SmartJS: Dynamic and Self-Adaptable Runtime Middleware for Next-Generation IoT Systems

Julien Gascon-Samson
University of British Columbia
Vancouver, BC, Canada
julien.gascon-samson@ece.ubc.ca

Mohammad Rafiuzzaman
University of British Columbia
Vancouver, BC, Canada
rafiuzzaman@ece.ubc.ca

Karthik Pattabiraman
University of British Columbia
Vancouver, BC, Canada
karthikp@ece.ubc.ca

Abstract

The Internet of Things (IoT) has gained wide popularity both in the academic and industrial contexts. However, IoT-based systems exhibit many important challenges across many dimensions. In this work, we propose *SmartJS*, a rich Javascript-based middleware platform and runtime environment that abstracts the complexity of the various IoT platforms by providing a high-level framework for IoT system developers. SmartJS abstracts large-scale distributed system considerations, such as scheduling, monitoring and self-adaptation, and proposes a rich inter-device Javascript-based code migration framework. Finally, it provides debugging and monitoring techniques to analyze performance and observe system-wide security properties.

CCS Concepts • **Software and its engineering → Development frameworks and environments**; **Runtime environments**; *System description languages*;

Keywords IoT, Internet of Things, Javascript, Scheduling, Code Migration, Dependability, Security

ACM Reference Format:
Julien Gascon-Samson, Mohammad Rafiuzzaman, and Karthik Pattabiraman. 2017. SmartJS: Dynamic and Self-Adaptable Runtime Middleware for Next-Generation IoT Systems. In *Proceedings of 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3135932.3135939

1 Introduction

The Internet of Things (IoT) refers to having several devices across many domains that are inter-connected in order to provide and exchange data. IoT systems exhibit important challenges across many dimensions, such as very high device heterogeneity, and the usage of many diverse programming languages, APIs and protocols. These factors not only render integration difficult, they also require developers to handle complex distributed systems and software dependability and security considerations, which can open the door to bugs and security vulnerabilities [2].

In this paper, we propose *SmartJS*, a rich Javascript-based middleware and runtime environment aiming at solving many of such challenges. While JavaScript has gained wide popularity as a programming language for web applications, it has also been proposed as a viable language for IoT. This is because of its event-driven nature, and large installed base of libraries and developers who know the language. Consequently, there have been a number of Virtual Machines (VMs) developed for running JavaScript code on IoT devices [1, 4].

SmartJS notably proposes a unified set of APIs and a set of high-level programming and communication paradigms that can be be used to efficiently write the code for the IoT system in JavaScript. SmartJS also abstracts developers from large-scale distributed system challenges and internally takes care of the dynamic scheduling and monitoring of the execution of the various components. In order to increase the dynamism of the system, SmartJS also provides, as a novel contribution, a state-preserving migration engine that transparently migrates the execution of Javascript-based applications across heterogeneous IoT devices. Finally, SmartJS also provides several debugging measures such as live system-wide invariant and performance monitoring and analysis.

2 SmartJS Ecosystem and Applications

From a holistic point of view, a SmartJS environment comprises a highly-distributed *SmartJS Application* and dynamically manages its execution over a set of heterogeneous *devices*. At it's core, a SmartJS application contains source code expressed in a high-level language (Javascript in our case), in a modular fashion (i.e., in the form a set of components), which allows developers to abstract out the platform-specific considerations and which allows the runtime to dynamically decide on the best placement of components to devices.

Device and Component Declaration. Developers must specify a set of devices on which the components will be run. Also, as there will likely be more than one instance of some

SPLASH Companion'17, October 22–27, 2017, Vancouver, Canada

ACM ISBN 978-1-4503-5514-8/17/10…$15.00
https://doi.org/10.1145/3135932.3135939

of the components (i.e., a building can contain many temperature sensors and actuators, but perhaps only one regulator), then developers must also specify the different instances of each component to be executed. Note that SmartJS allows for these to dynamically change, as devices and components can dynamically be added and/or removed.

