A WoT Approach to eHealth

RESTfull webservice in java

Bachelor Thesis

Mathieu Jean-Daniel

Mars 2013

Thesis supervisors:

Prof. Dr. Jacques Pasquier-Rocha
and
Andreas Ruppen
Software Engineering Group

Université de Fribourg Suisse
Universität Freiburg Schweiz

Software Engineering Group
Department of Informatics
University of Fribourg (Switzerland)

Table of Contents

List of Figures

List of Tables

List of Code extracts

1 Introduction

1.1 Motivation and Goals

Main purpose of the following work is to create a tool for caregivers in an eHealth environment, that facilitates work on a daily base and fully applies the web of things approach. All calculations and data modification must be reachable by any internet capable device, secured by basic authentication and use the RESTful standard. The server itself is written in Java and follows clearly defined designed patterns.

An important part of the work is focused on the usage of description languages such as XSD and WADL to describe the server before implementation. Such an approach allows to concentrate first on the software use cases and their description and helps implementation afterwards.

1.2 Organization

Introduction

Motivation, goals and overview of the following work.

Chapter 1: WoT and REST

This chapter tells the story of HTTP leading to REST and the WoT approach. While describing the basics required to understand the fundamental techniques behind HTTP and REST the crucial part of this chapter is focussing on the history of RESTful Web-Services and its influence on WoT.

1.3 Notations and Conventions

- Formatting conventions:
  - **Bold** and *italic* are used for first use of essential keywords;
  - `SansSerif` is used for web addresses;
  - `Code` is used in all Java code and generally for anything that would be typed literally when programming, including keywords, constants, method names, and variables, class names, and interface names;
- The work is divided into 5 chapters that contain section and subsections. Every section or subsection is organized into paragraphs, signaling logical breaks.
- Figures, Tables and Code extracts are numbered inside a chapter. For example, a reference to Figure j of Chapter i will be noted *Figure i.j*.
- As far as gender is concerned, I systematically select the masculin form due to simplicity. Both genders are meant equally.
- Source code is displayed as follows:

```java
public double division(int _x, int _y) {
  double _result;
  _result = _x / _y;
  return _result;
}
```

2
WoT & REST

2.1 The Story of HTTP

2.1.1 HTTP 0.7 and HTML

HTTP is a standard first introduced by Tim Berners Lee and his group as a part of a set of protocols for the world wide web project. The first version 0.7 could only transfer HTML documents, contained no authentication or security features and was essentially only used to transfer text. Technically anything bigger would have been an unreasonable amount of time wasted to load. Still HTML and HTTP remain the building blocks of internet pages. Until today a website is build around a HTML document, even if the content itself is loaded completely dynamically by script languages like JavaScript.

The evolution of HTTP is currently at version 1.1. This version adds the notion of session to a connection server-client and introduces an authentication method as well as a set of new headers. The RFC, request for comment community, is responsible to extend this standards and to allow individuals and companies to participate in the future of the internet. One of the leading authors for RFC 2616, the document that describes the HTTP 1.1 standard, is Dr. Roy Thomas Fielding, a doctor of philosophy of the university of California, Irvine. His role is going to be crucial for the following work.

2.1.2 HTTP into detail

HTTP is a request-response protocol that uses TCP/IP to transfer data. Every HTTP message consists of a header and a body. While information relevant for the connection and the data transfer is stored in the header, the payload itself is stored in the body.

HTTP Header

```
GET /eHealthServer/resources/caregivers/1 HTTP/1.1
Host: localhost:9090
Connection: keep-alive
Accept: application/xml
Authorization: Basic dGVzdDoxMjM0
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.22 (KHTML, like Gecko)
     Chrome/25.0.1364.97 Safari/537.22
Accept-Encoding: gzip,deflate,sdch
Accept-Language: de-DE,de;q=0.8,en-US;q=0.6,en;q=0.4
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Code extract 2.1: HTTP Request header

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 1611
Server: Jetty(8.1.2.v20120308)
```

Code extract 2.2: HTTP Response header

The first line always contains the HTTP version used. In a request the first word is always the method. Those, also called verbs, describe the type and effect of a message. A verb can be safe if its only intended for information retrieval and does not change the state of the server or it can be idempotent if repeatingly applying the same operation is not changing the result.

It is important to understand that those are guidelines and they are not necessary applied. Especially SOA Services often use GET with an XML Message to run a method on the server. Subsequent an overview of the HTTP methods[1]:

[1]Additional verbs are OPTIONS,TRACE,DEBUG,CONNECT,.. but are not needed for the work

HTTP Methods or Verbs

Verbs	Description	safe	idempotent
GET	Requests a representation of the specified resource.	✓	✓
POST	Requests that the server accept the entity enclosed in the body as a new subordinate of the resource identified by the URI.	✓	ϟ
PUT	Requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.	✓	✓
DELETE	Request that the resource under the supplied URI is deleted if present.	✓	✓

