Detecting Cross-browser Issues in Web Applications*

Shauvik Roy Choudhary
Georgia Institute of Technology, Atlanta, GA
shauvik@cc.gatech.edu

ABSTRACT

Cross-browser issues are prevalent in web applications. However, existing tools require considerable manual effort from developers to detect such issues. Our technique and prototype tool - WEBDIFF detects such issues automatically and reports them to the developer. Along with each issue reported, the tool also provides details about the affected HTML element, thereby helping the developer to fix the issue. WEBDIFF is the first technique to apply concepts from computer vision and graph theory to identify cross-browser issues in web applications. Our results show that WEBDIFF is practical and can find issues in real world web applications.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Reliability, Verification

Keywords

web testing, dynamic analysis, computer vision, tree matching

1. PROBLEM AND MOTIVATION

Web applications have gained popularity and widespread use in the past decade. Such applications follow the traditional client-server computing model where part of the web application, containing server side components runs on a web server and loads client side components in a web browser running on a remote user machine. The user interacts with these client side components which can make further requests from the server side and present information to the user. With recent advances in web technologies, the client side components have become more complex and richer along with the web browser platform. Due to the flexibility of the choice of the web browser, the client side environment is diverse among users. Recent web browser statistics reports the use of seven popular web browsers across different platforms [16]. Since client side technology standards are still evolving, web browsers often behave differently for the same web page. If a website is broken on one of these web browsers, it affects a class of users. Thus, it is crucial for developers to maintain compliance across these browsers by identifying such issues early and fixing them. This fix is usually a conditional client side scripts that run only in the affected browser to perform workarounds.

Cross-browser issues range from minor cosmetic defects to critical functional failures and result in the inability for the user to access a part of the web application. The Mozilla broken website reporter contains 1,767,900 websites running on 463,314 hosts, which were reported as broken by the users of the Firefox web browser [9]. In spite of such widespread impact of these issues, existing cross-browser testing techniques (as explained in Section 2.2) require significant amount of manual effort from the developer and are limited in the issues that they can find.

To address the limitations of existing technology, we present our technique and tool to automatically detect such cross-browser issues and help the developer provide a solution. In particular, the technique can (1) identify cross-browser issues automatically. (2) report the location of noticeable differences to the developer to assist him fix the issue.

*This paper summarizes some recent work published at ICSM 2010 [12, 13] and discusses some ongoing future work

ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

2. BACKGROUND AND RELATED WORK

2.1 Challenges

For achieving our goal of detecting cross-browser issues automatically, our technique needs to overcome some challenges. Firstly, any manual inspection for identifying such issues is expensive in terms of resources such as time and effort. Hence, the technique should be fully automated and should operate without requiring any manual effort from developers. Secondly, the technique should leverage the structural information of the web page across browsers. This can be obtained from the DOM[1] tree, which is maintained by each web browser when it loads any web page. However, the DOM tree for the same web page differs across browsers due to implementation differences and conditional scripts in web applications. Thus, automated comparison of this DOM tree across browsers is non-trivial. Before performing any comparison, our technique needs to match nodes between the different DOM trees obtained by loading the same web page across various web browsers. Our technique comprises of graph theory algorithms to perform both matching as well as structural comparison. Since the DOM tree can be mapped back to a source code element, it is helpful for the developer if the technique can identify differences in the corresponding tree nodes from different browsers. This information can then be translated to the sections of the HTML page and the server-side source code that generated it. Providing this information to developers allows them to localize and fix the issue effectively.

[1]Document Object Model http://www.w3.org/DOM/

An important challenge while performing cross-browser comparison is to actually compare how the web page elements appear on the screen. The DOM tree contains textual information about an element including the meta- information about its appearance (*e.g.,* dimensions, position etc.) However, it does not tell exactly how the element is rendered on the screen and presented to the user. For finding layout or rendering issues, it is essential to be able to mimic the end users' perception by considering the screen capture data from the web page. Hence, the technique should apply techniques from computer vision to match the visual appearance of corresponding elements of the web page across browsers.

Web applications often have variable elements such as advertisements and generated content (*e.g.,* time, date etc.) which are different across multiple requests. If such elements are not ignored, the technique might consider them as changes across browsers thereby resulting in false positives in the results. Hence, the technique needs to find and skip such elements during comparison. A final challenge is due to the inbuilt security measures in web browsers that pose a technical challenge for the technique to transparently extract information for comparison. The technique must overcome such security mechanisms reliably and collect all information needed.

Overcoming the aforementioned challenges allows WEBDIFF to identify dissimilarities in the same web page across two browsers and present them to the developer. More information about the steps involved in the technique have been presented in Section 3.

