PhD thesis proposal: JavaScript Ahead of Time compilation

INRIA Sophia-Méditerranée

supervisor: Manuel Serrano

2014-2017

1 Facts

location:	Inria Sophia Antipolis
funding:	36 months
date:	Oct 2014 - Oct 2017
supervisor:	Manuel Serrano
pdf:	these-hopjs.pdf

2 Introduction

Multitier Web programming is a paradigm for implementing Web applications [9]. It assumes a single programming language and a coherent runtime environment for all the components of the application. It makes reasoning about the programming easier as one single formalism is used for all the application. The benefits are a more compact and simple development model, more powerful analyses (for instance in security), more elaborated tools (debugging, profiling, etc.). However, multitier programming is not yet used by mainstream environments for the Web. For now, it still remains an academical research study. This PhD thesis will participate to the largest goal of popularized it by making it accessible to mainstream Web programming languages.

Multitier programming has been invented to simplify the development of Web applications. It is a programming paradigm where a single language and a single execution environment, although distributed, is used for programming the whole application. Multitier programming helps developing Web applications because it presents a global and coherent view of the whole application, which no longer consists of independent components merely connected to each others with URLs and HTTP communications. It has been pioneered by three programming languages (GWT[1], Hop [10], and Links [5]), which have all publicly appeared in 2006. Since then, other languages have been created (Ocsigen [12], Ur/Web [4], OPA [2], etc).

3 Context

Multitier programming exists in different flavors, from which two main trends can be distinguished. On the first one, many multitier languages rely on *tier annotations* that tell where a function, a variable, or an expression is to be evaluated. In this family, the core language can be a well known general purpose language. For instance, GWT relies on Java and Ocsigen relies on OCaml. It can also be a

[1] http:*www.gwtproject.org/*

new language specially invented for the Web, as Links. On the second trends, the separation between the tiers is implemented by meta-programming. This is the approach followed by Hop that considers client-side programs as values generated by server-side computations. This model differs from traditional programming as a significant part of the computation consists in computing the programs that will be executed elsewhere. This may seem exotic in the eyes of the programmer of classical application but this is actually as close as possible to the classical programming style of the Web. In traditional Web programming, the server side program generates strings of characters that represent an HTML document, which itself, contains strings of characters that represent client-side programs, namely, JavaScript programs. This is a simplified meta-programming model where generated programs consist of character strings. Hop, behaves similarly, a client-side program is represented by an explicit HTML document that embeds client-side expressions. The difference with classical Web programming is that Hop represents client-side programs as values of a concrete datatype and that the construction and the manipulation of these values is implemented by ad-hoc linguistic constructs.

Hop has been first released in 2006 as an open source software. Ever since, we have released new version every 6 months. We have studied its semantics [11, 3]. We have developed multitier tools [8]. Hop is a mostly academic programming language, based on the Scheme programming language˜“cite–scheme:r5rs″. It has an active but small programmer community. As we believe that multitier programming has invaluable virtues largely ignored by most programmers, we are re-designing and re-implementing Hop for a new core language in order to address a larger community.

For Hop, we have adapted the Scheme programming language to make it suitable for programming Web applications. We have added multithreading, object orientation, HTML as a primitive value, and we have created many Web-oriented libraries. We will apply similar modifications to JavaScript [6].

In the period 2005˜2010 JavaScript gained attention for Web server-side programming also. First, it started being used in popular NoSql projects (MongoDB, CouchDB, ...). Second, as the performance of its implementations significantly improved, it also started being used in the Web server, up to becoming the implementation language of a now popular Web server: Node.js. JavaScript is now commonly used on the browser side, on the server side, and JSON, the serialization format, that is a subset of the JavaScript literals, completes the whole chain of JavaScript based tools. JavaScript is then the current absolute dominant language of the Web. To popularize the principle of multitier Web programming we have decided to port the Hop principles to JavaScript, creating the Hop.js platform.

The new project will be decomposed in three tasks:

1. **Base Hop.js**: create a base implementation hosted by the Hop runtime environment.
2. **Multitier JavaScript**: extend JavaScript with multitier constructs.
3. **Tooling**: Create JavaScript development tools enabled by the multitier paradigm.
4. **Ahead-of-time compilation**: Study the ahead-of-time (AOT) compilation of JavaScript.