Physical and Logical Constraints. The *SmartJS Manager* schedules the placement of components on devices. For the *Manager* to make optimal choices, a set of constraints must be specified, as the set of devices which can hold a given component instance might be restricted. For instance, a sensor component instance might have to be bound to a specific device, due to the presence of the *physical* sensor. SmartJS proposes mechanisms to express such constraints, and distinguishes between two sets of constraints: (1) physical constraints, which model the system characteristics of the various SmartJS *devices* themselves (CPU, memory, bandwidth), and (2) logical constraints, which model the characteristics of the SmartJS application *components*.

Inter-Component Communications. In SmartJS, we require that all inter-component communications follow a topic-based publish/subscribe [3] (MQTT) model. The choice of this model was primarily motivated by the logical decoupling of content producers from content consumers that it provides, which allows for abstracting many networking-related considerations. Also, due to its lightweightness and simple yet flexible conceptual model, the use of topic-based publish/subscribe enjoys widespread popularity in IoT [5].

3 Monitoring and Self-Adaptation

SmartJS collects statistics for every component running on each device, at a regular time interval (every second). They include the current CPU and RAM usage, the bandwidth usage (incoming and outgoing) and the latency (incoming and outgoing) relative to each other component.

Machine Learning Model for Resource Prediction. In order to accurately schedule the placement of a given component C, one must be able to evaluate the *outcome* of executing C on the various available IoT devices, which are subject to various workload patterns. In that end, we first build a prediction model by learning and observing the outcome of executing various SmartJS components exhibiting different load profiles (as per the metrics described above), on various devices subject to various load patterns. This enables us to accurately *predict* the outcome of executing C given all available IoT devices and their current load.

Component Scheduling. Taking the predictive model in conjunction with the physical and logical constraints outlined in section 2, the *SmartJS Manager* statically and dynamically schedules the execution of all component instances across all devices in order to ensure that all constraints are met. More precisely, the *Manager* produces an initial configuration, which is then dynamically refined as the conditions and constraints across all components and devices evolve.

Code Migration. Upon the *SmartJS Manager* generating a new configuration in which some of the component instances are moved to a different device, it becomes necessary to dynamically *move* the execution of such components from the old device to the new one.

As a novel contribution, we propose a novel Javascript-based migration framework (*SmartJS Migrator*), which can transparently and dynamically checkpoint and migrate the execution of a given event-based Javascript application from one Javascript virtual machine (i.e., Node.js) to another, across heterogeneous IoT devices and in the cloud. Our method is inspired by the prior work [6, 7], in which the authors proposed a set of techniques for migrating the execution of Javascript-based web applications across different browsers. As these authors outlined, migrating a Javascript-based application is not trivial, mainly due to the fact that the language exhibits special constructs (i.e., closures, event queues, etc.) that increase the algorithmic complexity of saving and restoring the state.

4 Conclusion

In this work, we proposed *SmartJS*, a rich Javascript-based middleware platform and runtime environment which aims at solving many of today's challenges in developing large-scale IoT applications by proposing a rich and abstract programming and communication paradigm. SmartJS also aims at shielding developers from distributed systems and security challenges considerations by means of a declarative approach at expressing system-wide constraints, a dynamic scheduling and heterogeneous migration engine and a performance and invariant monitoring and analysis engine.

References

[1] 2014. Intel XDK. (2014). https://software.intel.com/en-us/xdk

[2] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. 2011. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proceedings of the 20th USENIX Conference on Security*. USENIX, Berkeley, USA.

[3] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.* 35, 2 (2003), 114–131.

[4] Evgeny Gavrin, Sung-Jae Lee, Ruben Ayrapetyan, and Andrey Shitov. 2015. Ultra Lightweight JavaScript Engine for Internet of Things. In *SPLASH Companion 2015*. ACM, New York, NY, USA, 19–20.

[5] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. 2015. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing* 3, 1 (2015), 11–17.

[6] Jin-woo Kwon and Soo-Mook Moon. 2017. Web Application Migration with Closure Reconstruction. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. Republic and Canton of Geneva, Switzerland, 133–142.

[7] James Teng Kin Lo, Eric Wohlstadter, and Ali Mesbah. 2013. Imagen: Runtime Migration of Browser Sessions for Javascript Web Applications. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW '13)*. ACM, New York, NY, USA, 815–826.