

Rigorous and Automatic Testing of Web Applications

Xiaoping Jia and Hongming Liu

School of Computer Science, Telecommunication and Information Systems

DePaul University

Chicago, Illinois, USA

email: {xjia, jordan}@cs.depaul.edu

ABSTRACT

As web applications become more and more prevalent, the quality assurance of web applications has become more and more important. Due to the complexity of the underlying technologies of web applications, it is more challenging to test web applications than conventional software. It is critical to develop effective methodologies and tools for testing web applications. In this paper, we propose an approach for rigorous and automatic testing of web applications using formal specifications. Our formal specification based approach is powerful, extensible, and versatile. It intends to address testing of various aspects of web applications, including functionality, security, and performance. We have developed a prototype tool based on the proposed approach, which accepts formal specifications in XML syntax as input, automatically generates test cases, executes the test cases and validates the test results.

KEYWORDS

web applications, formal specifications, software testing, XML

1 Introduction

Web applications are software programs or applications that receive input and deliver output through the Web, usually in the form of HTML or XML. Web applications are dynamic, interactive, often serve as the front end of complicated applications that often involve databases at the back-end. As more and more businesses being conducted through the web, it is critical for Web applications to be tested thoroughly. Due to the unique characteristics of web applications, conventional software testing tools are not adequate in dealing with web applications. It is critical to develop effective methodologies and tools for testing web applications.

Testing of web applications is a specialization of software testing, which has been studied for years by researchers. Several methodologies has been proposed to support web application testing [5, 6, 7, 8]. There are also some commercial automated Web application testing tools available [11, 12, 13]. While Web ap-

plication testing is a rather new area, most of these methodologies and tools have limitations especially in testing the overall functionality of Web application. We will cover more detail about these methodologies and tools in session 5.

In this paper, an approach for rigorous and automatic testing of web applications is presented. This approach uses formal specifications of web applications to test functionality, security and performance. Using the formal applications on functionality, security and performance of a web application, a test engine can automatically generate test cases, perform the test, and validate the test result against the formal specifications. A prototype tool *WebTest* has been developed and successfully demonstrates the feasibility and effectiveness of our approach.

The remainder of the paper is organized as follows. Section 2 presents an overview of the components of our approach for web application testing. Section 3 discusses the details of test specifications using several examples. Section 4 presents a prototype tool demonstrating our approach. Finally, section 5 gives a comparison and discusses future work.

2 Our Approach

To automate testing of web applications, we formally specify the testing process, as well as the functionality, security and performance of a web application using a formal specification language. A test engine will take the specification as the input, performance the test, and generate a test report. Figure 1 shows the overview of our approach.

The input to the test engine is a formal specification in XML syntax which specifies the functionality, security and performance of a web application needs to be tested. It contains templates of test cases. It specifies the input and the expected outputs of each test case. Test engine reads the input specification and generates test cases based on specification. Then the test engine executes the test cases and validates its result against the expected result specified in formal specification. At last, test engine generates a test report summarizing the results of all test cases. As shown in Figure 1, the test engine behaves as a web

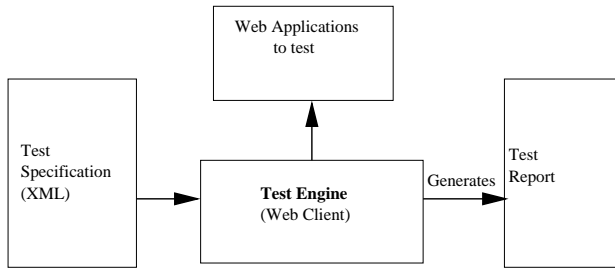


Figure 1. Web testing approach.

client accessing web applications.

