Asian Journal of Information Technology 15 (20): 3905-3911, 2016
ISSN: 1682-3915

A Bot Driven Framework for Testing Web Applications

K. Bhanu Sai Prasanth and G. Krishna Mohan
Department of CSE, KL University, Vaddeswaram, Guntur, India

Abstract: Dynamic web applications (web-apps) driven by various frameworks and AJAX depend on HTTP's asynchronous state-full communications between client and server. Handling dynamic modifications of the client side DOM tree with respect to user interactions is at the core of dynamic web-apps. This feature on a very basic level not just makes them unique but also efficient and flexible by avoiding page reloads to fetch data from server but compared to conventional web apps, additionally it is more inclined to bugs and crashes thus makes it very harder to test the quality of these web-apps. We propose a technique for testing dynamic web-apps naturally, using a light weight bot program driven by Selenium framework to deduce a state-flow nomogram for all user interface states. We distinguish between normal Create, Read, Update and Delete (CRUD) faults versus an AJAX based faults that can usually occur in user eventful states by generating a fixed DOM-tree oracles to handle the bugs. Our methodology upon actualization can serve as an application-specific state test suite generator and validate-or and highlight the level of automation of our testing approach achieves using test bots on dynamic web-apps with minimum human intervention.

Key words: Selenium, ajax, automated testing, dynamic web applications, user event testing, bots

INTRODUCTION

A bug database is a sub storage entity of software repositories for storing bug information and resolution status obtained by users and testers alike and is a critical component in developing quality applications. Since software development firms spend >45% of expense in rectifying bugs (Presan, 2010), we can conclude that they are inescapable and inevitable and settling them is as expensive as developing them. Expansive software programming ventures maintain bug archives to bolster data gathering for further version revisions and to help developers resolve them (Breu *et al*., 2010), (Fitzgerald, 2006).

There is a developing pattern to move applications towards the web. Some firms incorporate googl's mail, drive features for organization specific mailing domains and office utility tools like spreadsheets, word editors and to-do list calendar applications. The explanations behind this transition to the web are multi-fold and we list some of them:

- The web is not static with the advent of cloud computing the web is dynamic and can be accessed as our own local physical machine
- No installation and rigorous maintenance procedures for end-clients
- Automated updating and utilization of the latest software versions to all entitled clients, hence diminishing support costs
- Widespread access to the application as well as to the client information from any browser any location on any machine with Internet access

Throughout the wonderful benefits of these dynamic web applications is the use of "Asynchronous JAVASCRIPT and XML", in short AJAX. With AJAX, the client side entity such as a browser can offer dynamic page, data navigations through an event driven retrieval of HTML contents and supports enhanced collaboration by means of powerful user interface segments. While the utilization of dynamic web-apps absolutely influences ease of use and propels rich user interactions (Mesbah and Deursen, 2008) it includes some significant pitfalls since these applications are famously error prone because of:

- Their state-full, event based asynchronous mode of communication using dangling and spaghetti JAVASCRIPT
- The run time manipulation of client side(browser) Document-Object Model (DOM)
- The utilization of delta-correspondence between request seeker and response provider (Mesbah and Deursen, 2008)

Corresponding Author: K. Bhanu Sai Prasanth, Department of CSE, KL University, Vaddeswaram, Guntur, India

Keeping in mind the end goal to enhance the reliability of dynamic web-apps, static code analysis or testing procedures can be initialized although both approaches have certain draw backs. Lamentably, static code analysis methods cannot support a significant number of the dynamic conditions present in dynamic web applications. Besides, prior web testing tools depend solely on the established request seeker and response provider model, not considering client(browser) side configurations for getting data faster without page reloads regardless these tools seek a significant measure of manual endeavor's from a tester. Innovation of advanced testing tools such as Selenium, provides a capture event-and-replay model of testing taken after Macros for advanced web applications. While such testing tools are fit for generating and executing test oracles for dynamic web applications.

