# Reactive User Interfaces in the Web

M.Sc. Thesis
University of Turku
Department of Information Technology
Computer Science
2015
Johannes Dahlström

Supervisors:
First Supervisor
Second Supervisor

TURUN YLIOPISTO
Informaatioteknologian laitos

JOHANNES DAHLSTRÖM: Reactive User Interfaces in the Web

Pro gradu, 14 s., 0 liites.
Tietojenkäsittelytiede
Kuukausi 2015

---

Tarkempia ohjeita tiivistelmäsivun laadintaan löytyy opiskelijan yleisoppaasta, josta alla lyhyt katkelma.

Asiasanat: tähän, lista, avainsanoista

UNIVERSITY OF TURKU
Department of Information Technology

JOHANNES DAHLSTRÖM: Reactive User Interfaces in the Web

M.Sc. Thesis, 14 p., 0 app. p.
Computer Science
Month 2015

---

If your thesis is in english this might also be required???

Keywords: list, of, keywords

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Interactive software systems have become progressively more complex, driven by the demand for richer user interaction along with the growing multimedia capabilities of modern hardware. Applications are increasingly being written for the World Wide Web platform, and the inherently distributed nature of web applications brings forth its own challenges. Not only are they required to serve an ever-increasing number of users, but also, more and more commonly, to facilitate interaction *between* users.

Complex interaction requires complex user interfaces. In object-oriented programming, the usual means of implementing a user interface is to make use of the Observer design pattern. In this pattern, interested objects can register themselves as *observers*, or listeners, of events, such as mouse clicks or key presses, sent by user interface elements. When an observer is notified of an event, it reacts in an appropriate manner by initiating a computation or otherwise changing the application state. Similarly, the user interface may react to events triggered by some underlying computation, signaling the user that something requires his or her attention.

The traditional observer pattern has several disadvantages. Observers are difficult to compose and reuse, leading to instances of partial or complete code duplication. Events often lack important context, making it necessary to manually keep track of the program state and share the bookkeeping between different observers. The resulting code typically forms a complex and fragile state machine, making it difficult to reason about its behavior and correctness.

In the recent years, several mainstream object-oriented programming languages used in the industry have adopted concepts traditionally belonging to the relatively academic realm of functional programming. These include functions as first-class values; anonymous functions (lambdas); and combinators such as map, filter, and reduce. One impetus

for this paradigm shift has been the constantly increasing importance of concurrency and parallelism in software. Correctly managing and reasoning about mutable state shared between concurrent threads of execution is notoriously difficult, and the general disposition towards immutable state in functional programming has proven to be a useful basis for building better concurrency abstractions.

Reactive programming is a programming style centered on the concept of propagating change. In a reactive system, a variable can be bound to other variables so that its value changes automatically as a response to value changes in other components of the bound system. A common example of such reactivity is a spreadsheet application, where the value of a cell can be a formula referring to several other cells. The displayed value of the cell is refreshed whenever the value of a referenced cell changes. Another name for this approach is dataflow programming.

One way of improving upon the observer pattern is to elevate the abstract concept of an event stream to a first-class data type. Event streams, or *observables*, can then be merged, transformed, and otherwise manipulated in a declarative manner using the familiar set of functional tools.

Vaadin is a web application framework written in Java, aiming to provide a rich set of user interface components facilitating rapid application development. It also contains a data binding layer for propagating input and output between the user and a data model. Both the user interface and data binding are designed in terms of the traditional observer pattern.

This thesis seeks to answer the question of whether reactive programming techniques are useful in writing user interfaces in Vaadin. Furthermore, it seeks to analyze whether some of these techniques should be adopted by Vaadin itself instead of simply being built on top of the framework.

Chapter 2 provides an overview of the history of the Web as an application platform and the current state of the art of Web applications. Chapter 3 introduces the reactive programming paradigm and discusses its usage in the context of Web applications. In Chapter 4, a reactive framework, built on top of Vaadin, is presented. The viability and potential for further development is analyzed in Chapter 5, and Chapter 6 provides conclusions.

# Chapter 2

# Programming Web Applications

## 2.1 A Brief Introduction to the Web

The World Wide Web, or the Web for short, is a massive distributed information system in the Internet. It constitutes a large set of interlinked *hypertext documents*, accessed using a browser application.

The Web project was initiated by the British computer scientist Timothy Berners-Lee while working at the European Organization for Nuclear Research, or CERN, in the late 1980s and early 1990s. His original aim was to improve information sharing among scientists in the nuclear research community, but the Web soon expanded to more general use. [1] [2]

Hypertext, one of the central concepts of the Web, refers to electronic documents linked to other documents via embedded references called hyperlinks. An early vision of the concept was presented in the seminal 1945 essay *As We May Think* by Vannevar Bush [3]. In the 1960s, the term "hypertext" was coined by Ted Nelson, and a working hypertext system was showcased by Douglas Engelbart in the famous "Mother of All Demos".

