A Meta-Model to Support Regression Testing of Web Applications

Yanelis Hernandez, Tariq M. King, Jairo Pava and Peter J. Clarke
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
email: {yhern004, tking003, jpava001, clarkep}@cis.fiu.edu

Abstract

As businesses strive to keep pace with the rapid evolution of web technologies, their efforts to maintain automated regression testing strategies are being hindered. Technological migration of a web application can lead to test scripts becoming incapable of validating the migrated application due to differences in the testing platform. Regression tests that are still applicable to the application would therefore have to be re-written to be compatible with the new technologies.

In this paper, we apply a model-driven approach to the development of automated testing scripts for validating web applications from the client-side. We define a meta-model using UML 2.0 profiles, and describe the model transformations needed to automatically port regression tests to various platforms. A prototype of the test implementation for an e-commerce application is also presented.

Keywords: Regression Testing, MDSD, UML

1 Introduction

The many advances in web technologies has led to the development of web applications that compete in solution areas that traditional software previously addressed. Web applications are no longer simple streams of static web pages, but instead provide a collection of interactive services with the added flexibility, mobility, and connectivity of the Internet. These characteristics have made web-based solutions highly attractive to businesses, and this has led to the creation of many development and testing tools to support web programming [5, 13, 17, 18, 19]. However, a negative consequence of these advancements is the persistent growth in the complexity of web applications, and the rapid evolution of their supporting technologies.

Software testing is a very costly and time-consuming endeavor. Some studies indicate that the cost of software testing may account for between fifty to seventy-five percent of total development costs [7, 8]. In addition, testing costs tend to exceed those of design and implementation, and therefore the methodologies and tools employed at these stages are pertinent to the development of affordable quality software.

Automation is an effective way to reduce time and costs of software testing, and so many businesses conduct their testing process with some degree of automation. The level of automation of software testing typically exists at the test script level. Software testers encode a set of test cases for the application in a scripting language, and use the script as input to an automated testing tool which executes the tests. If subsequent changes are made to the system, the test script provides a means for automatically performing regression testing to determine whether new errors were introduced into previously tested components [6].

Script-level test automation becomes problematic when an application migrates to include technologies that are not supported by the testing tool currently being utilized. Regression tests that are still applicable to the migrated application therefore have to be re-written in the scripting language of a new testing tool, thereby defeating the purpose of test automation.

The model-driven software development (MDSD) paradigm emphasizes the use of models and model transformations to generate executable code for a specific platform. In this paper, we apply MDSD to the generation of an automated testing script for validating the client-side of a web application. To address the aforementioned problem of script-level automation, we propose that the test set for the web application be designed as a platform independent model which can be automatically transformed into a platform specific automated testing script.

The main contributions of this work are that it: (1)

presents a model-driven approach to the design and development of automated testing scripts to validate a web application; (2) provides meta-models for a subset of web-based development and testing technologies using UML 2.0 [12] profiles; and (3) elaborates on a case study developed using the proposed modeling approach to support testing.

This paper is organized as follows: the next section contains background information on web-based technologies, regression testing, and meta-modeling. Section 3 presents the proposed approach to support testing a migrated web application. Section 4 contains the meta-models used in our approach and describes the generation of the testing script. Section 5 provides the details of the case study. Section 6 presents related work, and in Section 7 we give concluding remarks and discuss future work.

2 Background

In this section we provide background information on the technologies commonly used to develop web applications. We then discuss the technique of regression testing, including tool support for automatically validating web applications. Approaches to meta-modeling are also described in this section.

2.1 Web-Based Technologies

There are two broad categories of web programming technologies used to develop web applications – *client-side* and *server-side* [4]. Client-side scripting technologies involve the use of a web-browser on the client machine to perform operations without having to communicate with the server. This type of scripting is generally used to dynamically modify the behaviors within a specific web page in response to user input [4]. Popular examples of client-side scripting technologies include [17, 19]: HTML and Javascript.

In contrast, server-side technologies perform operations on the web server instead of on the client machine. They are preferable when operations utilize information that is not available on the client, or when data storage from the client to the server is needed [4]. Dynamic operations on the server-side may involve changing the web page supplied to the client, or providing a new sequence of web pages to the browser. Active Server Pages (ASP) [4] and Hypertext Preprocessor (PHP) [18] are two examples of server-side scripting languages commonly used to develop web applications.

