International Journal of Software Engineering and Knowledge Engineering

Finding and Emulating Keyboard, Mouse, and Touch Interactions and Gestures while Crawling RIAs

FREDERIK H. NAKSTAD*, HIRONORI WASHIZAKI†, YOSHIAKI FUKAZAWA‡

Department of Computer Science and Engineering, Waseda University
Tokyo, Japan
**frederik@fuji.waseda.jp*
†*washizaki@waseda.jp*
‡*fukazawa@waseda.jp*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Existing techniques for crawling Javascript-heavy Rich Internet Applications tend to ignore user interactions beyond mouse clicking, and therefore often fail to consider potential mouse, keyboard and touch interactions. We propose a new technique for automatically finding and exercising such interactions by analyzing and exercising event handlers registered in the DOM. A basic form of gesture emulation is employed to find states accessible via swiping and tapping. Testing the tool against 6 well-known gesture libraries and 5 actual RIAs, we find that the technique discovers many states and transitions resulting from such interactions, and could be useful for cases such as automatic test generation and error discovery, especially for mobile web applications.

Keywords: crawling; gesture emulation; event handler analysis; RIA.

1. Introduction

Crawling JavaScript-heavy Rich Internet Applications (RIA) has been a hot topic in recent years, giving us automated tools and methods for extracting state diagrams for such highly dynamic web applications. These methods have spawned a variety of applications to automate beneficial tasks such as indexing for search engines [1], accessibility and usability evaluation [4], automatic test generation [6, 7], regression testing [8], validation [5], and security testing [3] to mention some.

However, existing crawling techniques tend to ignore user interactions beyond mouse clicking, and therefore often fail to consider potential mouse, keyboard and touch interactions. It seems reasonable to hypothesize that these kinds of interactions can be numerous in advanced interactive web applications as well as mobile-tailored web applications.

Fig. 1. Gallery with swipe interactions.

2. Motivating Example and Research Questions

As a motivating example, let's consider a photo gallery component containing an image and some descriptive text as pictured in Figure 1. In order to load the next picture and text caption in the gallery, you swipe left or right. These swipe interactions would be implemented by attaching the desired gesture type to a target DOM element via JavaScript. Since existing crawl techniques do not attempt to execute such advanced interaction events, the states loaded by performing these swipe gestures would not be found, leaving a good chunk of application functionality and content unexplored.

We propose a new technique for finding and exercising mouse, keyboard, and touch interactions when crawling interactive JavaScript-based websites by analyzing and exercising event handlers . Gesture emulation is employed to find states accessible via swiping and tapping. The research questions we try to answer are (1) how comprehensively can our technique capture and perform gesture interactions, (2) how often do various keyboard, mouse, and touch events lead to new states in modern RIA's, and which event types are more likely to induce new states, and (3) what DOM elements are more likely to be targets for interactions leading to new state transitions?

3. Technique

The technique, as depicted in Figure 2, works by executing a JavaScript module in each state before any other JavaScript is evaluated. This enables us to override the various event listener registration APIs of the browser, and thus collect all event handlers registered by the developer. Once this information is delivered back to the crawler, it can use it to decide what interactions to try out based on event handlers registered by the website developer. The technique is implemented as a set of extensions and modifications to Crawljax [1]. The tool went through two iterations, the first one using a MITM proxy[a] to intercept HTML content for the

[a]https://mitmproxy.org/

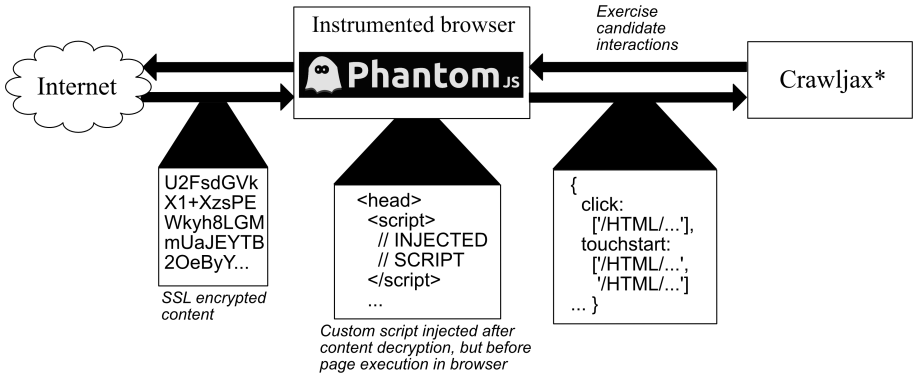

Fig. 2. Architecture of mobCrawler

Table 1. Gesture emulation.

Interaction	**Description**	**Fire on**
Event dispatch	Create and dispatch an event to the target element.	click, mouse, key
Mouse swipe	Dispatches a series of down, move and up events for mouse.	mousedown
Touch swipe	Dispatches a series of start, move and end events for mouse.	touchstart
Tap	Dispatches touchstart and touchend events in rapid succession.	touchstart
Double tap	Two tap gestures in quick succession.	touchstart
Tap hold	Tap gesture with longer timeout between the touchstart and touchend events.	touchstart

crawler, and then injecting the script. The second improved version relies on a modified version of the Phantom.js browser[b], where we inject the script after SSL decryption, but before the page is executed.

Our tool will programmatically construct and dispatch events to the various target elements according to what event handlers have been registered on them as outlined in Table 1. For mousedown and touchdown handler types we emulate various gestures: tap, double-tap, tap-hold, and swipes in different directions. If these interactions are emulated correctly they will elicit JavaScript code executions via the event callbacks the developer registered, which in turn may change the state of the DOM and give us a new state in the state graph.

