Journal of Theoretical and Applied Information Technology
31st August 2013. Vol. 54 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A SUSTAINABLE APPROACH TO AUTOMATE USER SESSION BASED STATE MACHINE GENERATION FOR AJAX WEB APPLICATIONS

[1]ANUJA ARORA, [2]MADHAVI SINHA

[1] Jaypee institute of information Technology, CSE/ IT Deptt.

[2]Birla Institute of technology, Mesra

E-mail: [1]anujaarora2909@gmail.com, [2]madhavisinha@sify.com

ABSTRACT

A whole new generation WEB 2.0 is being written to take advantage of extreme dynamism as in AJAX. AJAX is used to build rich internet applications that are more interactive, responsive, and easy to use. With the advent of Ajax which involves extreme dynamism, novel problems add to those already known in the Web testing area. For Testing an AJAX application, a sustainable approach is required to detect faults embedded area. Faults embedded areas are those areas where states of web application will get changed. Here, in this research paper, our focus will be on generating state machine of dynamic behavior of an AJAX web application. To detect faults in any web application optimum approach is model based testing. Here intention is to generate effective state machine which will be able to detect all dynamic states of user session applied on web application. Therefore, we designed a framework to generate state machine of user sessions performed on web application. A prototype tool is developed to automate and validate the whole process. We applied whole process on five case studies and results have shown that we are able to successfully detect whole Events, dynamically changing DOM Elements and all dynamically generated States.

Keywords: *User Session based Testing; State Machine based Testing ; Dynamic Object Model; Web Testing.*

1. INTRODUCTION

Recently, web applications reached to a new level of achievement. As earlier internet was just information based and data driven, nowadays instead of just serving data and flow of information is now interactive and social in nature. Web2.0 technology includes advanced and sophisticated user interaction because of introduced new technologies like AJAX. A web technology that has gained a striking position under the umbrella of Web 2.0 is AJAX (Garett, 2005[1]), in which a intelligent combination of JavaScript, Document Object Model(DOM) manipulation, along with asynchronous server communication is used to achieve high level of user interaction. Garrett [1] introduced AJAX to label the architecture behind the new generation of rich web application like Google+, Google Suggest, Google Map. AJAX boom come in limelight after various dynamic Google web application appeared. Anyone, surfing internet has already encountered AJAX through various web applications like Facebook[1], Google Map[2], Google Document[3], Netflix[4], Star Tribune [5]and many more [2].

Unlike any standard request/response approach found in a standard Web client, working of AJAX is little different because of technological novelties like web applications are free to interact with server asynchronously. This asynchronous feature leaves the user interface active and responsive and combined with the possibility to update a page dynamically through the DOM. AJAX is a client-side approach and can interact with J2EE, .NET, PHP, Ruby, and CGI scripts—it really is server-agnostic. One of the real strengths of this approach is that AJAX web application is fit for the heterogeneous and autonomous environment, and

[1] www.facebook.com
[2] Maps.google.co.in
[3] Docs.google.com
[4] www.netflix.com
[5] www.startribune.com

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

developers don't need to learn some new (technology, method or programming language) in server-side technology.

AJAX is one of the key enabling technologies currently used for building modern web applications [1]. The use of AJAX technology positively affects the user friendliness and interactivity of web application [36]. Since Ajax applications are heavily based on asynchronous messages and runtime DOM manipulation, we expect the faults associated with these two features to be relatively more common and widespread than in other kinds of web applications.

In this research work, out of all new technology, focus will be on AJAX web application testing because with the advent of AJAX, novel testing problems raised and added to the list of already existing problems in web testing area. AJAX poses novel, additional problems with respect to those already known in the web testing area[18]. AJAX potentially brings an end to the classical click-and-wait style of web navigation. The level of runtime dynamic manipulation of DOM tree as well as asynchronous client/ server interaction makes it hard to test during testing process [20]. AJAX is fault prone because of stateful client, asynchronous communication, delta updates, untyped JavaScript, client side DOM manipulation, event handling, timing, back/ forward button and browser dependence.

The overarching contribution of this research work is to automate strategies for detecting faults embedded area by using proposed framework generated state machine. These strategies are implemented as modules that plug into a testing framework customized for web applications. The author provides context for the overarching contributions by describing the automatically generated state machine model for AJAX web application. The prime purpose of modeling is to represent the web application at a higher level of abstraction. A state machine model can represent a static or dynamic aspect of a web application.

In this research paper, a detailed structure of State machine model is described which will provide us faults embedded area in an AJAX web application. Therefore, we bounded structure of this paper to explain how to generate state machine of dynamism of AJAX web application to detect fault embedded area. This research paper structure is as follows. We start out with section 2 about other researchers' contribution to test specifically AJAX web application using state based testing. Section 3 provides framework of methodology to provide state machine of any AJAX web application. Model extraction is discussed in section 4 and then next section 5 is about model generation. In this section discuss about inferring state machine. Chosen AJAX web application case studied to evaluate the research work is being detailed in section 6. Section 7 is representing experiments performed and later in section 8 proving result and experimental output of proposed methodology on various chosen case studies. Finally last section9 is about concluding remarks.

2. RELATED WORK

To Test Web application under test, we are formally using State machine instead of Use case diagram or Control flow graph etc [33]. Test case generation of a Web application under test is easier and effective through state machine than any other diagram. Indeed there are various diagrammatic ways to represent web application but represent dynamic behavior of web application with the help of state machine is most efficient for testing. State machine provides a convenient way to model software behavior in a way that avoids issues associated with the implementation [11]. Since, in Ajax based Web application states of various objects are changing as per user trigged event or changes are reflecting from server side dynamically that is why we are using state machine to show state changes as per changed behavior of web applications [12, 13].

Marchetto proposed state based testing technique [12,13], to bridge the gap between existing web testing techniques and new feature provided by AJAX. Idea was that the states of client side components of an AJAX application need to be taken into account during testing phase [33]. State based testing technique for AJAX is based on the analysis of all the states that can be reached by the client-side pages or server performed action on the application during its execution. Using AJAX, HTML elements—like TEXTAREA, FORM, INPUT, A, LI, SELECT,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OL, UL, DIV, SPAN, etc.—can be changed at runtime according to the user interactions or server side action. In this testing the HTML elements of a client-side page characterize the state of an AJAX Web page, and their corresponding values are used for building its finite state model.

Marchetto's work:

Marchetto used traces of the application to construct a finite state machine [12]. This technique was based on the dynamic extraction of finite state machine for a given AJAX application. Whereas in Marchetto's work, dynamic analysis was partial that is why he was using manual validation or refinement steps for model extraction. He accepted in his work that FSM recovery needs an improvement and is an unexplored area [12]. Dynamic extraction of states is quite tough to explore and needs constant attention in AJAX testing. Therefore, there was a need for Automatic Dynamic analysis for model construction. Later in his work Marchetto was mainly concerned to identify sets of "semantically interacting" events sequence, used to generate test suite of test cases [35]. His intuition was that longer interaction sequences have higher faults. The Conducted experiments showing that longer interaction sequence have higher fault exposing capability [7, 9, 34, 35, 39].

This technique generates high number of test cases involving unrelated events, for minimizing test cases using notion of semantically interacting events. So here Marchetto's main contribution is for analysis of "semantically interacting" events sequence and result proves that more faults at the time of long interacting sequence analysis. Sequences of semantically interacting events in the model are used to generate test cases once the model is refined by the tester.

A FSM based testing technique generates high number of test cases. These high number of test cases reaches to a State Explosion Problem. In Marchetto's work for minimizing test cases used notion of semantically interacting Events. This technique minimizes only asynchronous communication test cases. In Marchetto's work, This AJAX testing approach covers only two aspects of asynchronous communication swapped callback and dependable request. Other AJAX testing challenges like fetching of dynamic constructed states, transition, dynamic DOM are untouched till now. Other than this there is a need of extracted FSM of an AJAX application through dynamic analysis.

