Cross-Browser Testing in Browserbite

Tõnis Saar[1], Marlon Dumas[2], Marti Kaljuve[1], and Nataliia Semenenko[2]

[1] Software Technology and Applications Competence Center, Estonia
{tonis.saar, marti.kaljuve}@stacc.ee
[2] University of Tartu, Estonia
{marlon.dumas, nataliia}@ut.ee

Abstract. Cross-browser compatibility testing aims at verifying that a web page is rendered as intended by its developers across multiple browsers and platforms. Browserbite is a tool for cross-browser testing based on comparison of screenshots with the aim of identifying differences that a user may perceive as incompatibilities. Browserbite is based on segmentation and image comparison techniques adapted from the field of computer vision. The key idea is to first extract web page regions via segmentation and then to match and compare these regions pairwise based on geometry and pixel density distribution. Additional accuracy is achieved by post-processing the output of the region comparison step via supervised machine learning techniques. In this way, compatibility checking is performed based purely on screenshots rather than relying on the Document Object Model (DOM), an alternative that often leads to missed incompatibilities. Detected incompatibilities in Browserbite are overlaid on top of screenshots in order to assist users during cross-browser testing.

Keywords: Cross-browser compatibility testing, image processing.

1 Introduction

Cross-browser (compatibility) testing aims at finding incompatibilities in the way a Web page is rendered across different combinations of a browser, a browser setting, an operating system (OS) and a hardware platform (herein called a *configuration*). The exact meaning of the term "incompatibility" varies from one testing subject to another and hence cross-browser testing has to take into account the sensitivity of the intended user(s). Incompatibilities may range from missing buttons, to misaligned text blocks, broken images or misplaced elements. In the absence of tool support for cross-browser testing, testers have to open web pages manually and check for differences. This procedure is time-consuming, monotonous and non-scalable given the growing number of configurations that need to be supported by Web applications.

Existing automated methods for cross-browser testing are generally based on an analysis of the Document Object Model (DOM) [1][2][3]. However, the fact that a Web page has very similar DOM structure and parameters across different configurations does not guarantee absence of incompatibilities, as rendering engines may display similar DOMs in rather different ways. Thus DOM-based cross-browser testing techniques suffer from lower recall (high number of missed incompatibilities). Some techniques such as WebDiff [1] apply DOM-based web page segmentation in conjunction with image comparison over pairs of matching segments. But while the latter step improves recall, the DOM segmentation step may still hide incompatibilities.

In contrast to the above techniques, Browserbite employs image processing both for web page segmentation and segment comparison. Specifically, Browsebite combines an image segmentation technique based on detection of discontinuities and colour changes with an image comparison technique based on a combination of geometric features and histograms of pixel intensity distribution. These techniques are complemented by supervised machine learning, so as to take into account user sensitivity.

2 System Overview

Browserbite consists of three main components: screenshot generation, image segmentation and comparison, and classification, as shown in Fig. 1. These components are triggered sequentially when a user inserts a URL of a *web page under test* in Browserbite's interface. The URL is added to a queuing system implemented using Ruby Resque. Different Ruby workers then take specific tasks from the queue and perform the task in question, incl. generating screenshot, resizing image, comparing pair of images, or filtering potential incompatibility via a classification model.

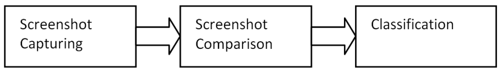

Fig. 1. Overview of Browserbite's tool chain

In addition to a URL, the user specifies a *baseline configuration*, that is a browser-OS-platform for which the user has verified correct rendering. On average, results are displayed in 30 – 45 seconds (with initial results shown incrementally). Detected incompatibilities are highlighted on top of the baseline configuration as shown in Fig 2.

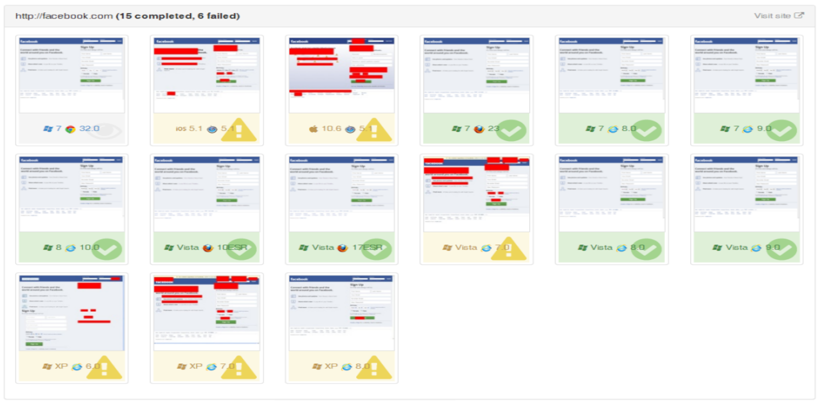

Fig. 2. Example of Browserbite report

2.1 Screenshot Capturing

A number of virtual machines are used to generate screenshots on different browsers and operating systems. Browserbite supports six OS (Windows XP, Vista, 7, 8, Apple OS X 10.6, iOS 6.0) and two browsers with default settings (Google Chrome, Firefox, IE and Safari). A screenshot is generated using the Selenium WebDriver library.

