



## Improved Static Code Analysis Techniques for Automated Testing of Ajax Apps

Sakeena Apsar SD, VenkataSubbaiah K

Dept of Computer Science and Engineering & JNTUA  
Hyderabad, India

---

**Abstract**— AJAX applications are well known for making errors due to their state fullness, asynchronous, event-built nature, the use of loosely typed, the client-side changes of the browser's Document-Object Model (DOM), and the use of delta communication between client and web server. Unfortunately, static code analysis techniques are not able to reveal many of the dynamic dependencies present in today's web applications. To that end, we propose an implementation named ATUSA in which we automatically derive a model of the user interface states of an AJAX application by "crawling" the AJAX application code to identify key events and components. In order to recognize logical failures in these executions, we propose the use of invariants obtained from the crawling process and for handling structural failures we propose to implement Metamorphic Relations based oracle. The results highlight the efficiency of the proposed approach in terms of fault-finding ability, ability to be changing in size, to increases the level of automation, and the usefulness of invariants and metamorphic relations.

**Keywords** — Ajax, DOM, JavaScript, web-server, Automated Testing, Web testing techniques.

---

### I. INTRODUCTION

Now a day's developing applications on the Web is the main aspect of research oriented peoples. For example we consider different applications like Google mail, Order Goods, Estimate, Educate, Search Information, Delivery News applications etc. These applications are easy to develop in web compared to standalone applications, because web application contain following benefits.

1. Zero Install
2. Reduce business cost
3. Centralized Data
4. Direct Access
5. Always up-to-date
6. Low space Pc's or smart phones

Above benefits is user friendly. And also they are developing these applications in software variances and testing variances based on dynamic web applications, because present days testing are the main problem for web application. So many people have developed different techniques for generating testing tools automatically. In software engineering there are many different types of testing strategies are proposed. Using testing strategies our developer proposed many techniques for generating testing tools automatically. This software testing tool streamlines teamwork by introducing a common platform for testers and developers to work together. It ships with a standalone application and a Visual Studio plug-in that use the same repositories and file formats. For today's web applications, one of the key technologies facilitating this move is AJAX(Asynchronous JAVASCRIPT And XML).AJAX applications are well known for making errors due to their statefulness, asynchronous, event-built nature, the use of loosely typed JAVASCRIPT, the client-side changes of the browser's Document-Object Model (DOM), and the use of delta communication between client and web-server. Recent tools such as Selenium offer a capture-and-replay style of testing for modern web applications. Whereas such tools are having the ability of executing AJAX test cases, they still demand some amount of manual effort from the tester. We propose to support automated testing of AJAX applications and therefore we propose an approach where we automatically derive a model of the user interface states of an AJAX application. We obtain this model by "crawling" an AJAX application code to identify key events and components. Invariants are properties of either the client-side DOM tree or the derived state machine that should hold for any execution. These invariants can be general or application-specific. The results highlight the efficiency of the proposed approach in terms of fault-finding ability, ability to change in size, increases the level of automation, and the usefulness of invariants. Static analysis is usually poor for identifying memory leaks and concurrent errors. To spot such errors you needs to execute a part of the program implicitly and It is very difficult to implement because such type of algorithms needs too much of memory and processor time. Static analyzers frequently limit themselves to identify simple cases. A more efficient way to spot memory leaks and concurrent errors is to use dynamic analysis tools. Automatically detecting dynamic structural and JAVASCRIPT invariants in modern web applications.

## II. RELATED WORK

The validation of Web applications that are based on asynchronous communication with the server and employ new technologies, such as AJAX, Flash, ActiveX plug-in components, is an area that deserves further investigation. For example, until now, little research efforts have focused on how to test Web applications employing AJAX. Since AJAX does not comply with the classical Web application model, several techniques presented in literature will not work any longer.

Traditional Web applications are based on the client-server model: a browser (client) sends a request asking for a Web page over a network (Internet, via the protocol HTTP) to a Web server (server), which returns the requested page as response. During this elaboration time the client must wait for the server response before visualizing the requested page on the browser, i.e., the model is based on synchronous communications between client and server.

