Documentation Guidelines

A Template Using The Software Engineering Group's Documentation Guidelines For Bachelor And Master Thesis

Master Thesis

Andreas Ruppen

July 2013

Thesis supervisors:

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

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

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

Software Engineering Group

Table of Contents

1. Introduction **2**

1.1. Motivation and Goals 2

1.2. Organization 2

1.3. Notations and Conventions 3

2. WoT: From HTTP to ROA and SQL **4**

2.1. HTTP: Hypertext Transfer Protocol 4

2.1.1. HTTP 0.7 and HTML 4

2.1.2. Elements of HTTP 4

2.2. REST: Representational State Transfer 6

2.2.1. Introduction 6

2.3. From World Wide Web to the Web of Things 7

2.3.1. World Wide Web 7

2.3.2. Internet of Things (IoT) 8

2.3.3. NoSQL 10

3. The eHealth Server: Design **12**

3.1. Overview 12

3.1.1. A RESTful Recipe 12

3.1.2. From an abstraction to a concrete implementation 13

3.2. Fourth: Expose a subset of the uniform interface 13

3.2.1. General overview of methods used 13

3.3. Fifth: Design representations 14

3.3.1. Using XSD and JAXB to obtain POJOs 14

3.3.2. Name-spaces: eHealth, JAXB and XSD 14

3.3.3. MySQL ERM 14

3.4. Web-Service Description 14

3.4.1. WSDL 14

3.4.2. WADL 15

3.5. Abstract of the design process 15

4. Conclusion **16**
4.1. Review 16
4.2. Outlook: Software Engineering methodology and client 16
4.3. Final statements 16

A. Common Acronyms **17**

B. License of the Documentation **19**

C. Website of the Project **20**

D. CD-Rom **22**

References **24**

Referenced Web Resources **24**

List of Figures

2.1. Connectedness of Resources within the eHealthServer 5
2.2. First website publicly available hosted by CERN 7

C.1. Screenshot of the project's official web-page 21

D.1. The CD-ROM of this project 23

List of Tables

2.1. Overview of HTTP methods 6
2.2. Elements of a resource by Fielding[2] 6

3.1. Overview of the HTTP methods used 14

Listings

2.1. HTTP Request header 4
2.2. HTTP Response header 5
2.3. Adapter2.java: unmarshal() and marshal() methods 8
2.5. application.wadl: A floating XML code example with file 8
2.4. A floating example 9
2.6. A floating XML code example 11

3.1. Declaration of a grammar 15
3.2. Declaration of a HTTP method in WADL 15

1 Introduction

1.1. Motivation and Goals **2**
1.2. Organization **2**
1.3. Notations and Conventions **3**

1.1. Motivation and Goals

Instead of simply defining another possible implementation, this work aims at identifying methods applicable to reduce the effort in building RESTful services for the Web of Things. Main purpose of the following work is therefore, to provide a methodology to create web-services for the Web of Things, using eHealth as an example. All Representational State Transfer (REST) constrains defined by Roy T. Fielding [2] must apply to the server and it must be based on the Resource Oriented Architecture (ROA) as defined by Richardson and Ruby [8]. The system should be secured using basic authentication and offer at least eXtensible Markup Language (XML) and Hypertext Markup Language (HTML) responses.

1.2. Organization

Introduction

The introduction contains the motivation and goals of this work, a short recapitulation of the structure of each chapter along with an overview of the formatting conventions.

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 technique HTTP and REST, the crucial part of this chapter is focussing on the history of RESTful Web-Services and its influence on WoT.

Chapter 2: The eHealth Server design

The subject is a design process introduced by Richardson and Ruby [8] and adapted for this particular use-case. Artefacts of this process are three elements: a XSD, a WADL

and a ERM file. Together describing the representation, manipulation and storage of resources.

Chapter 5: Conclusion

Describes the original idea and if that goal was reached. It describes the contribution of this work to the research field as well as personal challenges.

Appendix

Contains extracts of artefacts or service messages, abbreviations and references used throughout this work.

1.3. Notations and Conventions

- Formatting conventions:
  - Abbreviations and acronyms as follows Java Persistence API (JPA) for the first usage and JPA for any further usage;
  - `http://localhost:9090/eHealthServer` is used for web addresses;
  - Code is formatted as follows:

```
1 public double division(int _x, int _y) {
2     double result;
3     result = _x / _y;
4     return result;
5 }
```

- The work is divided into five chapters that are formatted in sections and subsections. Every section or subsection is organized into paragraphs, signalling logical breaks.
- Figure s, Table s and Listings s 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 masculine form due to simplicity. Both genders are meant equally.