The objective of this study is to bolster robotized testing of dynamic web-apps using selenium driven state aware bot scripts. To that end we propose a methodology in which we naturally infer a model of the client interface (UI) conditions of an application. We get this model by "slithering" the application using a bot and initiating automated inputs into UI data elements, actionable events such as clicks, scroll UI-components such as buttons etc thus upholding and validating client side functionality. With a specific end goal to perceive faults in these executions we propose utilization of bots that have defined properties of either the client side DOM-tree or the inferred state machine parameters that ought to hold good for any real time play out. These bots can be non specific or generic and will vary according to the UI and DOM of each HTML page subjected to testing.

Literature review: Present day web interfaces consolidates client (browser) side scripting (JavaScript) and clien's run time DOM modifications which are progressively isolated from server-side application domain (Stepien *et al*., 2008). Despite the fact that the field of rich web UI testing is fundamentally uncharted, a wealth of information might be obtained from two relatively close fields of software testing such as: conventional web testing using static code analytics and GUI testing.

Conventional web testing: Builds veriweb, a prototype for automated exploration of sites with hierarchical links using a web crawler and identifier for variations from the normal behavior of a link (dead or alive) often termed navigational failures. Veriweb utilizes mined smart templates to derive data values for input form based pages which has inspired our work to some extent. In spite of the fact that Veriweb's web crawler implementation has some backing for client side event handling, the study gives inadequate points of interest to figure out if it can be adapted to cutting edge modern dynamic web applications. VeriWeb offers no backing for producing test suites.

Testing tools like SecuBat (Kals *et al*., 2006), WAVES (Huang *et al*., 2005) have been proposed for surveying a web application's security aspects. The general methodology again utilizes a web crawler equipped for identifying potential information section leaks which can be exploited by malicious parties. Pernicious exploiting tools like SQL injection and XSS vulnerabilities are then infused into these section leaks identified earlier to deduces counter measures based on server output which is system scrutinized to fix section leaks of the web application.

Initiating static code analytics of server-side execution to comprehend the application functionality is another software validation methodology. Artzi proposed a procedure using a tool called apollo for discovering shortcomings in PHP web applications that depends on consolidated concrete and typical execution. Apollo can recognize run-time faults and distorted HTML content lacking syntax. Halfond and Orso (2007) present their static code analytics of servlets, JS's to fetch requests parameters and their values from client-server operations. They utilize (Halfond *et al*., 2009) typical execution of server-side code to distinguish conceivable interfaces of web applications. Such systems have confinements in uncovering bugs that are because of the complex browser side runtime functionality of dynamic web apps. Figure 1 illustrates the procedures of this approach.

Although (Ricca and Tonella, 2001) were the first one's to suggest model-based testing approach for web apps using Unified Modelling Language (UML) design centric tool called ReWeb for making a model of the web application which is utilized along-side complete scope criteria specification to render test-cases. Alfaro (2001) inspired from these approaches utilizing his testing tool called MCWEB applied model check analytics to validate quality of web apps. Their research, be that as it may was suitable mostly to standard CRUD based web apps with out AJAX dynamisms (Fig. 2).

Andrews *et al*. (2004) exhibited another methodology which depends on a finite-state automaton which in combination with functional requirements by the tester vaidates an application. All such model-construct testing