To achieve the above goals, we use W3C’s Extensible Markup Language [14] as the concrete syntax of our test specifications. XML is a widely used markup language for representing hierarchical structures and data. It has achieved broad acceptance across the industry, leading to the development of editors and parsers for a variety of platforms and operating systems. Using XML will allow our specification language to leverage many standard utilities and mechanisms defined on XML, such as DOM and XPath, etc. By formally specifying web applications’ intended functionality in test specification, we can perform test by comparing actual functionality with the intended functionality. Any web application functionality that can be specified can be tested. Besides functionality, security and performance are aspects of web application that can be tested using our approach. In more details, our approach can test:

- **Functionality:** specify appropriate path and parameters of web applications to test functions such as credit card processing. By simulating Web browser’s action, our approach can process through particular Web path and verify particular information.
- **Page structures:** specify if appropriate pages contain specific elements such as forms, text and images. Our approach can identify portions of a web page using XPath syntax and identify string pattern using regular expression. It also provides mechanism to do date and number comparison.
- **Security:** specify authentication protected area, session duration etc. For example, by specifying false user information and data such as user name, password, our testing can detect and report the vulnerability of security Web site.
- **Performance:** specify requests to mimic realistic user traffic. For example, it can specify responding time during different load level in the test specification and the test result will give a report of real performance counters.

```

TestSpec      ::= TestSuite*
TestSuite     ::= TestCase*
TestCase      ::= TestStep+
TestStep      ::= Name Conditionopt RequestSpec
                ResponseSpec TestStep*
RequestSpec   ::= URL RecursiveSpecopt
                HeaderSpec* ParameterSpec*
                VariableDecl*
ResponseSpec  ::= StatusCode ContentType
                HeaderSpec*
                VariableDecl* Predicate+
RecursiveSpec ::= Depth Domainopt
HeaderSpec    ::= Name CDATA optionalopt
ParameterSpec ::= Name CDATA optionalopt
Condition     ::= Predicate
Predicate     ::= not Predicate
                | Predicate and predicate
                | Predicate or Predicate
                | Predicate implies Predicate
                | ( forall VariableDecl+ Predicate )
                | ( exists VariableDecl+ Predicate )
                | SimplePredicate
SimplePredicate ::= MatchPred
                | ContainPred
                | ComparePred
MatchPred     ::= SelectExp MatchOp
                ( RegExp | CDATA )
MatchOp       ::= equals | contains
                | startsWith | endsWith
ComparisonPred ::= Exp ComparisonOp Exp
Exp           ::= NumberExp | DateExp
ComparisonOp  ::= == | != | < | <= | > | >=
ContainPred   ::= HrefSpec | FormSpec
HrefSpec      ::= URL CDATAopt VariableDeclopt
FormSpec      ::= ActionSpecopt MethodSpecopt
VariableDecl  ::= Name SelectExp

```

Figure 2. Abstract syntax of specification language.

3 Test Specification

A key component of our approach is a specification language for formally specifying the testing process and the functionality of the web applications to be tested. A BNF grammar of the abstract syntax of the specification language is shown in Figure 2. The concrete syntax of the specification language is in XML. The XML Document Type Definition (DTD) corresponding to the abstract syntax in Figure 2 can be found in our project page <http://jordan.cs.depaul.edu/research>. The examples of specifications in this section are give in XML syntax.

3.1 Specification Structure

A test specification is a hierarchy of test suite, test case, and test step.

- **Test Suite.** A specification consists of one or more test suites. Each test suite contains a set of test step for one feature of web application. For example, you can have different test suite for different

aspect of web functionality, security and performance.

- **Test Case.** A test case is designed to test a specific aspect of web application functionality, security or performance. Each test case consists of a set of test sequences, and each test sequence consists of a sequence of test step.
- **Test Step** A test step specifies one step in the interaction of a web application. Each test case consists of a pair of request specification and response specification.

In our test specification, a test case is defined as a tree of test step. Each path from the root of the tree to a leaf of the tree represents a test sequence. The nodes on a path represent test step to be carried out successively. In other words, a child node represents a test step to be taken following the test step represented by its parent. Sibling nodes represent alternative test step in test sequences.

The following is an example of a test suite”.

```
<testsuite>
  <testcase name="t1">
    <teststep name="a0">
      Request and Response elements of step a0
    <teststep name="a1">
      Request and Response elements of step a1
      <teststep name="a11">
        Request and Response elements of step a11
      </teststep>
      <teststep name="a12">
        Request and Response elements of step a12
      </teststep>
    </teststep>
  </testcase>
  <testcase name="t2">
    <teststep name="b0">
      Request and Response elements of step b0
    </teststep>
  </testcase>
</testsuite>
```

This test suite contains 2 test case: t1 and t2. test case t1 consists of two test sequences:

- Test sequence 1: test step a0, test step a1, test step a11
- Test sequence 2: test step a0, test step a1, test step a12

test case t2 contains only one test sequence, which consists of a single test step b0.