2

WoT: From HTTP to ROA and SQL

2.1. HTTP: Hypertext Transfer Protocol 4
2.1.1. HTTP 0.7 and HTML 4
2.1.2. Elements of HTTP 4
2.2. REST: Representational State Transfer 6
2.2.1. Introduction 6
2.3. From World Wide Web to the Web of Things 7
2.3.1. World Wide Web 7
2.3.2. Internet of Things (IoT) 8
2.3.3. NoSQL 10

2.1. HTTP: Hypertext Transfer Protocol

2.1.1. HTTP 0.7 and HTML

2.1.2. Elements of HTTP

Hypertext Transfer Protocol (HTTP) is an application-layer protocol that uses the Transmission Control Protocol (TCP) to assure integrity of transferred data and the Internet Protocol (IP) for address resolution. Every HTTP message consists of a header, containing connection relevant details and a body, containing the payload.

HTTP Header

```
1 GET /eHealthServer/resources/caregivers/1 HTTP/1.1
2 Host: localhost:9090
3 Connection: keep-alive
4 Accept: application/xml
5 Authorization: Basic dGVzdDoxMjM0
6 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.22 (KHTML) Chrome
7 Accept-Encoding: gzip,deflate,sdch
```

```
8 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

List. 2.1: HTTP Request header

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

List. 2.2: HTTP Response header

The first line declares a HTTP method, a Unified Resource Identifier (URI) path and a version tag. The method, also called verb, describes the action performed on the designated resource. The version tag describes for which version of HTTP this request is intended.

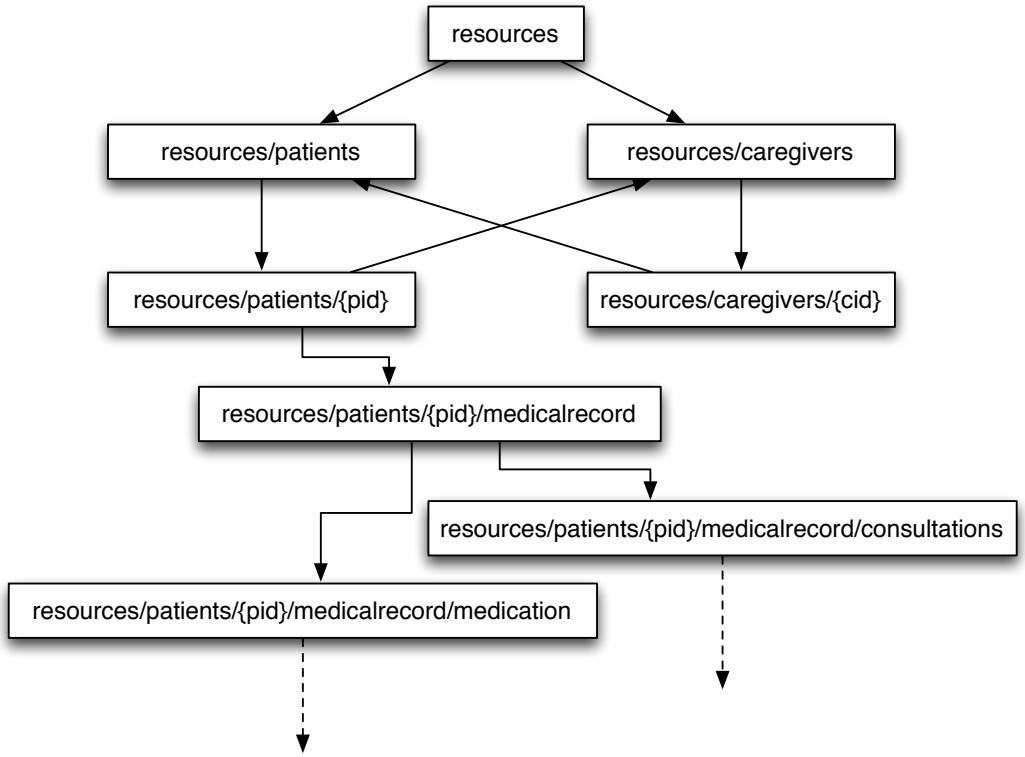


Fig. 2.1.: Connectedness of Resources within the eHealthServer

The next header lines describe different message properties. The Internet Assigned Numbers Authority (IANA) administrates a list[1] of valid and obsolete HTTP headers as well as a list of valid media-types[2]. Most relevant for the following work are the headers about the authorisation (`Authorization`), about the content of the body (`Accept`) and the host (`Host`).

HTTP Methods or Verbs

blah blah

[1]List of headers: `http://www.iana.org/assignments/message-headers/perm-headers.html`
[2]List of media-types (formerly MIME Types): `http://www.iana.org/assignments/media-types`

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.	↯	↯
PUT	Requests that the enclosed entity be stored under the supplied URI. If it refers to an already existing resource, it is modified. If it does not point to an existing resource, then the server can create the resource.	↯	✓
DELETE	Request that the resource under the supplied URI is deleted if present.	↯	✓

