International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-4, November 2019

Mutual Browser Conflicts Disclosure

Chandra Prakash Patidar, Meena Sharma

Abstract*: *In today's world everything is becoming web dependent, and due to the advances made in web technologies, web developers have to face various challenges. Every web application before being deployed goes through various phases which may look different on different browsers. It becomes difficult to identify correct web page when it gives differences across different browsers. The web pages may give significant differences and it is known as cross-browser inconsistency. A technology that has gained a prominent position known as AJAX (Asynchronous JavaScript and XML), in which the combination of JavaScript and Document Object Model (DOM) manipulation, along with asynchronous server communication is used to achieve a high level of user interactivity. With this change in developing web applications comes a whole set of new challenges, One way to address these challenges is through the use of a crawler that can automatically walk through different states of a highly dynamic AJAX site and create a model of the navigational paths and states. Identifying these conflicts manually is a laborious task. Mutual browser conflict disclosure presents a mechanism to identify conflicts.

Keywords*: *Browser conflicts; Web testing; Web applications, Inconsistencies.

I. INTRODUCTION

Web applications are all around us. To access web applications browsers are the primary requirement. Users of such applications might use any web browser to access them, and the application is expected to behave consistently across these different environments. However, web applications often exhibit differences when executed in different browsers, leading to browser conflicts[1]. This cause changes between a web application's appearance, behavior or both, when it is run on different environments. Once we access the Amazon website on Chrome, it shows inconsistencies. As compatibility testing comes under web testing and browser compatibility comes under the most influencing part of compatibility testing. That is the images are not properly shown, the content is not organized in a proper way. This gives a bad impression to the user and the user might switch the browser or the application itself. When the user chooses to change the website, it gives a bad impression for the previous website and the user may never want to return to use that website ever again[2]. When the user chooses to switch the browser, it gives a bad impression of the previous browser and the user is not satisfied with the previous browser and may never want to return to the same browser for use. In

general, if browser conflicts are not identified during testing, they can adversely degrade the experience of the users of the web application with the expected browser[3]. In fact, some inconsistencies completely prevent users from accessing the functionality offered by the web application, thereby rendering it useless on that particular platform. Browser conflicts are thus a serious concern for companies, which rely on such applications for business or for creating their public brand image. Our research work is aimed at finding inconsistencies of web applications.

Inconsistencies can be broadly classified as[4]:

1) Structural: Such conflicts affect the structure or layout of individual web pages. The web page structure is an arrangement of elements. For example, the improper alignment of one or more web page elements on a given web page, in a particular browser can lead to a structural conflict.

If the webpage shows something horizontally in Google Chrome but in Internet Explorer the same thing is vertical, this is structural inconsistency.

2) Content: Such differences can occur, where the visual appearance of a web page element or the textual value of an element, are different across two browsers. We further classify these two cases as visual-content and text-content conflicts. If the webpage is accessed in Internet Explorer the image is not present but when the same webpage is accessed in Google Chrome, the image is present. This is Content inconsistency.

3) Behavioral: These involve a difference in the behavior of individual components on a web page. An example of such would be a button that performs a particular action within one browser and totally different action, or no action at all, in another browser. For example hyperlink on the webpage in Google Chrome works as a hyperlink but this hyperlink behaves as normal text in Internet Explorer.

II. RELATED WORK

Web crawlers are the tools used to explore the web. A crawler is a computer program that visits a specific web page and also visits all other pages that are linked to that page. Search engines such as Google, Mozilla use crawlers continuously to crawl websites and keep them up to date. By exploring websites automatically, crawlers enhance user interaction, making them appropriate for automatic testing purposes[5]. Commonly used crawling tools are discussed below.

Manuscript published on November 30, 2019.
* Correspondence Author
Chandra Prakash Patidar*, Assistant Professor of Information Technology at the Devi Ahilya University, Indore, India.
Meena Sharma, Professor of Computer Engineering at the Devi Ahilya University, Indore, India.

1)WebSPHINX:

WebSPHINX (Website Specific Processors for HTML INformation eXtraction) is a Java class library and interactive development environment for web crawlers. A web crawler (also called a robot or spider) is a program that browses and processes Web pages automatically. It consists of two parts: the workbench and the class library. Workbench is a graphical user interface that is used to customize the crawler. It is used to save pages offline on disk, concatenate pages together, extract text from pages, visualize a collection of web pages as a graph. The class library provides support for writing web crawlers using java. It has features of tolerant HTML parsing, common HTML transformations, pattern matching, support for reusable page content classifiers, multithreaded web page retrieval. But the problem with WebSPHINX is that this tool does not provide us the option of running a web application across different browsers[8]. It only generates the graph of webpage/application across its default browser so we cannot compare the graph to find out if any inconsistency occurs. Websphinx is not designed for enormous crawls like the entire web as search engines do.