Mesbah's work:

Mesbah proposed an "Invariant based Automatic testing of AJAX user interface". In his work, first task was crawling of the AJAX application using CRAWLJAX tool, simulating real user events on the user interface and infer the abstract model from state flow graph [36, 37]. Mesbah's suggested further AJAX research topic out of that one was automatic invariant detection. His invariant based testing was dependent on CRWLAJAX. He accepted in his work that best path seeding practice in web application is capture and replay which he not applied in his work. Mesbah proposed in his latest work[38] that invariant based testing is a weak form of an oracle, which can be used to conduct basic sanity check on the DOM-tree or transition in the derived GUI state machine. For dynamic extraction of states best approach is using any capture and replay tool like Selenium otherwise AJAX is too dynamic that not able to test that correctly. Mesbah mentioned that best practice to design a model is by using Capture and replay. Our testing method will use capture and reply. Indeed, there is a need for automatic dynamic analysis using any Capture-Replay tool for model construction. It will reach to the faulty state, trigger faults in those states and propagate them so that failure can be determined.

3. FRAMEWORK: GENERATE STATE MACHINE MODEL OF AN AJAX WEB APPLICATION.

This section presents a novel approach State-Machine model framework. The work presented here is based on the state-based testing approach, originally defined for object oriented programs [3, 4]. The proposed framework constructs a state machine model of an AJAX web application using dynamic analysis and reconstruction of user interface state change. Proposed model covers all validation and verification of the static and dynamic behavior of the web applications. Proposed state machine model will show user triggered events, Dom changing elements and their Event-Element

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Relationship mapping for dynamic changes in DOM at the time of state change in the web application. The proposed Automatic Fault Embedded Area Detection Model (AFEAD), its conventions and its formal assembly on the basis of Event Element Mapping Function is also explained in this section.

A framework is a meta-level (high level of abstraction) concept through which a range of other concepts, models, techniques, methodologies can either be clarified and/or integrated [5]. The framework consist of guidelines for supporting the various phases of software development process, various tools for capturing information, modeling, designing, generating test cases etc. On these similar ways this section presents the 'Automatic fault embedded area detection model Framework' for generating state machine of dynamic changing behavior of web application which will further be mapped with supporting testing framework. To empirically explore the issues and challenges in developing strategies to automatically generate model of web applications and to enable evaluation of the techniques developed in this research work various prototype tools and manual evaluation of some researches work have been considered [6-10]. Based on that, we designed an efficacious framework and implemented a prototype of proposed framework to automate the process of generating state machine model of dynamic behavior of an AJAX web application.

The prime origination of the proposed state machine model framework was to automate the fault embedded area detection process of the web application. In case of AJAX web application, fault embedded area are those areas where dynamic changes will take place either because of server side or from client side effects. Therefore, with the help of prototype tool of proposed framework we are trying to detect all places where faults will affect the web application. The proposed methodology is designed to automate the state machine generation process and containing the following non functional challenges to find out fault embedded area in AJAX web application testing.

Effectiveness: The framework is able to detect all events, dynamic DOM changes associated with the events, able to detect states of web application and finally generating state machine in efficient manner. Framework is able to detect both client side and server side changes in the web application.

Scalability: Proposed framework is able to quantify faults in the web application. Scalability exists in proposed framework in order to detect concurrent changing states, event based system changes, and state transitions of various web applications. The key characteristic of the framework is that incremental state machine generation and states detection process does not require additional functionality and extensive modifications;

Flexibility: Framework elements like user session recorder, DOM tree viewer, Event and DOM change Violation detection can be replaced depending on the web application development platforms requirement. The framework independently can handle diverse web application technologies testing issues;

Generalization: Framework can be used to test any web application. We tested various AJAX, Java scripting, Java Servlet, JSP web applications. Therefore using proposed framework, there is no need to make changes to web application code for testing. This framework can easily be extended to other web technologies but not tried yet;

The underlying idea has been implemented in the tool called USSMG (User Session based State Machine Generator) which is one prototype tool of complete AFEAD framework and this USSMG Tool containing various subparts like: User Session Recorder, Event-Element relationship, DOM tree generator, State machine Generator. We have performed number of experiments to check overall performance of our USSMG tool on several AJAX web applications, evaluated the efficiency in retrieving all states, events, dynamic elements and their relationships.

In the remainder of this research paper, we discuss the framework's major components: User session recorder, Dynamic Object Model tree generator, Event-Element Mapping Function, and state machine generator. The flow and framework between various elements- from collecting Dynamic changes in web application to generating State machine of user session trigged on the web application and finally to detect faults embedded area in the web application is shown in figure 1.

Journal of Theoretical and Applied Information Technology
31st August 2013. Vol. 54 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. MODEL EXTRACTION:

In AJAX applications, the state of the user interface is determined dynamically; through event driven changes in the browser's DOM are visible only after executing the related Java script code. In this research work initial experiment was to model DOM tree invariants in Finite state machine has been performed. AJAX based state changes in user interface reflect in DOM tree. Dynamic changes caused either by server side or client side but these events handled by AJAX engine. The client side calls the server once an event has been triggered and then modifies the document as per response. Here we are mainly modeling changes where response is linked with the request (modeling of all server response linked with client calls).

Figure 2 shows the processing flow of application to generate finite state machine of web application under test (WAUT). In complete process two modules have been developed: FSM generator and FSM extractor. Here in figure2 both tools working presented in separate boundaries. Output of first FSM extractor boundary is two files: scenario execution traces log file and DOM elements and event extractor files. These files are used as input for other boundary which is FSM generator. In this research work introduced mapping function is used to generate Finite state machine on the basis of above mentioned two generated files. Other than this in this research work author using Selenium tool[26, 27] to record user session and with the help of selenium tool generated log files modeling dynamic behavior of an application in finite state machine. At the end, result will be in form of State transition text file which will show finite state machine in GVEdit tool of Graphviz [14]. Here, we briefly outline Finite state machine extraction technique of Web application under test. Here given approach has been applied on many case studies and various examples. In solution approach, trying to explain approach with the help of head rush Ajax book AJAX powered coffee Maker [15] and then in experimental results showing results of Tudu lists[16,17] case studies and further generated Finite State machine of the following case studies ajaxfilmdb[18], AJAX IM [19] and Break Neck Pizza[15].

4.1 User session recorder to provide changing behavior states of WAUT

Ricca and Tonella's [20] approach creates a model in which nodes represent web objects (web pages, forms, frames), and edges represent relationships and interactions among the objects (include, submit, split, link). One Limiting factor in the use of web application testing techniques such as Ricca and Tonella's is the cost of finding inputs that exercise the system as desired. Selection of such inputs is slow and must be accomplished manually [20]. User-session based techniques can help with this problem by transparently collecting user interactions and transforming them into test cases. Elbaum et al. [21,22] proposed a Web application testing approach that utilizes data captured in user sessions, stored in a modified log file, to create test cases automatically.

Here for User session recording using Selenium tool which is a functional testing tool and used here to access the run time Dom and browser history in Mozilla browser. In the proposed approach by mapping selenium tool generated XML with implemented DOM tree inspector, we test code of the web application and find out where fault lies with the help of generated state based machine of that particular web application under test(WAUT). Selenium Tool is used to record semantically interacting user events carried out on the web application under test, data is stored in log files and these log files are used as input in extracting dynamic content from the DOM tree.

