Firecrow – A tool for Web Application Analysis and Reuse

Josip Maras
University of Split
Croatia
josip.maras@fesb.hr

Maja Štula
University of Split
Croatia
maja.stula@fesb.hr

Jan Carlson
Mälaradalen University
Sweden
jan.carlson@mdh.se

ABSTRACT

This paper presents Firecrow – a tool for Web application analysis and reuse. The tool's primary function is to support reuse of client-side features, but it can also be used for feature identification, web application slicing, and generation of usage scenarios, i.e. sequences of user actions that cause the manifestation of application behaviors. The tool is in prototype stage and is accessible through a plug-in to the Firefox browser, but it can also be used as a library from other browsers (e.g. Chrome, Safari, and PhantomJs).

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.13 [**Software Engineering**]: Reusable Software

Keywords

Web; Tool; Feature Identification; Reuse; Testing

1. INTRODUCTION

The domain of web applications is one of the most pervasive and fastest growing application domains today. In the last twenty-five years, web applications have made significant advances, from simple static interconnected documents to full-blown applications that can almost match standard desktop applications in terms of responsiveness and functionality. However, at the same time, there exists a mismatch between their pervasiveness and the state of the art of tools and methods supporting their development. Compared to some other, more traditional software engineering domains, there is a lack of tools and methods for their development, analysis, testing, and reuse.

Constructing new software systems by reusing already existing artifacts can reduce development time and decrease defect density [12, 18, 6]. Reuse can even lead to increased productivity [3], improved quality [4], and more satisfied customers [19]. For these reasons, a number of approaches that facilitate reuse have been developed (e.g. component-based development [12], software product-lines [14]). Most of these approaches address pre-planned reuse, where software entities are built to be easily reusable. However, it would also be very beneficial if we could reuse existing code that was not developed with reuse in mind [5]. In such cases, identifying the exact code that we want to reuse, as well as integrating it into an already existing application can be a challenging and time-consuming task.

Web applications are composed out of two equally important parts: the *server-side* which is usually responsible for data-access and business logic, and the *client-side* which acts as an event-driven user-interface (UI) to the server-side. The two primary functions of client-side applications are: *i)* to communicate with the user over their UI, and *ii)* to communicate with the server by exchanging messages. For this reason, we have defined [8] that a client-side feature is a subset of the application's behavior, triggered by sequences of user-generated events, that manifest at runtime with: *i)* changes to certain parts of the application structure, and/or *ii)* communications with the server. Client-side applications are implemented through three programming languages: HTML for structure, CSS for the presentational aspects, and JavaScript for the behavior.

In our previous work, we have developed methods for supporting feature reuse in web application development; more specifically, methods that enable reuse of features that were not necessarily designed with reuse in mind. In order to do this, first we have to identify the source code that implements a particular feature [11], and then we have to integrate the code of the feature into the code of an already existing application [9]. The feature identification process and the feature integration process are both based on the dynamic analysis of web application execution. Since client-side web applications are user-interface (UI) applications, these executions are usually caused by certain sequences of user-generated events, i.e. usage scenarios. Because of this, we have also defined a method for generating usage scenarios [10] that cause the manifestations of particular application features.

In this paper we present Firecrow tool, an integrated tool that implements the functionalities of: *feature identification*, *automatic scenario generation*, and *feature reuse*, in the context of client-side web applications.

2. FIRECROW

The Firecrow tool is composed out of four subsystems (Figure 1): *i) DoppelBrowser*, that processes and interprets web application code, and creates a dependency graph and

ASE'14, September 15-19, 2014, Vasteras, Sweden.

http://dx.doi.org/10.1145/2642937.2648620.

execution summaries for a particular execution; *ii) Scenario Generator*, that automatically generates sequences of user actions (usage scenarios); *iii) Feature Locator*, that traverses the dependency graph, analyzes the execution summary, and identifies the code that implements a certain feature; and *iv) Feature Integrator*, that locates and fixes potential feature integration errors, and merges the feature code and the target application code. The *DoppelBrowser* is an underlying subsystem used by the other subsystems, while the *Scenario Generator*, the *Feature Locator*, and the *Feature Integrator* subsystems can be used in a stand-alone fashion by the user. Firecrow is available at github[1], and a video demonstration of the tool is available at `http://marjan.fesb.hr/~jomaras/tools.html`.

