FireCrystal: Understanding Interactive Behaviors in Dynamic Web Pages

Stephen Oney and Brad Myers
Human-Computer Interaction Institute - Carnegie Mellon University
{soney,bam}@cs.cmu.edu

Abstract

For developers debugging their own code, augmenting the code of others, or trying to learn the implementation details of interactive behaviors, understanding how web pages work is a fundamental problem. FireCrystal is a new Firefox extension that allows developers to indicate interactive behaviors of interest, and shows the specific code (Javascript, CSS, and HTML) that is responsible for those behaviors. FireCrystal provides an execution timeline that users can scrub back and forth, and the ability to select items of interest in the actual web page UI to see the associated code. FireCrystal may be especially useful for developers who are trying to learn the implementation details of interactive behaviors, so they can reuse these behaviors in their own web site.

1. Introduction

Examples serve as a crucial source of inspiration for designers while they are exploring possible designs [1]. While examples rarely provide exactly the functionality that a designer wants, they can serve as a foundation that can be customized and augmented. One development process frequently used by designers and programmers alike is to search for an example of what they want, copy that example, and then adapt it to fit their needs [2-4].

1.1. Interactive Behaviors

While it is often easy to replicate the look of a web page or other interactive application, it is very difficult to replicate the feel [5]. Designers interested in adopting an interactive behavior for their own uses would ordinarily have to search through HTML, JavaScript, and CSS code fragments, which are frequently poorly formatted, spread across many files, contain fragments unrelated to the behavior the designer is interested in, and give no hint to the parts of the behavior they are responsible for.

We have created a new tool called FireCrystal to help designers inspect interactive behaviors they find on web pages. While the Internet is a great repository of interactive behaviors for designers to use and incorporate into their own original designs, there have been several problems encountered by designers interested in adapting interactive behaviors from web sites, which are summarized in the next sections, along with FireCrystal's approach to helping with them.

1.2. HTML, CSS, Javascript, and the DOM

In most interactive web pages, the HTML and Javascript contribute to the page's *Document Object Model (DOM)*, which is an object model that controls the structure of the page. Usually, HTML represents the initial DOM, which is then manipulated by attached Javascript code to make the page interactive. CSS controls the look of the page, and sometimes controls the positions of elements. Even if a designer is familiar with HTML, Javascript, and CSS, the interaction and interplay between them can make it even more difficult to see what pieces of code are responsible for a particular interactive behavior.

1.3. Readability

Another source of difficulty in extracting behaviors is that there is no visible mapping between the source code and what the user sees on the web page. In addition to functionality being spread across three different languages, even within the Javascript code itself, small interdependent fragments of code are frequently spread across multiple files. Unlike many programs written in formal languages like Java, it is nearly impossible to trace execution paths to figure out how a program works, as there is no single entry-point into a program. Instead, many interactive behaviors run when triggered by input handlers attached to an element, one its parents, or a seemingly irrelevant element.

Web page code is almost never designed for readability; it is frequently poorly formatted, or sometimes intentionally rendered unreadable through