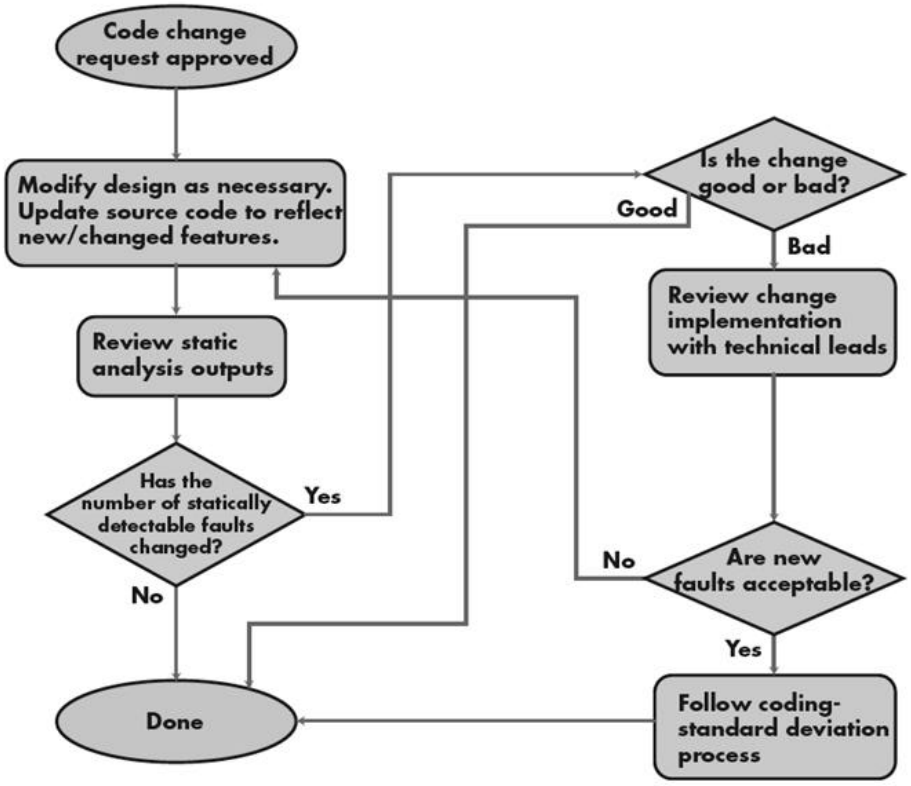

Fig. 1: Typical flow of operations used in static code analytics during application validations

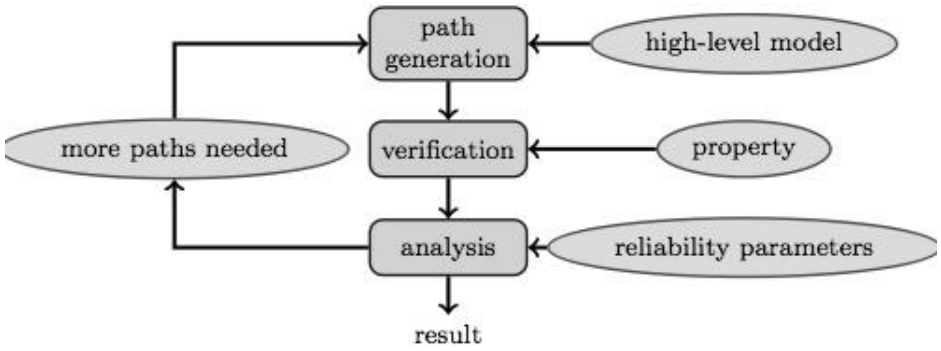

Fig. 2: Typical flow of operations used in mode-checking analytics during application validations

procedures center with respect to established multi-level CRUD based web applications. They generally utilize a crawler to gather a navigational model of the web. Tragically, conventional web crawlers are not ready to slither dynamic web-apps (Mesbah *et al*., 2008).

Logging client session information on the server is additionally utilized for obtaining testing automation (Elbaum *et al*., 2005), (Spxenkle *et al*., 2005). This methodology requires adequate cooperation of genuine real world web clients with the framework to create the fundamental logging information. Session-based testing methods are only centered around synchronous solicitations to the server and do not have the complete state data required for dynamic web-app testing. server responses (Mesbah and Deursen, 2008) to an event are difficult to break down all alone. The greater part of such redesigns get to be significant after they have been handled by the client side browser based plugin and infused into their DOM.

GUI (Desktop) applications testing: Memon (Marchetto *et al*., 2008) suggested output based input

