A Tool for Registering and Replaying Web Navigation

Alberto Bartoli
DI[3] - University of Trieste, Italy
email: bartoli.alberto@units.it

Eric Medvet
DI[3] - University of Trieste, Italy
email: emedvet@units.it

Marco Mauri
DI[3] - University of Trieste, Italy
email: marco.mauri@phd.units.it

***Abstract*—Web-based applications have an ever increasing impact on modern society. Organizations spend an ever increasing amount of resources developing and maintaining web applications that are in fact their public face. Hence, each problem that an external user encounters using these products—either due to an external attack or a simple defect in the underlying code—often results in a loss for the organization itself. In order to reduce defects and prevent attacks web applications require a thorough testing. Unfortunately, nowadays testing a web application is a very complicated issue, primarily due to the vast use of AJAX technologies. To simplify and improve the testing procedures, we propose a method to register and replay web navigation, i.e., sequences of user's interactions with web applications. Then, we developed a tool that implements this method and tested it on various web applications, both real and synthetic, obtaining very positive results.**

***Index Terms*—Web 2.0, Testing, Web navigation, JavaScript.**

I. Introduction

A complete testing is fundamental to develop and maintain web application but this task is made extremely complicated by the absence of testing tool both easy to use and capable to interact with modern, AJAX based, web applications.

Thus we decided to focus this article on both increasing the easiness of web application testing and finding a way to describe the interactions of AJAX based web applications.

We developed a method to register and replay sequences of user's interactions, called traces, with web applications with the following properties:

- the creation of a trace is completely automatic;
- no necessity to instrument the monitored web server;
- the replay algorithm is resilient to DOM variance.

This method aids web application testing in two ways: (i) it can easily navigate the browser to the page that is being tested even if it is deeply nested in the web application structure and requires login and (ii) it can also be used as a simple way to test that the modification being done to the web applications does not alter its structure in unexpected ways.

The innovative aspect of this method is the simultaneous presence of all of these three properties, as none of the cited works possesses all of them.

In particular, [1] describes a method that allows the registration and replay of traces of web application navigations. However, contrarily to our works, the registration of the trace is not automatic, but require manual marking of each event.

Similar methods are presented in [2], [3], but they cannot register events generated within HTML documents conforming to the last version of the HTML DOM specification, as they are limited to the so called version 0. For example, many of the buttons that compose the graphic interface of the iGoogle web application use the DOM 2 features to handle user events, so the approach taken by [2], [3] cannot register any event associated to those buttons, while the method described in this article can register those events. Furthermore, [2] is unable to create trace that spans over multiple pages.

Finally, another aspect of particular relevance of our work is the capability to cope with file uploads, which is a feature that is lacking from all of the cited works.

II. Architecture

Our system consists of two separate applications: the *trace recorder*, that registers the actions executed by the user in a browsing session, and the *trace replayer*, that replays a browsing session previously registered by the trace recorder.

The trace recorder consists of (i) a web application that we developed and that we call *GWT observer*, (ii) a preconfigured *proxy* and (iii) an instrumented browser. An overview of the trace recorder is plot in Fig. 1. The GWT observer is composed of a client-side code (Observer-C), which is executed within the browser and registers the user's actions, and a server-side code (Observer-S), which actually generates the traces. The proxy is placed in between the instrumented browser and the target web site and is configured to injects the Observer-C code into every HTML document sent to the browser. Hence, the user's actions can be registered transparently to the target site.

Observer-C is able to register all the events related to keys struck, mouse clicks, choice selection from a drop down list and the text typing into a form field.

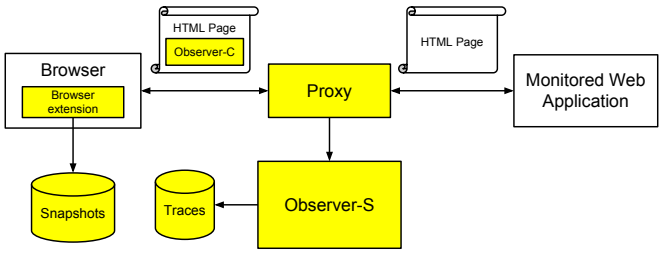

Figure 1. Trace Recorder logical architecture

The trace replayer reads the trace created by the trace recorder and then replay the registered actions using a instrumented version of the Mozilla Firefox web browser.