The application execution is traced and the execution is recorded in the log file. Execution traces can be traced using log files generated by real user interaction. Here using selenium tool generated log files, Execution traces exist in log file contains information about DOM states and call back. No of possible concrete states in AJAX web applications are huge and unbounded so can cause state explosion problem. Due to resolve this problem also, we have used User session Generated state machine in incremental manner. Each time generated state machine will be of a specific user session therefore, state explosion problem [23, 24, 25] will not arise in this methodology.

This is automated process for extracting dynamic states for model construction using user session tool selenium and generated Dom tree. The Selenium Tool generated log files are used as input

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

in extracting dynamic content from implemented DOM tree. For each user request or state change, system will analyze the modified DOM tree. The application execution traced and execution is recoded in a log file.

```
package com.example.tests;
import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.regex.Pattern;
public class Coffee Maker {
private Selenium selenium;
   @Before
    public void setUp() throws Exception {
      selenium = new DefaultSelenium ("localhost", 4444,
            "*chrome", "http://www.headfirstlabs.com/");
      selenium.start();
}
@Test
public void testCoffee Maker() throws Exception {
selenium.open("/books/hrajax/chapter03/coffee/coffee.html
?name=User2&size=small&beverage=mocha");
        selenium.type("id=name", "User1");
        selenium.click("css=input[type=\"button\"]");
        selenium.type("id=name", "User2");
       selenium.click("css=input[type=\"button\"]");
       selenium.type("id=name", "User3");
       selenium.click("css=input[type=\"button\"]");
       assertEquals("Sorry! Both coffee makers are busy.
                        Try again later.",
       selenium.getAlert());
       assertEquals("User1, your coffee is ready!",
       selenium.getAlert());
       assertEquals("User2, your coffee is ready!",
       selenium.getAlert());
       selenium.click("css=input[type=\"button\"]");
       selenium.type("id=name", "User4");
       selenium.click("css=input[type=\"button\"]");
       selenium.type("id=name", "user5");
       selenium.click("css=input[type=\"button\"]");
       assertEquals("Sorry! Both coffee makers are busy.
                        Try again later.",
        selenium.getAlert());
       assertEquals("User3, your coffee is ready!",
       selenium.getAlert());
       selenium.click("css=input[type=\"button\"]");
       assertEquals("User4, your coffee is ready!",
       selenium.getAlert());
       selenium.click("css=input[type=\"button\"]");  }
@After
public void tearDown() throws Exception {
selenium.stop();
}
}
```

Figure3: JUnit Test Case of onClick event of Coffee Maker web application

The proposed methodology starts from identifying the sequence of semantically interacting events from the Selenium TestRunner tool. Each sequence of semantically interacting events <E1, E2 ...En) used as a test case of length n means involving the execution of n callbacks.

In AJAX applications, the state of user interface is determined dynamically, so with the help of Selenium tool user triggered event driven changes in the Selenium generated HTML files are visible after executing the linked Java Script code, but those are only functional changes reflected by selenium, by linking selenium with DOM tree we can go for white box testing of the web application. Dynamically generated Dom state does not link automatically with the browser's history. Here in generated Tool, we are collecting all application execution traces using selenium generated HTML file as input and then matching user session events with DOM viewer. Finally extract dynamic events, other dynamic tags and complete DOM path of those dynamic events and tags from DOM viewer. The jUnit version of selenium generated log file of coffee maker example is presented in figure 3.

4.2 Generated DOM TREE VIEWER and Extracting components to find DOM Change Violation:

In traditional web applications, each state is represented by a URL and the corresponding web page. Whereas, in Ajax web application, it is the internal structure change of the DOM tree on the user interface that represents a state change. Therefore, to adopt a generic approach for all Ajax sites, we define a state change as a change on the DOM tree caused either by server side state changes propagated to the client, or client-side events handled by the Ajax engine.

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The DOM is an Application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. For instance, consider a table, taken from HTML document as shown in figure 4(a) and DOM tree of that table presented in figure 4(b).

In the DOM specification, the term "document" is used in the broad sense increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse

Journal of Theoretical and Applied Information Technology
31st August 2013. Vol. 54 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data [28].

```
<TABLE>
    <TBODY>
    <TR>
    <TD>Shady
Grove</TD>
    <TD>Aeolian</TD>
    </TR>
    <TR>
    <TD>Over the River,
Charlie</TD>
    <TD>Dorian</TD>
    </TR>
    </TBODY>
    </TABLE>
```

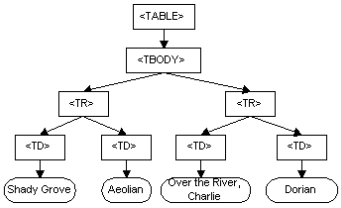

Figure 4: (a) Table taken from HTML document (b) Graphical representation of the DOM of the example table

With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model. It is based on an object structure that closely resembles the structure of the documents it models. In the implementation, The DOM tree viewer is implemented using JGrapht Library [29], JTree [30] library that displays a set of hierarchical data of a web document. DOM tree viewer shows the complete list of HTML elements of the current page and ordered list of HTML tag related to the element of the DOM structure of the target page. Therefore, a specific HTML element at a particular node in the Tree can be identified by path of the tree. Figure5 presents the implemented DOM Tree of an Online Pizza order web application. It contains the data and shows the structure of data in which it is designed.

Dynamic web application depends on asynchronous requests, which leave the user interface active and responsive, combined with the possibility to update a page dynamically through the DOM (Document Object Model). When client receives a response, it updates the web document by modifying one or more DOM objects. The document can be further processed and the results of that processing can be incorporated back into the presented page.

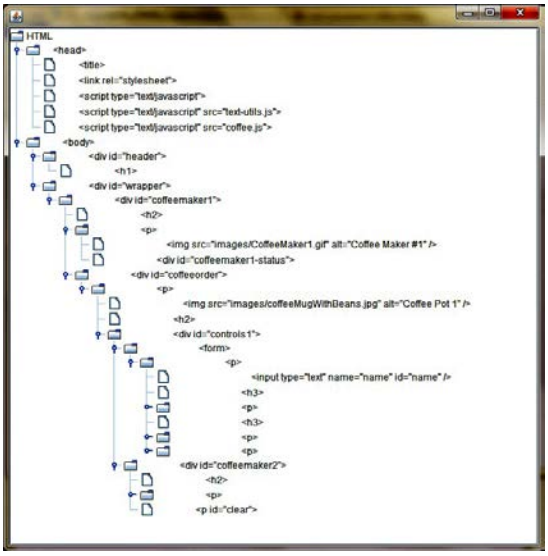

Figure5: Tool Designed DOM tree of Coffee Maker Application

The Dynamic objects are extracted from the required web application file and the DOM tree is constructed. Here Validating DOM is required because malformed HTML code can induce many errors, faults and browser portability issues and this is required to check that designed Dom is able to obtain what is needed. To prevent these faults, we should check and test DOM effectively. For validating DOM, we uses JTIDY[31], JTIDY provides a DOM interface to the document that is being processed. We validated designed DOM tree inspector with JTIDY designed DOM tree. We can't use open source JTIDY because this will not be able to execute our requirements. Implemented DOM tree viewer enriched with the capability as follows:

4.2.1 Navigation Bar:

User is able to navigate the web page to get idea about element and associative events. Dom viewer

Journal of Theoretical and Applied Information Technology
31st August 2013. Vol. 54 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

provides the user list of DOM elements of target page. Usually the list is very large. Therefore, to understand the web page navigation process, when user select element in DOM tree shown in the DOM tree viewer or click on DOM element List, tool highlight the selected DOM element area by red line. This is a vice-a-versa process means selected element will show DOM element area by red line and even DOM area selected on web page will also show elements coming inside that area see in figure 6. Figure 6 highlights the selected <p> tag by red line. Then, upon user click on display element list, Elements are added to the state element list.

