Understanding Behavioural Patterns in JavaScript

Saba Alimadadi
University of British Columbia
Vancouver, BC, Canada
saba@ece.ubc.ca

ABSTRACT

JavaScript is one of the most popular programming languages. However, understanding the dynamic behaviour of JavaScript apps is challenging in practice. There are many factors that hinder JavaScript comprehension, such as its dynamic, asynchronous, and event-driven nature, the dynamic interplay between JavaScript and the Document Object Model, and the asynchronous communication between client and server. In this research work, we have already proposed methods for understanding event-based and asynchronous JavaScript behaviour. To enhance the scalability of our methods, we propose a new technique that adopts bio-informatics algorithms to extract sequences of actions from execution traces that form higher-level patterns.

CCS Concepts

•**Software and its engineering** → *Software testing and debugging; Integrated and visual development environments;*

Keywords

JavaScript; comprehension; behavioural patterns

1. RESEARCH PROBLEM

Program comprehension is an essential first step for all software engineering tasks. Developers spend a considerable amount of time understanding code. They start with *searching* for cues in the code and the environment. Then they go back and forth on the incoming and outgoing dependencies to *relate* pieces of foraged information. Throughout the process, they *collect* information they find relevant for understanding the code [12]. However, developers fail often in searching and relating information, and lose track of relevant information. They collect information in their memory or use external sources to help them remember. They form a mental model of the entities, relations and the intent of the code, which they use throughout development to help them make decisions. Developers put much effort into creating and maintaining mental models. But exploring code is difficult and these models are almost always inaccurate [19]. It is shown that developers perform better using systematic strategies compared to "as-needed" strategies [24].

JavaScript has become the lingua franca of web development. It is voted as the most popular language [26] and is the most used language on GitHub [13]. Comprehending JavaScript applications entails a set of challenges for developers. JavaScript is single-threaded and thus callbacks are often exercised to simulate concurrency. Nested and asynchronous callbacks are used regularly to provide capabilities such as non-blocking I/O and concurrent request handling. This use of callbacks, however, can gravely complicate program comprehension – a problem coined as *callback hell*. The Document Object Model (DOM) and custom events, timers and XMLHttpRequest (XHR) objects interact with JavaScript code on the client and server to provide real-time interaction, all of which complicate understanding. Further, dynamic features of the JavaScript language pose a challenge to analysis techniques. For instance, almost everything in JavaScript, from fields and methods of objects to their parents, can be created or modified at runtime. Finally, client and server code communicate through XHR messages, and multiple messages (and their responses) can be in transit at a given time. As in any distributed system, there is no guarantee on the order or time of the arrival of requests at the server, and responses at the client. The uncertainty involved in the asynchronous communication makes the execution more intricate and thus more difficult to understand.

Our hypothesis is that we can improve developers' performance by inferring and creating higher-level models of the application's dynamic behaviour. Such models provide systematic means for assisting developers in the process of searching, relating and collecting the information necessary for understanding an application.

In this proposal, we introduce our approach for understanding the dynamic behaviour and execution patterns in JavaScript. We discuss our techniques for understanding event-based, dynamic, and asynchronous interactions in JavaScript. We then propose a technique for finding behavioural patterns in the execution traces of applications. Our goal is to provide a higher-level abstraction of execution that matches the mental model of the developers, instead of overwhelming them with the details of code-level execution.

2. RELATED WORK

There are numerous static analysis techniques proposed for JavaScript analysis in different domains [8, 10, 11, 20]. We did not choose a static approach, since many event-driven, dynamic and asynchronous features of JavaScript are not well supported statically. Dynamic and hybrid JavaScript analysis techniques have attempted to solve the shortcomings of static analysis [15, 21, 27]. However, existing techniques focus on the client-side of web applications alone, and do not consider the server. Also, non of these techniques provide higher-level patterns and semantic models of execution. Many papers have focused on locating the implementation of UI- and interaction-based features [14, 16, 28] in web applications. Record and replay techniques aid the understanding and debugging tasks of web applications [6, 7, 17, 18]. The goal of these techniques, however, is to provide a deterministic replay of UI events without capturing their consequences. Tracing techniques [5, 9, 22] collect traces of JavaScript execution selectively. Similar to our work, these tools provide a model and visualization of the trace. However, these methods do not extract behavioural patterns of execution and are not scalable to larger and more complex applications. There are many tools that use visualization to improve the process of understanding the behaviour of software applications [5, 22, 28]. However, their approaches are not concerned with creating a model of the web application, while ours is.

3. APPROACH

In this section, we explain our approach for assisting developers with comprehension of JavaScript applications.