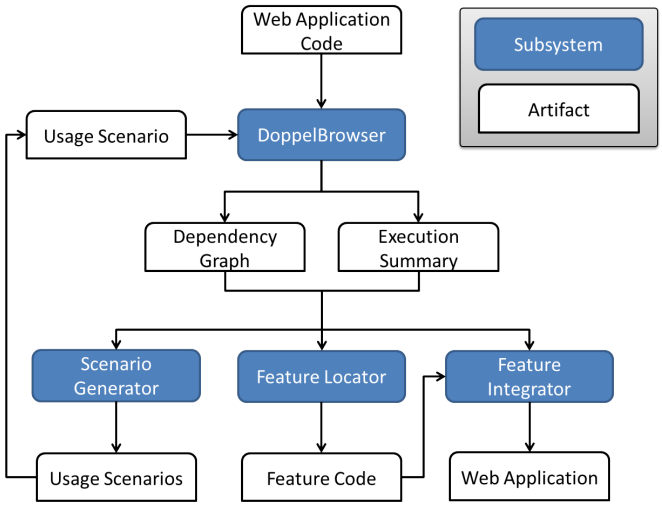

Figure 1: Firecrow subsystems and artifacts

2.1 DoppelBrowser

DoppelBrowser is a JavaScript library that interprets web application code according to standard rules of client-side web application interpretation. It includes a newly developed JavaScript interpreter that also keeps track of relationships between values and code expressions from where these values originate. This additional functionality enables it to construct a client-side dependency graph [11] that captures dependencies that exist in a particular execution (e.g. between an HTML element and a CSS rule that defines its visual properties, between a code expression that reads a value and the expression that assigns it, or between an HTML element and a JavaScript expression that modifies it). In addition, the library gathers execution summaries that capture important run-time information. The library is browser agnostic and we have used it from different browsers: Firefox, Chrome, Safari, and PhantomJs.

2.2 Scenario Generator

The *Scenario Generator* is a JavaScript library that makes use of the *DoppelBrowser* library and that automates the process of generating usage scenarios, i.e. sequences of user events that capture either the behavior of the whole application or the behavior of a particular feature. The process is similar to automatic web application testing, and is composed of two phases [10]: *i) Scenario Generation* and *ii) Scenario Filtering*. The *Scenario Generation* phase starts by creating an initial, empty scenario that represents the act of loading the page, without any user input. The approach then proceeds by iteratively selecting a scenario, executing it and dynamically analyzing the execution. New scenarios are then generated by: *i)* extending event chains – all event registrations and data-dependencies during the execution are tracked, and new scenarios are generated by extending the event chain of the current scenario with newly registered events, or with previously executed events whose execution potentially depends on the variables and objects modified by the current scenario; and by *ii)* modifying the input parameters, i.e. the internal state of the browser and the event parameters; the process tracks how the input parameters influence the control-flow of the application with concolic execution [17], and generates new scenarios by modifying those inputs. In the second phase of *Scenario Filtering*, execution traces of all executed scenarios are analyzed and the set of scenarios is filtered. If the process targets certain application features, all scenarios that do not cause the manifestation of those features are removed. In addition, we also remove scenarios whose removal does not lower the overall code coverage.