Tab. 2.1.: Overview of HTTP methods

HTTP Status Codes

blah

2.2. REST: Representational State Transfer

2.2.1. Introduction

As already mentioned the media-type is defined in the header of a request together with additional meta data and the resource data is stored in the body of a response. Table 2.2 provides an overview of all elements of a resource.

Component	Description	HTTP Element
Resource data	Intended conceptual target of a hyperlink.	Response Body
Identifier	Unique address assigned to the resource used to identify.	HTTP Header
Media Type	Encapsulation format of the resource.	HTTP Header
Meta Data	Additional data about the structure of the resource.	HTTP Header

Tab. 2.2.: Elements of a resource by Fielding[2]

Java enforces the "Write Once, Run Anywhere" principle.

A very good overview is given in Table 2.2.

These informations were retrieved from [12].

A more detailed description is provided by [5], [2] or [3, p. 54].

The class which is responsible for all the lookup and caching tasks is ch.unifr.ebay.util.Locator which implements the ch.unifr.ebay.Lookup interface. The lookup() method [3] is its most relevant service.

[3]This method is very analog to the lookdown() method.

Following file is the root [4] of my report `/home/genius/uni/master/report/src/main.tex` and this URL `http://diuf.unifr.ch/softenggroup/student-projects/completed/05-06/bobmarley/master.php` provides all the related information and stuff. You can send me your comments by email at following adress: `bob.marley@genius.org`

There are three possibilities: *(i)* the first; *(ii)* the second; *(iii)* the third; to achieve this goal!

2.3. From World Wide Web to the Web of Things

2.3.1. World Wide Web

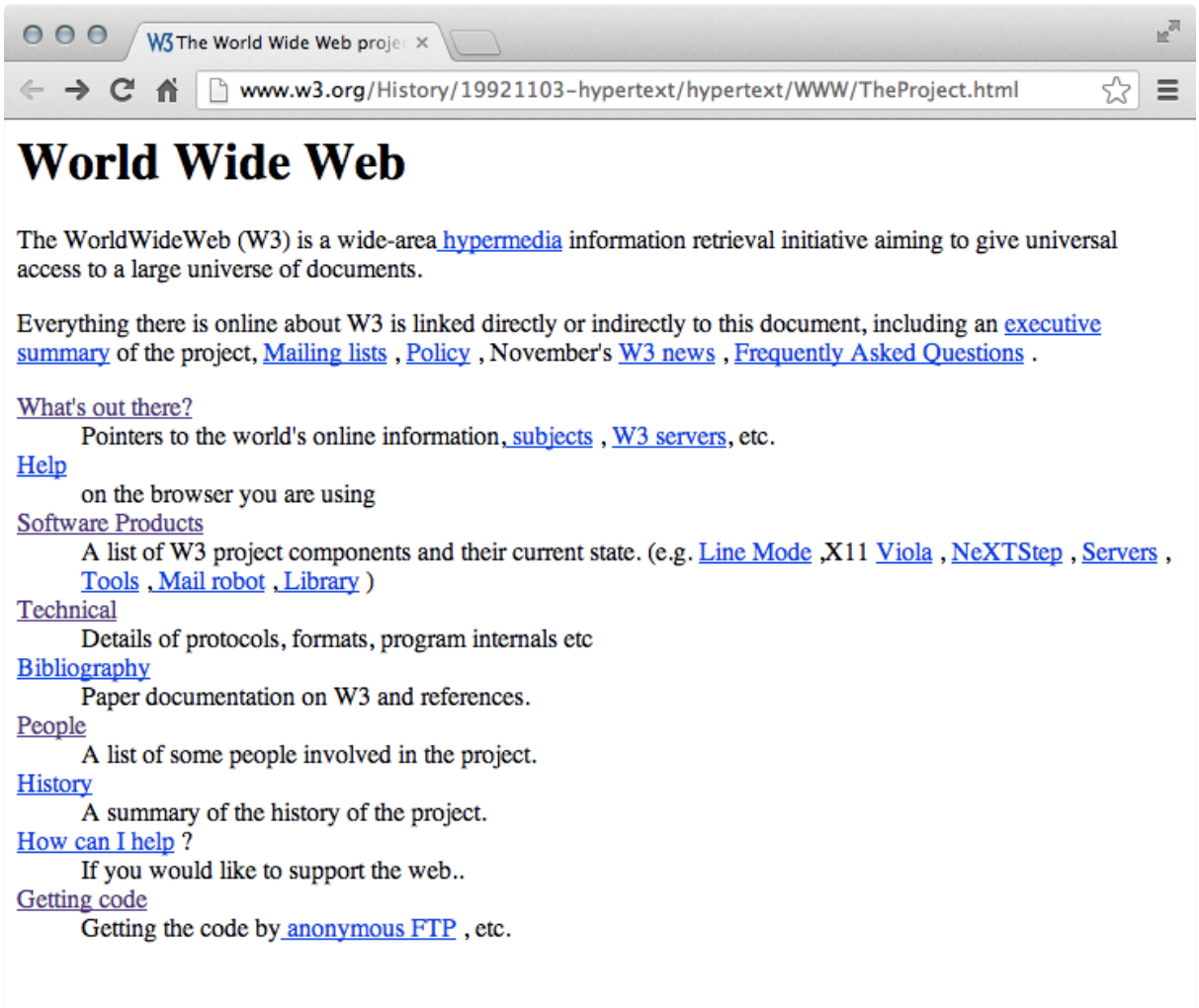


Fig. 2.2.: First website publicly available hosted by CERN

Do you know about eXtensible Markup Language (XML)? The word XML is used as an example of acronym here. Each time you use it, tpye XML and Latex will automatically

[4] This URL is a shortcut to the official website on the University server: `http://www.berkeley.com/cs/student-projects/bobby/index.jsp`

expand the acronym only for its first occurence! If we need it explained again, you can force the explanation of the acronym by writing eXtensible Markup Language (XML).

2.3.2. Internet of Things (IoT)

bla bla bla Internet of Things (IoT)

The Listing 2.3 shows how you can insert source code stored in a separate file (you can even choose the lines you want to display).