4 The subject

Base Hop.js is implemented as a new JavaScript engine in the Hop runtime environment. It is be composed of a native JavaScript compiler and a JavaScript server-side environment. It is compatible with Node.js, meaning, that *i)* it is fully compliant with the ECMAScript 262 standard, *ii)* it implements the same JavaScript extensions Node.js provides, and *iii)*/ it relies on the same module and packaging system. This strict compatibility will make it possible to re-use, as is, all JavaScript libraries and all the numerous Node.js contributions that already exist.

The Hop.js is not be based on an existing JavaScript implementation (such as V8[2] or SpiderMonkey[3]) for two main reasons. First we need the maximum flexibility to design, implement, and experiment with the JavaScript multitier extensions. Second, to implement multitier tools such as multitier debuggers, we need runtime constructs that are lacking in the JavaScript implementations, as they all consider the two ends of the Web application as two separated loosely connected processes.

JavaScript is a dynamic prototype-based programming language. Several characteristics makes it difficult to implement efficiently. Types are checked at runtime where most type conflicts are resolved by dynamically converting values from one type to another. Functions all accept a variable number of arguments. Properties can be added and removed dynamically from and to mostly all values. It lacks global variables, which are replaced by dynamic properties of a global object. It supports very few primitive types. For instance, it lacks integers. Array are extensible and potentially sparse. Etc.

Modern implementations have improved the performance of JavaScript so significantly that the language can now be used to implement applications that were considered out of reach a couple of years back. For instance, JavaScript is used to implement real-time graphical effects in Web video. It is used to implement reactive graphical games. It has even been used to implement a Web version of the QEmu emulator that is fast enough to boot a real Linux machine in the navigator.

State of the art JavaScript implementations (V8, SpiderMonkey) and academic researches [7, 1] all focus on dynamic optimizations and JIT compilation. This implementation technique is particularly well adapted to the context of client-side execution where a source code is shipped. It is questionable if it is bound to deliver best performances on the server side, where longer and ahead-of-time (AOT) compilation can be afforded. This is a subject largely unexplored both by industry and academia. Exploring the AOT compilation for JavaScript will be the subject of this thesis.

References

[1] W. Ahn, J. Choi, T. Shell, M. Garzaran, and J. Torellas. Improviing the JavaScript Performance by Deconstructing the Type System. In *Proceedings of the 35thACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, New York, NY, USA, 2014. ACM.

[2] H. Binsztok, A. Koprowski, and I. Swarczewskaja. *Opa: Up and Running*. O'Reilly Media, Feb. 2013.

[3] G. Boudol, Z. Luo, T. Rezk, and M. Serrano. Reasoning about Web Applications: An Operational Semantics for HOP. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 34(2), 2012.

[4] A. Chlipala. Static checking of dynamically-varying security policies in database-backed applications. In *OSDI'10: Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2010.

[5] E. Cooper, S. Lindley, P. Wadler, and J. Yallop. Links: Web Programming Without Tiers. In *5th International Symposium on Formal Methods for Components and Objects (FMCO)*, pages 266–296, Amsterdam, The Netherlands, Nov. 2006.

[6] ECMA. Ecma-262: Ecmascript language specification, 2009.

[7] B. Hackett and S.-y. Guo. Fast and precise hybrid type inference for javascript. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 239–250, New York, NY, USA, 2012. ACM.

[8] M. Serrano. A Multitier Debugger for Web Applications. In *Proceedings of the 10th WEBIST conference (WEBIST'14)*, Barcelona, Spain, Apr. 2014.

[2] `https://developers.google.com/v8/design`

[3] `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey`

[9] M. Serrano and G. Berry. Multitier Programming in Hop - a first step toward programming 21st-century applications. *Communications of the ACM*, 55(8):53–59, Aug. 2012.

[10] M. Serrano, E. Gallesio, and F. Loitsch. HOP, a language for programming the Web 2.0. In *Proceedings of the First Dynamic Languages Symposium (DLS)*, Portland, Oregon, USA, Oct. 2006.

[11] M. Serrano and C. Queinnec. A multi-tier semantics for Hop. *Higher Order and Symbolic Computation (HOSC)*, 23(4):409–431, 2012.

[12] J. Vouillon and V. Balat. From bytecode to Javascript: the Js_of_ocaml compiler. *Software: Practice and Experience*, doi: 10.1002/spe.2187, Feb. 2013.