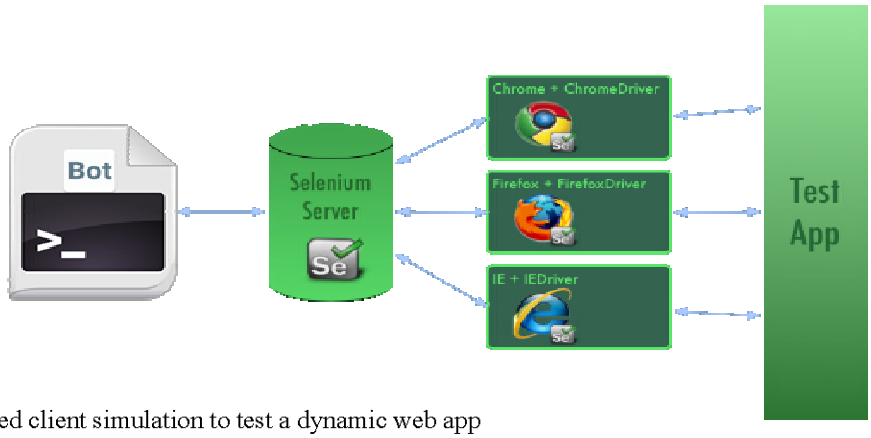

Fig. 3: Robotized client simulation to test a dynamic web app

often called reverse engineering process to figure out a model of already working gui app with a specific end goal to create test cases. Dynamic applications can be seen as an evolution of both desktop and web applications, since the UI is desktop app gui components like buttons, textboxes etc and another one being event based communication (Mesbah and Deursen, 2008). Be that as it may, dynamic web apps have particular features, for example, the asynchronous request seeker/response provider communications and run-time DOM manipulations which make them unique in relation to conventional GUI applications and along these lines definitely requires more evolved testing tools to handle these unique aspects.

MATERIALS AND METHODS

System model: The server-side of dynamic web apps can be tried with any traditional testing procedure. On the client side, testing can be performed at various levels. Unit testing frameworks such as js unit can be utilized to test JAVASCRIPT on a practical level. The most frequently utilized dynamic web app testing tools all have the same philosophy that is to catch/replay human interactions with the application with all possible set of values and ranges for example, WebKing and Sahi which permit DOM-based validations by capturing user actions during a request/response phase. These tools requires a generous measure of manual work load by a tester to generate a test suite, since each action trail and the corresponding DOM manipulations must be composed by the tester in the form of assertions.

Our bot based tool takes an alternate methodology: as opposed to being a javascript component running inside a web browser we utilize a wrapping component generated by a programmable script (Bot) in combination with Selenium framework gives Selenium API's to control the web browser to simulate virtual client while interacting with the dynamic web application in test. Our methodology is an efficient method to implement a robotized client simulation for validating dynamic web-apps that can replicate a genuine client actions on the web interface and inducing an event assertion model dynamically with ease. The following Figure describes the architecture (Fig. 3).

Proposed work

State-flow nomogram: The proposed slithering process using a bot on a dynamic web app fetches key elements of app pages JavaScript and html code and categorizes action snippets into two groups. One that cannot change states within the clients'(browser) DOM tree and one that can. We deduce a state-flow nomogram based on these snippet behaviors in influencing DOM tree modifications by capturing states of the user interface along with all their transitions based on all possible user events and values (Fig. 4).

Algorithm 1:
Crawling process with per/post crawling hooks
procedure start (url, set tags)
browest←init embedded browest(url)
robot←init robot()
sm←init state machine()
pre crawling plugins(browest)
crawl(null)
post crawling plugins(sm)
end procedure
procedure CRAWL (state ps)
cs←sm. Get current state()
Δupdate←diff(ps, cs)
f←analyse forms (Δupdate)
Set C←get candidate clickables (Δupdate, tages, f)
for c∈C do
generate even(cs, c)
emd for
end procedure

An algorithm to deduce the nomogram is described and invocation flow is depicted as.