```
1  package org.w3._2001.xmlschema;
2  import java.util.Calendar;
3  import javax.xml.bind.annotation.adapters.XmlAdapter;
4  public class Adapter2
5      extends XmlAdapter<String, Calendar>
6  {
7      public Calendar unmarshal(String value)  {
8          return (javax.xml.bind.DatatypeConverter.parseDate(value));
9      }
10     public String marshal(Calendar value) {
11         if (value == null) {
12             return null;
13         }
14         return (javax.xml.bind.DatatypeConverter.printDate(value));
15     }
16 }
```

List. 2.3: Adapter2.java: unmarshal() and marshal() methods

bla bla bla.

Here you see how you can typeset a single line of code in the text:

```
1 void setLocalTranslation(Vector3f localTranslation);
```

or a few lines of code (without caption) in the text:

```
1 Matrix3f rotMat = new Matrix3f();
2 rotMat.fromAngleNormalAxis( FastMath.DEG_TO_RAD * 45.0f, new Vector3f( 1.0f, 0.0f, 0.0
      f));
3 box.setLocalRotation( rotMat );
```

Another example is Listing 2.4 where you have a floting environment with the code given in the latex source directly.

You may note that one can reference the listing and a single line. For instance on line 7 of Listing 2.3 or on line 15 of the Listing 2.4.

bla bla bla

The two last examples Listing 2.5 and Listing 2.6 shows a listing with containing something different than Java code, here XML code.

```
1  <application xmlns="http://wadl.dev.java.net/2009/02">
2      <doc xmlns:jersey="http://jersey.java.net/"
3          jersey:generatedBy="Jersey: 1.12 02/15/2012 04:51 PM"/>
4      <grammars>
5          <include href="application.wadl/xsd0.xsd">
6              <doc title="Generated" xml:lang="en"/>
7          </include>
```

```
/**
 * This class moves the box along the negative x-axis
 */
class KeyNodeLeftAction extends KeyInputAction
{
    // the node to manipulate
    private Spatial node;

    KeyNodeLeftAction( Spatial node )
    {
    this.node = node;
    this.speed = 1;
    }

    public void performAction( InputActionEvent evt )
    {
        node.getLocalTranslation().x -= (speed * evt.getTime());
    }
}
```

List. 2.4: A floating example

```
    </grammars>
    <resources base="http://localhost:9090/eHealthServer/resources/">
        <resource path="/patients/{pid}/medicalrecord">
            <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="pid" style="
                template"
                type="xs:int"/>
            <method id="getMedicalRecordXML" name="GET">
                <response>
                    <representation xmlns:ns2="http://eHealth" element="ns2:
                        medicalRecord"
                        mediaType="application/xml"/>
                    <representation xmlns:ns2="http://eHealth" element="ns2:
                        medicalRecord"
                        mediaType="application/json"/>
                    <representation xmlns:ns2="http://eHealth" element="ns2:
                        medicalRecord"
                        mediaType="text/xml"/>
                </response>
            </method>
            <method id="getMedicalRecordHTML" name="GET">
                <response>
                    <representation mediaType="text/html"/>
                </response>
            </method>
            <method id="setMedicalRecordURL" name="PUT">
                <request>
                    <representation mediaType="application/x-www-form-urlencoded">
                        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="age"
                            style="query"
                            type="xs:string"/>
                        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="weight"
                            style="query" type="xs:string"/>
                        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="drugs"
                            style="query" type="xs:string"/>
```