Many web technologies can be integrated with others and hence web applications usually employ a myriad of technologies on both the client and the server. In essence, the classification of a scripting language depends on its implementation within the web application. For example, Flash [1] technologies may be implemented on the client using companion technologies such as HTML [19], or on the server by providing synchronized updates to the client.

2.2 Regression Testing

Software testing is the process of operating a software system under specified conditions, recording the results, and making an evaluation of some aspect of the software [10]. Testing is particularly useful for validating changes made to a system during software maintenance or evolution. Regression testing refers to rerunning test cases to determine whether or not new errors have been introduced into previously tested code [3]. In an effort to reduce costs, many testing strategies employ automated tools to support the performance of regression tests on modified software systems.

There has been a rapid growth of tools to support testing web applications on both the client and server sides. HTMLUnit [5] simulates the behavior of a web browser by providing an API to interact with web pages. Functional testing tools such as TestSmith [13] provide facilities for simulating mouse and keyboard events and hence can be used on the client-side to test Flash applications. On the server-side, PHPUnit [14], a member of the xUnit family of testing frameworks, is a unit testing solution for PHP [18].

2.3 Meta-Modeling

Model-driven software development (MDSD) focuses on the combined use of software models and associated transformations to build complete software systems. This typically involves the use of a source model or Platform Independent Model (PIM), and a target model or Platform Specific Model (PSM) [16]. The PIM does not rely on any specific technological platform that could be used to implement the software, and therefore represents the essence of the solution. A model transformation language can then be used to transform the PIM into a PSM that is executable on the target platform [16].

A technique known as *meta-modeling* is used to ensure the consistency of models during transformation. This involves defining the abstract syntax of models and the interrelationships between the model elements [15]. Meta-modeling should consist of orthogonal dimensions that support two forms of instantiation [2]: *linguistic* – relates to the language definition, and *ontological* – relates to the domain definition. In this paper

the ontological meta-modeling will be implemented using UML 2.0 [12] profiles.

3 MDSD Approach to Support Testing

In this section we define the scope of the problem being addressed in this paper. We then present our approach which applies MDSD to the generation of an automated testing script for validating a web application.

3.1 Problem Definition

Automating the process of testing a web application involves developing a script that can be recognized by a testing tool, which then applies predefined test cases to the application under test. This is depicted in the left-hand portion of Figure 1, where a test script `TS1` validates an application under test `AUT1`; both of which can be thought of as targeting the same set of web technologies `WT1`.

The migration from `AUT1` to `AUT2` in Figure 1 represents when a business updates their application to include a new set of web technologies, labeled `WT2`. However, this migration usually leads to the test script `TS1` becoming incapable of validating the application `AUT2`. Therefore, regression tests that are still applicable to `AUT2` would have to be re-written in a new test script `TS2` to be compatible with the testing tool for `WT2`.

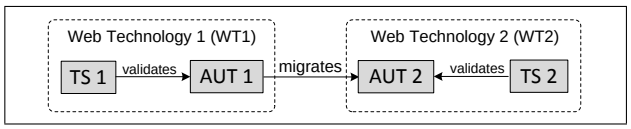

Figure 1. Testing a migrated web application.

3.2 Model-Driven Solution

Our approach harnesses the power of MDSD to automatically generate the testing script for a web application that has migrated to new technologies. Figure 2 shows the models and transformation processes used in our approach. A key aspect of our methodology is the use of a test script generator, shown at the center of Figure 2, to produce platform specific tests for the migrant web applications. Inputs to the generator are represented by dotted lines, while solid lines are used to represent the output.

First, a platform independent model, representing the essence of the test set for the web application, is input to the generator. This PIM is then combined with a model of the constructs used for testing a particular set of web technologies. For example, in Figure 2, `PI Test Set` would be combined with `WT Model 1`, which models constructs for testing HTML and Javascript, to automatically generate an HTMLUnit script `PS Test Set 1`. If the application later migrates to a new set of web technologies (e.g., Flash and Actionscript), the test set can also migrate automatically by combining `PI Test Set` and `WT Model 2`.

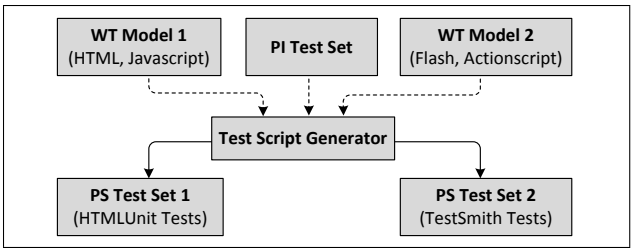

Figure 2. Model-driven test script generation.