After a web page is loaded, the browser window is maximized. Then the whole window (a.k.a. viewport) is saved as an image. In case of OS X a full-page screenshot has to be composed out of fragments, in a process of scrolling, screenshotting and re-stitching. In Windows, a full-page screenshot can be taken in a single step [4].

2.2 Segmentation and Comparison

In this stage, pairs of screenshot images are compared to find significant differences. One of the images is a baseline image and the other is an image under test.

As mentioned earlier, pixel-by-pixel comparison leads to excessive false positives. Indeed, small misalignments of even one pixel may cause pixel-by-pixel comparison to immediately fail. Accordingly, Browserbite adopts a two-step comparison process. First, images are segmented into smaller so-called regions, which are then matched pairwise. Segmentation helps to prevents false alarms caused by small misalignments of web page elements. The segmented regions can represent for example buttons, forms, headings, text blocks etc. Segmentation in Browserbite is based purely on visual features (discontinuity and colour changes) and is implemented using well-known image processing techniques [5].

In a second step, segments are compared pairwise (one baseline segment versus one image-under-test segment). Pairwise comparison is performed first on geometric features (position and size) and secondly on the values of the histogram of pixel density distribution, following a well-known histogram extraction technique used in computer vision [5]. Each baseline segment is matched to the most similar segment from the image-under-test. If two matched segments have differences in feature parameters beyond a *tolerance threshold*, or if a segment in one image has no matching pair in the other, the segment(s) is/are declared *potentially incompatible*. Tolerance thresholds have been tuned experimentally based on a corpus of images (see below) in such a way as to produce a small number of false negatives (2% of missed incompatibilities, i.e. 98% recall). These thresholds however lead to a precision of 66%. To strike a better tradeoff, Browserbite relies on an additional classification stage.

2.3 Classification

The classification stage is used to classify potential incompatibilities into actual incompatibilities versus false alarms. The classifier uses the same features mentioned above, which are extracted via image comparison (full list of features is given in [6]).

For training and testing the classifier, we used the 140 most popular web pages in Estonia (from the alexa.com list). These web pages were tested manually using Browserbite without the classifier component. As a result 20 000 potential differences were found. From this set 2700 segment pairs were randomly selected. 40 people were asked to classify these 2700 potential incompatibilities into the two classes. As a result 1350 positive and negative cases were obtained.

We tested both decision trees and neural networks (using the OpenCV library) as classification techniques. Neural networks give clearly better results [6]. Plain Browserbite without neural network classifier has precision of 66% and recall 98%. The neural network classifier improves precision to 96% with a recall of 89%, illustrating the trade-offs. Despite this imperfect result, Browserbite's accuracy (F-score) is superior to that of a state-of-the-art tool (Mogotest) [6].

On the background of these trade-offs, Browserbite has been made a commercial product, available on a software-as-a-service basis at: http://www.browserbite.com. It has a growing user base (over 10 000 registered users). At present, Browserbite can produce false positive results while testing pages with dynamic regions (e.g. animations). It is planned to add dynamic region suppression. Dynamic regions are detected by taking a screenshot of a web page with an interval in-between, and comparing the two screenshots using the same technique described above.

3 Conclusion

The Browserbite development experience demonstrates the feasibility and power of cross-browser testing based on image processing. Extensions include the ability to handle Web page flows (as opposed to individual pages) and the adaptation of Browserbite to non-traditional web platforms like smart TV's, billboards and GPS devices.

References

1. Choudhary, S.R., Versee, H., Orso, A.: WEBDIFF: Automated identification of cross-browser issues in web applications. 2010 IEEE International Conference on Software Maintenance (ICSM). pp. 1–10 (2010).
2. Choudhary, S.R., Prasad, M.R., Orso, A.: CrossCheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications. Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. pp. 171–180 (2012).
3. Mesbah, A., Prasad, M.R.: Automated cross-browser compatibility testing. Proceedings of the 33rd International Conference on Software Engineering. pp. 561–570 (2011).
4. Kaljuve, M. Cross-Browser Document Capture System. Master's Thesis, University of Tartu, June 2013. http://tinyurl.com/nlze7ub
5. Shapiro, L.G., Stockman G. C. *Computer Vision*. Prentice Hall (2001).
6. Semenenko, N., Dumas, M., Saar, T.: Browserbite: Accurate Cross-Browser Testing via Machine Learning Over Image Features. In Proceedings of the 28th International Conference on Software Maintenance (ICSM), pp. 528-531, IEEE Computer Society (2014)