```
37                      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="alcohol
                            "
38                          style="query" type="xs:string"/>
39                      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="nicotin
                            "
40                          style="query" type="xs:string"/>
41                  </representation>
42              </request>
43              <response>
44                  <representation xmlns:ns2="http://eHealth" element="ns2:
                        medicalRecord"
45                      mediaType="application/xml"/>
46                  <representation xmlns:ns2="http://eHealth" element="ns2:
                        medicalRecord"
47                      mediaType="application/json"/>
48                  <representation xmlns:ns2="http://eHealth" element="ns2:
                        medicalRecord"
49                      mediaType="text/xml"/>
50              </response>
51          </method>
```

List. 2.5: application.wadl: A floating XML code example with file

2.3.3. NoSQL

bla bla bla

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" jaxb:version="2.0" xmlns:tns="http://
        eHealth"
    targetNamespace="http://eHealth">

    <xs:annotation>
        <xs:appinfo>
            <jaxb:globalBindings>
                <jaxb:javaType name="java.net.URI" xmlType="xs:anyURI" parseMethod="
                    create" printMethod="toASCIIString"/>
                <jaxb:javaType name="java.util.Calendar" xmlType="xs:dateTime"
                    parseMethod="javax.xml.bind.DatatypeConverter.parseDate" printMethod
                    ="javax.xml.bind.DatatypeConverter.printDate"/>
                <jaxb:javaType name="java.util.Calendar" xmlType="xs:date" parseMethod="
                    javax.xml.bind.DatatypeConverter.parseDate" printMethod="javax.xml.
                    bind.DatatypeConverter.printDate"/>
            </jaxb:globalBindings>
        </xs:appinfo>
    </xs:annotation>




    <xs:element name="caregiver">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cid" minOccurs="1" maxOccurs="1" type="xs:int"/>
                <xs:element name="lastName" minOccurs="1" maxOccurs="1" type="xs:string"
                    />
                <xs:element name="firstName" minOccurs="1" maxOccurs="1" type="xs:string
                    "/>
                <xs:element name="speciality" minOccurs="1" maxOccurs="1" type="xs:
                    string"/>
                <xs:element name="monthsOfExperience" minOccurs="1" maxOccurs="1" type="
                    xs:int"/>
                <xs:element name="adress" type="tns:adressType" minOccurs="1" maxOccurs=
                    "1"/>
                <xs:element name="Links" minOccurs="1" maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <!-- Links to: Patients (which have this caregiver) -->
                            <!-- Links to: Devices (registered to caregiver for alarm,
                                software, ...) -->
                            <xs:element ref="tns:link" minOccurs="0" maxOccurs="unbounded
                                "/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="uri" type="xs:anyURI"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="patient">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="pid" maxOccurs="1" minOccurs="1" type="xs:int"/>
                <xs:element name="lastName" minOccurs="1" maxOccurs="1" type="xs:string"
                    />
                <xs:element name="firstName" minOccurs="1" maxOccurs="1" type="xs:string
                    "/>
                <xs:element name="gender" minOccurs="1" maxOccurs="1">
                    <xs:simpleType>
```

3
The eHealth Server: Design

3.1. Overview **12**
3.1.1. A RESTful Recipe 12
3.1.2. From an abstraction to a concrete implementation 13
3.2. Fourth: Expose a subset of the uniform interface **13**
3.2.1. General overview of methods used 13
3.3. Fifth: Design representations **14**
3.3.1. Using XSD and JAXB to obtain POJOs 14
3.3.2. Name-spaces: eHealth, JAXB and XSD 14
3.3.3. MySQL ERM 14
3.4. Web-Service Description **14**
3.4.1. WSDL 14
3.4.2. WADL 15
3.5. Abstract of the design process **15**

3.1. Overview

A continuous goal throughout the entire work is to follow a top-down approach, from an abstraction to a concrete implementation. This chapter will cover the design process of the web-service but in particular the description, manipulation and storage of resources defined to be relevant in the eHealthServer.

3.1.1. A RESTful Recipe

A recipe is similar to a design pattern and enforces a sequences of steps to follow to obtain a final product. The approach in this chapter follows the recipe, adapted to integrate intermediate artefacts, from the work of Richardson and Ruby[8]. Especially step five was modified and step seven newly introduced. The former describes the design of responses and requests simultaneously instead of separated. The latter provides a methodology to build databases for the storage of resources.

