Title Text

Subtitle Text, if any

Julien Richard-Foy Olivier Barais Jean-Marc Jézéquel
IRISA, Université de Rennes 1
{firstname}.{lastname}@irisa.fr

Abstract

This is the text of the abstract.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms term1, term2

Keywords keyword1, keyword2

1. Introduction

Rich Internet Applications have interesting properties: they require no installation or deployment on clients and enables large scale collaborative experiences. However, writing large Web applications is known to be difficult [6, 7]. One challenge comes from the fact that the business logic is scattered into heterogeneous client-side and server-side environments [4, 8]. For instance, HTML fragments may be built from the server-side when a page is requested by a client, but they may also be built from the client-side to perform an incremental update subsequent to a user action. How could developers write HTML fragment definitions once and render them on both client-side and server-side? The more interactive the application is, the more logic needs to be duplicated between the server-side and the client-side.

Using the same programming language on both server-side and client-side could reduce the gap between client-side and server-side environments, however the JavaScript language – which is currently the only action language natively supported by all Web browsers – has several drawbacks making it hardly suitable for large code bases (*e.g.*no static typing, no module system, verbose syntax, *etc.*). An increasing number of programming languages support generating JavaScript (*e.g.*GWT, SharpKit[1], Dart, Kotlin[2], ClojureScript [5], Fay[3], Haxe [2] or Opa[4]), increasing the panel of programming languages available to write both the client-side and the server-side code of a Web application.

[1] http://sharpkit.net

[2] http://kotlin.jetbrains.org/

[3] http://fay-lang.org/

[4] http://opalang.org/

[Copyright notice will appear here once 'preprint' option is removed.]

Having a full featured, cutting edge, programming language that runs on both client-side and server-side can help developers to write more maintainable code, however by abstracting over the differences of the client-side and the server-side environments, the code may suffer from performance issues. Performance is a primary concern in Web applications, because they are supposed to run on a broad range of devices, from the powerful desktop personal computer to the less powerful smartphone. "Every 100 ms delay costs 1% of sales", said Amazon in 2006.

For instance, the boundaries of the code parts to emit on client-side are less visible when you share code between client-side and server-side so transitive dependencies may pull a lot of code, causing a high download overhead. Moreover, generating efficient code for heterogeneous platforms is hard to achieve in an extensible way. (TODO Give a concrete example for each problem)

On one hand, for engineering reasons, developers want to write Web applications using a single programming language, abstracting over the target platforms differences. But on the other hand, for performance reasons, they want to keep control on the way their code is compiled to each target platform. How to solve this dilemma?

Lightweight Modular Staging [9] is a framework for defining deeply embedded DSLs in Scala. It has been used to define high-performance DSLs for parallel computing [1] and can be used to generate JavaScript code [3].

On top of this previous work, this paper presents several new APIs (?) that can be shared between client-side and server-side code and/or that are efficiently translated on each target platform. More precisely, our contributions are:

- Type-directed ad-hoc polymorphism on client-side without runtime dynamic dispatch logic;
- Use monads without extra container object creation;
- Ability to define DOM fragments using a common language for server-side and client-side, but that generates code using standard APIs on both server-side and client-side ;
- An API for searching in the DOM, that exposes a single entry point but that generates code potentially using more optimized native APIs

The remainder of this paper is organized as follows. The next section introduces existing approaches for defining cross-compiling languages. Section 3 presents our contribution. Section 5 evaluates our contribution. Section 6 concludes.

2. Related Works

2.1 Fat Languages

2.2 Thin Languages

2.3 Deeply Embedded Languages

3. High-Level Abstractions Generating Efficient (and Heterogeneous) Code

3.1 Ad-Hoc Polymorphism

3.2 Monads Sequencing

3.3 DOM Fragments

3.4 Selectors

4. Implementation

5. Evaluation

5.1 Real World Application

Chooze.

5.2 Several Implementations

5.2.1 Vanilla JavaScript

5.2.2 jQuery

5.2.3 GWT

5.2.4 SharpKit

5.2.5 Js-Scala

5.3 Benchmarks, Code Metrics

6. Conclusion, Future Work

Acknowledgments

Acknowledgments, if needed.

References

[1] K. J. Brown, A. K. Sujeeth, H. J. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun. A heterogeneous parallel framework for domain-specific languages. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 89–100. IEEE, 2011.

[2] N. Cannasse. Using haxe. *The Essential Guide to Open Source Flash Development*, pages 227–244, 2008.

[3] G. Kossakowski, N. Amin, T. Rompf, and M. Odersky. JavaScript as an Embedded DSL. In J. Noble, editor, *ECOOP 2012 – Object-Oriented Programming*, volume 7313 of *Lecture Notes in Computer Science*, pages 409–434, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi: 10.1007/978-3-642-31057-7_19. URL `https://github.com/js-scala/js-scala/`.

[4] J. Kuuskeri and T. Mikkonen. Partitioning web applications between the server and the client. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 647–652, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-166-8. doi: 10.1145/1529282.1529416. URL `http://doi.acm.org/10.1145/1529282.1529416`.

[5] M. McGranaghan. Clojurescript: Functional programming for javascript platforms. *Internet Computing, IEEE*, 15(6):97–102, 2011.

[6] T. Mikkonen and A. Taivalsaari. Web applications - spaghetti code for the 21st century. In *Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications*, pages 319–328, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3302-5. doi: 10.1109/SERA.2008.16. URL `http://dl.acm.org/citation.cfm?id=1443226.1444030`.

[7] J. C. Preciado, M. L. Trigueros, F. Sánchez-Figueroa, and S. Comai. Necessity of methodologies to model rich internet applications. In *WSE*, pages 7–13. IEEE Computer Society, 2005. ISBN 0-7695-2470-2.

[8] R. Rodríguez-Echeverría. Ria: more than a nice face. In *Proceedings of the Doctolral Consortium of the International Conference on Web Engineering*, volume 484. CEUR-WS.org, 2009.

[9] T. Rompf. *Lightweight Modular Staging and Embedded Compilers*. PhD thesis, IC, Lausanne, 2012. URL `http://library.epfl.ch/theses/?nr=5456`.