# Towards The Reactive Web

Master Thesis

Dominic Bosch
Departement Mathematics and Computer Science
University of Basel

May 4, 2014

# Acknowledgment

This master thesis is dedicated to all my lovings

# Contents

# List of Figures

# List of Tables

# Listings

# Chapter 1

# Introduction

The fast evolving web has brought up a trend towards easy to master interfaces to services, the so called WebAPIs. They do not only provide access to mere services but whole applications that allow access over WebAPIs. These trending WebAPIs benefit from a RESTful architecture which predominantly uses HTTP and thus relies on the most basic and powerful operations and the basis of the Web itself, the HTTP protocol.

quick (handling/mastering) accessible services and even whole web applications through so called Web APIs. WebAPIs provide powerful tools to govern data and functionality in the web independently of any user interface from the service provider. The relatively Allowing access to these services via API is increasingly popular and allows to mash up these services

Practically all services flood the user with events The web should be event driven, that's why we need an engine that deals with events and makes the web reactive There's still the challenge of filtering What's important to whom Plus the user needs to have tools to combine and add programmability to the combination,( such as conditions, selection of provided arguments and so on)

## 1.1 Related Work

### 1.1.1 WebAPI Mashups

Mashups combine information and functionality of more than one resource in a single place. The mashing up of such resources allows new points of view on data, or even ways to interact with them. Simple functions are combined into more powerful ones which influence data and services in a way their founders eventually didn't even think of. They have been developped ever since services in the web started to exist and were accessible in a more or less convenient way. One of the earliest inventors of such a webservice mashup is Paul Rademacher. In the same year after Google Maps came up in 2005, he invented a site[7, 5] that displayed Craigslist houses on a Google Map. With no Google Maps API at that time, he needed time and skills to reverse engineer Google Map's functionalities.

A large number of such "static" mashups were and are still developped. They are static in the way that they aggregate a fixed (and mostly low) number, of either data or functionality resources, to provide an enhanced resource in a specialized domain. Of course Mashups can be mashed up again, to provide even more sophisticated functionality and data. Some latest example Mashups, taken from the ProgrammableWeb[3] directory, are:

- Wifi and Plugs[1]: MapBox, Google Docs and Import.io API's used to display where Wi-Fi and plugs are available in London.
- MapLight[2]: GovTrack.us and OpenSecrets API's used to combine political results with financial contributions to show how capital contributions affect voting.

- Shared Count[4]: Facebook, LinkedIn, Pinterest and Twitter API's used to display informations about how well spread a URL is on social media sites.

In the past few years, research and development for platforms to allow users to flexibly mashup WebAPIs got attention. With IFTTT and Zapier, two platforms have evolved out of this process. Users that register on those platforms are provided with a multitude of WebAPI functions that act as event triggers and such that are used to execute actions. The user is then free to combine these event triggers and actions in the way it suits best, creating helpful WebAPI mashups on their own.

# Chapter 2

# Conceptual Model for Reactive Web Systems

## 2.1 Event Condition Action (ECA) Model in the Web

Existing ECA systems (List examples) all act on local data. Looking at (Wikipedia...) their definition is actions on local data. This does only add reactivity to these systems and not to the Web per se.

Such systems are merely event sinks which add fairly any value to the Web, except for the individual users and the system itself. We are taking a step further and allow not only the chaining up of several remote ECA engines, but also the invocation of actions on any arbitrary Web accessible service.

wow

# Chapter 3

# The "XYZ" Prototype System

## 3.1 Event Triggers

Event Gathering is the E in ECA and without one of these letters such a system would not run. It is of utmost importance to find as much as possible ways to get data into a system.

### 3.1.1 Polling for Events

### 3.1.2 Webhooks

Which

## 3.2 Actions

## 3.3 ECA Rules

## 3.4 Architecture

## 3.5 Asynchronous Systems & Closures

Often optimization approaches and programming language concepts require special attention to avoid common pitfalls. When closures are used in asynchronous systems, developers need to be very careful not to end up with randomly inconsistent states.

Looking at an example of sequential code execution in figure 3.1, we see that function execution of `fA` is halted until function `fB` is finished. If `fB` happens to be a latency-driven I/O operation the completion of `fA` could be deferred for a relatively long time. While the application waits for the completion of the I/O operation, some remaining operations in `fA` could eventually already be executed without causing any race conditions.

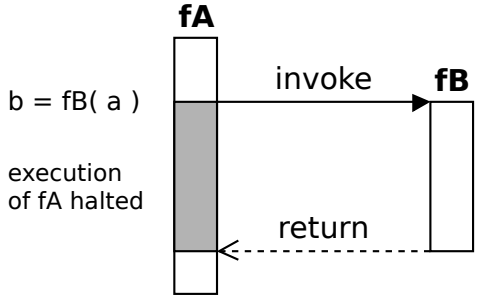


Figure 3.1: Synchronous Function Call

Asynchronous code execution, as shown in figure 3.2, allows non-blocking and thus scalable applications. Non-blocking operations are a remedy for optimzed resource allocation and open ways to overcome previously described unnecessary resource bindings. Processing any kind of latency-driven I/O operation asynchronously ( e.g. filesystem access and socket communication ) exploits resources that would otherwise be bound while waiting for completion. Such operations are processed and completed whenever required resources are available.