1. **Evaluate Data Sets**
   Evaluates a real case medical record to extract the entity sets.
2. **Split data sets into resources**
   Using a meta model, the sets are split into compositions and resources.
3. **Name the resources with URI's**
   Build a URI schema of the resources identified.
4. **Expose a subset of the uniform interface**
   Decided on what method of the uniform interface is applicable to what URI.
5. **Design representations**
   Define messages served by the server to the client and vice-versa.
6. **Integrate resource into existing resources, using hypermedia links and forms**
   Usage of comments to describe which resources are inter-linked.
7. **Resource storage**
   Design the entity database relationship model.

3.1.2. From an abstraction to a concrete implementation

blah blah blah

3.2. Fourth: Expose a subset of the uniform interface

3.2.1. General overview of methods used

blah blah blah

Special Case: Medication and preconditions

blah blah blah

Method	URI
authority	
GET	`http://localhost:9090/eHealthServer`
path	
GET	`.../`
GET, POST	`.../patients`
GET, PUT, DELETE	`.../patients/{pid}`
GET, PUT	`.../patients/{pid}/medicalrecord`
GET, POST	`.../patients/{pid}/medicalrecord/consultations`
GET, DELETE	`.../patients/{pid}/medicalrecord/consultations/{consultationID}`
GET, POST	`.../patients/{pid}/medicalrecord/preconditions`
DELETE	`.../patients/{pid}/medicalrecord/preconditions?type={type}`
GET, POST, DELETE	`.../patients/{pid}/medicalrecord/medications`
DELETE	`../patients/{pid}/medicalrecord/medications?typeOfMedication={typeOfMedication}`
GET, POST	`.../caregivers`
GET, PUT, DELETE	`.../caregivers/{cid}`

Tab. 3.1.: Overview of the HTTP methods used

3.3. Fifth: Design representations

3.3.1. Using XSD and JAXB to obtain POJOs

blah blah blah

3.3.2. Name-spaces: eHealth, JAXB and XSD

blah blah blah

3.3.3. MySQL ERM

blah blah blah

3.4. Web-Service Description

3.4.1. WSDL

blah blah blah

3.4.2. WADL

Web Application Description Language (WADL) is blah blah blah

```
4   <grammars>
5       <include href="application.wadl/xsd0.xsd">
6           <doc title="Generated" xml:lang="en"/>
7       </include>
8   </grammars>
```

List. 3.1: Declaration of a grammar

blah blah blah

```
346         <resource path="/caregivers/{cid}">
347             <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="cid" style="
                    template"
348                 type="xs:int"/>
349             <method id="getCaregiverXML" name="GET">
350                 <response>
351                     <representation xmlns:ns2="http://eHealth" element="ns2:caregiver"
352                         mediaType="application/xml"/>
353                 </response>
354             </method>
355             <method id="getCareGiverHTML" name="GET">
356                 <response>
357                     <representation mediaType="text/html"/>
358                 </response>
359             </method>
360             <method id="setCaregiverURL" name="PUT">
361                 <request>
362                     <representation mediaType="application/x-www-form-urlencoded">
363                         <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="
                                firstName"
364                             style="query" type="xs:string"/>
```

List. 3.2: Declaration of a HTTP method in WADL

3.5. Abstract of the design process

blah blah blah

4 Conclusion

4.1. Review

This work started out as an implementation for an eHealth Service, delivering resources over a REST interface and complying to the constrains defined by Roy T. Fielding and Richardson and Ruby. Using both sources the process became a structure usable by any other similar implementation. Instead of focusing on the integration of a multitude of similar resources, I tried to find a general applicable set of recipe-like steps to follow. Intermediate products are not simple artefacts and references but key dependencies. They can be reused to generate new code, new method stubs and extend the set of resources with new entities.

4.2. Outlook: Software Engineering methodology and client

These step-by-steps instructions comply to most currently used software project management methodologies and can therefore easily be integrated in a pre-existing development cycle. For instance in an Agile environment, it is easy to modify the resource schema during the requirement engineering phase, declare new resources, integrate them into the WADL and XML Schema Definition (XSD) and automatically generate new method stubs for the implementation.

4.3. Final statements

This bachelor thesis grew organically around the ideas and discussions with caregivers and other persons. First as simple implementation work based on a real use-case and afterwards as a study on methodology.