Figure6: Navigation Bar of Break neck Pizza web application

4.2.2 DOM State Event Selector

There is no straight way of acquiring all clickable events solely by DOM TREE. Therefore our user session recording approach makes use of a set of all events, which are all exposed to an event type.

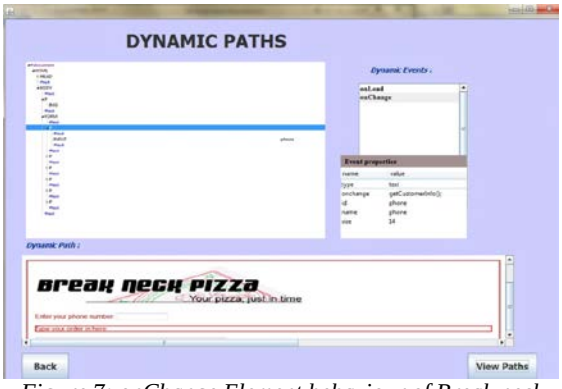

Figure 7: onChange Element behaviour of Break neck Pizza web application

In our coffeemaker example, we are using 'onClick' event to present our example. However, the other event types can be used just as well to analyze the effects on the DOM in the same manner. We are extracting and obtaining all events associated with an user session in automatic manner. At the time of recording of user session and changes will reflect in DOM tree, system will extract Events either server side state changes propagated to the client, or client-side events handled by the Ajax engine.

DOM Events selection is very important part for dynamic content testing in web application because DOM events are making dynamic changes in DOM elements values.

In AJAX web application even some specific tags like div, P etc typically used for styling or structuring have a relationship with events attached to it. Events make elements capable for changing the internal DOM state of the application when response is linked with request.

DOM event selection of a particular state provides all the properties/ behaviour of complete list of DOM nodes available in that particular state. By default, all the events will display in dynamic events list of a particular state. To simplify the event selection process, when one will click on a particular event or element all properties associated with that will be visible like node names and their node values etc. See in figure 7, in the selection of onChange event of 'Order Pizza' web application, one pop window will display and show all the properties of onChange event. Further these properties will be useful at the time to create a mapping between user session and DOM tree.

4.2.3 DOM State Element Selector

The total number of DOM elements and possible concrete DOM states is usually huge and unbounded. Hence, we consider only a restricted set of DOM elements coming under DOM Pages of a specific user session , also called 'state elements', that mainly characterize the state of the application GUI, and we consider user triggered states instead of the concrete ones, following an approach similar to the one implemented in the tool Adabu[32].

Here to extract all dynamic elements, we implemented one event-element model to save elements and events relationship mean on a specific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

event what all elements would change their behaviour. Here Implemented event-element model will extract and create a list of element those are changing behaviour with respect to associated event. In this process extracted all events and changed behaviour element using following definition:

Definition I:

Let d be a DOM object.

ρ(d)=(param1…….paramn)

e(d) is set of events of a particular set of values of types of d.

e(d)=(event1,event2,…..,…eventm)

d.function, d.event is the event which starts d.function.

For each event there is one set

ρ(ele)=(element1,……elementn), set of elements changing dynamically on execution of an event.

4.2.4 COMPUTING EVENT-ELEMENT MAPPING FUNCTION(EEMF)

Mapping function is simulating an environment to map user trigged events with DOM Tree inspector. Mapping function will automatically map user session scenario with updated behaviour of DOM. The mapping function reflexively will select changed events and elements from DOM Tree Inspector and will generate FSM test file based on the execution trace of user session data. By using the collected execution traces of the user session log files, the Finite State Machine of the sample Web Application is constructed where states represent Document Object Model (DOM) instances and the transitions of which represent the effects of callback executions. So finally we are getting a relationship mapping between events and changed behaviour elements. This Event-Element relationship mapping function is generated by following definition.

Definition II:

$\forall xevent(A)$ for all x elements event (A) is true means all x element are dynamically changing on change of event(A).

For event and element relationship using this definition and storing event element relationship in one file with the help of mapping function. Mapping function is used to find HTTP Request Violation. Mapping function provides relationship in request with the corresponding DOM element and from which Event that element is originated. Once we know which element is causing the request, we can analyze the behaviour and decide whether a violation has occurred. Here in mapping function, we intended to solve the issue, we analyzed the DOM in every state and exposed properties of the DOM that must be verified.

5. MODEL GENERATION:

In traditional web application, every web page URL contains a state and represented by a resultant web page, However in AJAX, Internal structure of web page change means state change can be identified only through DOM and changes reflects on that same web page.

In the proposed approach, we exercise on the event performed on the AJAX web application. From these events, we construct Finite state machine of dynamic behaviour of set of user session performed on an Ajax web application. An Overview of our approach is already visualized in Figure1 AFEADM framework (Automatic Fault embedded area detection Model) and processing view of that has shown in figure 2. In this section, we summarize the state machine generation from the extracted important component like User Session Recorder to give changing behavior and then for those changing behavior extraction of Dom change Violation and finally computing Event-Element Mapping relation.

5.1 Inferring state machine:

The State machine is generated in the incremental manner. The algorithm written to infer the state machine is shown in Algorithm 1in figure 8. The start procedure (lines 1-7) takes care of initializing the various components and processes involved. The actual recursive extraction of states, events and element procedure starts at line 10. Here we are considering state machine as directed graph where nodes representing states and edges representing state transitions and output functions. Nodes are labeled with state name, state (idle, changed, etc) and finally nodes representing states of all elements those can change their state and what are the changes reflecting on the corresponding elements dynamically.

Journal of Theoretical and Applied Information Technology
31st August 2013. Vol. 54 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

As an example of generated Finite state machine, Figure 9 depicts the visualization of the state machine of a coffee maker example of AJAX web application. It Illustrate how on a single web page four different states has been processed.

Table1 depicts Finite state machine structure of dynamism of coffee maker web application. In this table, it's showing changes in DOM structure at the time of state change in web application. As in given example, initially in stage 1 web application is in state 'ST_0', both coffee Makers are in 'idle' state. On click on ordercoffee Button, 'onClick'event will execute and run javaScript function 'ordercoffee' and changes will reflect on user interface, state 'ST_1' coffeemaker1 is in Busy state and preparing coffee for user1, This is stage2.

In stage 3 that same time user2 ordered for a coffee. In this stage, state ST_2, both coffee makers coffeemaker1 and coffeemaker2 are in busy state. In this situation, now user3 wants to order for a coffee, but both coffeemakers are in busy state, therefore AssertAlert@user3 means user3 is getting alert, "Both Coffee Makers are busy". This is stage3, but in this stage no DOM change will reflect on web application. Web application is on that same previous state ST_2.

Here in this web application, we have one more stage means state ST_3, when we are getting alert and as well as state of web application is also changing. This time coffee maker1 is in idle state and coffeemaker2 is in busy state. Finally reaching back to root state ST_0, when coffee 2 is ready. Both coffeemaker are again in idle state which is ST_0.

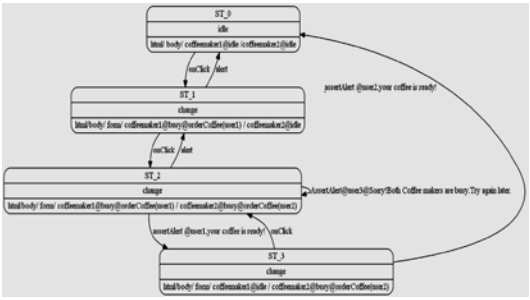

Figure 9: Onclick Element Behaviour State Machine Of Coffee Maker Web Application

6. CASE STUDIES:

To validate proposed approach and to prove that proposed methodology is able to obtain all non functional challenges in practical sense, we applied proposed methodology on various AJAX web application case studies as shown in Table 2.