Hypertext documents in the Web are written in HTML (Hypertext Markup Language), which is based on the older SGML (Standardized General Markup Language). A HTML document is a structured text file consisting of a hierarchy of nested elements, representing the logical and visual structure of the document. A Web browser interprets the HTML and renders a graphical representation of it to the user, managing layouting, typesetting, multimedia, and possible interactivity as specified in the document. [4]

The Web is based on a client-server architecture utilizing the Hypertext Transfer Pro-

tocol (HTTP). A client application, typically a Web browser, connects to a HTTP server, requesting a *resource* such as a Web page, an image, or other type of file. Resources are identified via unique textual identifiers, URIs. [5]

A single server can attend to several clients—constrained by the available resources—but the clients cannot communicate directly with each other. The server must work as a mediator in any client-to-client interaction. Another limitation in the HTTP model is that the server cannot proactively send messages to the clients; it may only respond to requests initiated by them. Furthermore, the protocol is stateless; two consecutive requests by the same user are independent and not associated with the same session without separate bookkeeping.

A Web resource need not be a physical file served from mass storage; the server may instead elect to generate the response partially or fully programmatically. Thus, when reloading a page, its content may change dynamically without manual maintenance. For instance, the server might do a database query and present the results to the client as a HTML document. In practice, it is useful to modularize a Web server so that it can delegate request handling to a set of subprograms on a request-by-request basis.

A simple protocol, *Common Gateway Interface* (CGI), was developed to facilitate such delegation from a Web server to an auxiliary program. Early CGI applications were typically written in C or Perl. For each request, the server would execute the associated CGI application as a separate process, passing it details of the request in environment variables. The application would write a HTTP response to its standard output stream and the server would send the response to the client. [6]

In the early days of the Web, the only form of interaction between the user and the Web server was requesting pages either by typing an explicit URI or following hyperlinks. Use

Listing 2.1: A small HTML document

```
<!-- TODO This is just a test example -->
<!DOCTYPE html5>
<html>
  <head>
    <title>Test</title>
  </head>
  <body>
    test
  </body>
</html>
```

cases soon emerged for the ability for the user to send input data to the server; the latter in turn serving another page based on the user input and possibly save the input to persistent storage for future use. In 1993, Mosaic, one of the first graphical browsers, added to its HTML dialect a rudimentary set of input elements, including text fields, buttons, and list boxes. These elements were later included in the HTML 2 standard [4].

Compared to regular desktop applications, this rudimentary interactivity was rather slow and awkward. To process any user input, the browser would have to send a HTTP request to the server, where it would be processed and a new Web page, generated based on the input, served to the client. Fetching up-to-date content from the server would require the user to manually ask the browser to refresh the page.

To achieve more fulfilling interaction, a client-side programming model was necessary. In 1995, Brendan Eich developed the first version of *JavaScript*, an interpreted language executed by the browser. The language could be used to dynamically manipulate the document, typically interactively as a response to input events such as mouse clicks and key presses. For security reasons, JavaScript code in a browser is executed in a ”sandbox”, a secure virtual machine, and the language initially offered practically no access to standard operating system services such as the file system or the network. Partially due to these limitations, client-side scripting was considered by many a novelty at best and a nuisance at worst.

To better utilize the distributed aspect of the Web in client-side programming, a method for making programmatic HTML requests, without necessitating the reloading the whole page—and losing all client-side state—was required. Such functionality was developed by Microsoft in the late 1990s and became known as Ajax (Asynchronous JavaScript and XML). It is arguable that Ajax was the technology that started the ongoing age of Web applications.

As the name implies, Ajax requests are *asynchronous*. Because JavaScript programs are single-threaded, they cannot simply wait for the response without pausing user inter-

Listing 2.2: A small JavaScript program

```
1 // TODO This is just a test example
2 var btn = document.getElementById("button");
3
4 btn.addEventListener("click", function () {
5    alert("Clicked a button!");
6 });
```

action. Instead, the browser initiates the request in the background and notifies the script of its eventual completion. As a security measure, Ajax connections can, by default, only be made to the server from which the JavaScript code itself was requested.

HTML 5 is a common name for various technologies that aim to improve the capabilities and richness of interaction of Web applications. Several of these new features are programming interfaces that expose operating system services to JavaScript in a controlled fashion, including persistent storage, networking, and accelerated 3D graphics. [7]

The Web has been transformed from a simple document retrieval system to a full-fledged distributed application platform. Despite their potentially awkward user interfaces, Web applications have several advantages. They do not require a separate installation step, they run on any platform with a modern Web browser, and they are intrinsically network-aware, permitting interaction with not just the server, but also other users connected to the same server. While historically user interactions in the Web may have had a latency of several seconds, with modern Web technologies even highly interactive applications such as fast-paced multiplayer computer games are feasible.

## 2.2 Web Application Architecture

In short, a Web application is a program accessed with a Web browser or a specialized HTTP client. HTML pages, either computer-generated or hand-written, are used to present a graphical interface to the user, and HTTP requests are utilized to transmit information between the client and the server as necessary. JavaScript is used to provide low-latency interactivity in the browser, and there has been a constant push to move more and more application logic to the client side. Consequently, there has been a rising demand to improve the performance capabilities of JavaScript programs, and browser vendors have responded to that demand.