4. Experiment and Case Study

We tested the tool against the 6 most popular gesture libraries for JavaScript according to GitHub. This was done with an experiment where we created a simple website for each gesture library, and implemented event handlers for gestures listed in Table 2. Each event handler would trigger a callback introducing a small write

[b]http://phantomjs.org/

Table 2. Gesture support.

Library	**Tap**	**Double tap**	**Tap hold**	**Swipe**			
				Left	**Right**	**Up**	**Down**
Hammer.js	✓	✓	✓	✓	✓	✓	✓
Quo.js	✓	×	×	×	×	×	×
dojox.gesture	✓	✓	✓	✓	✓	✓	✓
touchSwipe	✓	✓	✓	✓	✓	✓	✓
Touchy	×	×	×	×	×	×	×
jGestures	✓	N/A	N/A	✓	✓	✓	✓

Note: All instances marked × were successfully found and emulated if configured to look for custom event types.

Table 3. Event type distribution for state transitions

	keyboard	**mouseswipe vertical**	**mouseswipe horizontal**	**mouse hovers**	**click**	**touchswipe vertical**	**touchswipe horizontal**	**touch taps**
C1	0	0	0	42	22	0	0	0
C2	38	19	28	19	60	3	6	11
C3	5	14	16	1	23	44	58	60
C4	3	0	0	0	61	0	1	3
C5	0	26	25	0	100	11	27	19

Table 4. Element type distribution for state transitions

	div	**html**	**a**	**span**	**body**	**button**	**input**	**li**
C1	0	0	4	60	0	0	0	0
C2	8	167	16	0	0	0	0	0
C3	175	46	0	0	0	0	0	0
C4	33	0	29	0	4	0	1	1
C5	128	6	35	0	25	14	0	0

to the DOM, so that it would be picked up by the crawler. The crawler was then run on each website in turn to see if it could detect the gesture registrations and successfully emulate them. In answer to RQ1 we found that the technique was successful in emulating all attempted gestures for all gesture libraries as long as we instruct mobCrawler to also look for common non-W3C standard event types.

Following up we performed a case study on 5 RIAs: 2 desktop-targeted, and 3 mobile-friendly web applications. For RQ2, as shown in Table 3, we found that there was a significant number of states and transitions found by non-click interactions. Most non-"click" interactions were found in the more interactive sites, but even the simple informational sites had a fair number of such transitions. The desktop applications had a large number of mouse and keyboard interactions leading to new states and transitions, while the mobile pages had many touch interactions. Swiping horizontally was more fruitful than vertical swipes. We witnessed an explosion in transitions leading to the same state for some sites, and advise careful configuration of the crawl parameters to fit your purpose. For RQ3, as shown in Table 4, we found that there was a significant amount of transitions caused by elements of other type than <button>and <a>. However, the elements used for interaction targets varied widely from site to site. The most consistent ones, aside from <a>, were , <html>and <body>. The latter two were often used as targets for interactions which could take place on the entire surface of the screen.

5. Related Work

There has been performed previous research utilizing event handlers by tools such as ARTEMIS [6] and FEEDEX [9]. These tools analyze event handler registrations, and then use the event handlers to prioritize what crawl action to take, though they dont attempt gesture emulations and focus on mouse clicking. [10] introduces a symbolic execution framework for finding security vulnerabilities in web applications. It utilizes event handler analysis and can perform simple event dispatches, but does not attempt to emulate gestures. In contrast to our tool, none of these approaches try to find and emulate complex gestures, and most of them, except [10], ignore simple event dispatches.

6. Conclusion and Future Work

Our findings indicate the technique could be combined with existing techniques to improve recall for automatic test generation [2], content indexing, and automatic evaluations of errors and security [3], especially for mobile web applications with advanced interaction options. In the future we will perform more case studies with the improved version of the tool to further clarify the applicability of the technique.

References

[1] A. Mesbah, E. Bozdag, and A. van Deursen, Crawling ajax by inferring user interface state changes, in *ICWE'08*, Yorktown Heights, NY, USA, 2008, pp. 122-134.
[2] A. Mesbah and A. van Deursen, Invariant-based automatic testing of ajax user interfaces, in *ICSE'09*, Vancouver, Canada, 2009, pp. 210220.
[3] C. P. Bezemer, A. Mesbah, and A. van Deursen, Automated security testing of web widget interactions, in *ESEC/FSE'09*, Amsterdam, Netherlands, 2009, pp. 8190.
[4] F. Ferrucci, F. Sarro, D. Ronca, and S. Abraho, A crawljax based approach to exploit traditional accessibility evaluation tools for AJAX applications, in *Information technology and innovation trends in organizations*, (Springer, 2011), pp. 255-262.
[5] Y. Maezawa, K. Nishiura, H. Washizaki, and S. Honiden, Validating ajax applications using a delay-based mutation technique, in *ASE'14*, Vsters, Sweden, 2014, pp. 491-502.
[6] S. Artziz, J. Dolby, S. H. Jensen, A. Moller, and F. Tip, A framework for automated testing of javascript web applications, in *ICSE'11*, Honolulu, HI, USA, 2011, pp. 571-580.
[7] A. Marchetto, P. Tonella, and F. Ricca, State-based testing of ajax web applications, in *ICST'08*, Lillehammer, Norway, 2008, pp. 121-130.
[8] D. Roest, A. Mesbah, and A. van Deursen, Regression testing ajax applications: Coping with dynamism, in *ICST'10*, Paris, France, 2010, pp. 127-136.
[9] A. M. Fard, and A. Mesbah, Feedback-directed exploration of web applications to derive test models, in *ISSRE'13*, Pasadena, CA, USA, 2013, pp. 278-287.
[10] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, A symbolic execution framework for javascript, in *IEEE S&P'10*, Oakland, CA, USA, 2010, pp. 513-528.