Our selection criteria used AJAX case sites mentioned in table 2 because these sites changes the state of the application using JavaScript, Assigning events to HTML elements, Changing Dom State at run time on the basis of client request and changes affecting system asynchronously through delta updates. The all sites are open source sites and we have access to their source code. Last two sites are small, But these sites are most useful and having variety of dynamically handled events and having their own importance at the time of modeling the framework.

Table 2: Selected Case Studies And Examples

S.No	Case study Name	Source (url)	Dom size in bytes	Description
1	Filmdb	http://sourceforge.net/projects/ajaxfilmdb/	38914 bytes	AJAX-FilmDB is designed in PHP/ MySQL /AJAX and a film database application.
2	Tudu list	http://tudu.sourceforge.net/ http://www.julien-dubois.com/tudu-lists	24009 bytes	Tudu Lists is an J2EE application for managing todo lists. With Tudu Lists, todo lists can be easily accessed, edited and shared on the Web.
3	AJAX IM	http://ajaxim.com/	34578 bytes	Ajax IM ("Ajax Instant Messenger") is a browser-centric instant messaging framework.
4	Coffee Maker	http://www.headfirstlabs.com/books/hrajax/chapter03/coffee/coffee.html	574 bytes in 7 files	An AJAX powered Coffee maker application
5	Break Neck pizza	http://www.headfirstlabs.com/books/hrajax/chapter02/breakneck/pizza.html	6033 bytes in 5 files	An Ajax powered pizza delivery application

Our First Site FilmDB is Sourceforge.net open source software community provided site. This is built on PHP, MySQL and AJAX. This is a single page site and broadly providing eight features: View, Display, Info, buttons, Navigation, selection, sorting and search without changing web page.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Each feature contains variety of sub features like one feature view provides four sub features: Poster view, Film View, Row View and list view.

Our chosen second Case study ToDo List is again a source forge provided site. Tudu Lists is a J2EE application for managing todo lists of a user. This web application is selected to show how J2EE plate form can be used in AJAX web application.

Third case study is AJAX IM, an instant message framework. This application create a real-time (or near real-time) IM environment that can be used in conjunction with existing community and commercial software, or simply as a stand-alone product.

Our chosen fourth and fifth case studies are simple web application and having short code are head rush book codes and mainly these case studies are having vast role in designing the proposed framework.

7. EXERIMENT PERFORMED

The starting point of our dynamic analysis is set of execution traces, as one set of execution traces shown in Table 3 of TuDu List an AJAX web application that emulates simple To do list. This TuDu List web application allows user to add a new todo list, delete todo list, edit todo list etc and further sub functionalities of each and every feature. User session traces obtained from selenium tool saved log file generated by real user interaction, similar kind of user session traces approach proposed by Elbaum [48].

Table 3: Add Todo User Session Traces Event Sequence For Tudulist AJAX Web Application

Trace Sequence	Event sequence
1	SHOW My ToDos
2	ADD a New List
3	ADD a New List
4	Quick ADD ToDo
5	Quick ADD ToDo
6	ToDo Completed
9	Reopen Todo
10	ToDo Completed
11	Delete Completed ToDos

The states of AJAX web application can be inferred by analyzing Event-Element mapping function formed by dynamically changing behavior of DOM elements and user session traces recorded by selenium tool. The representation of USSMG tool designed state machine for mentioned traces is shown in figure10. Here we are using TuDu AJAX Web application's execution traces to model and present proposed state space reduction methodology to avoid state space explosion problem.

8. RESULT AND EXPERIMENTAL OUTPUT:

All User session, DOM Inspector analyzed and generated State machine. Generated coffee Maker state machine of set of user session is shown in figure9. Figure10 represents generated state machine of ADD ToDo user session of TuDu list case study.

Table 4 depicts results of designed framework on various case studies. Table 4 presents detected events, changing behavior DOM elements, Detected States and tags those are changing their value on the basis of retrieved user sessions.

As defined in Framework, the proposed methodology is designed to automate the fault embedded area detection and catering various non functional challenges. Goal of experimental result is also to validate results with respect to these challenges. Fault seeding and Test case coverage are results of whole work, therefore discussed in section results. Here we are validating result of non functional challenges:

8.1 NF1: Effectiveness

To check effectiveness of the designed framework, we applied this complete process on 5 sites. Out of all 5, first 3 are successfully running web applications and last two are small but containing different ways of showing AJAX based dynamism in web applications. Figure11 is comparison graph of events and Figure12 is comparison graph of states and these graph shows that system is able to retrieve 90% states as expected. Even we applied this complete process on various other small size case studies as Board 'R' Us, Top 5 CD Listing and well known on faceCart Ajax shopping cart also. For View module of case study1 18 states are detected out of 20 states and Total number of events was 16 and detected events

Journal of Theoretical and Applied Information Technology
31st August 2013. Vol. 54 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

are 15. In case study 2 in three modules- Add, Edit and Delete Number of able to retrieve all events as expected but not able to form all states, it is generate 30 states out of 35 states. Our chosen third case study, reacted in a opposite manner, as number of events were less in this and every time DOM was changing, So this retrieved more number of states as expected. In case of fourth and fifth case study, it's fetching all events and all number of states.

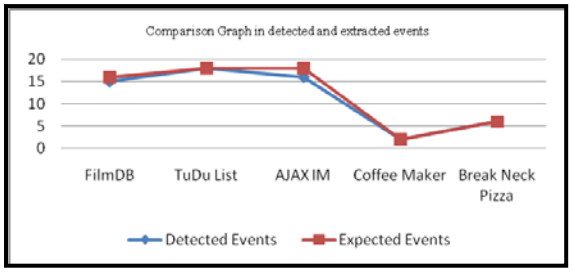

Figure11: Comparison in detected and extracted events

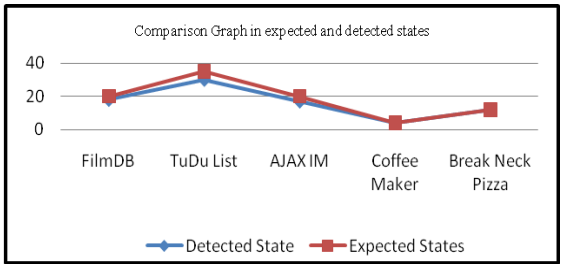

Figure12 Comparison in detected and extracted States

8.2 NF2: Scalability

Proposed approach is capable to achieve any number of states due to incremental approach applied to generate state machine. To validate scalability in proposed methodology, size of case studies vary from 500 -15000 bytes. Figure 13 size Graph shows size variation in selection of case studies.

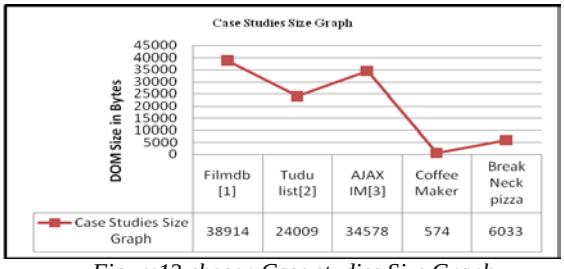

Figure13 chosen Case studies Size Graph

8.2 NF3: Flexibility

We have chosen case studies developed on variety of platforms. Case study 2 is a J2EE application, this application case study consist of server side part written in J2EE and client side portion written in Java script. Whereas all other case study are in PHP and java script. Proposed methodology is able to design correct state machine for both kind of application.