### 2.2.1 Communication

HTTP, the protocol underlying the Web, is request-oriented and stateless. For the original use case of document retrieval this was more than sufficient, but issues arise when it is attempted to use as the communication protocol of a distributed application.

-Synchronous request/response

-Ajax async request/response

-Polling

-Push: Comet, Websocket

### 2.2.2 Single-page Web Applications

-Only one, initial, full page load

-Initial page fully created on server, or bootstrapped with JS

-UI events cause Ajax requests, server replies with page fragments or instructions controlling JS

-Push

### 2.2.3 Model-View-Controller

## 2.3 User Interface Programming

### 2.3.1 Widgets

-User interface elements

-Basic division: layout and input

-Layouts have children, forming a tree

-

### 2.3.2 Events and Observers

-Event loop

-Observer pattern

## 2.4 Servlets

## 2.5 Vaadin

Vaadin is a Web application framework written in Java that aims to make the design and maintenance of high-quality Web-based user interfaces easy. It attempts to abstract away many of the more inconvenient aspects of the Web platform, trying to provide an experience similar to that offered by traditional desktop user interface frameworks.

-Abstract away client whenever feasible

-Client access is there if needed

-Recognized that the abstraction is leaky

-Write server-side UI code, client side rendered automatically

-Events and updates transmitted automatically

### 2.5.1 Architecture

-GWT

-Clientside counterparts for server-side components maintained automatically

-Ajax or push

-Shared state and RPC

### 2.5.2 Component Model

-Basic observer pattern

### 2.5.3 Data Model

-Bind fields to data

-Data source can be memory, DB, Web service, ...

-Bidirectional propagation of change

-Converters

-Transactionality

-Based on observer pattern

# Chapter 3

# Reactive Programming

## 3.1 Reactive Primitives

### 3.1.1 Behaviors and Events

### 3.1.2 Futures and Promises

### 3.1.3 Observables

### 3.1.4 Async and Await

### 3.1.5 Actors

### 3.1.6 Composability

## 3.2 Reactive libraries

### 3.2.1 Rx

### 3.2.2 React.js

# Chapter 4

# Case Study: Reactive Vaadin

# Chapter 5

# Validation

# Chapter 6

# Conclusions

# References


[1] T. Berners-Lee. Information Management: A Proposal. CERN internal proposal, March 1989.

[2] T. Berners-Lee and R. Cailliau. WorldWideWeb: Proposal for a HyperText Project. CERN internal proposal, November 1990.

[3] V. Bush. As We May Think. *The Atlantic Monthly*, 176(1):101–108, July 1945.

[4] T. Berners-Lee and D. Connolly. Hypertext Markup Language – 2.0. RFC 1866, RFC Editor, November 1995.

[5] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.

[6] D. Robinson and K. Coar. The Common Gateway Interface (CGI) Version 1.1. RFC 3875, RFC Editor, October 2004.

[7] I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E.D. Navara, E. O'Connor, and S. Pfeiffer. HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Proposed Recommendation, W3C, September 2014.

[8] R. Cailliau. A Little History of the World Wide Web, 1995.

[9] A. Barth and U.C. Berkeley. HTTP State Management Mechanism. RFC 6265, RFC Editor, April 2011.

[10] Anne van Kesteren. XMLHttpRequest Level 2. W3C working draft, W3C, January 2012.

[11] ECMAScript® Language Specification. Standard ECMA-262 Edition 5.1, Ecma International, June 2011.

[12] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 3 Core Specification. W3C Recommendation, W3C, April 2004.

[13] M. Grönroos. *Book of Vaadin*. Vaadin Ltd, Vaadin 7 edition, April 2013.

[14] D. Coward and Y. Yoshida. Java™ Servlet 2.4 Specification. JSR 154, Sun Microsystems, Inc., November 2003.

[15] R. Mordani. Java™ Servlet Specification, version 3.0. JSR 315, Sun Microsystems, Inc., December 2009.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, October 1994.

[17] A. Leff and J.T. Rayfield. Web-application development using the Model/View/-Controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*, pages 118–127, 2001.

[18] J. Conallen. Modeling Web Application Architectures with UML. *Commun. ACM*, 42(10):63–70, October 1999.

[19] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[20] M. Jazayeri. Some Trends in Web Application Development. In *Future of Software Engineering, 2007. FOSE '07*, pages 199–213, May 2007.

[21] L. Shklar and R. Rosen. *Web Application Architecture: Principles, Protocols and Practices*. Wiley, 2nd edition, April 2009.

[22] Microsoft Patterns & Practices Team. *Microsoft® Application Architecture Guide (Patterns & Practices)*. Microsoft Press, 2nd edition, November 2009.

[23] M. Galli, R. Soares, and I. Oeschger. Inner-browsing extending the browser navigation paradigm, May 2003.