3.1 Accomplished Work

First, we proposed a generic, non-intrusive technique, called Clematis, for supporting comprehension of event-based interactions in web applications [3]. Through a combination of automated JavaScript code instrumentation and transformation, we capture a detailed trace of a web application's behaviour during a particular user session. Our technique transforms the trace into an abstract behavioural model. The model is then presented to the developers as an interactive visualization that depicts the creation and flow of triggered events, the corresponding executed JavaScript functions, and the mutated DOM nodes, within each episode. To evaluate this approach, we conducted a controlled experiment with 20 professional web developers. The results showed that using Clematis significantly improved developers' performance in completing comprehension tasks, by improving task completion duration and accuracy by 61% and 89% on average, respectively.

Next, we proposed an approach to assist understanding the impact of change in the presence of dynamic, DOM-related and asynchronous features of JavaScript [1]. We introduced a hybrid change impact analysis technique for client-side JavaScript, called Tochal. Tochal augments static analysis with dynamic analysis to enable a DOM-sensitive and event-aware change impact analysis method. The results of conducting a controlled experiment with 12 professional web developers showed that Tochal was effective in improving developers' performance significantly. The participants that used Tochal had an average of 105% more accuracy, while spending 56% less time.

Developers use JavaScript not only for the client-side but also for server-side programming, leading to full-stack applications written entirely in JavaScript. The two previous techniques were agnostic of the server, where most of the program logic is located in full-stack applications. We proposed Sahand [2], a technique for capturing a behavioural model of full-stack JavaScript. Our approach captures the context-sensitive executions of events and functions during their lifespan. We designed a model to accommodate the temporal nature of function executions and the asynchronous scheduling mechanisms of full-stack JavaScript. The model is finally visualized for the developers. We designed another controlled experiment with 12 graduate students to evaluate Sahand. The results showed that using Sahand helped developers complete comprehension tasks 3 times more accurately, in about the same time.

To assist the process of searching, relating and collecting information, the aforementioned techniques collect execution traces, analyze them, and visualize the results for the developers. However, the collected information can become very large and complex, very fast. As a result, the visualized data itself can become incomprehensible for developers. To mitigate this problem, we propose a technique discussed in the next section.

3.2 Going Forward

As the next step, we propose to investigate the means of helping the comprehension process even further by providing a semantic model of the execution patterns. The goal is to find a higher-level model that better represents the mental model of developers. The model provides an overview of the application behaviour, and reveals the code-dependent details systematically based on user demand. Our proposed approach consists of the following steps.

First, we instrument JavaScript on the fly. Next, we intercept JavaScript and collect a trace of execution. Then, we start the process of extracting execution patterns from the trace. For building our pattern finding algorithm, we are inspired by genomics, where comparing a sequence to other sequences is an important goal. This is typically done by comparing a query sequence with sequences already stored in a database. We use *local sequence alignment* to align subsets of trace sequences. BLAST (Basic Local Alignment Search Tool) is the most common local algorithm [4].

The algorithm starts by finding exact matches of length k between the query sequence and the database sequences. These matches are then used as seeds to extend the alignment in both directions. At this point, we want to find similar patterns (not just the exact matches) to provide a degree of flexibility in our algorithm. Hence, we use the Smith-Waterman algorithm [25], which is a dynamic programming technique. This algorithm quantifies the alignment process by assigning scores for matches and mismatches, and penalties for gaps (scoring matrices). Aligned sequences are then found by searching for the highest scores in the scoring matrices.

We design a scoring matrix based on our specific domain. For instance, two functions in two sequences match if their names match. The match is strong if the arguments match as well. If the functions are not matched, a gap penalty is included in the score, which will increase by the number of gaps. The base scores for matches and penalties are determined using empirical data.

At the final step, we visualize the extracted patterns. We will also take advantage of information visualization techniques to facilitate developers' understanding. Enabling visual pattern recognition and clustering methods in an interactive visual interface can greatly benefit the viewers. Interaction mechanisms such as querying, filtering, semantic zooming, bookmarking, and adding notes can be utilized to help developers locate and understand their required information easier, faster, and more accurately.

4. EVALUATION

To evaluate our proposed approach, we should investigate its usability for developers. We will conduct a controlled experiment to assess the effectiveness of our technique for developers in practice. We divide the participants into control and experimental groups. The experimental group use our approach, while the control group use the tool of their choice. The developers will accomplish a set of comprehension tasks, and their performance will be measured. The tasks are designed based on common software comprehension activities [23]. Similar to our previous studies [1, 2, 3], we define the performance of a developer by the combination of time and accuracy of completing the tasks. Our hypothesis is that using our approach will enhance developers' performance in understanding the overall behaviour, main usecases, and recurring patterns of a web application.

5. REFERENCES

[1] S. Alimadadi, A. Mesbah, and K. Pattabiraman. Hybrid DOM-sensitive change impact analysis for JavaScript. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 321–345. LIPIcs, 2015.