9. CONCLUSION:

In this research paper, AJAX application testing is directed towards revealing faults related to incorrect manipulation of DOM. Therefore, aim of this research paper was to design a state machine which will be able to detect all states of AJAX web application whenever there web application behaviour will change. Result have shown that we are able to detect all events and Dom changing elements of user sessions and successfully able to detect all states and on the basis of events able to form state machine of chosen web application. We detected event, elements and state using user session tool and DOM manipulation tool and validated results also at each and every step. A prototype tool is also developed to apply this process on multiple case studies. We applied tool on 5 case studies and able to achieve correct and efficient results. Till now, we applied this process separately on modules of chosen WA. Further we will apply this on whole web application and plug in this prototype tool with testing tool and reveal faults related to incorrect manipulation of the DOM for AJAX web applications.

REFRENCES:

[1] Garrett, J., Ajax: a new approach to web applications", Adapt. Path,2005, http://adaptivepath.com/ideas/ajax-new-approach-.web-applications,November 2011

[2] Sebestien Salva, Patrice Laurencot: Automatic Ajax Application Testing. ICIW 2009:229-234

[3] R. Binder. State-based testing. Object Magazine, July-Aug 1995.

[4] C. D. Turner and D. J. Robson. The state-based testing of object-oriented programs. IEEE Conference on Software Maintenance (ICSM), September 1993.

[5] *Jayaratna, N. "Understanding and Evaluating Methodologies: NIMSAD, A Systematic Framework," McGraw-Hill Companies, London, pp.288, 1994.*

[6] Sreedevi Sampath, Valentin Mihaylov, Amie Souter, and Lori Pollock. Composing a framework to automate testing of operational web-based software. In Proceedings of the 20th

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IEEE International Conference on Software Maintenance (ICSM), pages 104–113, Washington, DC, USA, September 2004. IEEE Computer Society.

[7] S. Sampath, V. Mihaylov, A. Souter, and L. Pollock. A scalable approach to user-session ased testing of web applications through concept analysis. In Proceedings of the Automated Software Engineering Conference, September 2004.

[8] S. Sampath, A. Souter, and L. Pollock. Towards defining and exploiting similarities in web application use cases through user session analysis. In Proceedings of the Second International Workshop on Dynamic Analysis, May 2004.

[9] Sara Sprenkle, Holly Esquivel, Barbara Hazelwood, and Lori Pollock. WebVizOr: A visualization tool for analyzing test results of web applications. Technical Report 2006-335, University of Delaware, 2007.

[10] Sara Sprenkle, Emily Gibson, Sreedevi Sampath, and Lori Pollock. Automated replay and fault detection for web applications. In Proceedings of the International Conference on Automated Software Engineering (ASE), pages 253–262, New York, NY, USA, November 2005. ACM Press.

[11] Anneliese Andrews, Jeff Offutt, and Roger Alexander. Testing web applications by modeling with FSMs. Software Systems and Modeling,4(2):326–345, April 2005.

[12] Marchetto, A., Tonella, P., and Ricca, F. (2008b). State-based testing of Ajax web applications. In Proc. 1st IEEE Int. Conference on Sw. Testing Verification and Validation (ICST'08), pages 121–130. IEEE Computer Society.

[13] Marchetto, A., Ricca, F., and Tonella, P. (2008a). A case study-based comparison of web testing techniques applied to ajax web applications. Int. Journal on Software Tools for Technology Transfer, 10(6):477–492.

[14] www.graphviz.org/

[15] http://headfirstlabs.com/books/hrajax/

[16] http://sourceforge.net/projects/tudu/

[17] http://www.julien-dubois.com/tudu-lists

[18] http://sourceforge.net/projects/ajaxfilmdb/

[19] http://www.ajaxim.com

[20] F. Ricca, P. Tonella, Analysis and testing of Web applications, in: Proceedings of the International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos (CA), 2001, pp. 25–34.

[21] S. Elbaum, S. Karre, and G. Rothermel. Improving Web application testing with user session data. In Proceedings of the 25th International Conference on Software Engineering (ICSE), pages 49.59, Portland, USA, May 2003. IEEE Computer Society.

[22] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher. Leveraging user session data to support web application testing. IEEE Transactions of Software Engineering, 31(3):187. 202, March 2005.

[23] S. Park and G. Kwon. Avoidance of state explosion using dependency analysis in model checking control flow model. LNCS, 2006.

[24] Valmari, A.: The state explosion problem. In: Lectures on Petri Nets I: Basic Models. Lecture Notes in Computer Science, vol. 1491 pp. 429–528. Springer, Berlin (1998)

[25] Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Publishers, Cambridge (1999)

[26] http://www.openqa.org/selenium/

[27] Prasanth Yalla, Dr. L S S Reddy, M.Srinivas, T.Subha Mastan Rao. Framework for Testing Web Applications using Selenium Testing tool with respect to Integration Testing. IJCST Vol. 2, Issue 3, September 2011.

[28] www.w3.org.

[29] http://jgrapht.sourceforge.net

[30] http://docs.oracle.com/javase/1.4.2/docs/api/javax/swing/JTree.html

[31] http://jtidy.sourceforge.net

[32] Dallmeier, V., Lindig, C., Wasylkowski, A., Zeller, A.: 'Mining object behavior with ADABU'. Proc. Fourth Int. Workshop on Dynamic Analysis (WODA), Shangai, China, 2006

[33] T. Ball, D. Hoffman, F. Ruskey, R. Webber, and L. White. State generation and automated class testing. Software Testing, Verification and Reliability (STVR), 10(3):149.170, July-Aug 2000.

[34] Marchetto A, Tonella P (2010). Using search-based algorithms for Ajax event sequence generation during testing.

[35] Marchetto A, Tonella P (2009) Search-based testing of ajax web applications. In: Proc. of IEEE international symposium on search based software engineering (SSBSE). IEEE Computer Society, Windsor, pp 3–13

[36] Mesbah, A., Bozdag, E., and van Deursen, A. (2008). Crawling Ajax by inferring user interface state changes. In Proc. 8th Int. Conference on Web Engineering (ICWE'08), pages 122–134. IEEE Computer Society.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[37] A. Mesbah and A. van Deursen. Invariant-based automatic testing of Ajax user interfaces. In Proceedings of the 31st International Conference on Software Engineering (ICSE'09), pages 210–220. IEEE Computer Society, 2009.
Arie van Deursen and Ali Mesbah. Research Issues in the Automated Testing of Ajax Applications. In Proceedings 36th International Conference on Current Trend in Theory and Practice of Computer Science (SOFSEM), pp. 16-28. Lecture Notes in Computer Science 5901, Springer-Verlag, 2010.

[39] Anuja Arora , Madhavi Sinha, "Applying variable chromosome length Genetic Algorithm for testing Dynamism of Web Application,3rd International Conference on Recent Trends in Information Technology, Proceeding in IEEE Xplore, July 2013 ISBN : 978-1-4799-1024-3/13 25-27 pp 539-545.