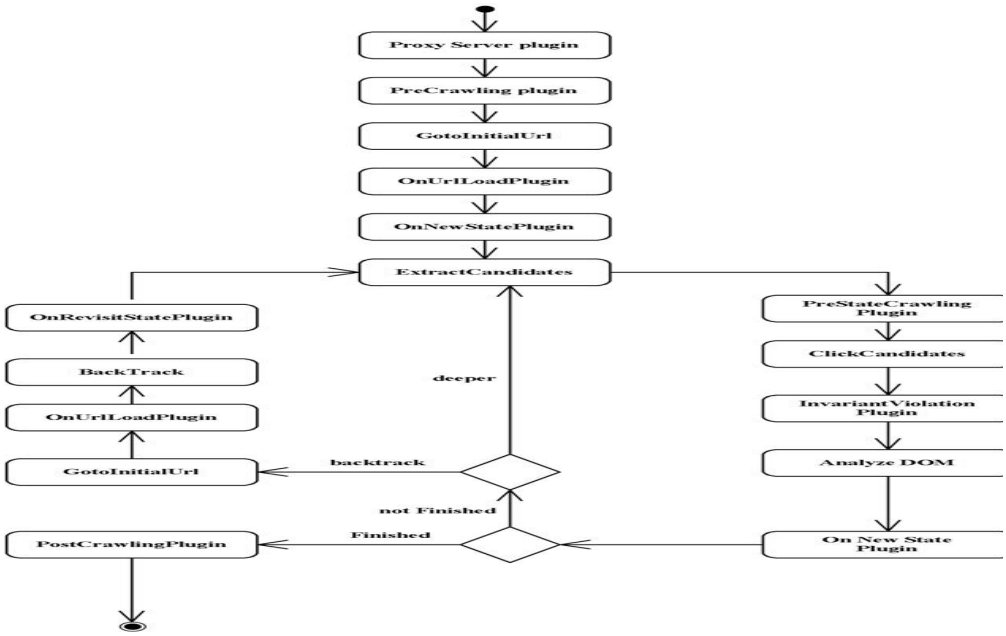

Fig. 4: Bot invocation and subsequent validation process

The proposed bot is capable of deriving a state machine from the dynamic web app using the above nomogram. which is contracted and inferred by the bot for a specific page that is being tested. The nomogram inference involves opening the dynamic web app in a selectively and programmatically chosen web browser, the investigating the DOM-tree and obtaining possible UI elements to trigger action events upon those and identifying subsequent UI state changes by listening to AJAX state transformations.

We implement these analytics and element navigations methods iteratively for all pages and for all states within the web application. One example being providing a series of random values of various types to input fields in the form when no pre-defined data is available. The bot is extended to serve various options for controlling the slithering phase.

RESULTS AND DISCUSSION

We design the implementation on a standard machine with the configurations intel(R) pentium(R) D CPU 2.8 Ghz, 2GB RAM. 32-bit windows 7 operating system. There are variations across state space acquisitions of a small scale and an enterprise dynamic web app. It might be in normal ranges for small scale apps but is huge for enterprise apps and can sometimes lead to state explosion problem and subsequent browser crashes. To handle the state explosion problem, we provision the bot with the ability to set configurable options such as specifying the following:

- Maximum link depth extraction level
- Similarity margins for different state comparisons
- Maximum number of states that can be inferred per page/domain
- Slithering process and it's duration threshold
- Link filtrations for cross domain using pre-defined regular expressions

Among the above specified options, the potential component that can influence performance with respect to scalability is the slithering process. The efficiency of slithering an entire web app site relies on many external issues such as:

- The rate at which the server responds
- The rate at which client(browser) side JAVASCRIPT based on the servers response can update the interface
- The size of the DOM tree

Application size: Our dynamic web app prototype approximately contained 12 pages of html rendering server side code containing at least 15,000 lines of ui, database, ajax, JAVASCRIPT, application codes and possessing around thousands of dynamic DOM states for ajax pages themselves. Dynamic analysis results compared to static code analysis of prior approaches are obviously independent of the size of the code which wont be a detrimental factor for our current approach. A better performance indicator is the number of dynamic states

inferred during the nomogram construction which depends on size of the DOM-tree along with amount of memory available. nomogram inference states can be estimated by the equation which is size of (memory) to 3 times the size of (DOM).

According to our demonstration on the above mentioned enterprise dynamic web application the average size of the DOM is approximately 0.20 MB. On the machine with resources mentioned above this would produce approximately 5000 states, that is suitable for most real world dynamic web applications without crashing the web browser.

