Tierless Modules

Gabriel Radanne
IRIF UMR 8243 CNRS
Univ Paris Diderot, Sorbonne Paris Cité
gabriel.radanne@irif.fr

Jérôme Vouillon
IRIF UMR 8243 CNRS
Univ Paris Diderot, Sorbonne Paris Cité
BeSport
jerome.vouillon@irif.fr

ABSTRACT

We present a module system for Eliom, a tierless web programming language that extends OCaml. Our module language integrates well with both the Eliom and OCaml languages and supports nice properties of module systems such as a strong support for abstraction and separate compilation.

KEYWORDS

Web, client/server, OCaml, ML, Eliom, functional, module

1 INTRODUCTION

Traditional Web applications are composed of several distinct tiers: Web pages are written in HTML and styled in CSS; these pages are produced by a server which can be written in just about any language: PHP, Ruby, C++ ...; their dynamic behavior is controlled through client-side languages such as JavaScript. The traditional way to compose these languages is to write separate programs for the client and the server. Then, the programmer is expected to respect a common interface between the two programs. This constraint is usually not checked automatically, and it is the responsibility of the programmer to ensure that the two programs behave in a coherent manner. Of course, such checking is often error-prone. This issue, present in the Web since its inception, has become even more relevant in modern Web applications. Furthermore, the unit of composition here is a whole file (or compilation unit): files contain either client code or server code but cannot be composed of both client and server code. Such composition is very coarse-grained and hinders the modularity of Web programming libraries.

One goal of a modern client-server Web application framework should be to make it possible to build dynamic Web pages in a *composable* way. One should be able to define on the server a function that creates a fragment of a page together with its associated client-side behavior; this behavior might depend on the function parameters. The so-called *tierless* languages aim to solve such modularity issues by allowing to compose tiers inside expressions, by allowing to freely intersperse the client and server parts of the application in one language with seamless communication. For most of these languages, the program is sliced in two: a part which runs on the server and a part which is compiled to JavaScript and runs on the client. This allows to write libraries and widgets with both client and server behaviors. It also provides static guarantees about client-server separation and a fine-grained notion of composition.

However, programming large-scale software and libraries still requires a form of larger-scale composition. Indeed, parts of a library could be entirely on the server or on the client. Most tierless languages do not support such modular approach to program architecture and do not handle separate compilation. To solve this problem, we propose to leverage a well-known tool: ML-style modules. In this article, we show how to extend the ML module system with tierless annotations. By doing so, we gain a convenient paradigm for organizing large-scale software and support for separate compilation on top of the gains provided by tierless programming languages. Our module system is built as a complement to Eliom, a tierless web programming language built on top of OCaml, and retains its good properties such as static typing of client-server communications and an efficient execution model.

1.1 Modules

In modern ML languages, the module language is separate from the expression language. While the language of expression allows to program "in the small", the module language allows to program "in the large". In most languages, modules are compilation units: a simple collection of type and value declarations in a file. The SML module language (MacQueen 1984) uses this notion of collection of declarations (called *structure*) and extends it with types (module specifications, or *signatures*), functions (parametrized modules, or *functors*) and function application, forming a small typed functional language. Such a module system can account for separate compilation (Leroy 1994) and provides support for datatype abstraction (Crary 2017; Leroy 1995), which allows to hide the implementation of a given type in order to enforce some invariants. In the history of ML-programming languages, ML-style modules have been informally shown to be very

This work was partially performed at IRILL, center for Free Software Research and Innovation in Paris, France, http://www.irill.org.
Conference'17, July 2017, Washington, DC, USA
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

expressive tools to architect software. Functors, in particular, allow to write generic implementations by abstracting over a complete module. The OCaml language provides such a module system, extended with various other constructs such as package types (Russo 2000) (also known as first-class modules). One distinctive feature is that modules in OCaml are runtime entities. In contrast to systems such as MLton (2014), they are not eliminated at compile time.

1.2 Eliom

Eliom (Radanne et al. 2016a,b) is an extension of OCaml for tierless programming that supports composable and typesafe client-server interactions. It provides fine-grained modularity by allowing to manipulate *on the server*, as first class values, fragments of code which will be executed *on the client*. Eliom is part of the larger Ocsigen (Balat et al. 2009; Eliom 2017) project, which also includes the compiler js_of_ocaml (Vouillon and Balat 2014), a Web server, and various related libraries to build client-server applications. Besides the language presented here, Eliom comes with a complete set of modules, for server and/or client side Web programming, such as RPCs; a functional reactive library for Web programming; a GUI toolkit; a powerful session mechanism and an advanced *service identification mechanism* (Balat 2014). The Ocsigen project started in 2004, as a research project, with the goal of building a complete industrial-strength framework.