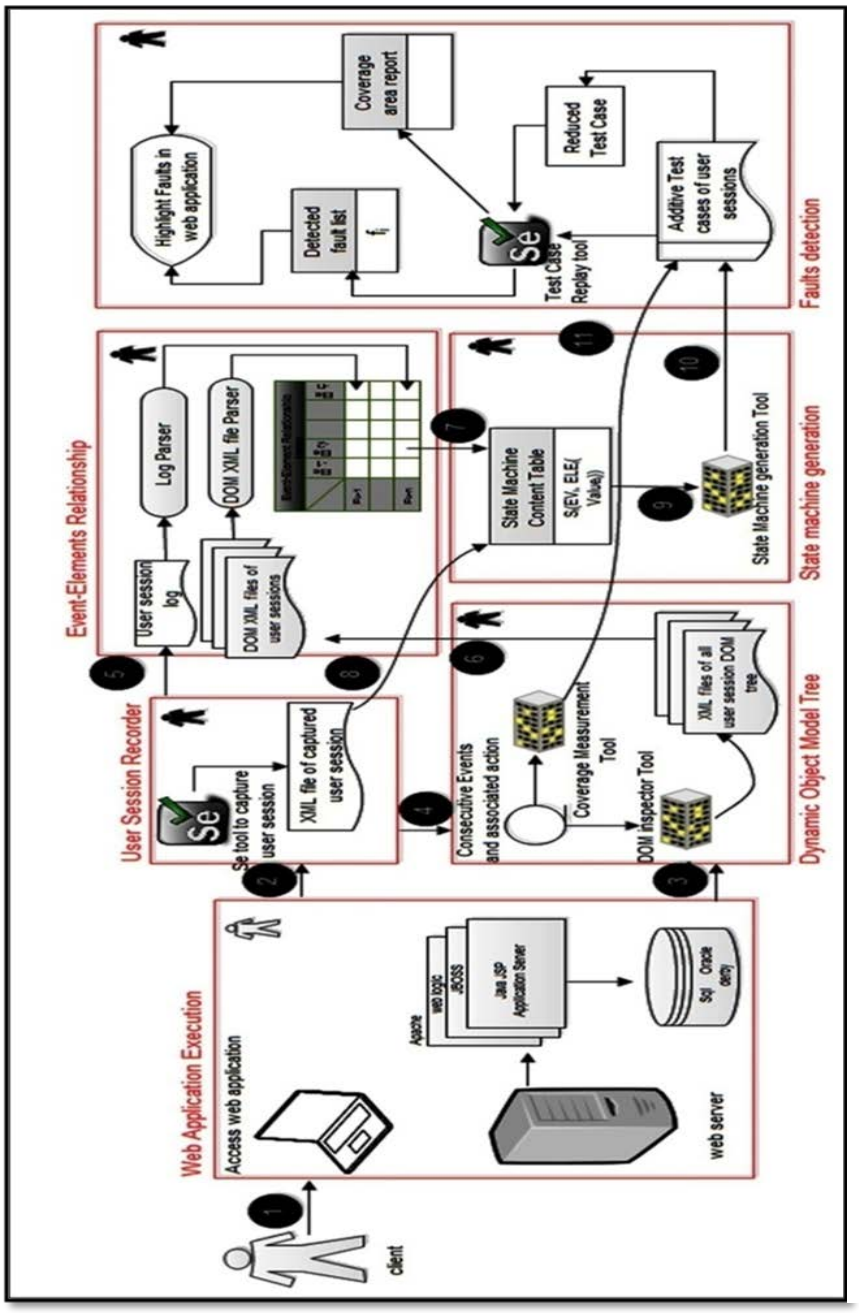

Figure 1: Automatic Fault embedded area detection model (AFEADM) framework for Testing AJAX Web Application

Journal of Theoretical and Applied Information Technology
31st August 2013. Vol. 54 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