978-1-4244-4876-0/09/$25.00 ©2009 IEEE

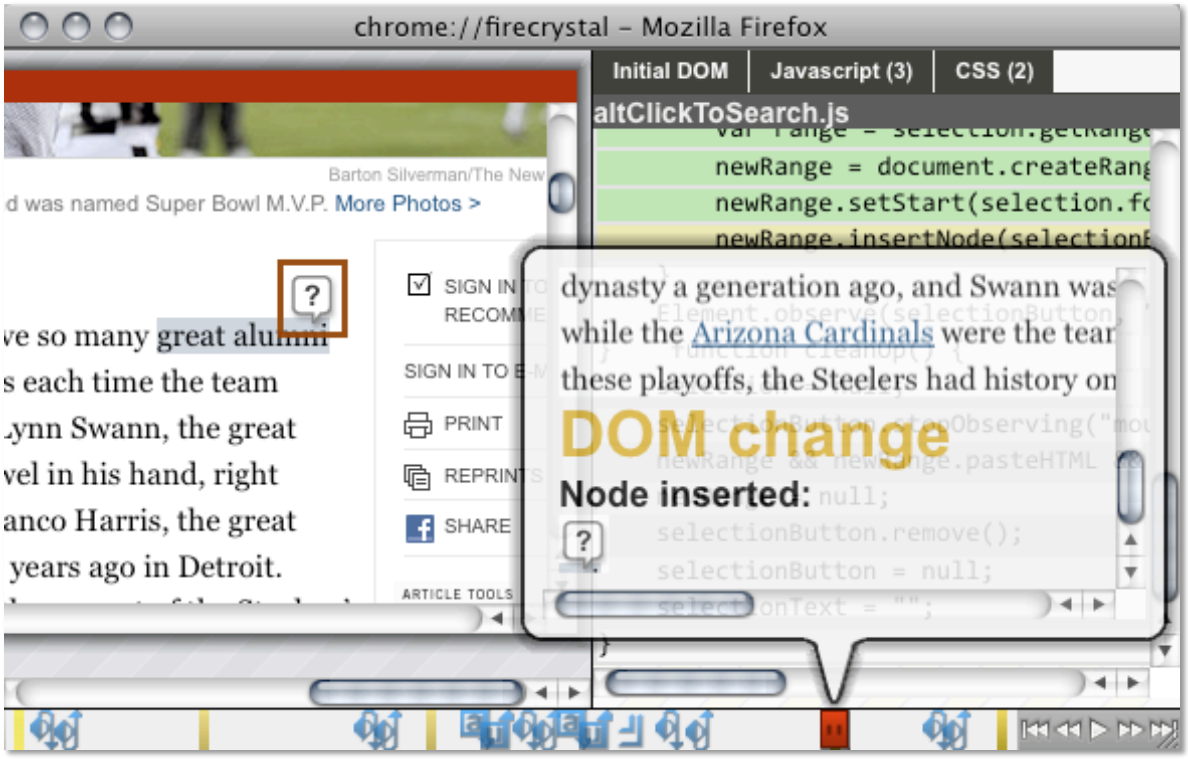

Figure 1 - FireCrystal replaying an interaction with an article on the New York Times website (http://www.nytimes.com/). In this interaction, the question mark box (outlined in red) appears after the user has highlighted a phrase ("great alumni" in this case). The timeline on the bottom of the screen shows user input events (in blue, from left: mouse down and up events, keyboard keys down and up, and window resizing, etc.) and DOM changes (as yellow bars). As the user drags the timeline handle (in red) to different times, a callout box gives more details on the event, the interaction is shown on a clone of the web page (left panel), and relevant code (Javascript, HTML, or CSS) is highlighted (right panel)

code obfuscation methods ranging from eliminating whitespaces to renaming variables or even inserting nonfunctional code fragments.

1.4. Our Approach

In light of these problems, we created FireCrystal, an add-on for the Firefox web browser that allows users to record execution paths while they are interacting with an interactive web page. To use FireCrystal, users tell it to start recording, then demonstrate the interactive behavior they are interested in extracting. FireCrystal records the interaction, keeping track of DOM changes, Javascript code executions, and user input events. When the user is finished recording, they can replay the interaction, and FireCrystal displays the HTML, CSS, and Javascript code that affected a particular element at any specific time.

Suppose a user wants to understand the "define" behavior of the New York Times (www.nytimes.com) web page, which allows users to highlight a word to pop up the question mark, as shown in Figure 1. The user can then click on the question mark to be shown the definition of the word. Without FireCrystal, a designer would have to carefully look through the Javascript files included by the page (in this case, over 50 files and thousands of lines of code).

With FireCrystal, the user can demonstrate the behavior while FireCrystal is recording. FireCrystal then displays the window shown in Figure 1. The replay window initially displays the web page in the same state as when the user started recording. The user can then use the timeline to find the point of interest. The timeline shows where DOM changes were made as well as where user input events happened. The user can click on an action in the time line, or else can focus in on the element of interest by clicking on it in the replay window.

FireCrystal then displays the HTML, JavaScript, and CSS code that caused that action or affected that element by highlighting the lines and files. Javascript code that was just run is highlighted in green, and Javascript code that makes a DOM change is highlighted in yellow. With FireCrystal, the user can

easily see what user input this interactive behavior reacts to and how it works.

