domain， existing WAs are modified frequently and quickly [8]. As a result, test data for WA should be general, adaptive and reusable. In this situation, data pool is very appropriate for WA testing. A data pool is a collection of data partitions or explicit values that are used by test engine; it provides a means for providing values or data partitions for repeated tests [10]. The forms contained in the client page to be accessed could be assigned by the data selected from data pool.

3.3 Test execution view

In the test execution view, two kinds of models are defined: the test deployment model and the test control model. Because test cases of WAs usually need to be run on several client computers, test deployment model is required to describe the test environment. And the test control model is used to plan test execution process.

The test deployment model extends the deployment diagram of UML2. It has three basic elements: *test center*, *test node*, and *test script*. The two previous elements generalize the *Node* class from UML2; the *test script* generalizes the *Artifact* class from UML2. The *test center* stands for the computer which controls *test nodes*, and collects test results from them. A *test node* is a computer that actually executes test. *Test scripts* would be automatically generated from test cases, and deployed on these *test nodes*.

The test control model extends the UML activity diagram. The *TestInvocation* class is defined by generalizing the *Action* class from UML2. It associates with *test node*. The relation between *TestInvocation* objects controls the execution order of the *test nodes*.

In order to organize and manage the test deployment model and the test control model, we defined the test architecture model based-on U2TP. The test architecture model is not shown to users. The *Test context*, which is a collection of test cases together with a test configuration on the basis of which the test cases are executed [10], is defined in this model.

3.4 Test result view

Test result model is defined to save and present test results. And besides being shown in reports, test results would be associated with test case models.

Test results are classified as test trace result and test verdict result. The test trace result is a behavior description of execution. Each test trace result records execution time and error or exception information during the execution. The test verdict result is the assessment of the correctness of system under test. The test verdict result of a test case is calculated by comparing with the excepted test results.

4. The framework of MDWATP

MDWATP offers a framework to support the

model-driven WA testing defined in section 3. It provides two ways to build the model of the tested WAs: forward and reverse modeling.

Figure 2 shows MDWATP's architecture. It is developed as Eclipse Plug-in. The *Modeler* takes in charge of customizing, building, visualizing and saving test models. The *Tester* is responsible for recovering the WANM, generating test cases and executing test.

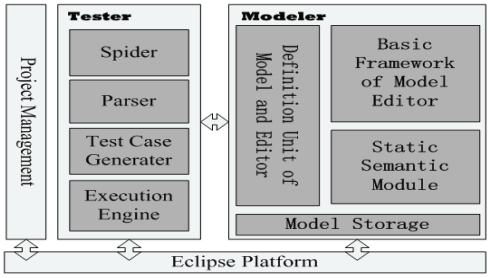

Figure 2. Architecture of MDWATP

The *Modeler* and the *Tester* are separated to make MDWATP more extensible and flexible. The *Modeler* is an independent Eclipse plug-in which implements two kinds of extension mechanisms of UML2: MOF and Profile. Therefore the *Modeler* not only implements meta-models defined in section 3, but also can specify any other models extended from UML2. And for the other test models, the *Tester* in MDWATP could be replaced independently.

4.1 Modeler

The *Basic Framework of Model Editor* is the core of the *Modeler*. It implements common functions of the *Modeler*, including interpretation and processing of user input, configuration of interface, connection with the *Static Semantic Module* and definition and checking of abstract syntactic, edit operations and geometrical relations.

The *Static Semantic Module* contains the definition of meta-models. In MDWATP, all the meta-models defined in section 3 are implemented in this module.

The *Definition Unit of Model and Editor* consists of a meta-graph library, a model editor definition file and an interface description file. The interface description file defines the main interface of the model editor. When a model editor is running, this information would be loaded to create the corresponding model editor. After the model editor is loaded, all of the predefined models in the *Static Semantic Module* could be created and edited.

The *Model Storage* is responsible to save all of the generated models in database.

4.2 Tester

The *Tester* semi-automatically recovers the WANM, generates test cases, and automatically executes testing.

The *Spider* and the *Parser* in the *Tester* abstract navigation and form information from HTML pages. Because some client pages are dynamically created on server based-on user input, the WANM couldn't be adequately retrieved only by the *Spider* and the *Parser*. Therefore, MDWATP offers two means to extend the WANM. Firstly, HTML elements defined in the WANM could be manually added and edited. Since developers know well of the result page that will be generated after a form is submitted, they can extend the model directly. Secondly, user could input data for the form in client page and let the *Tester* extends the WANM automatically base-on the input data. The generated models are visualized and saved by the *Modeler*.