Table 2.1: Overview of the HTTP methods

HTTP Status Codes

Status codes indicate a successful, pending or failed request. They can also be used to inform about the status of the server or the connection. Every HTTP Response contains a status code and the according text. Those can be used to troubleshoot request problem. Following an overview of the most frequent status codes[2] and their usage:

100	**Information**		
102	Indicating a request is being processed.	**118**	Connection has timed out.
200	**Successfull operations**		
200	Operation was successful.	**201**	Resource was created.
202	Accepted but execution delayed.	**204**	Accepted but response has on purpose no content.
400	**Client Error**		
400	Request contains errors.	**401**	Authentication failed.
404	Resource cannot be found.	**405**	HTTP Method is not allowed on this resource.
406	The requested resource is not available in the desired form.		
500	**Server Error**		
500	A not further described internal server error.	**501**	The requested method at the given URI is not yet implemented or is missing.
505	HTTP Version is not supported.		

Table 2.2: Status Codes for HTTP

[2]The full list is defined in the RFC-2616 and is subject to sporadic change.

2.2 REST

2.2.1 The invention of REST

The term REST stands for Representational State Transfer and was first introduced and described by Roy T. Fielding who is also main author of the HTTP Standard RFC-2616. In short REST can be seen as a Architectural style that describes a specific set of web services based on HTTP. As the biggest implementation one can see the world wide web.

2.2.2 REST and RESTful

Fielding introduced a very general classification of services and remained very abstract about a concrete implementation. His constrains are not concrete implementations but classification of hypermedia systems. Whereas ROA is a concrete and applicable implementation by Richardson and Ruby[2]. RESTful is the term used to describe an implemented service that is applying the REST constrains. ROA is such a RESTful webservice.

2.2.3 REST by Fielding: Architectural style based on constrains

"REST is an abstraction of the architectural elements within a distributed hypermedia system." [1] Starting with a null style, constrains are added and with them properties are induced to the final system. There are five mandatory and one optional properties. Additionally the uniform interface property requires a specific design in resources one needs to follow while designing.

1. Client-Server Architectural Style

The first constrain requests the separation of the user interface from the data storage. For instance the entire world wide web is based in storages (Databases, Web Servers, ...) and clients (Web Browsers, Web Service Clients, ...).

- ✓ `Improved Portability`: allowing multiple different clients for the same service;
- ✓ `Improved Scalability`: every request is atomic and the operation terminates with the response;
- ✓ `Separate development`: client and server can develop independently.

2. Stateless: Restriction to server-client interaction

Any request is to be considered independently and contains everything needed for its execution. It is not allowed to make requests dependent on preliminary information exchange. As a consequence the application state is entirely stored on the client side.

- ✓ `Visibility`: No need to wait for additionally data to execute a request, all needed details are in the request;
- ✓ `Reliability`: Recovery from partial error by repeating single request;

✓ `Scalability`: Any application state on the server side start and stops with a single request, this helps manage computational resources.

ϟ `Increase in Network traffic`: since every request needs to contain all data needed for the reqeust;

ϟ `Reduction in Application control by server`: since state is entirely controlled by the client.

3. Cache

Every request must be explicitly or implicitly labelled as cacheable or non-cacheable.

✓ `Reduction in network traffic`: therefore increase in efficiency, scalability and user-perceived performance;

ϟ `Reliability`: is reduced if response to the same request differ largely.

4. Uniform Interface

The software engineering principle of generality is applied to the component interface, removing any unnatural restriction or limitation. Implementations are decoupled from the service they provide. Every response is structured similarly[3].

✓ `Simplified architecture and improvement of visibility`;

ϟ `Inefficient`: since informations are transferred standardized and a lot of overhead is generated;

ϟ `Additional constrains to the interface`.

5. Layered System

A REST Service is structured into layers for which each participant sees only to its direct interacting partner. This separates a client server layer from a server database layer and so on.

✓ `Load Balancing`: Distribute load across multiple instances of resource providers (multiple databases, ...);

✓ `Encapsulating and outsourcing`: High frequency services can be distributed to multiple layer instance where as low frequency services can be encapsulated together, still sharing the same interface;

ϟ `Added overhead and latency`: used to select and transfer request to matching service.

(6.) Code-On-Demand

REST allows to outsource functionality from client to server and reduce pre-implemented code on server side. This also reduces visibility and can be seen as an optional constrain.

[3]As a specific implementation for a REST Service the eHealthServer uses an XSD file to described the XML messages exchanged.

2.2.4 REST by Fielding: Definition of Resources and their representation

A resource is any information that can be named especially in a context that could be the target of a hyperlink reference. Therefore any system that serves informations over a network is considered a web service. The data to be transferred is encapsulated in a commonly readable form and contains meta data to read those informations. Especially in the case of XML the data itself becomes humanly readable and only by following the structure of the data and the information stored in the meta data one can consume the service and all of its functionality.