2. Related Work

Crystal, the namesake of FireCrystal, is a framework aimed at enabling application developers to create applications that allow users to debug user-level features of the interface [6]. FireCrystal is also aimed at making the interface "clear", but focuses on web page code instead of user actions in complex applications.

The other major influence on FireCrystal is the Whyline [7], which allows users to ask "why" and "why not" questions about their programs. The web poses a unique challenge, however, because web pages use a mixture of imperative (Javascript) and declarative (HTML and CSS) syntaxes, unlike the strictly imperative syntaxes of Java and Alice addressed by the Whyline.

There are many widely used Javascript debuggers in Firefox and other browsers. According to the Firefox add-on download site, the most popular debugger is Firebug [8]. Firebug allows users to perform standard debugger actions, such as pause, step over, step into, and step return Javascript applications. It also has tools to analyze the CSS and DOM trees of a particular web site. However, it is still difficult to debug interactive behaviors in Firebug – stepping often does not work because users have to invoke the interactive behavior in order to get the code to run. In addition, there is no built-in way to log what happened during a particular interaction.

Also related are the tools that help users find relevant examples, such as Mica [9] and BluePrint [10]. However, these tools do not help users understand the examples.

3. FireCrystal

FireCrystal is a standalone Firefox add-on, written in a combination of XUL and Javascript. Upon activation, the user is given the option to start recording on command, or to reload the target page and start recording upon reload, to catch events that occur as soon as the page loads. When FireCrystal starts recording, it saves a complete copy of the target page's DOM, which will be used for the replay later on.

While FireCrystal is recording, it listens to DOM changes, user input events, and Javascript executions. All of these types of events are placed into a single "event log" that is used for the replay.

FireCrystal uses the standard HTML event/listener framework, as provided by the Firefox browser, to add hooks for DOM change events. Every time the DOM changes, an object describing that DOM change is sent to FireCrystal. FireCrystal then generates two methods – one called "next" that can replicate that DOM change and another called "prev" that can nullify the effects of the change. Together, these methods will be used in the replay window to allow the user to scrub back in forth while replaying their interaction with the target page. For example, if an element's color is changed from blue to red, FireCrystal generates a "next" method that changes the color of a clone of that element to red, and a "prev" method that changes it to blue. By generating a stack of methods rather than making complete DOM copies, FireCrystal saves processor time and memory. An object with details of the DOM change and these two methods is then placed onto the event log.

FireCrystal also uses the standard HTML event/listener framework from Firefox to add hooks so it is notified about most types of possible user input events, including mouse movement, mouse clicks, key presses, scrolling, and window resizing. Any time one of these events occurs, an object with the details of the input is placed on the event log.

FireCrystal uses the native Firefox debugging service to record every time a line of Javascript code is run on the target page. FireCrystal only records the line number and source file that is executing and does no further analysis on the Javascript code itself. This step is the most processor-intensive of any FireCrystal feature and sometimes slows down pages that execute a lot of Javascript when the user is recording.

When FireCrystal finishes recording, the target window is hidden and the "replay window" shown in Figure 1 appears. The timeline allows users to move back and forth through the recording and displays markers representing DOM changes and small icons representing the different types of user input events. The replay window starts by showing the page as it looked when the user started recording (using a complete DOM clone) and then uses the methods in the event log to change it to show the user interface state at the point of time that the user selected along the timeline. The replay window uses a copy of the original page's CSS to mimic the look of the original window. The replay window also shows certain user interactions, such as mouse movement (with a mouse cursor image), clicks (with ripples on the page where the user clicked), and window resizes (by resizing the internal frame). Javascript that would normally run on the host page is disabled to prevent possibly harmful side effects of re-running code. The replay window also shows the names of all of the files used by the web page during the user's interaction and highlights files that contain code that is responsible for a DOM change (in yellow). If the user has selected a particular element

to focus on in the replay window (such as the question mark box in Figure 1), files with code that affected that element, whether it is HTML, Javascript, or CSS are highlighted. All code is also automatically formatted for readability.