A trace contains a sequence of *event descriptions*. An event description consists of (i) the event type, (ii) the time at which the event occurred (timestamp) and (iii) the description of the target element of the event.

To properly replay the trace we introduced the concepts of relevant and unrelevant events. A *relevant* event is an event whose replay is essential to properly replay the entire trace. An *unrelevant* event is an event whose replay does not influence the final outcome of the replay of the trace. An event can be marked as relevant or unrelevant only upon the complete registration of the trace: thus, the trace recorder registers all events. The replay of unrelevant events is counter-productive, because it could slow down the replay of the whole trace.

An example of unrelevant events are those generated to select a text field inside a form, because the subsequent event of typing inside that field implies the selection of the text field itself.

We consider as relevant all the events of related to mouse clicks and double clicks and keys struck.

The trace replayer preprocess the trace by filtering out unrelevant event and sorting the remaining events into a single sequence S according to their timestamp.

III. Trace replay

The replay of the trace is performed by the trace replayer which drives a real browser (Mozilla Firefox) through a freely available tool (Webdriver).

The algorithm used to replay the trace is the following. For each event E in the sequence S, the trace replayer searches the current web document for the associated target element T_E; if found, replays the event using T_E as target using WebDriver. If not found, the trace replayer repeats the search a fixed number of times, waiting half second between every repetition. The rationale of this procedure is in that the searched element could be generated later by a JavaScript code executed by the web application. If even after repeating the search multiple times the corresponding element is not found, the trace replayer aborts the replay and notifies the error.

When all the events of S have been correctly replayed, the trace replayer notifies the user of the positive outcome.

IV. Target element search algorithm

The search of the correct target element is a fundamental, yet complex procedure. On one hand, in the vast majority of cases the element to search for is not marked with an unique identifier. On the other hand, there are often significant differences between the web document as seen in the registration phase and in the replay phase.

For these reasons, we designed a series of heuristics to find the correct target element for an event. We designed three heuristic tailored respectively to find media element, form inputs, generic textual content. Furthermore, we designed a fourth generic heuristic which is valid for all the other cases.

We have not created a labeled dataset to quantitatively evaluate specifically the performance of our heuristics. Yet, we performed a set of experiments on real and complex web applications and found that the heuristics did not generate any false positive nor any false negative—i.e., they did not find any element that was not the one seen during the registration.

V. Experiments

In order to test our proposed method and tool, we performed a series of experiments on various web applications: each experiment consisted in the registration of a trace and in multiple replay of such trace to verify the repeatability of the replay.

We experimented on the following modern web applications:

- Amazon
- Facebook
- Google Groups
- Stack Overflow
- WackoPicko
- WIVET

The last two web applications are designed specifically as a testing ground for security scanner tools and crawler respectively.

We could correctly register and replay all of these traces. On the contrary, the other cited works could not. In particular, [2] could not register any of these web applications because they all spans over multiple pages; [3] could not register any trace on Google Groups because this web application makes large use of the DOM 2 technology.

VI. Conclusions

This paper presents a novel method and a tool to create and replay sequences of user's interactions with web applications. The tool is easy to use and works with real and complex web applications, whereas preexisting similar tools do not.

This method is useful to improve web application testing by reducing the time needed to thoroughly test the web application. Finally, it does not require to instrument the web application by building specific and complex tools.

References

[1] P. Montoto, A. Pan, J. Raposo, F. Bellas, and J. López, "Automating navigation sequences in ajax websites," in *Web Engineering*, ser. Lecture Notes in Computer Science, M. Gaedke, M. Grossniklaus, and O. Díaz, Eds. Springer Berlin / Heidelberg, 2009, vol. 5648, pp. 166–180, 10.1007/978-3-642-02818-2_12. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02818-2_12

[2] K. Pattabiraman and B. Zorn, "Dodom: Leveraging dom invariants for web 2.0 application robustness testing," in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ser. ISSRE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 191–200. [Online]. Available: http://dx.doi.org/10.1109/ISSRE.2010.17

[3] M. Álvarez, A. Pan, J. Raposo, and J. Hidalgo, "Crawling web pages with support for client-side dynamism," in *Advances in Web-Age Information Management*, ser. Lecture Notes in Computer Science, J. Yu, M. Kitsuregawa, and H. Leong, Eds. Springer Berlin / Heidelberg, 2006, vol. 4016, pp. 252–262, 10.1007/11775300_22. [Online]. Available: http://dx.doi.org/10.1007/11775300_22