3.2 Request and Response Specification

Each test step consists of a pair of *RequestSpec* and *ResponseSpec* specifications. A test step may also

have an optional boolean condition. If the condition is present, the test step will be carried out only if the condition is true.

- *RequestSpec* The *RequestSpec* specification specifies a pattern of HTTP requests, which consists of the URL of the request, a (possibly empty) list of HTTP *HeaderSpec*, and a (possibly empty) list of *parameterSpec*.
- *ResponseSpec* The *ResponseSpec* specification specifies the assertions on the HTTP response generated from the HTTP request in the same test step. Each *RequestSpec* specification consists of a (possibly empty) list of *HeaderSpec* followed by a list of *Predicate*. Both *RequestSpec* and *ResponseSpec* specifications may contain a list of *VariableDecl*, which bind values to variables that can be used elsewhere in the specification

The attributes of *RequestSpec* specification are explained below:

- *url* specifies the URL of initial request web page.
- *recursiveSpec* specifies the *Depth* of the request recursion. It also specifies the restrict *Domain* of the request.

Example 1. Link Check

Hyperlink integrity is an important aspect of web application functionality. The following example shows a specification which tests a web site's links recursively. It will recursively check any links less than 3 steps from the initial URL: <http://www.cs.depaul.edu>, to see if all the links are valid.

```
<testcase name="Link check">
  <teststep name="link check 1">
    <request url="http://www.cs.depaul.edu",
      recursive="true", recursivedepth="3"/>
    <response statuscode="200"/>
  </teststep>
</testcase>
```

The above test case requests a web page with initial URL <http://www.cs.depaul.edu>. The specification `recursive="true"` and `recursivedepth="3"` indicates that this request will continue recursively once it finds hyperlink in the requested page until it reaches the third level. The response specification only specifies `statuscode="200"`, which means the test specification is expecting a status code of 200, i.e., link okay, for all the response.

3.3 Predicates

The predicate in response specification specifies the assertions on the results of testing.

- *Predicate* Two kinds of predicate are defined: simple predicate and compound predicate.

- *Simple Predicate* Three simple predicates are defined to verify the contents of the response pages: *MatchPred*, *ContainPred* and *ComparePred*.
- *Compound Predicate* Compound predicates can be formed using the logical connectors such as: **not**, **and**, **or**, or **implies**. Quantified predicates are also be used.

The *MatchPred* checks the contents of the response document. It takes the following arguments:

- *SelectExp*, which is an XPath expression [15], selecting a segment from the response document; XPath is a language for selecting parts from an XML document. It also provides basic facilities for manipulation of strings, numbers and booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values.
- A target string, which can be either a literal string or a regular expression.
- *MatchOp* operator, which is one of the following:
 - **equals**: the selected segment matches the target string exactly;
 - **contains**: the selected segment contains the target string;
 - **startsWith**: selected segment has the target string as a prefix;
 - **endsWith**: selected segment has the target string as a suffix.

The *ContainSpec* predicate checks the presence and the contents of elements such as `<href>`, `<form>`, or `<table>`, etc. The *ComparePred* predicate checks whether to objects, such as strings or dates, are equal.

While using Xpath in *SelectExp*, our approach treats HTML document returned by Web application as a Document Object Model(DOM). The DOM technology enables HTML document to be manipulated by exposing each HTML document element as node with properties and Xpath provides a function that mapping DOM to a single or a set of nodes.

$$f_n : DOM \rightarrow \mathbb{P}Node$$

The richness and expressiveness of the predicates and expressions defined in the specification language allows complete and precise specifications to be written on the behavior of request and responses. The goal of the specification language is to support the level of expressiveness similar to that of generic formal specification languages, such as *Z* in the context of web applications.