2) CRAWLJAX: It is a tool which is used to create a document object model of web applications by deploying it across different browsers. It can easily be used with the command prompt of the system. It is an open-source java tool for crawling modern web applications. It uses the event-driven dynamic crawling engine and can explore Javascript-based Ajax applications. It is also used for nonfunctional testing like accessibility, validation, etc. It can detect broken links, images, tooltips. It can easily be extended through its easy to use plugin architecture. But the problem with Crawljax is that it cannot be operated on different browsers[7]. The command specified in its documentation is not working for different browsers even with changing the environment variables and contacting with Github society. So we were not able to find the solution for running the webpage/application on a different browser (other than Firefox).

3) SCREAMING FROG SEO SPIDER: The SEO Spider is a desktop program that can be directly installed on the computer, Mac or Linux. It crawls websites, links, applications for evaluation. The SEO spider tool is flexible and can crawl in short time duration allowing us to analyze the results in real-time. This tool is good for analysis of larger websites in which checking all the pages is laborious and where redirections, meta refresh or duplicate page issues can easily arrive. This tool exports the data (URL, page title, etc.) to Excel so it can be used as a base for SEO recommendations. It is used to crawl a website instantly and find broken links, identify redirect chains and loops, or upload a list of URLs to audit in a site migration. It is also used to analyze page titles, meta descriptions, collect data from the HTML of a web page using CSS Path, XPath. But the problem with this tool is that it can only be used on Mac or Linux operating systems. In the free version, only 500 URLs can be tested, for the complete tool we have to get the paid version.

4)GRAPHWALKER: GraphWalker is a Model-Based testing tool. It reads models in the shape of directed graphs, and generate paths from these graphs. The model is a collection of arrows and nodes and together they create a graph. An arrow represents an action and a node represents a verification. GraphWalker by mathematical algorithms generates a path which corresponds to your test idea, for this GraphWalker used generator rule and a model. The aim of the test design under test is to describe the expected behavior. The way it works is that you in a finite state diagram, express an action as a directed edge. An edge is corresponding to a transition. The edge points to a vertex, known as a node or state, where the results or the consequence of the previous action is verified. To use GraphWalker Either download the standalone jar file or include GraphWalker in your java project. But the problem with this is that it is browser-independent. The graph it generates is irrespective of the browser.

Other related tools in this field include [10]:

1) **Carejax**: A tool built around Crawljax, which is named Carejax, a combination of Careweb and Crawljax. Carejax will provide the foundation for crawling. Careweb is the first step towards automated regression testing. The main contribution to the existing works is that state-based crawling is applied to an industrial rich internet application. There are a few characteristics of the Careweb application, one is authorization and user accounts. To use Careweb, a user has to log in with valid credentials. Choosing a suitable crawl depth is important. The crawl depth determines the maximum number of events that should be executed consecutively from the start state. We have encountered the following difficulties: Incorrect state identification, state-space explosion, limited reliability and hard to analyze crawl results. For Careweb, solving incorrect state identification was possible through DOM strippers. Limited reliability can be addressed through improvements to the crawler, while the analysis of large crawl results could be made easier with more sophisticated tooling. Nevertheless, by crawling Careweb, a first step has been set towards crawling-based regression testing of the application. We think such automated regression testing is feasible for applications such as Careweb, provided state space explosion is controlled and an adequate level of robustness is guaranteed.

2) X-PERT: The technique starts by crawling the web application, in an identical fashion, in each of the browsers. In this process, it records the observed behavior as navigation models. The model is captured as a labeled transition system, which represents the top-level structure of the crawled web application. In the model, the states correspond to web application screens, and each transition is labeled with a widget action that leads to screen navigation[4]. In the X-PERT navigation model, record the screen image and the DOM structure of the elements on each observed screen. Any web application that runs on desktop browsers supported by X-PERT. Python and Java are used to write the code of X-PERT. Popular desktop operating systems, including Windows, Mac OS X, and Linux can be used to run X-PERT.

III. TECHNIQUE OVERVIEW

Most of the part of cross-browser inconsistency detection comes under the category of compatibility testing and some part comes under the category of functionality testing [9][12].

