PhD thesis proposal: Reactive Web Programming

INRIA Sophia-Méditerranée, Collège de France

supervisors: Manuel Serrano, Gérard Berry

2014-2017

1 Facts

location:	Inria Sophia Antipolis
funding:	36 months
date:	2014 - 2017
supervisor:	Manuel Serrano & Gérard Berry
pdf:	these-hiphop.pdf

2 Introduction

Asynchrony being consubstantial with the Web, asynchronous communication and control patterns are ubiquitous at multiple scales in Web programs. They are used to program Web clients as well as Web servers. Browser HTML GUIs raise asynchronous events in reaction to user interactions. The Web browser DOM (Document Object Model) creates Web page elements asynchronously. Browser-to-server (AJAX) and server-to-server (*program-as-a-service* (PAAS) and Web APIs) communications are physically asynchronous as they involve network traversals that take wall-clock time. Web-of-things and diffuse robotics applications that are based on device-to-machine communications make asynchronous event raising even more frequent.

The objective of this PhD thesis is to help programming this family of rich Web applications by creating a programming language dedicated to deal with asynchronous events and orchestrate the tasks they trigger or terminate. This language will rely on the former development of synchronous languages directed towards embedded systems (Esterel, Lustre/SCADE, etc.). Despite the apparent name antagonism, synchronous languages are very efficient for programming complex reactions to asynchronous events and synchronize them. They have already be used extensively for robotics and GUI applications (including full airplane GUIs).

3 Context and Subject

Here is the state of the art w.r.t. web event handling. A myriad of ad-hoc solutions keep emerging here and there. The popular IFTTT[1] platform relies on an overly simple language for programming recipes. It spawns actions in reaction to events: *I*f *T*his event occurs *T*hen do *T*hat action. The events are generated by Web applications such as Twitter, Facebook, or other any other Web site. Typical

[1] `http://ifttt.com`

actions are "send me a text message" or "create a status message on Facebook". Although rudimentary, this system is gaining attraction as it allows non expert users to program simple but useful tasks. More elaborate solutions require fancier programming languages, usually extensions of JavaScript that reigns here unchallenged. During the last years, many JavaScript libraries implementing asynchronous operators have been released publicly. Let us name a few: async[2], async.js[3], bacon.js[4], RxJS [6][5], slide-flow-control[6], step[7]. These libraries implement asynchronous streams and some sort of *futures* [4]. They support higher-order stream operators such as map, filter, and reduce. More elaborated systems such as Flapjax [7] and Elm [3] make the functional reactive programming (FRP) paradigm available within JavaScript. They promote asynchronous streams to first-class values of the language, *i.e.*, streams can be stored into regular data structures, passed as arguments to functions, returned as results, and even be composed into stream of streams. Flapjax and Elm define themselves as extensions of JavaScript, but, since they change the meaning of the control flow operators, a compilation-to-JavaScript phase takes place before execution.

Each of the aforementioned system or language addresses a specific aspect of Web asynchronous programming, but none aims at offering a universal solution to the global management and synchronization of asynchronous events and tasks. For instance, neither the asynchronous JavaScript libraries nor the FRP JavaScript extensions support preemption, which limits them to crude ad-hoc solutions for dealing with cancellations and errors. FRP design choices have another not necessarily desirable consequence: promoting asynchronous streams as first-class values demolishes the fences that separate the classical algorithmic core language from its asynchronous extensions. The genuine semantics of the core language is then altered. Flapjax and Elm are based on JavaScript versions that obey their own semantics rules and that differ from the official ECMAScript 5 language rules.

The synchronous programming model of languages like Esterel or Reactive-C rely on a different form of concurrency based on the perfect synchrony hypothesis: a synchronous reactive program repeatedly reacts in conceptual zero-delay to input events by generating output events or calling computation actions; synchronization and communication between synchronous parallel statements is also performed in conceptual zero-delay. Perfect synchrony makes concurrent programs deterministic and deadlock-free, the only non-determinism left being that of the application environment. These languages are nowadays used to build complex and critical systems such as airplane control or cockpit systems.

In the synchronous reactive programming model, two languages levels coexist: the core algorithmic language and the orchestration language. How these languages are exposed to the programmer varies from one particular synchronous language to another, but all share a strong design axiom: the core algorithmic language and the orchestration language are separated by an hermetic membrane that can only be traversed by well-identified operators that have clearly (*i.e.*, formally) established semantics. The synchronous model is very appealing for programming the asynchronous patterns of Web applications, because it makes synchronization trivially explicit and deterministic, which simplifies decision taking. Synchronous languages are much more expressive than asynchronous languages, since they support rich concurrency and event-handling operators: parallelism, synchronization, preemption, and exception and error handling. The synchronous model has been extensively studied, is mathematically well-founded, and has become an industrial standard for embedded systems. It does

[2] `https://github.com/caolan/async`
[3] `https://github.com/fjakobs/async.js`
[4] `http://baconjs.github.io`
[5] `https://github.com/Reactive-Extensions/RxJS`
[6] `https://github.com/isaacs/slide-flow-control`
[7] `https://github.com/creationix/step`

not impose to create a brand new programming language since it can be implemented as a DSL hosted by an existing algorithmic language as already done for C [2] and OCaml [5].

The objective of this PhD thesis is to construct a JavaScript synchronous DSL for orchestrating Web applications. Will the language be presented with its own syntax requiring a front-end compilation or will it be exposed as a pure JavaScript API remains an open choice. The former offers more flexibility in the design of the language and probably and ease of programming. The latter is much in line with modern practice of releasing JavaScript libraries such as Node.js, Bower, or Grunt packages. That would make it be much easier to deploy and more likely to attract a much larger audience, with a potentially important impact outside the academic sphere. The previous HipHop experiment that we have conducted for Hop environment will be of course an important source of inspiration [1].

References

[1] G. Berry and M. Serrano. Hop and Hiphop : Multitier Web Orchestration. In *Proceedings of the ICDCIT 2014 conference*, pages 1–13, Feb. 2014.

[2] F. Boussinot. Reactive C: An extension of C to program reactive systems. *Software: Practice and Experience*, 21(4):401–428, 1991.

[3] E. Czaplicki and S. Chong. Asynchronous functional reactive programming for GUIs. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 411–422, New York, NY, USA, June 2013. ACM Press.

[4] D. Friedman and D. Wise. The Impact of Applicative Programming on Multiprocessing. In *International Conference on Parallel Processing*, pages 263–272, 1976.

[5] L. Mandel and M. Pouzet. ReactiveML, a reactive extension to ML. In *Proceedings of 7th ACM SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP'05)*, Lisbon, Portugal, July 2005.

[6] E. Meijer. Your Mouse is a Database. *ACM Queue*, 10(3), Mar. 2012.

[7] L. A. Meyerovich, A. Guha, J. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi. Flapjax: A programming language for ajax applications. *SIGPLAN Not.*, 44(10):1–20, Oct. 2009.