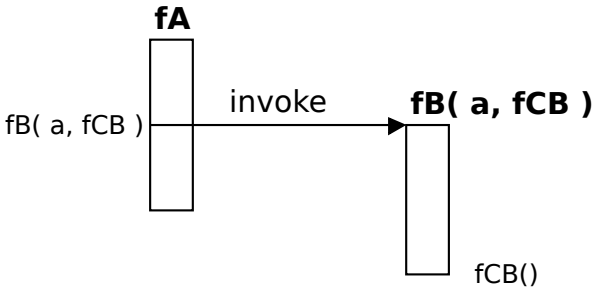


Figure 3.2: Asynchronous Function Call

Often other operations depend on the completion of asynchronous operations, hence their execution needs to be deferred. This necessary code execution deferral is achieved through the use of callback functions, denoted `fCB` in figure 3.2. Any code placed in a callback function, which is assigned to an asynchronous operation, is only executed after the respective asynchronous operation completed. This allows stacking of functions and operations upon each other which results in a flexible, event driven application.

If we look at the example in figure 3.2 and assume closures as defined in [6], .

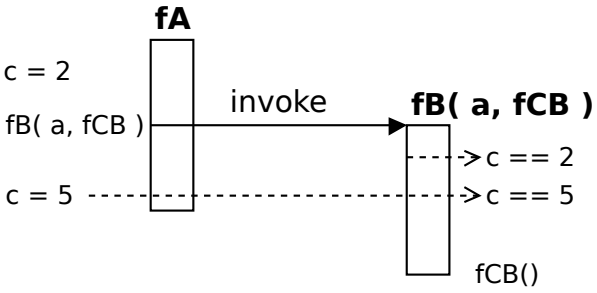


Figure 3.3: Closure Scope

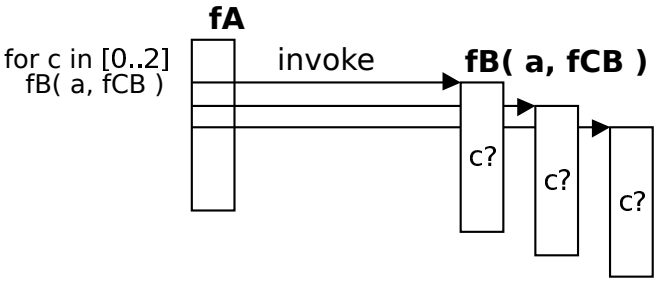


Figure 3.4: Closure Scope

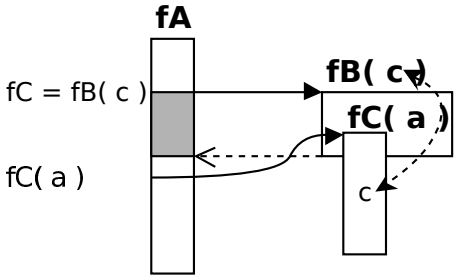


Figure 3.5: Closure Scope

# Chapter 4

# Discussion & Results

### 4.0.1 Use Cases

## 4.1 Future Work

We have seen that the ECA approach is already a powerful one to make the web reactive. A future improvement of this could be to adopt Complex Event Processing (CEP). This would mean that several events could be stored in a rule and be evaluated in terms of time constraints. Through this more complex events can be created as a result of several atomic events which would lead into semantically more complex events. A change in paradigm will result in an approach where events are not just processed when they are entering the system and evaluated against rules, but these events would need to be stored for quite a long time. Also the rules will not all be checked for each event but they are subject to a scheduler. It can be decided when and how often a rule is evaluated and all events will be checked at these point in times, whether they are candidates for firing the rule. A relational database will be needed in order to search through the timestamps

```
Name.prototype = {
  methodName: function(params){
    var doubleQuoteString = "some text";
    var singleQuoteString = 'some more text';
    // this is a comment
    if(this.confirmed != null && typeof(this.confirmed) == Boolean && this.
        confirmed == true){
      document.createElement('h3');
      $('#system').append("This looks great");
      return false;
    } else {
      throw new Error;
    }
  }
}
```

Listing 4.1: My Javascript Example

# Bibliography

[1] Free Wifi & Plugs — Your comprehensive list of where to work around London. http://wifiandplugs.co.uk. Accessed: 2014-05-02.

[2] MapLight - Money and Politics — U.S. Congress Campaign Contributions and Voting Database. http://maplight.org. Accessed: 2014-05-02.

[3] ProgrammableWeb: APIs, mashups and code. Because the world's your programmable oyster. http://www.programmableweb.com. Accessed: 2014-05-02.

[4] Shared Count. http://lab.neerajkumar.name/sharedcount. Accessed: 2014-05-02.

[5] Adam DuVander. 5 Years Ago Today the Web Mashup Was Born. http://blog.programmableweb.com/2010/04/08/the-fifth-anniversary-of-map-mashups-on-the-web, 2010. Accessed: 2014-05-01.

[6] ECMA International. *Standard ECMA-262 - ECMAScript Language Specification*. 5.1 edition, June 2011.

[7] Joshua Porter. Holy Amazing Interface, Batman! Paul Rademachers Brilliant Lodging Finder. http://bokardo.com/archives/holy-amazing-interface-batman, 2005. Accessed: 2014-05-01.