Apart from this we also work on automation testing to find out the cross-browser inconsistency. In this technique, we use the principle of Behaviour Driven Development and Acceptance Test-Driven Development. We use TestNG, ReportNG,Selenium and Selenium Grid. We also crawl and compare extracted attributes to compute inconsistencies. In this method we divide the process into five different modules Web crawler, Attribute extractor, Comparator, Classifier, and Report generator.We compute visual inconsistency by RGB index and histograms. In this technique we tested http://www.dauniv.ac.in/. We divide the process into four parts: State of the link, RGB difference, Histogram difference, and Coordinate difference. We performed our experiments on Google Chrome [v14.0.1] and Internet Explorer [v9.0.9]

We have performed our experiments on Internet Explorer (11.0), Google Chrome(70.0) and Mozilla Firefox(61.0).
We must fix the number of browsers used in the testing process.
We use jsoup a java parser to parse the web page. Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. jsoup implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do. It scrape and parse HTML from a URL, file, or string. Also find and extract data, using DOM traversal or CSS selectors.
Jsoup manipulate the HTML elements, attributes, and text and clean user-submitted content against a safe white-list, to prevent XSS attacks. jsoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; jsoup will create a sensible parse tree. For parsing the CSS, we need to use CSSOMPARSER, CSSSTYLESHEET, CSSRULELIST library. After Parsing CSS, We get the RULELIST means particular styling in one line.Then, Form a ArrayList of this rule LIST to check if the property is present or not.
It extracts the Document Object Model. In jsoup, we supply browsers by method *jsoup.connect()*, We get the parsing code. After Parsing the Html code, we need to extract the link tags from it.Link tags contains the css links which we need to parse.
We store the parsing code to our database for further processing. Apply search algorithm by selecting properties one after another from our database. If it is available then there is an inconsistency. Classifier gives the type of inconsistency. Our proposed and implemented model is depicted in Figure 1 and the algorithm is depicted in Algorithm 1.
Execution time ***'t'*** will be a factor of the following terms:
B= Number of Browsers
x=Number of lines parsed
Hence, t=B*x

Algorithm 1:

Input: Web Page, Browser and Database

(*w, Br1, Br2, Br3, D*)

Output: Inconsistency (*XBI*) exists or not.

Method: *jsoup (https://webpage)*

Get Parse Code (PC=PC_1...PC_m)

Do

Search in PC Ɛ properties (P_1...P_n) stored in D

For(each P_1 to P_n)

{

For(each PC_1 to PC_m)

{

If ($P_{i==}$ PC_j) // Where i=1...n and j=1...m

{

XBIClassifier Database:

Case 1: Structural

Case 2: Content

Case 3: Behavioral

}

Else

{

No XBI

Exit

}

}

}

IV. RESULTS

We categories our test on three categories. First three test page used as test cases. Next three are the websites that are used by us in 2016 for knowing the existence of cross-browser inconsistencies. Next three are from http://www.uroulette.com/ online random link generator. Table 1 shows the obtained results.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Table 1: Results

S. No.	URL	Inconsistencies		
		Structural	Content	Behavioral
1	Test Case 1	Yes	No	No
2	Test Case 2	No	No	No
3	Test Case 3	No	Yes	No
4	http://www.emeraldheights.edu.in/Day_School.html	No	No	Yes
5	http://www.tripadvisor.com/	No	No	No
6	http://www.holidayiq.com/	No	No	No
7	https://www.ffe.com/	Yes	No	No
8	http://www.stahnsdorf.de/	No	Yes	No
9	https://www.niel.be/	No	No	No

V. DISCUSSION

Currently, all the testing of web pages/applications is done manually by running everything on different browsers which consumes a lot of time. There should be a tool that can directly run the web pages automatically on all the web browsers without any problems. By which we will be able to compare in order to find conflicts.

VI. CONCLUSION

We started our work from finding inconsistencies manually in different websites. At that time we find many websites were tested manually for cross-browser inconsistency because that website contains an inconsistency. It is a very tedious task to find out inconsistency because many times it happened that websites that produced inconsistency did not exist or they removed the inconsistency. Initially, we tried to compute inconsistency by extracted attributes with the help of crawler. Further, we compute inconsistency by RGB index and histogram values. Finally, we conclude our methodology using jsoup parser and creating our own database. In this way, we can easily trace out the inconsistency.