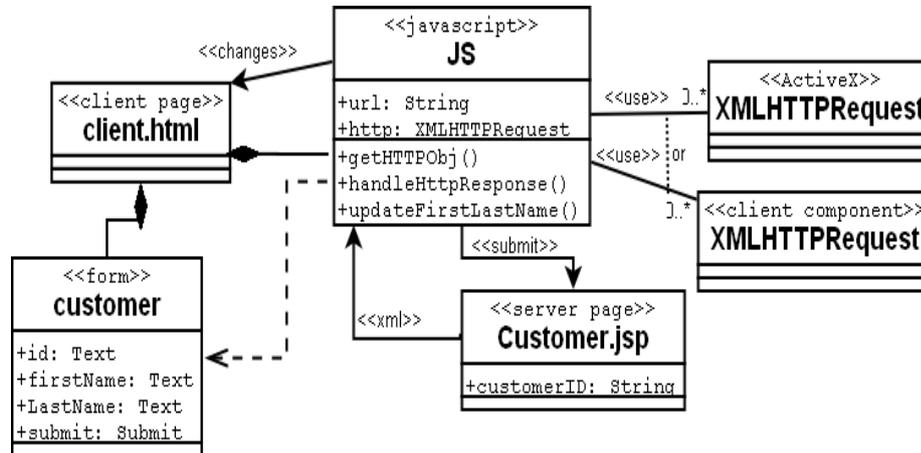


Fig.1. customer ID UML model

AJAX (Asynchronous JAVASCRIPT and XML) is a bundle of existing technologies used to simplify the implementation of rich and dynamic Web applications. HTML and CSS are used to present the information, the Document Object Model is used to dynamically display and interact with the information and the page structure, the XMLHttpRequest object is exploited to retrieve data from the Web server, XML is used to wrap data and JAVASCRIPT is exploited to bind “everything together” and to manage the whole process.

Automatic detection of invariants is another direction that has increased momentum. The known work of Ernst et al on Daikon, a tool capable of inferring likely invariants from program execution traces. A more recent tool is DoDom, capable of inferring DOM invariants. We also look at the ways of automatically detecting DOM and JAVASCRIPT invariants in web applications. The rest of this paper proposes section III describes existing techniques in testing applications Section IV describes proposed process for application development Section V describes performance analysis with results.

## III. EXISTING SYSTEM

The traditional way of placing content from multiple external vendors on one page is by using IFrames. The advantage of IFrames is that they provide an isolated environment for the widgets because the Same Origin Policy (SOP) constraints them from accessing other parts of the page. The main problem with using IFrames is, however, that they do not provide real integration on the page. IFrames impose undesired limitations, for instance on, the page layout, use of CSS style sheets of the framework, and rich widget interactions such as resizing. Static program analysis is the analysis of computer software that is performed without actually executing programs (analysis carried out by executing programs is known as dynamic analysis). In many situations the analysis is performed on some version of the source code and in other situations some form of the object code. The term is normally applied to the analysis, carried out by an automated tool. Today’s dynamic web applications using one of the driving technologies is AJAX(Asynchronous JAVASCRIPT and XML)AJAX web browsers not only suggest the user navigation across a sequence of HTML pages, but also dynamic rich client interaction via graphical user interface(UI)components. AJAX applications are well known for making errors because of their state fullness, asynchronous, and event-built nature, the use of loosely typed JAVASCRIPT, the client-side changes of the browser’s Document-Object Model (DOM), and the use of delta communication between client and web-server.

Furthermore, traditional web testing techniques are based on the classical page request/response model, does not take client side functionality into their account. While such tools are capable of executing AJAX test cases, they still demand a considerable amount of manual effort from the tester. So we propose better automated testing system for quality of AJAX applications.