Figure 3 demonstrates an example of WANM. The WA under testing is a digital library. The first two columns of client pages and forms contained in them are recovered by the *Tester*. The filled square nodes stand for server pages. The client pages *P, A, Welcome, History* and *Error* are directly added into this model by user through the *Modeler*. The two *Search Results* client pages are generated automatically after the *Check Form* is input and submitted to server.

The *Test Case Generator* in the *Tester* generates test cases based-on the WANM. Paths in WANM could be covered to generate test cases. And the *Test Case Generator* supports to record the user's clicking action on the WANM. The test data needed for test case could be selected from data pool. And the expected result is edited in an arbiter view.

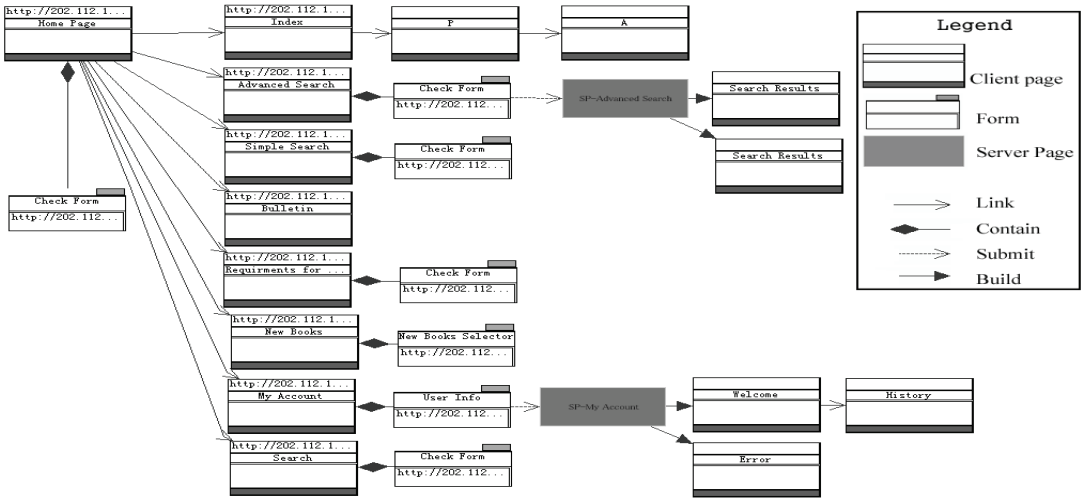

Figure 3. Demonstration of a WANM

Figure 4 depicts a test case model generated by recording the sequence of user clicks on the WANM shown in figure 3. Firstly, the *Home Page* was accessed. Then the hyperlink of *My Account* on the *Home Page* was clicked. After the *User Info* form was inputted by the data selected from its data pool and submitted to server, *Error* page was returned. Finally, the *History page* was accessed by inputting its URL in a browser. The data in the first square brackets on each message describes the mode of page transfer. And in the second square brackets, the data briefly shows the expected results after the action is executed. For example, *"Please log in…"* would be presented in the *History page,* if it was accessed before the correct user name and password were input.

Figure 4. Demonstration of a test case model

Test scripts would be automatically generated from test cases, and executed by the *Execution Engine* based-on the test deployment model and the test control model. Finally, all test results distributed in different computers are collected, and shown by a test report while appended to test case models. Figure 5 and figure 6 show examples of a test deployment model and a test control model designed in a deployment model editor and a control model editor respectively.

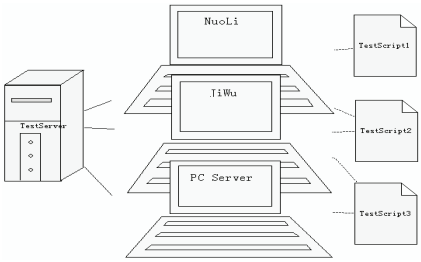

Figure 5. Example of a test deployment model

The test deployment model depicted in figure 5 contains a test center (*TestServer*), three test nodes (*NuoLi, JiWu* and *PC Server*) and three test scripts. *TestServer* controls the execution process and collects test results.

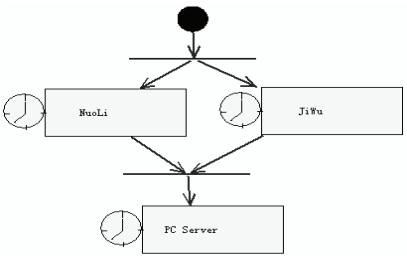

Figure 6. Demonstration of a test control model

Figure 6 shows the execution flow of the three test nodes. The test scripts deployed on *NuoLi* and *JiWu* would be parallel executed. Then the *PC Server* would be invocated when both of the execution on *NuoLi* and *JiWu* finished.

