Journal of Logical and Algebraic Methods in Programming 85 (2016) 737–758

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Declarative layout constraints for testing web applications ☆

Sylvain Hallé *, Nicolas Bergeron, Francis Guérin, Gabriel Le Breton, Oussama Beroual

Laboratoire d'Informatique Formelle, Département d'Informatique et de Mathématique, Université du Québec à Chicoutimi, Canada

ARTICLE INFO

Article history:
Received 4 April 2015
Received in revised form 29 April 2016
Accepted 29 April 2016
Available online 24 May 2016

ABSTRACT

The paper focuses on bugs in web applications that can be detected by analyzing the contents and layout of page elements inside a browser's window. Based on an empirical analysis of 35 real-world web sites and applications (such as Facebook, Dropbox, and Moodle), it provides a survey and classification of more than 90 instances of layout-based bugs. It then introduces Cornipickle, an automated testing tool that provides a declarative language to express desirable properties of a web application as a set of human-readable assertions on the page's HTML and CSS data. Such properties can be verified on-the-fly as a user interacts with an application.

1. Introduction

The use of the web has seen significant changes since the inception of HTML in the early 1990s. Thanks to the use of databases, client- and server-side code, web sites have moved from being mere repositories of simple and static pages to become complex, interactive and stateful applications in their own right. Sites like Facebook or LinkedIn have become common fixtures of web users, and vendors like Microsoft now maintain a fully featured web-based version of their popular Office suite. Recent years have even seen the advent of full-fledged operating systems relying solely on the web stack of HTML/CSS/JavaScript, such as FirefoxOS,[1] Chromium OS[2] and eyeOS.[3] The web has now become a large and mature software ecosystem on a par with the complexity found in traditional desktop applications.

Due to the somewhat complex relationship between HTML, CSS and JavaScript, the layout of web applications tends to be harder to properly specify in contrast with traditional desktop applications. The same document can be shown in a variety of sizes, resolutions, browsers and even devices, making the presence of so-called layout "bugs" all the more prevalent. Such problems can range from relatively mundane quirks like overlapping or incorrectly aligned elements, to more serious issues compromising the functionality of the user interface. The problem is exacerbated by the current lack of tools to test web applications with respect to their layout, and in particular the short supply of formally defined languages for expressing the desirable properties of a web application's content and display.

As proof of this assertion, in this paper, we provide a summary of a study of 90 instances of layout-based bugs in 35 real-world web sites and applications, including Facebook, Dropbox, Moodle, LinkedIn, various banks, airlines and online

☆ The authors gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

* Corresponding author.
E-mail address: shalle@acm.org (S. Hallé).

[1] https://www.mozilla.org/en/firefox/os/.
[2] https://www.chromium.org/chromium-os.
[3] http://www.eyeos.com.

http://dx.doi.org/10.1016/j.jlamp.2016.04.001
2352-2208/

Table 1
Web sites and applications for which at least one layout-based bug has been found.

• Academia.edu	• Dropbox	• NSERC
• Acer	• EasyChair	• OngerNeige
• Adagio Hotels	• Elsevier	• ProQuest
• Air Canada	• Evous France	• Rail Europe
• Air France	• Facebook	• ResearchGate
• AllMusic	• IEEE	• St-Hubert
• American Airlines	• Just for Laughs	• SpringerOpen
• Boingo	• LinkedIn	• TD Canada Trust
• Canadian Mathematical Society	• Liveshout	• Time Magazine
• Customize.org	• Microsoft TechNet	• Uniform Server
• Digital Ocean	• Monoprix	•YouTube
	• Moodle	
	• Naymz	

stores. Each bug instance is documented in an online repository where the exact page contents are saved for later study. Section 2 shall describe and classify these bugs.

We then define the syntax of a declarative language borrowing from first-order logic, linear temporal logic and existing test tools like Cucumber [1]. We show in Section 3 how each bug from our empirical study can be expressed as a declarative, human-readable property on the page's HTML and CSS properties.

We finally design and implement in Section 4 a testing tool, called *Cornipickle*, that can automatically verify properties written in that language as a user interacts with an application. Cornipickle's architecture does not rely on browser plugins or JavaScript frameworks, and can hence work in unusual browser/OS combinations outside the reach of most testing tools. Most notably, we describe how, thanks to the logic-based nature of Cornipickle's specification language, useful and intuitive feedback can be provided to the user in the case a specification becomes violated. An automated counter-example generation procedure can be used to pinpoint specific elements of a document "responsible" for the violation and highlight them in real time.

One particular use of Cornipickle is *regression testing*: once a layout bug has been found, developers can write a rule and use Cornipickle to ensure this bug never reappears in future modifications made to the application. In terms of maturity, Cornipickle can be classified as a functional proof of concept. It is freely available under an open source license[4] and is expected to undergo steady development in the near future.

2. Layout-based bugs in web applications

A *layout-based bug* is a defect in a web system that has visible effects on the content of the pages served to the user. This content can be anything observable by the client, including the structure of the page's DOM and the dimensions and style attributes of elements. Hence we exclude from this definition anything that relates to the inner workings of the application: obviously any server-side state information (session variables, back-end database contents), but also any reference to client-side code, such as values of variables in local JavaScript code. In a nutshell, layout-based bugs are those that can be detected by client-side black-box testing.

This section shall first show that, although seemingly mundane, layout-based bugs are widely present in a large number of real-world applications and web sites. It will then demonstrate how, although based on page layout, bugs of this kind are not limited to simple, static presentation problems, and can in many cases reveal defects in the *behaviour* of the application.

2.1. Prevalence and types of layout bugs

We performed a survey of more than 35 web applications over a ten-month period in 2014–2015, and collected any bugs having an impact over the layout or contents of their user interface. The sites were surveyed in an informal way, by simply collecting data on bugs through the authors' daily use of the web. Table 1 gives the list of web sites and applications for which at least one layout bug has been found.

Every time such a bug was found, a bug report was created. This report contains a short description of the bug (or the expected content of the page), a screenshot of the offending part of the page, as well as a snapshot of the page's contents in the Mozilla Archive File Format (MAFF) or MIME HTML (MHT) [2]. Such formats save in a single archive file all the resources referenced by the page (JavaScript code, images, CSS files) and rebase all URLs so that they point to the archive. This way, the page can be reopened at any point in time without the need to rely on online resources that may have moved or disappeared; as such, a MAFF or MHT archive can be seen as a faithful snapshot of the page's state at the moment the bug was found.

All our subsequent testing has been done on snapshots of the pages taken at the moment the bugs were discovered. Therefore, it is possible that some of the bugs mentioned in this paper have since been fixed.

In the following, we briefly present some of the bugs we discovered; many are manifestations of more than one type of layout problem, so that the classification given below is not a *partition* of all the encountered bugs. We shall highlight the

[4] https://bitbucket.org/sylvainhalle/cornipickle.