Example 2. Content Check (simple predicates) The following is an example of a test case with predicate elements. It checks the content of page `http://www.cs.depaul.edu/program` to see if it contains certain key words. This test case uses the match predicates to check the content of the response.

```
<testcase name="Content check using predicates">
  <teststep name="Content check step one">
    <request url="http://www.cs.depaul.edu/program"/>
    <response statusCode="200">
      <and>
        <or>
          <match op="contains" regexp="false"
            select="/html/body" value="Undergraduate Degree"/>
          <match op="contains" regexp="false"
            select="/html/body" value="Bachelor Degree"/>
        </or>
        <match op="contains" regexp="true"
          select="/html/body value="[M|m]aster [D|d]egree"/>
      </and>
    </response>
  </teststep>
</testcase>
```

In this test case, the request specification specifies the initial URL `http://www.cs.depaul.edu/program`. In this test case, the `select="/html/body"` means that the match predicate is searching the body of the response document. With the string value or regular expression specified by value attribute, the above match predicate specify that the response page contains literal string `Undergraduate Degree` or `Bachelor Degree` and regular expression `[M|m]aster [D|d]egree`.

Quantified Predicates

Two quantified predicates, **forall** and **exists**, are defined in the specification language. A quantified predicates contains a list of variable declarations and a predicate. A variable declaration consists of the variable name and a select expression in XPath [15], which selects a set of one or more fragments from the response document. Each of the fragments are bound to the variable name. The predicate **forall** returns true if the component predicate evaluates to true for *all* variable bindings, i.e., the predicate evaluates to true for all the selected fragments. The predicate **exists** returns true if the component predicate evaluates to true for *at least one* variable binding, i.e., the predicate evaluates to true for at least one of the selected fragments.

Example 3. Content Check (quantified predicates) The following test goes to a sample page and check if all the references in that page are using http protocol.

```
<testsuite>
  <testcase name="Content check using quantified
    predicates">
    <teststep name="content check step one" >
      <request url="http://jordan.cs.depaul.edu
        /webtest/testhref.htm">
      </request>
```

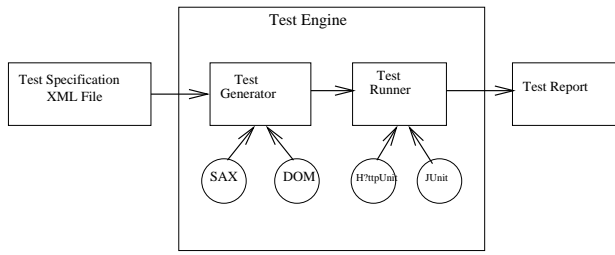


Figure 3. A prototype tool — WebTest.

```

<response>
  <forall>
    <varbale name="l" select="descendent::a/@href"/>
    <match op="startswith"
      select="$l"
      regexp="false"
      value="http://"/>
    </forall>
  </response>
</teststep>
</testcase>
</testsuite>
  
```

In this test case, `select="descendent::a/@href"` selects all the `href` attribute of the `a` elements in the response page. The `match` predicate checks if each of the URL begins with `http://`. In other words, this test specification checks if all the URL in the response page are using the HTTP protocol with an absolute URL.

4 Prototype Tool

We have developed a prototype tool, WebTest, to demonstrate our approach. It allows users to load a test specification file, generate and run the test. Figure 3 is an overview of the design of WebTest. The input of WebTest is an XML file containing the test specifications. Test Generator and Test Runner together form Test Engine, which reads the input XML file and generates a Test Report as output. Test Generator is a Java program reads the XML file, parses it and generates the testing source code — the Test Runner. It uses available XML processing software or API such as Java API for XML Processing (JAXP). Both of these API supports processing of XML documents using DOM and SAX. Test Runner is a Java program generated by Test Generator. It uses JUnit [16] a testing framework to implement tests and uses HttpUnit [17] Java API. Test Report gives the result of test. It contains information such as how many cases are tested, time taken by each of the test case and result of each test case.

5 Related Work

While the growth of Web applications has been extraordinary, only few works related to Web application testing are devoted to Web application testing. The contribution of this paper to the available literature on Web application is in the introduction of a new approach based on formal specification which conducts rigorous and automatic testing of Web application, and in the study of the formal specification that can be effectively applied to Web application testing. Several methodologies have been proposed to support Web application testing.