2.2 Related Work

Currently, web application developers render a given web page in different web browsers and manually inspect them for issues. Commercial tools can assist manual inspection by presenting a side by side rendering in two browsers [1, 8]. A research tool by Eaton and Memon [4] predicts faulty html tags for a particular browser. However, it requires a lot of manual classification and only considers HTML tags, ignoring other client side components (*e.g.,* CSS, JavaScript). Another tool by Tamm [15], identifies a class of browser rendering issues. However, it focuses only on a specific issue related to text elements. Moreover, it requires the re-rendering of the web page multiple times, making this technique very expensive.

A recent tool, Browsera [3] can detect some of the problems automatically. However, it is closed and performs naive DOM based comparison (in our experience using the tool). Mesbah and Prasad [7] recently describe a technique to extract finite-state machine based models of the web page from different browsers and compares them to find discrepancies. However, this technique is also DOM based and misses any graphical differences among browsers.

3. TECHNIQUE

In this section, we summarize our WEBDIFF technique [13] for finding dissimilarities between elements of a web page across multiple browsers. WEBDIFF considers one browser as "reference" and compares the behavior of the web page in it with that in other browsers. The WEBDIFF technique is composed of five different phases, each of which is described below.

3.1 Data Collection

In this phase, the technique first loads the web page in all web browsers. For each web browser, WEBDIFF equalizes the visible area of the web pages using automated browser window resizing. This allows the technique to obtain screen captures of the same size making them ideal for comparison in the later stages. Next, again in each browser, the technique captures the DOM tree of the loaded web page along with its screen captures. For each node in the DOM tree, the technique collects and stores its properties by querying the DOM API in each web browser from a JavaScript program. The approaches followed in this technique are browser independent and can be applied to any modern browser for collecting both the DOM tree and the screen capture of the web page.

3.2 Data Preprocessing

The data collected in phase 3.1 might contain variable elements. To identify these, WEBDIFF loads the web page again in the reference browser and collects the DOM tree and screen capture. Then it compares the two DOM trees from the same web page (loaded twice in the reference browser) using a pairwise breadth first traversal. Nodes with differences are marked as variable. For nodes without a match, based on the node heuristics, the technique checks the corresponding screen locations for observable differences. The heuristics points nodes that cannot be guaranteed to be the same visually by just matching its DOM properties. Thus, the technique compares such elements using a histogram based distance measure [14] from computer vision to find the dissimilarity between the two images. At the end of this phase, the technique marks the variable nodes in the DOM tree and discolors corresponding regions on the screen captures to ignore them in the next phases.

3.3 Cross-browser Matching

As discussed before (Section 2.1), the DOM trees captured across different browsers may not be structurally similar. This phase matches the DOM tree from a non-reference browser (target) to that of the reference browser to find corresponding nodes. It does so by traversing two trees in parallel to find a pair of nodes with the best match. Matching is perfect if a unique equality is found in identifying properties (*i.e.,* HTML id and xpath) of the node pair. Otherwise, WEBDIFF finds corresponding pair of nodes based on the similarity in their location in the DOM tree as well as their DOM properties.

3.4 Cross-browser Comparison

This phase performs the actual comparison of the information from each non-reference browser with that of the reference browser. For each pair of matched nodes in two DOM trees, this phase find differences in properties if they exist. For nodes whose properties match, this phase compares the corresponding sections of the screen captures to find differences. To perform this visual comparison, WEBDIFF uses the same histogram based matching technique as used in phase 3.2. After this comparison,WEBDIFF produces a list of nodes with dissimilarities found across browsers.

3.5 Report Generation

This phase generates the report from the differences obtained in phase 3.4. For compactness, it clusters multiple DOM nodes responsible for the same issue into one item. These items are then visualized as shown in Figure 1. The top section shows the screenshot from two browsers (reference browser on the left and target browser on the right). The bottom section shows issues found by WEBDIFF. Each item in the issue list indicates the type of the issue, the location on the screen screen where it was identified and the DOM xpath of the element that is involved in this issue.

4. EVALUATION AND DISCUSSION

To evaluate our technique, we implemented it in a tool [12] and performed experiments by exercising it on nine web pages as described in [13]. More specifically, we investigated the following two research questions related to the usefulness of the technique:

RQ1: Can WEBDIFF detect cross-browser issues in web applications?

RQ2: Can WEBDIFF detect such issues without generating a large number of false positives?

For our experiments, we used three popular browsers : Internet

Figure 1: Report generated by WEBDIFF.