Current & Future Developments

Currently, in many organizations, cross-browser testing is done manually. As we know the future era is of web and to access web browsers are a primary component. We try to automate inconsistency. Currently, we consider popular three browsers and desktop platform. In the future, we will increase the number of browsers and try to consider the mobile platform also.

ACKNOWLEDGMENT

Thanks to Dr. Meena Sharma (Professor, IET DAVV, Indore M.P., India) for the constant guidance and support in this work.

REFERENCES

1. https://www.w3schools.com/html/html5_webstorage.asp. [Accessed: Jan 12, 2018].
2. R. Gunasundari and S. Karthikeyan, "A study of content extraction from web pages based on links", International Journal of Data Mining and Knowledge Management Process, Vol. 2, pp. 23-30, 2012.
3. https://www.computerhope.com/jargon/w/webpage.htm. [Accessed: Jan 10, 2018].
4. https://techterms.com/definition/crossbrowser. [Accessed: Feb 08, 2018].
5. Ochin and J. Gaur, "Cross-Browser Incompatibility: Reasons and Solutions", International Journal of Software Engineering & Applications, Vol. 2, pp. 66-77, 2011.
6. https://www.siteground.com/kb/why_does_my_website_look_different_on_different_browsers/. [Accessed: February 02, 2018].
7. N. Barskar and C.P. Patidar, "A Survey on Cross Browser Inconsistencies in Web Application", International Journal of Computer Applications, Vol. 137, pp. 37-41, 2016.
8. S.R. Choudhary, M.R. Prasad, and A. Orso, "X-PERT: Accurate Identification of Cross-Browser Issues in Web Applications", In Proc. IEEE International Conference on Software Engineering (ICSE'13), 2012, pp. 702-711.
9. C.P. Patidar and M. Sharma, "An Automated Approach for Cross-Browser Inconsistency detection", In Proc. 9th Annual ACM COMPUTE 2016 held at DAIICT Gandhinagar Gujrat, 2016, pp. 141-145.
10. C.P. Patidar, M. Sharma and V. Sharda, "Detection of Cross Browser Inconsistency by Comparing Extracted Attributes", International Journal of Scientific Research in Computer Science and Engineering, Vol. 5, pp. 1-6, 2017.
11. https://www.sciencedaily.com/terms/web_crawler.htm. [Accessed: Feb 05, 2018].
12. A. Pranav and S. Chauhan, "Efficient Focused Web Crawling Approach for Search Engine", International Journal of Computer Science and Mobile Computing, Vol. 4, pp. 545-551, 2015.
13. N. Kowsalya, "An Approach of Web Crawling and Indexing of Nutch", International Journal of Scientific & Engineering Research, Vol. 5, pp. 766-772, 2014.

14. http://today.java.net/pub/a/today/2006/01/10/introduction-to-nutch-1.html. [Accessed: Feb 20, 2018].
15. http://crawljax.com/about/. [Accessed: Feb 15, 2018].
16. https://www.cs.cmu.edu/~rcm/websphinx/. [Accessed: January 10, 2018].
17. S.R. Choudhary, M.R. Prasad and A. Orso "CROSSCHECK: Combining Crawling and Differencing to Better Detect Cross-Browser Incompatibilities in Web Applications", In Proc. IEEE 5th International Conference on Software Testing, Verification and Validation (ICST'12), 2012, pp. 171-180.
18. A. Mesbah, E. Bozdag and A.V. Deursen "Crawling Ajax by Inferring User Interface State Changes", In Proc. IEEE Computer Society 8th International Conference on Web Engineering (ICWE'08), 2008, pp. 122–134.
19. A. Mesbah and M.R. Prasad, "Automated Cross-Browser Compatibility Testing", In Proc. ACM 33rdInternational Conference on Software Engineering, 2011 pp. 561-570.

AUTHORS PROFILE

Chandra Prakash Patidar received the B.E. degree in Information Technology and M.E. degree in Computer Engineering. He is an Assistant Professor of Information Technology at the Devi Ahilya University, Indore, India. His research interests are in Cross Browser Testing,GPGPU Computing, CUDA Programming, MultithreadedArchitecture, Compiler and MemoryArchitecture of Computers.

Dr. Meena Sharma received the B.E. degree in Computer Engineering and M. Tech degree in Computer Science in 1992 and 2004 respectively. She received the Ph. D. Degree in Computer Engineering in 2012.She is a Professor of Computer Engineering at the Devi Ahilya University, Indore, India. Her research interests are in Software Engineering, Software Quality Matrices and Object Oriented Modelling and Design.