A
Common Acronyms

AJAX	Asynchronous JavaScript And XML
ANSI	American National Standards Institute
CERN	European Organization for Nuclear Research
CSS	Cascading Style Sheet
CGI	Common Gateway Interface
cURL	Client for URL
DBMS	Database Management System
DOM	Document Object Model
ERM	Entity Relationship Model
HL7	Health Level 7
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IoT	Internet of Things
IP	Internet Protocol
IPSec	Internet Protocol Security
JAXB	Java Architecture for XML Binding
JAX-RS	Java API for RESTful Web Services
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JSON	JavaScript Object Notation
JSP	Java Server Pages
JSTL	Java Server Tag Library
NCSA	National Center for Supercomputing Applications
PHP	Hypertext Processor
POJO	Plain Old Java Object
POM	Project Object Model
QR	Quick Response
RFID	Radio Frequency Identification
REST	Representational State Transfer
ROA	Resource Oriented Architecture
SAX	Simple API for XML
SOA	Service Oriented Architecture
SQL	Structured Query Language

SPDY	SPeeDY, an open networking protocol
TCP	Transmission Control Protocol
URI	Unified Resource Identifier
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Service Description Language
WoT	Web of Things
XML	eXtensible Markup Language
XSD	XML Schema Definition

B

License of the Documentation

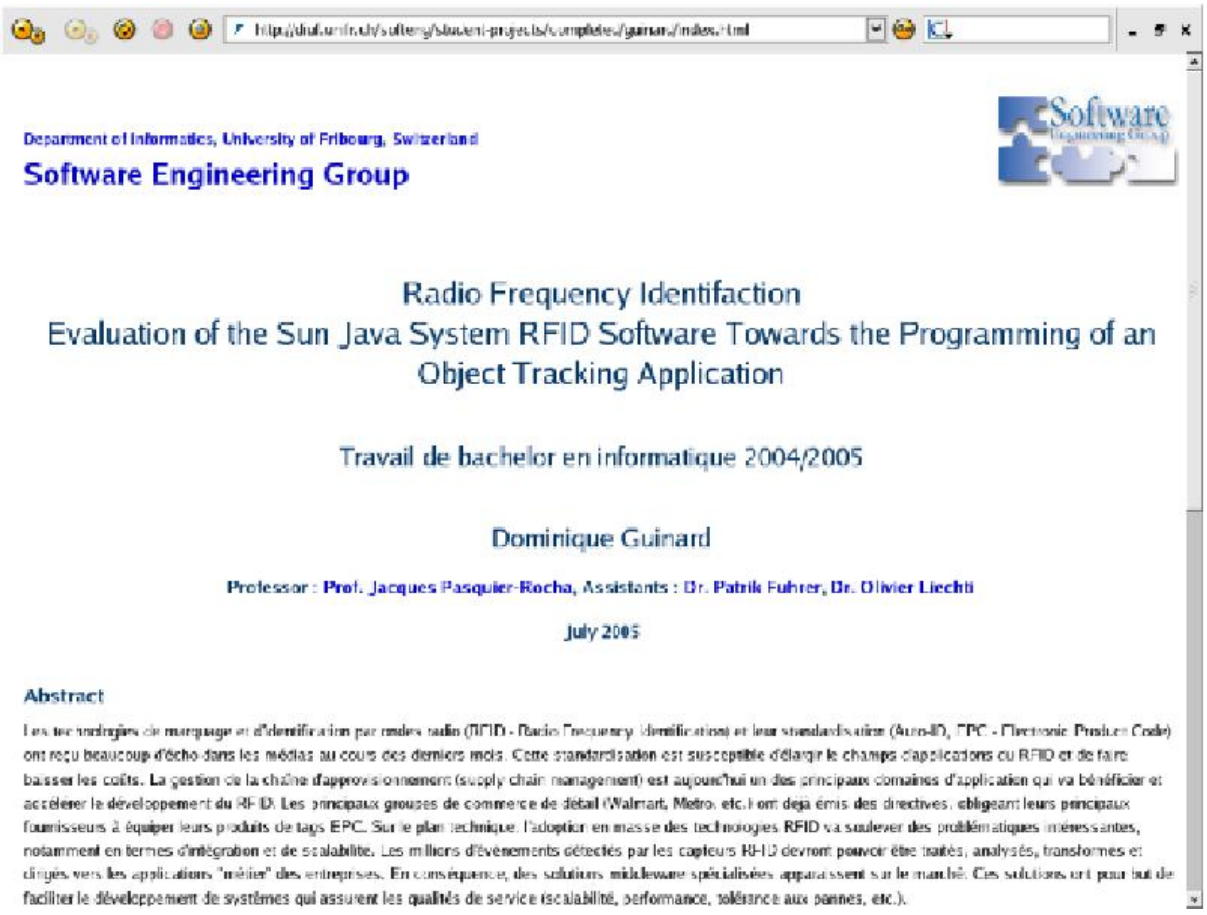
C
Website of the Project

A web-page was created for this project: http://www.gmipsoft.com/rfid[1]. On this page you will find:

- The API of the project.
- This binaries and sources of this documentation.
- The binaries and sources of the.
- Parts of the content (see D.1).

