Extracting Client-Side Web User Interface Controls*

Josip Maras[1], Maja Štula[1], and Jan Carlson[2]

[1] University of Split, Croatia
[2] Mälardalen Real-Time Research Center, Mälardalen University, Västerås, Sweden
{josip.maras,maja.stula}@fesb.hr, jan.carlson@mdh.se

Abstract. Web applications that are highly dynamic and interactive on the client side are becoming increasingly popular. As with any other type of applications, reuse offers considerable benefits. In this paper we present our first results on extracting easily reusable web user-interface controls. We have developed a tool called Firecrow that facilitates the extraction of reusable client side controls by dynamically analyzing a series of interactions, carried out by the developer, in order to extract the source code and the resources necessary for the reuse of the desired web user-interface control.

1 Introduction

Web developers now routinely use sophisticated scripting languages and other active client-side technologies to provide users with rich experiences that approximate the performance of desktop applications [1]. Unfortunately, because of the very short time-to-market and fast pace of technology development, reuse is often not one of the primary concerns. When developers are building new applications, they often encounter problems that have already been solved in the past. Rather than re-inventing the wheel, or spending time componentizing the already available solution, they resort to pragmatic reuse [2]. This is especially true for web development [3], where basically all client side code is open.

In web development, a web page, whose layout is defined with HTML (HyperText Markup Language), is represented with the Document Object Model (DOM). All client side interactions are realized with JavaScript modifications of the DOM and the presentation of the web page is usually defined with CSS (Cascading Style Sheets). So, in order to be able to understand a web page and extract a reusable control, a developer has to be familiar with HTML, JavaScript and CSS, and has to be able to make sense of the interactions between the three separate parts that produce the end result. This is not a simple task, since there is no trivial mapping between the source code and the page displayed in the browser; the responsible code is often scattered between several files and is often intermixed with code irrelevant for the reuse task. For example, even extracting the required CSS styles is not a trivial process for the developer since CSS styles are inherited from parent elements, they are often defined in multiple files and

* This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS, and the Unity Through Knowledge Fund supported by Croatian Government and the World Bank via the DICES project.

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 502–505, 2010.

can be dynamically changed while the application is executing (either with JavaScript or with CSS pseudo-classes). Required resources, such as images, can be included either directly as HTML nodes, or via CSS styles or dynamically with JavaScript code, and when transferring them the developer has to locate all used ones, copy them and adjust for the changed location (e.g. if some resources are included via absolute paths).

In this paper we propose a way of semi-automatically extracting reusable client-side controls. Using a tree-based DOM explorer the developer selects the HTML nodes that represent the desired user-interface (UI) control on the web page and executes the interactions that represent the behavior he/she wants to reuse. Based on the analysis done during the execution, all resources such as images, CSS styles, JavaScript and HTML code snippets required for the inclusion of the selected UI control are extracted.

The work presented in this paper is related to web application program slicing and extracting code responsible for the desired behavior of the selected web UI control. In the context of web engineering, Tonella and Ricca [4] define web application slicing as a process which results in a portion of the web application which exhibits the same behavior as the initial web application in terms of information of interest displayed to the user. There also exist two related approaches and tools that facilitate the understanding of dynamic web page behavior: Script InSight [5] and FireCrystal [6] that have a similar functionality of relating elements in the browser with the code responsible for them. However, they are both different to FireCrow in that they do not provide support for code extraction.

2 Method

If we set aside plugins, such as Java Applets, Flash or Silverlight, the client side of the web page is usually composed of three different parts: HTML code that defines the structure and the layout of the page, CSS code that defines the style and the presentation, and JavaScript code that is responsible for executing client-side business logic and interactive layout manipulation. Based on their mutual interactions and resources such as images, the browser engine renders the web page.

In order to extract the minimum amount of resources necessary for the reuse of the selected web UI control we have developed Firecrow, an extension to the Firebug[1] debugger, which supports a three-step extraction process (as shown in Figure 1).

In the fist step the developer chooses the part of the web page user interface that he/she wishes to reuse. Firecrow uses the Firebug's HTML DOM tree which makes it easy for the developer to choose HTML nodes that constitute the desired section of the user interface.

In the second step the developer starts the recording phase. This can be done in three modes: *(i)* Without tracking the execution of the JavaScript code — this mode is useful when the developer only wishes to reuse the layout and styles of the selected HTML nodes. No attempts for extracting the JavaScript code are made. *(ii)* Simple tracking of the executed JavaScript code — in this mode the tool keeps track of the executed JavaScript code. In that way, when the process is finished the tool extracts only the code

[1] Firebug is a plugin for the Firefox browser available for download from: `http://getfirebug.com`

that was executed while in the recording phase. But, often web application code is not designed for reuse, so it is likely that the executed code will not only be concerned with the desired functionality. In that case the developer has to manually locate and remove all statements that could cause the extracted application to break (e.g. by accessing HTML nodes that are removed in the process of extraction). *(iii)* Advanced tracking of executed JavaScript code — in this mode the tool analyzes all executed JavaScript statements searching for statements that are in any way connected with HTML nodes that are removed in the process of extraction. After the process of extraction is completed, the tool annotates the source code with warnings that make it easier to locate those lines.