[2] S. Alimadadi, A. Mesbah, and K. Pattabiraman. Understanding asynchronous interactions in full-stack JavaScript. In *Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 1169–1180. ACM, 2016.

[3] S. Alimadadi, S. Sequeira, A. Mesbah, and K. Pattabiraman. Understanding JavaScript event-based interactions. In *Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 367–377. ACM, 2014.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403 – 410, 1990.

[5] D. Amalfitano, A. Fasolino, A. Polcaro, and P. Tramontana. The DynaRIA tool for the comprehension of Ajax web applications by dynamic analysis. *Innovations in Systems and Software Engineering*, 10(1):41–57, 2014.

[6] S. Andrica and G. Candea. WaRR: A tool for high-fidelity web application record and replay. In *Proceedings of the International Conference on Dependable Systems & Networks (DSN)*, pages 403–410. IEEE Computer Society, 2011.

[7] B. Burg, R. Bailey, A. J. Ko, and M. D. Ernst. Interactive record/replay for web application debugging. In *Proceedings of the Symposium on User Interface Software and Technology (UIST)*, pages 473–484. ACM, 2013.

[8] A. Feldthaus, M. Schäfer, M. Sridharan, J. Dolby, and F. Tip. Efficient construction of approximate call graphs for JavaScript IDE services. In *Proceedings of International Conference on Software Engineering (ICSE)*, pages 752–761. IEEE, 2013.

[9] J. Hibschman and H. Zhang. Unravel: Rapid web application reverse engineering via interaction recording, source tracing, and library detection. In *Proceedings of ACM User Interface Software and Technology Symposium (UIST)*, pages 270–279. ACM, 2015.

[10] S. H. Jensen, M. Madsen, and A. Møller. Modeling the HTML DOM and browser API in static analysis of JavaScript web applications. In *Proceedings of the Symposium on Foundations of Software Engineering (ESEC/FSE)*, pages 59–69. ACM, 2011.

[11] V. Kashyap, K. Dewey, E. A. Kuefner, J. Wagner, K. Gibbons, J. Sarracino, B. Wiedermann, and B. Hardekopf. Jsai: A static analysis platform for JavaScript. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 121–132. ACM, 2014.

[12] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987, Dec 2006.

[13] A. La. Language trends on GitHub. https://github.com/blog/2047-language-trends-on-github, 2015.

[14] J. Lo, E. Wohlstadter, and A. Mesbah. Imagen: Runtime migration of browser sessions for JavaScript web applications. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 815–825. ACM, 2013.

[15] M. Madsen, F. Tip, and O. Lhoták. Static analysis of event-driven node.js JavaScript applications. In *Proceedings of ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 505–519. ACM, 2015.

[16] J. Maras, M. Stula, and J. Carlson. Generating feature usage scenarios in client-side web applications. In *Proceeding of the International Conference on Web Engineering (ICWE)*, pages 186–200. Springer, 2013.

[17] J. Mickens, J. Elson, and J. Howell. Mugshot: Deterministic capture and replay for Javascript applications. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 159–174. USENIX Association, 2010.

[18] P. Montoto, A. Pan, J. Raposo, F. Bellas, and J. López. Automating navigation sequences in Ajax websites. In *Proceedings of the International Conference on Web Engineering (ICWE)*, pages 166–180. Springer, 2009.

[19] G. C. Murphy, D. Notkin, and K. Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In *Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '95, pages 18–28, New York, NY, USA, 1995. ACM.

[20] H. V. Nguyen, C. Kästner, and T. N. Nguyen. Building call graphs for embedded client-side code in dynamic web applications. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 518–529. ACM, 2014.

[21] H. V. Nguyen, H. A. Nguyen, A. T. Nguyen, and T. N. Nguyen. Mining interprocedural, data-oriented usage patterns in JavaScript web applications. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 791–802. ACM, 2014.

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

[23] M. J. Pacione, M. Roper, and M. Wood. A novel software visualisation model to support software comprehension. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pages 70–79. IEEE Computer Society, IEEE, 2004.

[24] M. P. Robillard, W. Coelho, and G. C. Murphy. How effective developers investigate source code: an exploratory study. *IEEE Transactions on Software Engineering*, 30(12):889–903, Dec 2004.

[25] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[26] Stack Overflow. Developer survey. http://stackoverflow.com/research/developer-survey-2015, 2015.

[27] S. Wei and B. G. Ryder. State-sensitive points-to analysis for the dynamic behavior of JavaScript objects. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*, pages 1–26. Springer, 2014.

[28] A. Zaidman, N. Matthijssen, M.-A. Storey, and A. van Deursen. Understanding Ajax applications by connecting client and server-side execution traces. *Empirical Software Engineering*, 18(2):181–218, 2013.