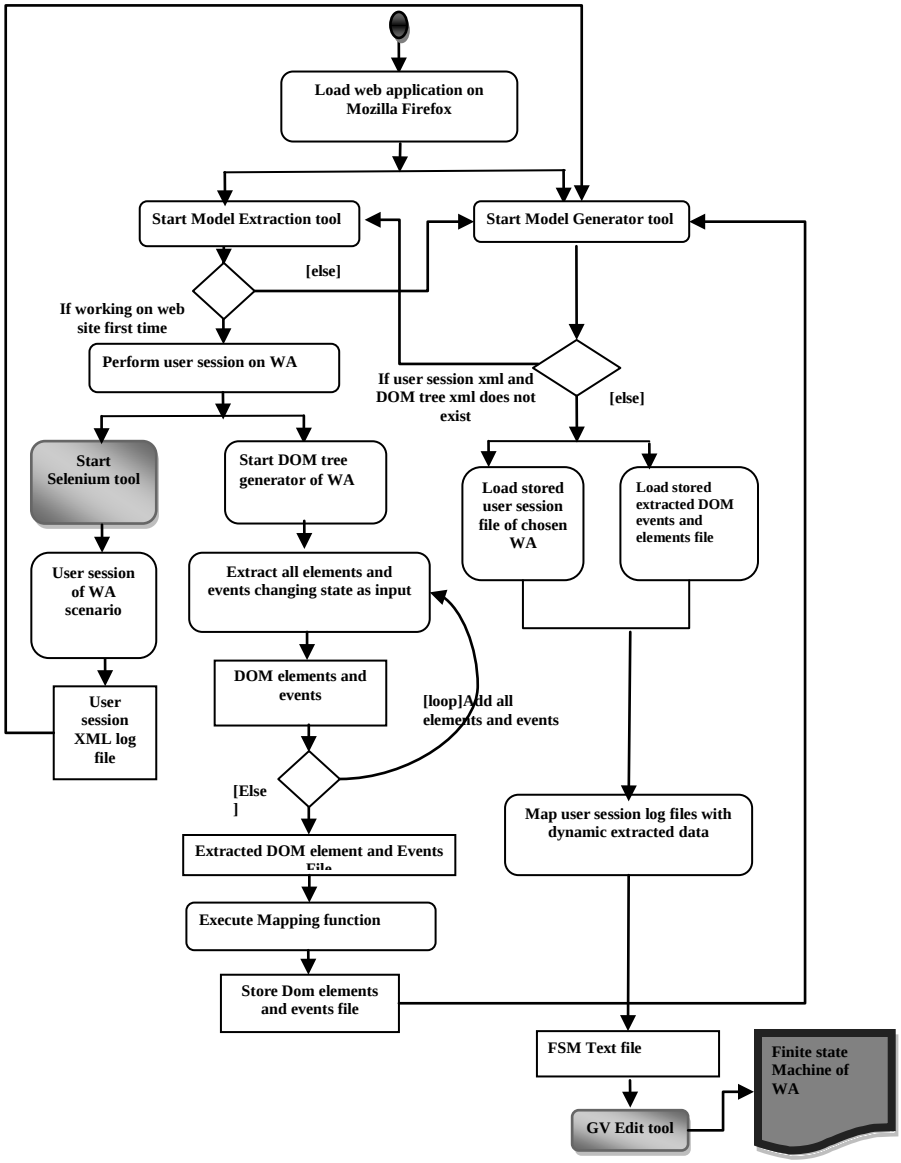

Figure2: Processing view of Finite State Machine Generation of Web Application

ISSN: 1992-8645 E-ISSN: 1817-3195

Figure8: Algorithm for inferring State Machine of an AJAX Web Application

Algorithm 1: Inferring State Machine
Input: Ajax web Application
Output: All recorded user session XML files; DOM tree inspector files for set of DOM in for all User Sessions; Event-Element Mapping Function generated Event-Element relationship metrics; State machine of dynamically changing behaviour.

```
1. procedure Start(url)
2. Browser     initialize browser(url)
3. rsm←  initialize rootStateMachine()
4. statersm ← initialize State()
5. Navigate(rsm)
6. SaveAndcreate(selog.xml)            //Save and create user session log file
7. end procedure
8.
9. procedure infersm(url, selog)                  //Event-Element mapping function
10. domrsm ← browser.fetchDom(selog.rsm)
11. se= parse(selog)
12. extract(seCommand)
13. while(selog!=EOF)
                // Store DOM for all selenium tool given command
14.     StoreAndSave domcs[ ]=browser.fetchDom(selog.secommand)
15.     if(Diff(domrsm,domcs[0])>0)
16.             rsm.addState(domcs[0])
17              Extract element.list(changed tag.list, tag.value)
18.             else
19.             domrsm=domcs
20.     endif
20.     for(domcs.length)
21.             if(Diff(domcs(i), domcs(i+1))> 0
22.                     rsm.addState(domcs)
                // If DOM Change from  client side means User did some action
23.             If(selog.secommand= “action”
24.                             event<-getselog.secommand
25.                             domcs.statename = ST_(name of secommand)
26.                             rsm.addEdge(event)
27.                     Extract ElementList(TagName, TagValue, TagIndex)
                        //Server side changes reflect on WA
27.             elseif(selog.secommand= “AssertAlert”)
28.                             rsm.addEdge(event@seTarget)
29.                     Extract ElementList(TagList, TagValue, TagIndex)
30.             endif   endif   endif
33.     endfor
34. endloop
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Extracted Structure Of Generated Coffee Maker Onclick Event State Machine

S No	Initial State	Final State	Event Performed	Changes in DOM	State
1	ST_0	ST_0	onLoad()	ST_0: html/body/div[4]@Idle, html/body/div[8]@Idle	Coffeemaker1 Idle, Coffee maker2 Idle
2	ST_0	ST_1	onClick()	ST_0: html/body/div[4]@Idle, html/body/div[8]@Idle ST_1: html/body/div[4] @ Brewing USER1's small mocha, html/body/div[8]@Idle	Coffeemaker1 Busy, Coffee maker2 Idle
3	ST_1	ST_0	Assert Alert()	ST_1:html/body/div[4]@Brewing User1's small mocha, html/body/div[8]@Idle ST_0: html/body/div[4]@Idle, html/body/div[8]@Idle	Coffeemaker1 Idle, Coffeemaker2 Idle
4	ST_1	ST_2	OnClick()	ST_1:html/body/div[4]@Brewing User1's small mocha, html/body/div[8]@Idle ST_2:html/body/div[4]@Brewing User1's small mocha, html/body/div[8]@Brewing User2's small mocha	Coffeemaker1 Busy, Coffeemaker2 Busy
5	ST_2	ST_3	AssertAlert()	ST_2: html/body/div[4]@Brewing User1's small mocha. html/body/div[8]@Brewing User2's small mocha ST_3: html/body/div[4]@Idle, html/body/div[8]@Brewing User2's small mocha.	Coffeemaker1 Idle, Coffeemaker2 Busy
6	ST_2	ST_1	AssertAlert()	ST_2: html/body/div[4]@Brewing User1's small mocha. html/body/div[8]@Brewing User2's small mocha ST_1:html/body/div[4]@Brewing User3's small mocha, html/body/div[8]@Idle	Coffeemaker1 Busy, Coffeemaker2 Idle
7	ST_3	ST_2	onClick()	ST_3: html/body/div[4]@Idle, html/body/div[8]@Brewing User2's small mocha. ST_2: html/body/div[4]@Brewing User3's small mocha. html/body/div[8]@Brewing User2's small mocha	Coffeemaker1 Busy, Coffeemaker2 Busy
8	ST_3	ST_0	AssertAlert()	ST_3: html/body/div[4]@Idle, html/body/div[8]@Brewing User2's small mocha. ST_0: html/body/div[4]@Idle, html/body/div[8]@Idle	Coffeemaker1 Idle, Coffeemaker2 Idle

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

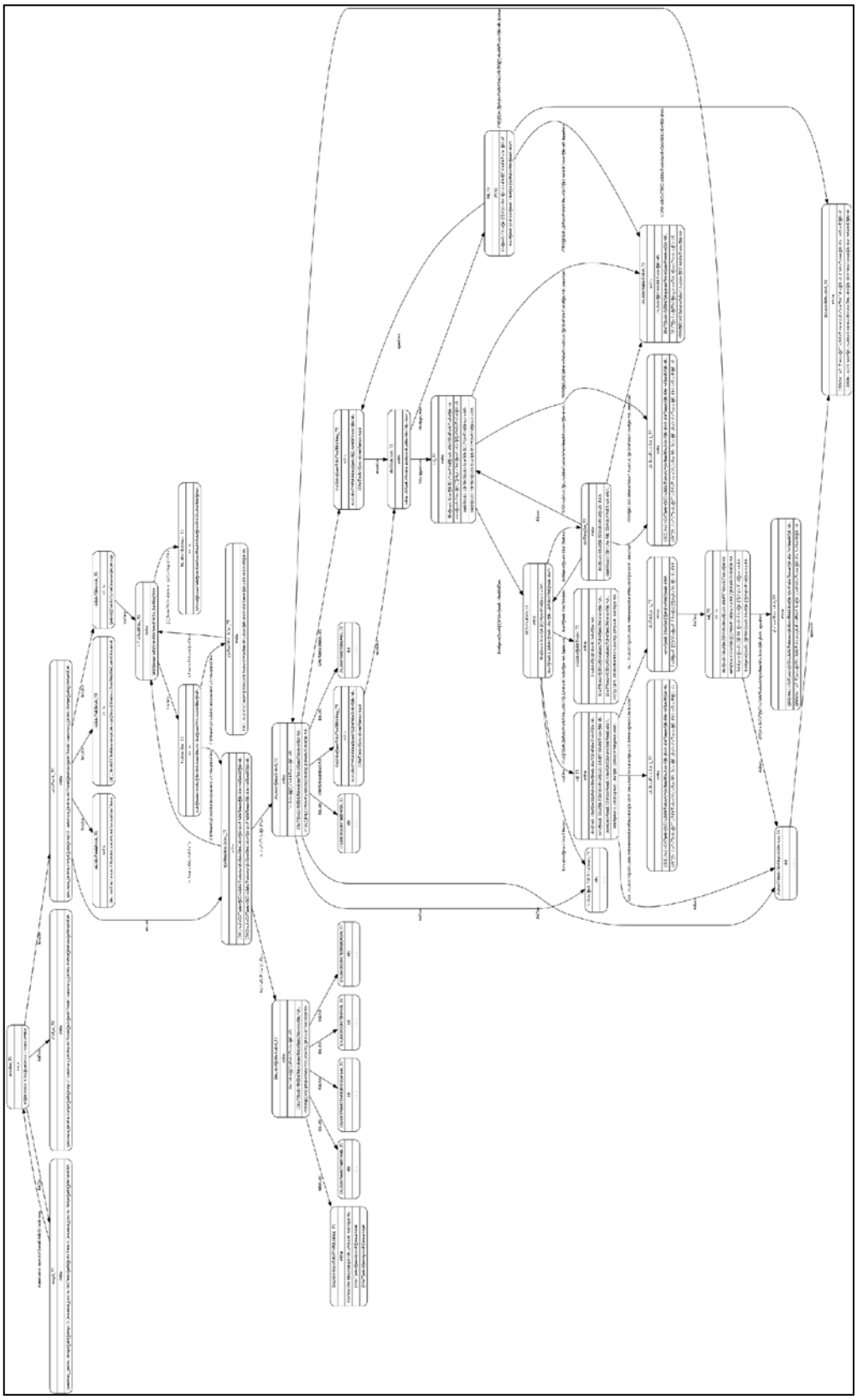

Figure 10: USSMG tool generated State Machine of user session traces mentioned in Table3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 4 Results Of Generated State Machine Of Various Web Applications

Case Study	Case Study Name	User session features	DOM String Size/ User Session	Detected Events	Expected Events	Detected DOM Changed Elements	Detected States	Expected States
1	FilmDB	View	2000 bytes	23	25	48	41	47
2	TuDu List	Add, Edit, Detete	15054 bytes	32	32	63	63	65
3	AJAX IM	Send Message	10356 bytes	16	30	30	20	15
4	Coffee Maker	Order Coffee	574 bytes in 7 files	2	2	16	4	4
5	Break Neck Pizza	Order Pizza	6033 bytes in 5 files	10	10	27	23	23