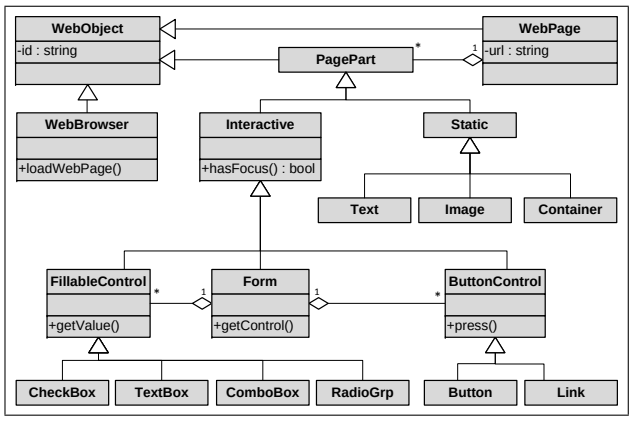

Figure 3. Conceptual model of a web interface.

4 Meta-Models

In this section we provide a conceptual model depicting abstractions for a subset of the visual elements of a web application. We then present a UML 2.0 profile of a meta-model to support testing web applications based on the conceptual model.

4.1 Conceptual Model of a Web Interface

Figure 3 shows a conceptual model for the user interface of a web application. The purpose of the model is to provide abstractions for the visual elements of the web application that are relevant to testing. At the top of the hierarchy of object types is the `WebObject` (top-left of Figure 3), which is a general representation for any element of the web interface. These object types include web pages and the elements contained within them, which are classified as follows: (1) *interactive* – allows for dynamic user interaction, e.g., forms,

Stereotype	Base Class	Tagged Value	Constraints
<<TestSet>>	Class	id: String	May only declare instances of classes stereotyped TestCase. id is unique.
<<TestCase>>	Class	id: String	May only declare instances of classes stereotyped TestSection, and be associated with at most one instance each of classes stereotyped Setup, Precondition, Input, Postcondition, and Rollback. id is unique.
<<TestSection>>	Class	id: String	id is unique.
<<TestCommand>>	Class	id: String	May be associated with at most one instance of a class stereotyped TestSubject. id is unique.
<<WebObject>>	Class	id: String	id is unique.
<<TestSubject>>	WebObject		
<<Setup>>	TestSection		May only declare instances of classes stereotyped CreateCommand.
<<Precondition>>	TestSection		May only declare instances of classes stereotyped InputCommand.
<<Input>>	TestSection		May only declare instances of classes stereotyped InputCommand.
<<Postcondition>>	TestSection		May only declare instances of classes stereotyped AssertCommand.
<<Rollback>>	TestSection		May only declare instances of classes stereotyped DestroyCommand.
<<CreateCommand>>	TestCommand		
<<InputCommand>>	TestCommand		
<<AssertCommand>>	TestCommand		
<<DestroyCommand>>	TestCommand		
<<contains>>	Association		Connects instances of TestSet with instances of TestCase, and instances of TestCase with instances of TestSection
<<manipulates>>	Association		Connects instances of Command with instances of TestSubject.

Table 1. UML profile of the test model for a web application.

textboxes, buttons; and (2) *static* – remains fixed regardless of external stimuli, e.g., text, images, tables.

An interesting aspect of the model is that the type WebBrowser is also derived from WebObject. This is because the browser allows users to break the normal flow of control of the application, and testing should address such scenarios. For example, a user may press the Back button of the browser during the execution of the application causing unexpected results [20]. In addition, the WebBrowser type facilitates changes to the browser configuration, and allows testing to simulate the use of a specific browser; both of which can affect the behavior of the web application.

4.2 Meta-Model to Support Testing

The UML 2.0 profile for the test model of a web application is shown in Table 1. It consists of four kinds of artifacts indicated by the column headings (from left to right): (1) stereotype – represents specific meta-classes; (2) base class – denotes an extension relationship from a UML meta-class or inheritance from another stereotype; (3) tagged value – defines attributes of the stereotype; and (4) constraints – enforce restrictions on how the meta-model may be used. For example, in Row 1 of Table 1 the stereotype TestSet extends of the UML meta-class named Class; contains the tagged value id of type String; and may only contain variables of a class whose stereotype is TestCase. The constraints for this table entry also specify that the tagged value id should only hold unique values. It should be noted that the stereotype TestSubject in Row 5 extends WebObject, which is the base class from the conceptual model of a web interface presented in Subsection 4.1.