C.1 provides a screenshot of this website.

[1]This URL is a shortcut to `http://diuf.unifr.ch/drupal/softeng/teaching/studentprojects/ehealth`



Department of Informatics, University of Fribourg, Switzerland

Software Engineering Group

Radio Frequency Identifaction
Evaluation of the Sun Java System RFID Software Towards the Programming of an Object Tracking Application

Travail de bachelor en informatique 2004/2005

Dominique Guinard

Professor : Prof. Jacques Pasquier-Rocha, Assistants : Dr. Patrik Fuhrer, Dr. Olivier Liechti

July 2005

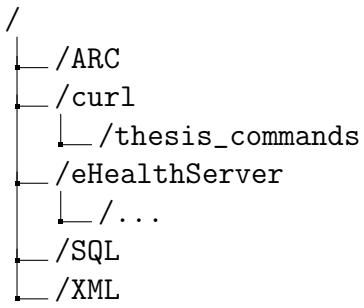
Abstract

Les technologies de marquage et d'identification par radio radio (RFID - Radio Frequency Identification) et leur standardisation (Auto-ID, EPC - Electronic Product Code) ont reçu beaucoup d'écho dans les médias au cours des derniers mois. Cette standardisation est susceptible d'élargir le champs d'applications ou RFID et de faire baisser les coûts. La gestion de la chaîne d'approvisionnement (supply chain management) est aujourd'hui un des principaux domaines d'application qui va bénéficier et accélerer le développement du RFID. Les principaux groupes de commerce de détail (Walmart, Metro, etc.) ont deja émis des directives, obligeant leurs principaux fournisseurs à équiper leurs produits de tags EPC. Sur le plan technique, l'adoption en masse des technologies RFID va soulever des problématiques intéressantes, notamment en termes d'intégration et de scalabilité. Les millions d'événements détectés par les capteurs RFID devront pouvoir être traités, analysés, transformés et dirigés vers les applications "métier" des entreprises. En conséquence, des solutions middleware spécialisées apparaissent sur le marché. Ces solutions ont pour but de faciliter le développement de systèmes qui assurent les qualités de service (scalabilité, performance, tolérance aux pannes, etc.).

Fig. C.1.: Screenshot of the project's official web-page

D
CD-Rom

Overview of the content



Short description

ARC

Contains a JSON formatted file, that can be imported as project in the Advanced Rest Client, a plugin for Chrome. It contains a full set of all possible resource operations for testing.

curl

Contains all files described in the appendix for the eHealthServer documentation.

eHealthServer

Contains the latest snapshot of the eHealthServer taken on February 17, 2020. It can be imported as is into Netbeans.

SQL

Contains the ERM file and a SQL dump file for a MySQL Server.

XML

Contains the WADL and XSD file.

Fig. D.1.: The CD-ROM of this project

References

[1] R. Fielding et al. *Hypertext Transfer Protocol - HTTP/1.1*. 1999. RFC 2616.

[2] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. iv, 2, 6

[3] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of software engineering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[4] Marc Hadley. Web application description language (wadl). Technical Report TR-2006-153, Sun Microsystems, April 2006.

[5] Ian Jacobs. Uris, addressability, and the use of http get and post. World Wide Web Consortium, TAG Finding, March 2004.

[6] Andreas Meier. *Relationale und postrelationale Datenbanken*. eXamen.press. Springer, Berlin [u.a.], 6., überarb. und erw. edition, 2007.

[7] Goestenmeyer Ralph and Rehn-Goestenmeier Gudrun. *Das Einsteigerseminar Linux*. Verlag moderne industrie Buch AG & Co. KG, Landsberg, Königswinterer Str. 418, 35227 Bonn, Deutschland, 4., überarb. edition, 2003.

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

[9] Andreas Ruppen and Jacques Pasquier-Rocha. A Meta-Model for the Web of Things. Internal Working Paper 13-03, Department of Informatics, University of Fribourg, Switzerland, June 2013.

[10] Alexander Schatten. *Best Practice Software-Engineering*. Springer DE, 2010.

Referenced Web Resources

[11] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (accessed July 30, 2005).

[12] Website of the Sun Microsystems. http://www.sun.com/software/solutions/rfid/ (accessed July 28, 2005).

Index

ERM, 14

HTML, 4
HTTP, 4
    Status Codes, 6
    TCP/IP, 4

Internet of Things, 8

JSON, 14

NoSQL, 10

REST
    Uniform Interface, 13

SQL, 14

WADL, 15
World Wide Web, 7
WSDL, 14

XML, 14
XSD, 14