CONCLUSION

In this study we have proposed a technique for automated qualitative analysis of dynamic web applications. Our present research comprises of expanding the a bot program considerably to support automated testing using a modular progression based robotized client to generate and initiate a test suite. To outline the contributions:

- Test models generation using DOM analysis
- State flow estimations using nomogram
- Test suite implementation using automated bot client using web browser driven by selenium

An observational assessment of an enterprise application, uncovering the bot's abilities and it's adaptability on the level of automation achieved.

RECOMMENDATIONS

Our future work will incorporate directing further contextual analyses and the improvement of additionally testing modules for detecting development bugs and security hacks or loopholes in dynamic web apps.

REFERENCES

Alfaro, L.D., 2001. Model Checking the World Wide Web?. In: International Conference on Computer Aided Verification. Berry, G., H. Comon and A. Finkel (Eds.). Springer Berlin Heidelberg, Berlin, Germany isBN: 978-3-540-44585-2, pp: 337-349.

Andrews, A., J. Offutt and R. Alexander, 2004. Testing web applications by modeling with FSMs. Software Syst. Modeling, 4: 326-345.

Breu, S., R. Premraj, J. Sillito and T. Zimmermann, 2010. February Information needs in bug reports: Improving cooperation between developers and users. Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, February 06-10, 2010, ACM, New York, USA. isBN: 978-1-60558-795-0, pp: 301-310.

Elbaum, S., G. Rothermel, S. Karre and M. Fisher, 2005. Leveraging user-session data to support web application testing. IEEE. Trans. Software Eng., 31: 187-202.

Fitzgerald, B., 2006. The transformation of open source software. MIS Q., 30: 587-598.

Halfond, W.G. and A. Orso, 2007. Improving test case generation for web applications using automated interface discovery. Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, September 3-7, 2007, ACM, New York, USA. isBN: 978-1-59593-811-4, pp: 145-154.

Halfond, W.G., S. Anand and A. Orso, 2009. Precise interface identification to improve testing and analysis of web applications. Proceedings of the 18th International Symposium on Software Testing and Analysis, July 19-23, 2009 ACM, New York, USA. isBN: 978-1-60558-338-9, pp: 285-296.

Huang, Y.W., C.H. Tsai, T.P. Lin, S.K. Huang, D.T. Lee and S.Y. Kuo, 2005. A testing framework for Web application security assessment. Comput. Networks, 48: 739-761.

Kals, S., E. Kirda, C. Kruegel and N. Jovanovic, 2006. Secubat: A web vulnerability scanner. Proceedings of the 15th International Conference on World Wide Web, May 22-26, 2006, ACM, New York, USA. is BN:1-59593-323-9, pp: 247-256.

Marchetto, A., P. Tonella and F. Ricca, 2008. State-based testing of Ajax web applications. Proceedings of the 2008 1st International Conference on Software Testing, Verification and Validation, April 9-11, 2008, IEEE, Lillehammer, Norway isBN: 978-0-7695-3127-4, pp: 121-130.

Mesbah, A. and A.V. Deursen, 2008. A component-and push-based architectural style for ajax applications. J. Syst. Software, 81: 2194-2209.

Mesbah, A., E. Bozdag and A.V. Deursen, 2008. Crawling Ajax by inferring user interface state changes. Proceedings of the 8th International Conference on Web Engineering ICWE'08, July 14-18, 2008, IEEE, Yorktown Heights, New Jersey isBN: 978-0-7695-3261-5, pp: 122-134.

Presan, R.S., 2010. Soflware Engheering: A PracIilioner's Approach. 7th Edn., McGraw-Hill, New York, USA.,.

Ricca, F. and P. Tonella, 2001. Analysis and testing of web applications. Proceedings of the 23rd International Conference on Software Engineering, May 12-19, 2001 IEEE Computer Society, Washington, USA. is BN:0-7695-1050-7, pp: 25-34.

Sprenkle, S., E. Gibson, S. Sampath and L. Pollock, 2005. Automated replay and failure detection for web applications. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Nov. 7-11, Long Beach, USA., pp: 253-262.

Stepien, B., L. Peyton and P. Xiong, 2008. Framework testing of web applicalions using 'TI'CN-3. Int. J. Software Tools Technol. Transfer, 10: 37l-38l.