5 Case Study

In this section we present a case study developed as an initial proof of concept realization of our methodology. We first outline the features of the application, and describe the technologies and test support tools required to setup the experiment. We then provide details on a test set implementation that uses the proposed approach, including the generation of test scripts. The findings and limitations of the study are also discussed in this section.

5.1 FastBooks Application

FastBooks is a small e-commerce application for purchasing college textbooks on-line. Users may choose to purchase their textbooks in three different formats (Print, Audio, or Electronic), and then submit their billing and shipping information for validation. Two versions of the FastBooks application were developed to set up the scenario of a business migrating from one web platform to another that uses technologies unsupported by the current testing tool.

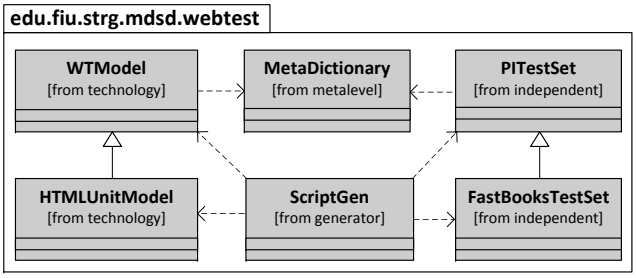

Figure 4. Minimal class diagram of the prototype.

The first version of the application was developed using HTML 4.01 and Javascript on the client-side, and implemented its automated testing using HTMLUnit 1.4. The second version of the application was developed using Flash 9 and Actionscript 3.0 on the client-side. Both versions used PHP 5.25 on the server-side for transferring data to and from persistent storage, while Apache HTTP Server 2.2 provided the web server functionality.

5.2 Test Implementation

We developed a prototype in Java 5.0 to implement the model-driven testing solution presented in SubSection 3.2. First, test cases for the FastBooks application were designed using boundary value analysis and equivalence partitioning techniques. The initial test set, consisting of 12 test cases, was then encoded using the constructs and rules of the testing meta-model proposed in this paper.

The package labeled `edu.fiu.strg.mdsd.webtest` in Figure 4 shows the main communicating classes from various sub-packages of the prototype. The types from the conceptual model of a web interface were stored in the class `MetaDictionary`, which was used to design generalized classes for modeling the web testing technology `WTModel` and platform independent test set `PITestSet`. These two classes were then specialized to create objects for holding the HTML and Javascript testing constructs, as well as the test cases designed for the FastBooks application; represented by the classes `HTMLUnitModel` and `FastBooksTestSet` respectively.

5.3 Generation of Test Scripts

The class `ScriptGen` in Figure 4 is responsible for iterating through `FastBooksTestSet`, and generating a test script using the platform specific constructs in `HTMLUnitModel`. This is achieved by retrieving the test case definitions that are represented as abstract test commands, along with variable names and their associated values. These abstract test commands are then mapped to the HTMLUnit constructs that also contain placeholders for the variable names and values. The generator then overwrites the placeholders with the actual variable names and values stored in `FastBooksTestSet`, and appends the completed instruction to the output file `FastBooks.htmlUnit`.

5.4 Discussion

The purpose of the case study was to demonstrate the feasibility of applying a model-driven approach to the design and development of testing scripts for a web application. Implementing the prototype gives credence to the claim that the strategy can be used to define platform independent tests, and convert them into scripts for an automated testing tool. All of the base test cases developed for the case study were successfully transformed into syntactically correct HTMLUnit tests. This suggests that the abstract constructs used in the current prototype were therefore sufficient for representing the platform specific constructs required to validate the FastBooks application. Although the current version of the prototype does not implement the technology model for Flash, this could be easily incorporated by extending the generalized classes provided by the infrastructure.

Conducting the study also provided us with insight into the intricacies of developing a framework for the proposed approach. Although the FastBooks test set only required the prototype to address a limited number of test scenarios, designing the framework to maintain independence among the test model, technology model, and script generator was very challenging. Limitations of the current prototype include coverage of only a subset of web controls and widgets, and manual detection of model constraint violations. However, the latter could be solved through the use of model-driven architecture tools such as the Eclipse Modeling Framework.