Note that FireCrystal is able to do this without interpreting or analyzing the code itself – it assumes that any lines of code that are executed before the selected DOM change is relevant to that change.

4. Limitations and Future Work

While FireCrystal helps designers extract interactive behaviors, it only works with code that is run and viewable on the client-side. Thus, it does not work with compiled languages like Adobe Flash or with server-side code written in PHP, ASP, etc. In addition, while FireCrystal takes measures to deobfuscate the Javascript, HTML, and CSS code of web pages – by automatically formatting the code and highlighting relevant code, there are obfuscation techniques that FireCrystal is still vulnerable to. For example, if a web developer intentionally inserts large amounts of code that is executed with no effect, FireCrystal has no easy way of distinguishing this code from the code responsible for the behavior, as FireCrystal currently does not try to "understand" the Javascript code, and Firefox does supply any tools to help with such analysis.

Modifications to the Firefox development platform would enable many additions to FireCrystal. In particular, the Firefox platform does not support developers listening for changes made in CSS "pseudo-classes", which can be responsible for certain types of animations that currently do not show up as DOM changes. In addition, the Firefox add-on development API does not enable in-depth analysis of Javascript code that might enable FireCrystal to automatically extract and adapt behaviors.

Users might also benefit if FireCrystal were integrated with other Javascript debugging platforms like Firebug [8]. We need to determine the best way to integrate the features of FireCrystal and Firebug so that users can take advantage of the strengths of both add-ons. Finally, we have plans for many more features that will help designers copy and paste the interactive behaviors into their own code, after they have used FireCrystal to isolate and understand the example's code.

5. Conclusion

FireCrystal is a tool aimed at helping designers and developers extract interactive behaviors that they can then adapt for use in their own interactive applications. FireCrystal is unique because of its focus on interactive behaviors in an environment that involves three languages – two declarative languages (HTML and CSS) and one imperative language (Javascript). FireCrystal allows the user to see what particular code or declarations affect an element at any time.

6. Acknowledgements

The authors would like to thank Andrew Faulring, Thomas LaToza, Christopher Scaffidi, Jeff Stylos, and Jeff Wong for their invaluable assistance. We would also like to thank the ARCS Foundation. Adobe has provided wonderful financial support and advice. This research was also supported by the National Science Foundation under grants IIS-0757511 and CCF-0811610. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

7. References

[1] C. Eckert, and M. Stacey, "Sources of inspiration: a language of design," *Design Studies,* vol. 21, no. 5, pp. 523-538, 2000.

[2] M. B. Rosson, and J. Carroll, "The reuse of uses in Smalltalk programming," *ACM TOCHI,* vol. 3, no. 3, pp. 219-253, 1996.

[3] G. Fischer, "Cognitive view of reuse and redesign," *IEEE Software,* vol. 4, no. 4, pp. 60-72, 1987.

[4] J. Brandt, P. J. Guo, J. Lewenstein *et al.*, "Two studies of opportunistic programming: interleaving web foraging, learning, and writing code," *Proceedings of the 27th international conference on Human factors in computing systems*, 2009.

[5] B. A. Myers, S. Y. Park, Y. Nakano *et al.*, "How Designers Design and Program Interactive Behaviors," *VL/HCC,*, pp. 177-184, 2008.

[6] B. A. Myers, D. A. Weitzman, A. J. Ko *et al.*, "Answering why and why not questions in user interfaces," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Montreal, Quebec, Canada, 2006.

[7] A. J. Ko, and B. A. Myers, "Debugging reinvented: asking and answering why and why not questions about program behavior," *ICSE*, Leipzig, Germany, 2008.

[8] I. Parakey. "Firebug: Web Development Evolved," http://getfirebug.com/.

[9] J. Stylos, and B. Myers, "Mica: A Web-Search Tool for Finding API Components and Examples," *VL/HCC*, pp. 5-7;195-202, 2006.

[10] J. Brandt, M. Dontcheva, M. Weskamp *et al.*, "Example-Centric Programming: Integrating Web Search into the Development Environment," *Stanford University Technical Report, CSTR*, no. 2009-01, 2009.