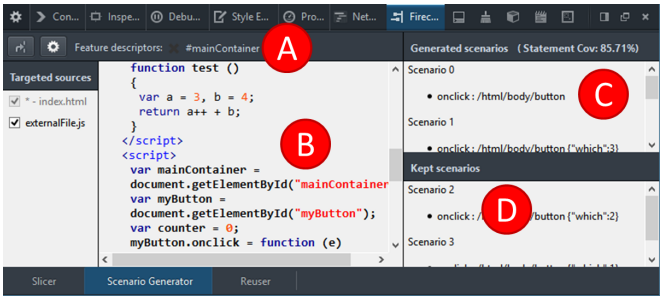

Figure 2: Scenario Generator used as a Firefox plugin. A – toolbar; B – web application code; C – Generated Scenarios; D – Kept scenarios

Figure 2 shows the UI of the Scenario Generator subsystem, when accessed through a Firefox plugin. Mark A indicates the toolbar that is used to specify feature selectors and to start the process; mark B shows the source code of the application, where the bold text denotes parts of the source code that was executed by at least one scenario; mark C shows all scenarios that were generated in the process, and mark D shows the scenarios that were kept after the scenario filtering phase.

2.3 Feature Locator

Feature Locator is a JavaScript library developed with the purpose of identifying code that implements a particular feature. It operates on the artifacts created by the *DoppelBrowser* library: the dependency graph and the execution summary. As the *DoppelBrowser* library, it can be used from different browsers.

From the user's perspective, a client-side application offers a number of features that are relatively easy to distinguish. However, the same can not be said for their implementation details. A feature is implemented by a subset of the application's code and resources, and identifying the exact subset is

[1]https://github.com/jomaras/Firecrow

a challenging task. In our approach, we execute the application with a certain scenario that causes the manifestation of a feature described by a set of feature descriptors (i.e. CSS selectors[2] or XPath expressions[3]). During this execution the *DoppelBrowser* builds a client-side dependency graph, identifies points in the execution where a feature behavior can be observed (feature manifestation points), and gathers an execution summary necessary for the accurate identification and extraction of feature code. Once the application execution is completed, the process traverses the dependency graph for each part of the feature structure and for every feature manifestation point, and in that way identifies the feature code [11]. In essence, the process is performing dynamic program slicing [1] with the feature manifestation points and parts of the feature structure as a slicing criteria.

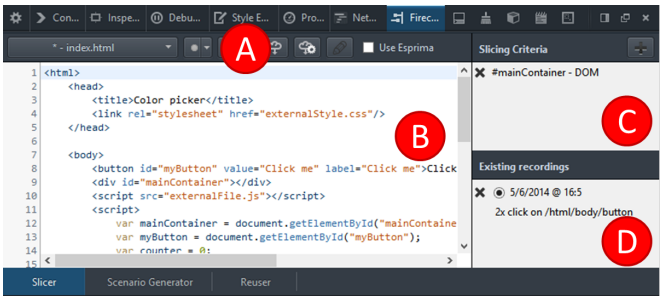

Figure 3: Feature Locator used as a Firefox plugin. A – action toolbar; B – DOM viewer; C – Slicing Criteria; D – Scenario description

Figure 3 shows the UI of the *Feature Locator* subsystem, when accessed through a Firefox plugin. Mark A indicates the toolbar that allows browsing though application source code, recording scenarios, and initiating the feature identification process; mark B shows the DOM viewer which enables easy specification of parts of web page structure where the feature manifests; mark C shows a container with slicing criteria (either points in the source code, or CSS selectors that point to certain parts of the page); mark D shows a list of all events for the recorded scenario.

2.4 Feature Integrator

The *Feature Integrator* is a JavaScript library that provides the functionality of detecting and fixing conflicts, and integrating feature code identified with the *Feature Locator* subsystem into an already existing application. It uses the functionalities of the *DoppelBrowser* library to execute and analyze the application, and the *Feature Locator* application to identify the feature code that will be reused.

Once the feature code has been identified, in order to achieve reuse, we have to integrate it with the code of the target application. This is a complex task, because by doing this we are changing the environment on which both the feature code and the target application code rely for their behavior. This can lead to a number of errors in behavior and differences in presentation for both the extracted feature and the target application. These errors occur because of conflicts between two code bases. So in order to integrate the code of the feature into the code of the target application, we have to detect and fix all possible conflicts. In our previous work, we have defined all types of conflicts, and have specified methods for their resolving [9].