## IV. PROPOSED SYSTEM

We propose to support automated testing of AJAX applications. To the end, we propose an approach where we automatically derive a model of the user interface states of an AJAX application. We obtain this model by “crawling” an

AJAX application code to identify key events and components. Invariants are properties of either the client-side DOM tree or the derived state machine that should hold for any execution. These invariants can be general or application-specific. In order to recognize failures in these executions, we propose the use of invariants obtained from the crawling process. We offer an implementation of the proposed approach in an open source, plug in-based tool architecture. It also consists of a crawling structure which is similar to CRAWLJAX and as well as a series of testing specific extensions. Our proposed system is referred to as ATUSA (Automated Testing of User Specific Ajax). We have applied these tools to a series of AJAX applications. The results highlight the efficiency of the proposed approach in terms of fault-finding ability, ability change in size, increases the level of automation, and the usefulness of invariants.

## V. OUR APPROACH

Automatically detecting dynamic structural and JAVASCRIPT invariants in modern web applications is complicated due to short comings of Static code analysis techniques. **Static code analysis' disadvantages**

1. Static analysis is usually poor for identifying memory leaks and concurrent errors. To spot such errors you need to execute a part of the program implicitly and It is very difficult to implement because such algorithms take too much memory and processor time. Static analyzers limit themselves to identify simple cases. A more efficient way to spot memory leaks and concurrent errors is to use dynamic analysis tools.
2. A static analysis tool notice you about the code fragments that is, the code can actually be quite correct. It is also known as false-positive reports. Only the programmer understands that the analyzer estimates the real error or it is a false-positive. The need to review false-positives takes work time and weakens attention to those code fragments that really contain errors.

Instead of employing Static code analysis based oracles for initiating testing we propose to implement metamorphic testing. Metamorphic testing is a technique for the verification of software output without a complete testing oracle. Metamorphic testing observes that even if the executions do not result in failures, they still takes useful information. Additional test-cases should be constructed from the original set of test-cases with reference to selected essential properties of the specific function. Such essential properties of the function are called metamorphic relations. The subject program is verified through metamorphic relations (MR). It is unlikely for a single MR to detect all possible faults. Therefore, 4 MRs that are different from one another with a view to spot various faults will be used. Finding good MRs requires knowledge of the problem area, knowing the user requirements and some creativity. These Metamorphic relations are identified according to equivalence and nonequivalence relations among regular expressions. Therefore, this type of testing help in an automated addressing of all possible forms of failures be it structural, syntactical or logical.

## VI. PERFORMANCE ANALYSIS

A metamorphic property can be defined as the relationship by which the change to the output of a function can be predicted based on a transformation of input .Study a function that calculates the standard deviation of a set of numbers. Some transformations of the set would be expected to produce the same result: for example, allowing the order of the elements should not affect the calculation; nor would multiplying each value by -1, since the deviation from the mean would still be the same. Clearly metamorphic testing can be very useful in the absence of an oracle: regardless of the values that are used in the test-cases, if the relations between the inputs and their respective outputs are not as predicted, then a fault must exist in the implementation. That is, even if there is no test oracle to indicate whether  $f(x)$  is correct, if the output  $f(t(x))$  is not as predicted, and the metamorphic property is sound, the property is considered as violated, and therefore a fault must exist.