As already mentioned resource representation contains the data, the metadata and additional control data that gives details about the state of the requested resource. As for definitions the format of data is called the **media type**[4]and the resource identifier **URI**[5]. Summing up a resource served by a service has the following parts:

Component	Description	HTTP representation
Resource	Intended conceptual target of a hypertext reference.	Response Body
URI	Identifier of a resource.	URI (Host, Path)
Media Type	Encapsulation format for the resource.	Header (Accept, Content-Type)
Meta Data	Additional data about the structure of the resource.	Header

Table 2.3: Fielding: Resource properties

2.2.5 REST by Fielding: Conclusion

A REST WS is not a definition of a specific web service but a number of constrains that are applicable to a hypermedia system. Each adding properties and allowing for further scalability, improved visibility and reliability. The drawbacks can be summed up by the added overhead and implications to the design of the elements that are served. Especially the resource design becomes an essential part of any such service.

2.2.6 RESTful by Richardson and Ruby

[4]Standard media types are: text/html, text/plain, application/xml, application/json, etc ...
[5]Uniform Resource Identifier

2.3 From Web 1.0 to the web of things

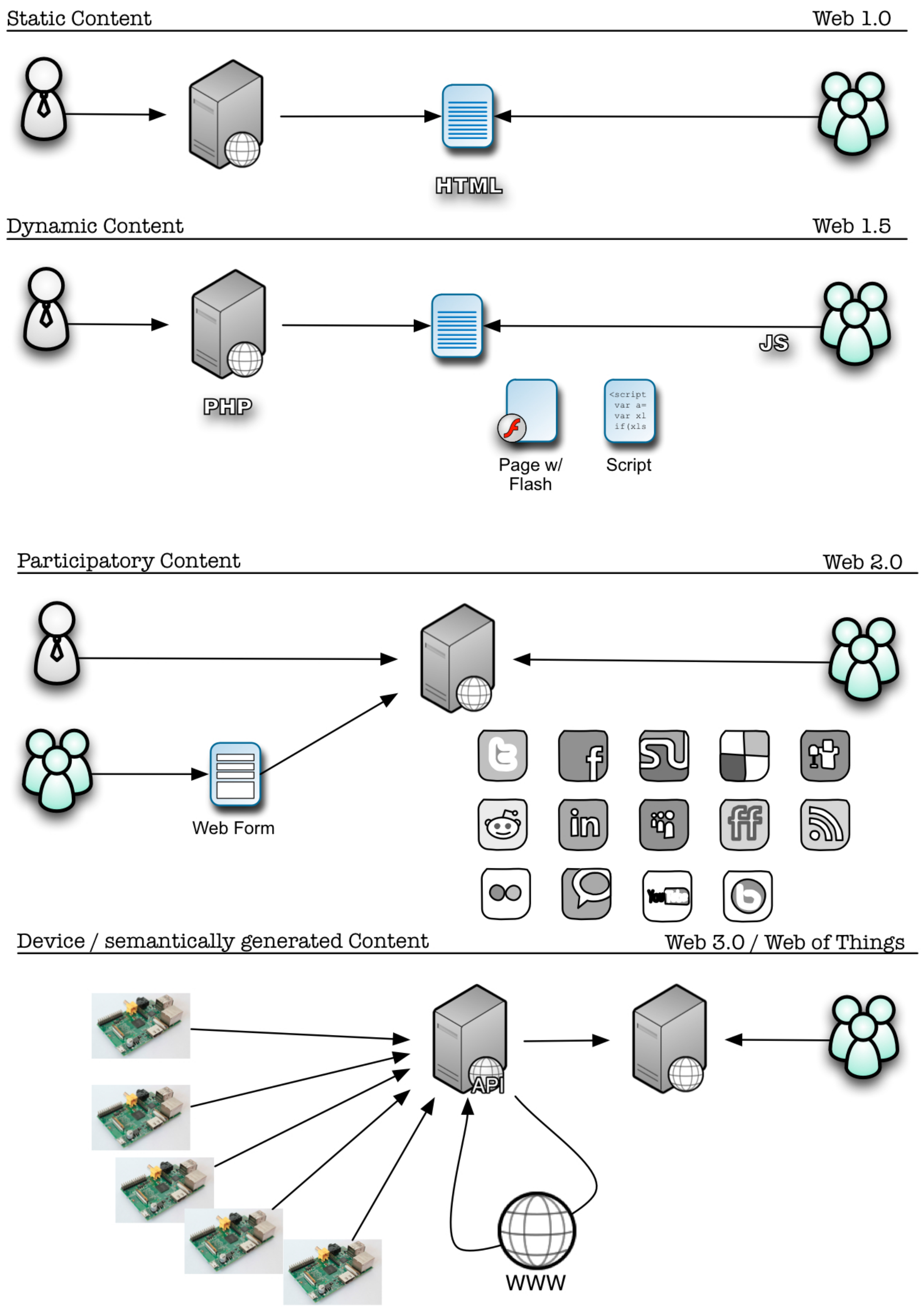

Figure 2.1: Evolution from web 1.0 to Web of things

References

[1] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. AAI9980887. 7

[2] Leonard Richardson and Sam Ruby. *Restful Web Services*. O'Reilly Media, 2007.