- Hieatt and Mee [9] introduces a mechanism called *acceptance testings*. *Acceptance testing* allow product managers to express tests at the scenario level. It develops a testing tool that expressed operations in the system and the expected results as XML. The using of XML to specify expected result is similar to our test specification. However, Hieatt and Mee is more focus on unit testing while our approach concern more about the overall testing of Web application.
- A UML model of Web application is proposed in Ricca and Tonella [5] for Web application presentation, which drives Web application testing, in that it can be exploited to define white box testing criteria and to semi-automatically generate the associated test case. This mechanism focus more on validating paths in a web site, while our approach deals with broader functionality aspects. A few similar object oriented models are also proposed in [7, 8].
- Performance testing is proposed in Subraya and Subrahmanya [6]. It addresses a new testing process uses the concept of decomposing the behavior of the Web site into testable component, which are mapped into testable objects. Different from this approach, ours test Web application's behavior as a whole and don't have the process of decomposing the Web site.

6 Conclusions

In this paper, we have presented a novel approach to conduct rigorous and automatic web application testing. The approach is based on formal specification of Web applications' functionality, security and performance. Our main contributions are:

- Using formal specifications, our approach can precisely capture and specify the functionality of web applications. This is due to the expressiveness of our test specification language with the inclusion of simple and compound predicates.

- This approach is versatile and flexible for testing a wide variety of different aspects of web applications., including link validity, functionality, security, and performance.
- Our approach can perform web application test automatically once test specification is feed into test engine. Same test can be repeated for regression test of web applications. Test specification can evolve along with the web applications.
- This approach is especially suited for incremental and iterative development processes often adopted for web application development.
- A prototype tool, WebTest has been developed to demonstrate the feasibility and effectiveness of this approach.

Our future work will involve enhancing our approach by focusing on enhancing the ability and flexibility of test specifications to specify functionality, security, and performance of web applications. To accomplish that we will enrich our specification language to cover more testing aspects of Web Application. We will also further enhance the capability of the prototype tool WebTest to deal with large and complex web applications.

References

- [1] B. Beizer, *Software Testing Techniques*, 2nd edition. International Thomson Computer Press 1990
- [2] X. Jia, *A Pragmatic Approach to Formalizing Object-Oriented Modeling and Development*, Proc. of the 21st Annual IEEE International Computer Software and Applications Conference, August 1997, Washington, D.C., USA. pp. 240–245.
- [3] X. Jia, *An Approach to Animating Z Specifications*, Proc. of the 19th Annual IEEE International Computer Software and Application Conference. August 1995, Dallas, TX. pp. 108-113.
- [4] X. Jia, *Model-Based Formal Specification Directed Testing of Abstract Data Types*, Proc. of the 17th Annual IEEE International Computer Software and Application Conference, Phoenix, Arizona, November 1993, pp. 360-366.
- [5] F. Ricca and P. Tonella, *Analysis and Testing of Web Applications*, Proc. of the 23rd International Conference on Software Engineering, 2001, Toronto, Ontario, Canada. pp.25-34
- [6] B.M. Subraya and S.V. Subrahmanya, *Object driven Performance Testing of Web Applications*, The First Asia-Pacific Conference on Quality Software, October 2000 HongKong, China.
- [7] J. Yang, J. Huang and F. Wang, *An Object-Oriented Architecture Supporting Web Application Testing*, The 23rd Annual International Computer Software and Application Conference, October 1999, Phoenix, Arizona, USA.
- [8] C. Liu, D. Kung and P.Hsia, *Object-Based Data Flow Testing of Web Application*, The First Asia-Pacific Conference on Quality Software, October 2000 HongKong, China.
- [9] E. Hieatt and R. Mee, *Going Faster:Testing the Web Application*, IEEE Software March/April 2002(Vol.19,No.2)
- [10] ParaSoft, *WebKing White Paper*, <http://thewebking.com/products/webking/papers>
- [11] Rational Software, *Visual Test 4.0 White Paper*, <http://www.rational.com/products/visual.test/>
- [12] Softbridge Inc., *Web Analyst*, <http://www.softbridge.com>
- [13] Sun Microsystems, *SunTest Suite*, <http://www.sun.com/suntest>
- [14] *Extensible Markup Language (XML) 1.0*, Second Edition, W3C Recommendation, 6 October 2000
- [15] *XML Path Language (XPath Version 1.0)*, W3C Recommendation, 16 November 1999
- [16] *JUnit Frameworks*, <http://www.junit.org>
- [17] *HttpUnit Java API*, <http://www.httpunit.org/doc/api/index.html>