[2]http://www.w3.org/TR/CSS2/selector.html
[3]http://www.w3.org/TR/xpath/

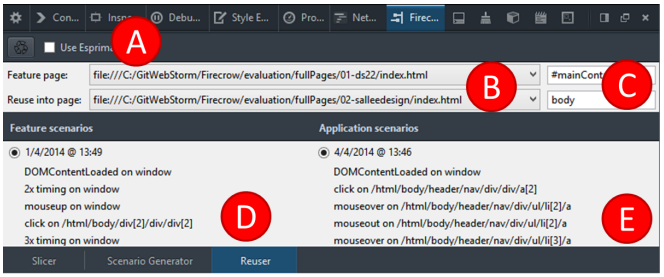

Figure 4: Feature Integrator used as a Firefox plugin. A – toolbar; B – Setting the feature and the target page; C – Specifying the parts of the page structure that define the feature and the position where the feature will be reused; D – Feature scenarios; E – application scenarios

Figure 4 shows the UI of the Feature Integrator subsystem, used as a Firefox plugin. Marks A, B, and C indicate the toolbar that allows the user to select the feature page and specify parts of the page structure where the feature manifests, and the page where the feature will be reused into, as well as the exact position in the page structure where the feature will be integrated; mark D shows the feature scenarios, and mark E the scenarios of the target application.

3. RELATED WORK

A number of approaches that support some form of reuse have been developed: Hunter Gatherer [16], HTMLviewPad [21], and Internet Scrapbook [20] in the web domain; and G&P [5] in the Java domain.

HunterGatherer [16], Internet Scrapbook [20], and HTMLviewPad [21] are similar approaches related to clipping fragments of Web pages. Users can create personalized pages that gather data from different sources. Since these approaches were developed in the 1990's and early 2000, when web page development was not as dynamic on the client-side, currently their usability is quite limited.

For Java applications, G&P [5] is a tool suite composed of two tools, Gilligan and Procrustes, that support pragmatic reuse tasks. Gilligan enables the investigation of dependencies from a desired functionality and the construction of a reuse plan, while Procrustes extracts the relevant code from the originating system, makes modifications that reduce the errors in compilation and includes it into the target system. The main difference between our approaches is that G&P statically analyzes Java applications; while some of the ideas and end goals are similar, these tools can not be used to support reuse on the client-side web applications.

Our tool is related to tools for automatic testing of web applications, e.g. Kudzu [15] and Artemis [2]. Kudzu explores the application's event space with GUI exploration, i.e. by searching the space of all event sequences with a random exploration strategy; and the application's value space by using dynamic symbolic execution. In the process, they have also developed a string constraint solver that deals with the particularities of string constraints for JavaScript. Artemis [2] uses feedback directed testing of JavaScript ap-

plications and is based on dynamic analysis of web application execution. Neither of these tools enable users to target specific client-side features, nor do they filter the generated scenarios in order to remove the unnecessary ones.

There are also tools that support the understanding of dynamic web page behavior: FireCrystal [13] and Script InSight [7]. FireCrystal supports the understanding of interactive behaviors by recording user interactions and logging DOM changes, user generated events, and JavaScript code executions. The user can then use a time-line to study the code executed for the particular behavior. Script InSight relates the elements in the page with the lower-level JavaScript syntax, by gathering data during the script's execution and building a context-sensitive, control-flow model with tracing information. Compared to our approach they make no attempts to track data dependencies between different code expressions, nor to identify individual features in the analyzed code.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a prototype open-source tool for client-side web application analysis and reuse – Firecrow. The tool provides the functionality of automatic usage scenario generation, feature identification, and feature integration. These three functionalities target the general goal of facilitating reuse in the domain of client-side web applications. The tool is developed as a JavaScript library and can be used from different browsers: Firefox, Chrome, Safari, PhantomJs. However, due to the advantages of the Firefox platform, the tool is integrated into the Firefox browser, where it can be used as a plugin to Firefox Web Developer Tools.