1.3 A module language for tierless programming languages

All of the modules and libraries in Ocsigen, and in particular in the Eliom framework, are implemented on top of a core language described in Radanne et al. (2016b). The design of this core language is guided by four complementary goals: easy composition of client and server code, type-safe communication between client and server, explicit communications that are easy to reason about and efficient execution model. We introduce additional properties that will drive the design of our module language:

Integration with the host language. Eliom is an extension of OCaml. We should be able to leverage both the language and the ecosystem of OCaml. OCaml libraries can be useful on the server, on the client or on both. As such, any OCaml file, even when compiled with the regular OCaml compiler, is a valid Eliom module. Furthermore, we can specify if we want to use a given library on the client, on the server, or everywhere.

Abstraction. Module languages are powerful abstraction tools. By only exposing part of a module, the programmer can safely hide implementation details and enforce specific properties. Eliom leverages module abstraction to provide encapsulation and separation of concern for widgets and libraries. By combining module abstraction and tierless features, library authors can provide good APIs that do not expose the details of client-server communication to the users.

These properties lead us to define a module language, Eliom$_m$, that extends the tierless core language presented in Radanne et al. (2016b). During the workshop, we will present both the expression and the module language used in Eliom with examples of client-server tierless programs. We will also present the underlying theoretical aspects, with a focus on the novelty introduced by our module system.

A preprint of the complete paper is also available (Radanne and Vouillon 2017).

REFERENCES

Vincent Balat. 2014. Rethinking Traditional Web Interaction: Theory and Implementation. *International Journal on Advances in Internet Technology* (2014). http://www.iariajournals.org/internet_technology/

Vincent Balat, Jérôme Vouillon, and Boris Yakobowski. 2009. Experience report: Ocsigen, a Web programming framework. In *ICFP*, Graham Hutton and Andrew P. Tolmach (Eds.). ACM, 311–316.

Karl Crary. 2017. Modules, abstraction, and parametric polymorphism. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 100–113. http://dl.acm.org/citation.cfm?id=3009892

Eliom 2017. *Eliom web site.* https://ocsigen.org/eliom.

Xavier Leroy. 1994. Manifest Types, Modules, and Separate Compilation. In *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, Hans-Juergen Boehm, Bernard Lang, and Daniel M. Yellin (Eds.). ACM Press, 109–122. DOI: http://dx.doi.org/10.1145/174675.176926

Xavier Leroy. 1995. Applicative Functors and Fully Transparent Higher-Order Modules. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, Ron K. Cytron and Peter Lee (Eds.). ACM Press, 142–153. DOI: http://dx.doi.org/10.1145/199448.199476

David B. MacQueen. 1984. Modules for Standard ML. In *LISP and Functional Programming*. 198–207.

MLton 2014. MLton. (2014). http://mlton.org/Home

Gabriel Radanne, Vasilis Papavasileiou, Jérôme Vouillon, and Vincent Balat. 2016a. Eliom: tierless Web programming from the ground up. In *IFL 2016, Leuven, Belgium, August 31 - September 2, 2016*, Tom Schrijvers (Ed.). ACM, 8:1–8:12. DOI: http://dx.doi.org/10.1145/3064899.3064901

Gabriel Radanne and Jérôme Vouillon. 2017. Tierless Modules. (March 2017). https://hal.archives-ouvertes.fr/hal-01485362 Submitted to ICFP 2017.

Gabriel Radanne, Jérôme Vouillon, and Vincent Balat. 2016b. Eliom: A Core ML Language for Tierless Web Programming. In *APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings (Lecture Notes in Computer Science)*, Atsushi Igarashi (Ed.), Vol. 10017. 377–397. DOI: http://dx.doi.org/10.1007/978-3-319-47958-3_20

Claudio V. Russo. 2000. First-Class Structures for Standard ML. *Nord. J. Comput.* 7, 4 (2000), 348–374.

Jérôme Vouillon and Vincent Balat. 2014. From bytecode to JavaScript: the Js_of_ocaml compiler. *Software: Practice and Experience* 44, 8 (2014), 951–972. DOI: http://dx.doi.org/10.1002/spe.2187