Figure 7 shows the execution result of the test case shown in figure 4. The results are also modeled and take similar form of the original test case. The rectangle on each lifeline displays the test results of

each step in the test case. And the last one is the final test result of the test case.

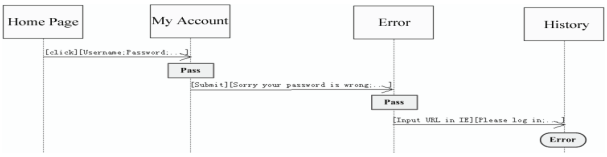

Figure 7. Demonstration of a test result model

5. Conclusions

A serial of models for MDT of WA is defined and a framework supporting them is presented in this paper. The framework supports building tested WA model and test case model by a semi-automatic procedure, enables testers to design test deployment model and test control model, and implements distributed test execution. It is designed as Eclipse plug-in, and separated into a *tester* and a *modeler*. The *tester* and the *modeler* are independently. The *modeler* not only supports to build the test models defined in this paper, but also could specify any other models extended from UML2.

Compared with the TPTP and the MDT package of Haifa, MDWATP focuses on WA's testing, allows user to define meta-models, and offers a more friend user interface. MDWATP supports the transition of models. If the WANM changes, the others will automatically updated.

Since MDWATP is still under development, more work is needed to enable testing majority of real WAs. The automatic generation of test data is still a difficult problem. The log files in server may be utilized to help the generation of test data. Furthermore, rules can be concluded to help user to modify model for test cases generation.

References:

[1] R.Hower, Web Site Test Tools and Site Management Tools, http://www.softwareqatest.com/qatweb1.html. Last modified in January 28, 2006./Referenced in February 21, 2006.

[2] Model Driven Testing Tools Whitepaper. IBM Haifa Research Laboratory. 2003-07-31 http://www.haifa.ibm.com /projects/verification/mdt/papers/MDTWhitePaper.pdf

[3] J.Conallen, "Modeling Web Application Architectures with UML", *Communications of the ACM*, Vol.42, No.10, 1999, pp. 63-70.

[4] L. Baresi, F. Garzotto, and P. Paolini, "Extending UML for Modeling Web Applications". *Proceedings of the 34th Annual Hawaii International Conference on System Sciences.* IEEE Computer Society, Maui, Hawaii, 2001.

[5] J. Li, J. Chen, and P. Chen. "Modeling Web Application Architecture with UML", *Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems*, IEEE Computer Society, Xi'an, 2000, pp. 265-274.

[6] G. A. Di Lucca, A. R. Fasolino, F. Faralli, and U. De Carlini. "Testing Web Applications", *Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002), Maintaining Distributed Heterogeneous Systems*, IEEE Computer Society, Montreal, 2002, pp. 310-319.

[7] D. C. Kung, C. H. Liu, P. Hsia, "An object-oriented web test model for testing Web applications", *Proceedings of the 1st Asia-Pacific Conference on Quality Software*, IEEE Computer Society, Hong Kong, 2000, pp. 111-120.

[8] G. A. Di Lucca, A. R. Fasolino, F. Faralli, and U. De Carlini, "WARE: a tool for the Reverse Engineering of Web Applications", *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering,* IEEE Computer Society, Budapest, 2002, pp. 241-250.

[9] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications", *Proceedings of the 23rd IEEE International Conference on Software Engineering*, IEEE Computer Society, Toronto, Ontario, 2001, pp.25-34.

[10] UML 2.0 Testing Profile Specification. OMG Adopted Specification. 2003-08-03.

[11] Eclipse Test & Performance Tools Platform Project. http://www.eclipse.org/tptp/. Last modified in September 15, 2005/ Referenced in February 21, 2006.

[12] Meta Object Facility (MOF) 2.0 Core Specification. OMG Adopted Specification. 2003-10-04

[13] N. Li, J. Wu, M.Z. Jin, and C. Liu. "The Design of a Web Application Testing Model", *Acta Electronica Sinica*, l33 (12A), 2005, pp. 2376-2380.

[14] A. Hartman and K. Nagin, "The AGEDIS Tools for Model Based Testing", *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis,* ACM 2004, Boston, 2004, pp. 129-132.