For future work, we plan to explore the usage of some of its functionalities for different purposes than reuse, e.g. feature identification could be used to facilitate debugging, code understanding, maintenance, and even for deriving various software metrics. Web applications in general are composed out of two parts: the server-side application which implements business logic and data access, and the client-side which represents the UI of the application. In its current state, Firecrow deals exclusively with the client-side, and as part of future work, we plan to extend it in order to support server-side applications also.

5. REFERENCES

[1] H. Agrawal and J. R. Horgan. Dynamic program slicing. *ACM SIGPLAN Notices*, 25(6):246–256, 1990.

[2] S. Artzi, J. Dolby, S. H. Jensen, A. Møller, and F. Tip. A framework for automated testing of Javascript web applications. In *Software Engineering, ICSE 2011, 33rd International Conference on*, pages 571–580. ACM, 2011.

[3] V. R. Basili, L. C. Briand, and W. L. Melo. How reuse influences productivity in object-oriented systems. *Communications of the ACM*, 39(10):104–116, 1996.

[4] W. B. Frakes and G. Succi. An industrial study of reuse, quality, and productivity. *Journal of Systems and Software*, 57(2):99–106, 2001.

[5] R. Holmes and R. J. Walker. Semi-Automating Pragmatic Reuse Tasks. Automated Software Engineering, pages 481–482. IEEE Computer Society, 2008.

[6] C. W. Krueger. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183, 1992.

[7] P. Li and E. Wohlstadter. Script insight: Using models to explore Javascript code from the browser view. In *International Conference on Web Engineering*, pages 260–274, 2009.

[8] J. Maras. *Automating Reuse in Web Application Development*. PhD thesis, Mälardalen University, April 2014.

[9] J. Maras, J. Carlson, and I. Crnković. Towards automatic client-side feature reuse. In *Web Information Systems Engineering–WISE 2013*, pages 479–488. Springer, 2013.

[10] J. Maras, M. Štula, and J. Carlson. Generating feature usage scenarios in client-side web applications. In *Web Engineering*, pages 186–200. Springer, 2013.

[11] J. Maras, M. Stula, J. Carlson, and I. Crnkovic. Identifying code of individual features in client-side web applications. 2013.

[12] M. D. McIlroy, J. Buxton, P. Naur, and B. Randell. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98. sn, 1968.

[13] S. Oney and B. Myers. FireCrystal: Understanding interactive behaviors in dynamic web pages. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 105–108. IEEE Computer Society, 2009.

[14] D. L. Parnas. On the design and development of program families. *Software Engineering, IEEE Transactions on*, (1):1–9, 1976.

[15] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. A symbolic execution framework for Javascript. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 513–528. IEEE, 2010.

[16] M. Schraefel, Y. Zhu, D. Modjeska, D. Wigdor, and S. Zhao. Hunter Gatherer: Interaction Support for the Creation and Management of Within-Web-Page Collections. World Wide Web, pages 172–181, 2002.

[17] K. Sen, D. Marinov, and G. Agha. *CUTE: a concolic unit testing engine for C*, volume 30. ACM, 2005.

[18] T. A. Standish. An essay on software reuse. *Software Engineering, IEEE Transactions on*, (5):494–497, 1984.

[19] G. Succi, L. Benedicenti, and T. Vernazza. Analysis of the effects of software reuse on customer satisfaction in an RPG environment. *Software Engineering, IEEE Transactions on*, 27(5):473–479, 2001.

[20] A. Sugiura and Y. Koseki. Internet scrapbook: creating personalized world wide web pages. Human Computer Interaction, pages 343–344. ACM, 1997.

[21] Y. Tanaka, K. Ito, and J. Fujima. Meme Media for Clipping and Combining Web Resources. *World Wide Web*, 9:117–142, 2006.