Figure 2 demonstrates an example of a metamorphic property for system testing as specified in an XML file. On line 2, the name of the program to be run (in this case, Marti Rank) is specified, and on line 3 the names of the command-line parameters are given. For the purposes of metamorphic testing, the input is given the placeholder name "training data" and the output is given the name "model"; other parameters specific to the execution of Marti Rank are also specified on line 3. On lines 5 and 8, the types of files for the input and output are given, respectively. The metamorphic properties to be tested are declared in lines 10-22. On line 11, we say that there is to be a "main" execution, the output of which will be shown to the user. On lines 12-14, we specify a test case called "test1" in which the input is to be permuted. On lines 15-17, we also specify another test case in which the elements of the training data should all be multiplied by 10. Lines 19 and 20 specify how to compare the outputs for the two metamorphic properties: in both cases, the new output (as a result of the test case) is expected to be the same as the original from the "main" execution

```
1 <TESTDESCRIPTOR>
2 <EXECUTION>/usr/bin/c-marti</EXECUTION>
3 <PARAMETERS>@input.training_data@output.model- -no-permute- -no-prob-dist</PARAMETERS>
4 <INPUT>
5 <VAR TYPE="csv_file" NAME="training_data"/>
6 </INPUT>
7 <OUTPUT>
8 <VAR TYPE="text_file" NAME="model"/>
9 </OUTPUT>
10 <POST_TEST>
11 <BRANCH NAME="main"/>
```

```

12 <BRANCH NAME="test1">
13 @op_permute(@input.training_data)
14 </BRANCH>
15 <BRANCH NAME="test2">
16 @op_multiply(@input.training_data,10)
17 </BRANCH>
18 <PROPERTY>
19 <ASSERT> @op_equal(@main.output.model, @test1.output.model) </ASSERT>
20 <ASSERT> @op_equal(@main.output.model, @test2.output.model) </ASSERT>
21 </PROPERTY>
22 </POST_TEST>
23 </TESTDESCRIPTOR>

```

Figure2: Specification of metamorphic property for system-level testing.

**Results:**

Each of the four techniques described above (partial oracle, pseudo-oracle, metamorphic testing, and runtime assertion checking) was applied to all mutated variants of the three applications investigated in this experiment. The goal was to determine what percentage of the mutants was killed by each approach.

Table 1: Distinct defects detected in Study.

App.	# Mutants	Partial Oracle	Metamorphic Testing	Assertion Checking
C4.5	28	22 (78.5%)	26 (92.8%)	28 (100%)
MartiRank	69	47 (68.1%)	69 (100%)	57 (82.6%)
SVM	85	78 (91.7%)	83 (97.6%)	68 (80.0%)
<b>Total</b>	<b>182</b>	<b>147 (80.7%)</b>	<b>178 (97.8%)</b>	<b>153 (84.0%)</b>

Table 1. Shows the electiveness (percentage of distinct defects found) for the partial oracle, metamorphic testing, and assertion checking. We have omitted the results of testing with pseudo-oracles for reasons that are described at the end of the Discussion section. Overall, metamorphic testing was most elective, killing 97.8% of the mutants in the three applications. Assertion checking found 84% of the 182 defects, and the partial oracle detected 80.7%.

**VII. CONCLUSION**

In this paper, we propose an approach in which we automatically derive a model of user interface states of AJAX applications based on support automated testing. The results highlight the efficiency of the proposed approach in terms of fault-finding ability, ability to change in size, increases the level of atomization, and the usefulness of invariants. But we faced a problem on memory space, so we propose metamorphic system testing based on their relations and properties. Our approach gives efficient results for generated automatic testing in various applications using AJAX and JAVA SCRIPT.

**REFERENCES**

[1] Mesbah and A. van Deursen, "Invariant-Based Automatic Testing of Ajax User Interfaces," Proc. IEEE 31st Int'l Conf. Software Eng., pp. 210-220, 2009.

[2] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock, "Automated Replay and Failure Detection for Web Applications," Proc. IEEE/ ACM 20th Int'l Conf. Automated Software Eng., pp. 253-262, 2005.

[3] F. Groeneveld, A. Mesbah, and A. van Deursen, "Automatic Invariant Detection in Dynamic Web Applications," Technical Report TUD-SERG-2010-037, Delft Univ. of Technology, 2010.

[4] M. Barnett, R. Deline, M. Fahndrich, K. Rustan, M. Leino, and W. Schulte, "Verification of Object-Oriented Programs with Invariants," J. Object Technology, vol. 3, no. 6, pp. 1-30, 2004.

[5] Memon, "An Event-Flow Model of GUI-Based Applications for Testing: Research Articles," Software Testing, Verification and Reliability, vol. 17, no. 3, pp. 137-157, 2007.

[6] C.-H. Liu, D. C. Kung, P. Hsia, and C.-T. Hsu. An object based data \_own testing approach for web applications. *International Journal of Software Engineering and Knowledge Engineering*, 11(2):157.179, April 2001.