Explorer[2] (Version 8.0), Mozilla Firefox[3] (Version 3.6) and Google Chrome[4] (Version 4.1) on a system running the Windows XP operating system. Out of the subject web pages, one web page was a selected subject with a known issue and the rest were obtained from a random URL generator service from Yahoo![5]. For each issue reported by WEBDIFF we manually classified it as a true or false positive. In these subjects, WEBDIFF reported 121 true issues while generating 21 false positives (17% ratio). Since WEBDIFF could find issues in real web pages, RQ1 was found to be true. The false positives were due to minor differences which were not noticeable by human eye but were caught by visual comparison and due to presence of some variable elements that WEBDIFF failed to identify. As far as RQ2 is concerned, we find our results to be encouraging. WEBDIFF is the first tool to find these issues automatically and we see a potential of eliminating the false positives in the future.

We are currently working on ways to improve the performance and precision of WEBDIFF. The most expensive part of the analysis is the visual comparison. We can reduce the number of comparisons by clustering the DOM nodes into regions and only performing visual comparison once per region. Further, in the data preprocessing phase, we plan to extract image features from the screen capture and use them along with the DOM node properties for matching. This will not only improve phases 3.3 and 3.4 but will help to prioritize the reported issues based on number of mismatches. In addition to these, we want to combine WEBDIFF with a test generation technique (*e.g.,* [2, 5, 6, 10, 11]) to exercise it on the complete web application, rather than individual web pages. Finally, we would like to perform user studies by deploying WEBDIFF in the field and make improvements based on feedback from users.

5. CONCLUSION

Cross-browser issues are relevant and a serious concern for web application developers. Current techniques are limited in finding these issues and require considerable manual effort. Our technique WEBDIFF identifies such issues automatically and helps the developer to fix them. To achieve this, WEBDIFF applies concepts from graph theory and computer vision to the data collected from the web page (DOM tree and screen capture respectively) across different browsers. Our results suggest that WEBDIFF is practical and can find issues in real world applications.

[2] `http://www.microsoft.com/windows/internet-explorer/`
[3] `http://www.mozilla.com/firefox/`
[4] `http://www.google.com/chrome`
[5] `http://random.yahoo.com/bin/ryl`

6. REFERENCES

[1] Adobe. Browser lab. `https://browserlab.adobe.com/`, May 2010.

[2] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst. Finding bugs in dynamic web applications. In *ISSTA '08: Proceedings of the 2008 international symposium on Software testing and analysis*, pages 261–272, 2008.

[3] Browsera. Automated browser compatibility testing. `http://www.browsera.com/`, December 2010.

[4] C. Eaton and A. M. Memon. An empirical approach to evaluating web application compliance across diverse client platform configurations. *Int. J. Web Eng. Technol.*, 3(3):227–253, 2007.

[5] W. G. J. Halfond and A. Orso. Improving test case generation for web applications using automated interface discovery. In *ESEC-FSE '07: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 145–154. ACM, 2007.

[6] X. Jia and H. Liu. Rigorous and automatic testing of web applications. In *In 6th IASTED International Conference on Software Engineering and Applications (SEA 2002*, pages 280–285, 2002.

[7] A. Mesbah and M. Prasad. Automated cross-browser compatibility testing. In *ICSE '11: Proceedings of the 33rd ACM/IEEE International Conference on Software Engineering*. ACM, 2011.

[8] Microsoft. Expression web. `http://www.microsoft.com/expression/products/Web_Overview.aspx`, May 2010.

[9] Mozilla. Firefox broken website reporter. `http://reporter.mozilla.org/app/stats/`, July 2010.

[10] F. Ricca and P. Tonella. Analysis and testing of web applications. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 25–34, Washington, DC, USA, 2001. IEEE Computer Society.

[11] D. Roest, A. Mesbah, and A. v. Deursen. Regression testing ajax applications: Coping with dynamism. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pages 127 –136, 6-10 2010.

[12] S. Roy Choudhary, H. Versee, and A. Orso. A cross-browser web application testing tool. In *26th IEEE International Conference on Software Maintenance (ICSM 2010) – Formal research demonstration*, 2010.

[13] S. Roy Choudhary, H. Versee, and A. Orso. Webdiff: Automated identification of cross-browser issues in web applications. In *ICSM '10: Proceedings of the International Conference on Software Maintenance*. IEEE, September 2010.

[14] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[15] M. Tamm. Fighting layout bugs. `http://code.google.com/p/fighting-layout-bugs/`, October 2009.

[16] W3Schools.com. Browser statistics month by month. `http://www.w3schools.com/browsers/browsers_stats.asp`, May 2010.