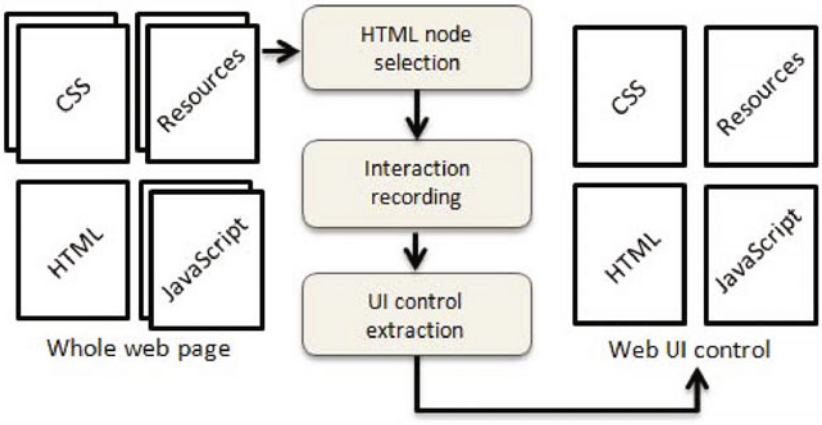

Fig. 1. The Firecrow extraction process

The reason for providing three different degrees of analysis is that each additional layer of analysis comes with a performance cost. For example, in the case of JavaScript intensive web applications, the last mode of advanced JavaScript code tracking could in some cases make the UI non-responsive.

The third step starts when the recording phase is finished — the tool, based on the selected resources, and the code executed while interacting with the UI control, extracts the necessary HTML code, CSS styles, resources and JavaScript code. This significantly simplifies the process of reuse, since the developer only has to include the extracted CSS styles and JavaScript code, and copy the extracted HTML code to the desired location.

2.1 Firecrow

Firecrow is realized as an extension to Firebug, which is a plugin for the Firefox browser. This allows taking advantage of the functionality provided by Firebug to analyze the CSS dependencies and the DOM of selected web page. Also, Firecrow is connected to the Firefox JavaScript engine in order to dynamically analyze the executed JavaScript code. An evaluation version of the tool, as well as a video showing how it can be used, is available from `www.fesb.hr/~jomaras/Firecrow`.

3 Limitations and Future Work

Firecrow is primarily designed to extract easily reusable client side web UI controls developed with a combination of JavaScript, HTML and CSS code. For those reasons there is currently no support for the extraction of Silverlight nor Flash UI controls. Also, since Firecrow is directly connected with the Firefox JavaScript engine, code specifically developed to be executed only in some other browser (e.g. Internet Explorer, Opera, Safari, etc.) will not be extracted.

An important remark is that, even though Firecrow does relatively simple dynamic analysis, in the case of web applications that are very JavaScript intensive there is a significant drop of performance. This is especially noticeable when doing advanced dynamic analysis. Currently we offer one way of tackling this problem: today many applications use JavaScript libraries which provide common infrastructure for easier development of JavaScript applications (out of Alexa Top 10000 Web sites 32,5% use at least one of JavaScript libraries [7]). These libraries are complex and are usually often reused as is. The developer can exclude those scripts from the dynamic analysis phase, in that way focusing only on the code that is realizing the desired functionality on a higher level of abstraction and not on the common library code that will most probably be reused as is.

So far, Firecrow only annotates the source code with warnings that show code statements that can potentially stop the application execution. As future work, we plan to expand this with the additional functionality of removing all source code statements that are not related with DOM modifications of the selected UI controls. Also, we will have to find a way to avoid performance problems when doing dynamic analysis in JavaScript intensive applications.

References

1. Wright, A.: Ready for a Web OS? ACM Commun. 52(12), 16–17 (2009)
2. Holmes, R.: Pragmatic Software Reuse. PhD thesis, University of Calgary, Canada (2008)
3. Brandt, J., Guo, P.J., Lewenstein, J., Klemmer, S.R.: Opportunistic programming: How rapid ideation and prototyping occur in practice. In: WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering, pp. 1–5. ACM, New York (2008)
4. Tonella, P., Ricca, F.: Web application slicing in presence of dynamic code generation. Automated Software Engg. 12(2), 259–288 (2005)
5. Li, P., Wohlstadter, E.: Script insight: Using models to explore javascript code from the browser view. In: Proceedings of Web Engineering, 9th International Conference, ICWE 2009, San Sebastián, Spain, June 24-26, pp. 260–274 (2009)
6. Oney, S., Myers, B.: Firecrystal: Understanding interactive behaviors in dynamic web pages. In: VLHCC '09: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 105–108. IEEE Computer Society, Los Alamitos (2009)
7. BackendBattles: Javascript libraries (2010), `http://www.backendbattles.com/JavaScript_Libraries`