Title Text

You don't have to trade abstraction for control

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

Web applications are attractive because they require no installation or deployment on clients and enable large scale collaborative experiences. However, writing large Web applications is known to be difficult [9, 10]. One challenge comes from the fact that the business logic is scattered into heterogeneous client-side and server-side environments [7, 11]. This gives less flexibility in the engineering process and requires a higher maintainance effort: once you decided to implement a feature on client-side, changing your *mind* means completely rewriting the feature on server-side (and *vice versa*). Even worse, logic parts that need to run on both client-side and server-side are duplicated. 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 an 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 (explain why?).

Using the same programming language on both server-side and client-side can improve the software engineering process by enabling code reuse between both sides. Incidentally, the JavaScript language – which is currently the only action language natively supported by almost all Web clients – can be used on server-side, and an increasing number of programming languages or compiler backends can generate JavaScript code (*e.g.*Java/GWT [3], SharpKit[1], Dart [5], Kotlin[2], ClojureScript [8], Fay[3], Haxe [2] or Opa[4]).

However, this engineering comfort may come at the price of an inefficient runtime: abstracting over platform differences often means restricting to a subset of common features and losing opportunities to perform platform specific optimizations. Performance is a primary concern in Web applications, because they are expected 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, because the boundaries of the code sent to the client are less visible when you share code between client-side and server-side, transitive dependencies may pull a lot of code on the client, causing a high download overhead. Moreover, generating efficient code for heterogeneous platforms is hard to achieve in an extensible way: the translation of common abstractions like collections into their native counterpart (JavaScript arrays on client-side and standard library's collections on server-side) may be hardcoded in the compiler, but that would not scale to handle all the abstractions a complete application may use (*e.g.*HTML fragment definitions, form validation rules, or even some business data type that may be represented differently for performance reasons).

On one hand, for engineering reasons, developers want to write Web applications using a single 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?

Compiled domain specific embedded languages [4] allow the definition of domain specific languages (DSLs) as libraries on top of a host language, and to compile them to a target platform. The deep embedding gives the opportunity to control the code generation scheme for a given abstraction and target platform.

This paper presents such a DSL allowing developers to write Web applications in a single language which code fragments can be shared between client and server sides, and which is efficiently compiled to each side. More precisely, we demonstrate the following features:

- Type-directed ad-hoc polymorphism on client-side without runtime dynamic dispatch logic;
- Usage of 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;

[1] http://sharpkit.net

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

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

[4] http://opalang.org/

- 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 Work

2.1 Fat Languages

2.2 Thin Languages

2.3 Deeply Embedded Languages

Lightweight Modular Staging [12] 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 [6].

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] P. Chaganti. *Google Web Toolkit: GWT Java Ajax Programming*. Packt Pub Limited, 2007.

[4] C. Elliott, S. Finne, and O. De Moor. Compiling embedded languages. *Journal of Functional Programming*, 13(3):455–481, 2003.

[5] R. Griffith. The dart programming language for non-programmers-overview. 2011.

[6] 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/`.

[7] 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`.

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

[9] 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`.

[10] 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.

[11] 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.

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