6 Related Work

There has been great effort in the research community to assure the quality of web applications through effective testing methodologies. However, most of the work that focuses on the use of models for testing web applications relies heavily on specific platforms for the creation of their models. In contrast, our approach uses a platform independent test model to facilitate the generation of test scripts.

The work presented by Li et al. [11] is most closely related to our work. It proposes a model-driven testing methodology for web applications. A model of the

web application is built to describe the system under test, and test cases are developed based on that model. Test scripts are generated from the test cases and executed by a test engine. Our approach differs from [11] in that we consider the negative impact that migration has on the ability to automatically validate web applications. Therefore, the technique proposed in this paper for modeling the technological constructs could be used in [11] to provide a more extensible solution.

The Object Management Group (OMG) [12] extended UML with testing concepts such as test architecture, test data, and test behavior. Similar to the profile presented in this paper, the UML Testing Profile 1.0 is based on the UML 2.0 specification and provides a modeling language that can be used to design, visualize, and specify the artifacts of a test system. The meta-models presented by [12] encapsulate a broad view of testing as a process. However, in this paper we focus on defining a profile for the structure of a test set for the client-side of a web application, and leverage the resulting models to generate test scripts.

Heckel et al. [9] present an approach for testing web applications designed with a model-driven approach. Design patterns such as Bridge and Proxy are used to execute the same test cases in a local and distributed testing environment, respectively. Their methodology takes advantage of the separation of PIMs and PSMs on both the model-level and implementation-level. Although the scope of our work does not include test execution, the strategies in [9] could be used to design a test harness for scripts generated by our approach.

7 Concluding Remarks

In this paper we presented a model-driven technique for designing platform independent tests for validating web applications. Our approach leverages these test case models for the generation of automated test scripts, thereby addressing the problems associated with technological migration of the web application. An e-commerce application was used as the basis of our study and a prototype was implemented.

Future work calls for deeper investigation into the problem by: (1) extending the prototype to include the technology model for Flash and Actionscript; (2) formulating additional test scenarios for the application used in the case study, and (3) developing a platform independent model for tests that target server-side scripting languages.

References

[1] Adobe Systems Inc. Flash 9, April 2007. `http://www.adobe.com/products/flash/` (Mar. 2008).

[2] C. Atkinson and T. Kühne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20(5):36–41, 2003.

[3] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, second edition, 1990.

[4] D. Buser, C. Ullman, J. Duckett, J. Kauffman, J. T. Llibre, and D. Sussman. *Beginning Active Server Pages 3.0*. Wrox Press Ltd., Birmingham, UK, UK, 2000.

[5] Gargoyle Software Inc. HTMLUnit 1.14, Jan 2008. `http://htmlunit.sourceforge.net/` (Mar. 2008).

[6] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM Trans. Softw. Eng. Methodol.*, 10(2):184–208, 2001.

[7] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

[8] M. J. Harrold. Testing: a roadmap. In *ICSE - Future of SE Track*, pages 61–72, 2000.

[9] R. Heckel and M. Lohmann. Towards model-driven testing. *Electr. Notes Theor. Comput. Sci.*, 82(6), 2003.

[10] IEEE Computer Society. Std 610.12-1990(r2002): Standard glossary of software engineering terminology. Technical report, 2002.

[11] N. Li, Q. qin Ma, J. Wu, M. zhong Jin, and C. Liu. A framework of model-driven web application testing. In *COMPSAC (2)*, pages 157–162, 2006.

[12] Object Management Group. Unified modeling language. `http://www.uml.org/` (Mar. 2008).

[13] Quality Forge. TestSmith 1.40, Sept 2005. `http://agilethinking.net/qualityforge/testsmith/` (Mar. 2008).

[14] Sebastian Bergmann. PHPUnit 3.2.15, Feb. 2008. `http://www.phpunit.de/` (Mar. 2008).

[15] S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, 2003.

[16] T. Stahl, M. Voelter, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[17] Sun Microsystems, Inc. JavaScript, December 1995. `http://java.sun.com/javascript/` (Mar. 2008).

[18] The PHP Group. PHP 5, Nov. 2007. `http://www.php.net/` (Mar. 2008).

[19] W3C. HyperText Markup Language 4, December 1999. `http://www.w3.org/TR/REC-html40/` (Mar. 2008).

[20] Y. Wu and J. Offutt. Modeling and testing web applications. Technical report, GMU ISE Technical ISE-TR-02-08, November 2002.