UMEÅ UNIVERSITY
Department of Computer Science
Report

January 15, 2010

A RESTful service

Service-Oriented Architectures, HT-09

Umume

Anton Johansson, dit06ajn@cs.umu.se
Jonny Strömberg, dit06jsg@cs.umu.se

Start-server: `dit06ajn~/edu/soa/umume> ./start-server.sh`

Start-gui: `dit06ajn~/edu/soa/umume> ./start-gui.sh`

Supervisors
P-O Östberg, p-o+soa@cs.umu.se

Contents

1 Introduction

This report describes the work done designing and implementing a RESTful web service providing information about employees and students at Umeå university. For this web service a client web site is done to highlight possible uses of the web service.

The main idea with the web service is to combine information from the existing information about every employee/student with a possibility for persons to add information about themselves. This information can for example be a brief description, visiting location and usernames for other web services such as Twitter[1] to enable mashup implementations.

The web service will hereby be called *Umume-rest*, a combination of *Umu* as in Umeå university and me because it provides a service for students and employees. The client web site is called *Umume-website*. All students and employees at Umeå university are potential users of our service, they will herby be refereed to as users.

These implementations are done with the programming language *Java*[2] and uses the framework *Spring Framework*[3] for the web site, and *Jersey*[4] for the RESTful Web service.

The original project specification for this work is shown in Appendix A.1.

2 Problem analysis

Since the *Umume-rest* is created to enable third party developers to access information and create their own web service, it was decided to make the web service RESTful instead of using the SOAP protocol. This decision was made because in a RESTful web service every resource have a unique identifier easily accessible with a simple HTTP GET method call. For example the information about a person could be fetched by pointing a web browser to `http://example.com/users/aonjon04`. This for example enables the development of simple AJAX Javascript clients that use our service, this is demonstrated and described in Section **??**.

The web service *Umume-rest* builds mainly on two existing services provided by Umeå university, a Lightweight Directory Access Protocol directory service (LDAP) and a Central Authentication Service (CAS).

2.1 LDAP

All preexisting informations about users is fetched from an LDAP directory service provided by the university at `ldap://ldap.umu.se`. This directory service contains information about every student and employee. Every user is uniquely identified by a 8 character string such as ”aonjon04”. With every unique user comes extra information such as faculty membership, email addresses, phone numbers and so on.

An initial idea was to use address information about every user and map this information to a longitude and latitude to plot the users positions an a map, for example by using Google Maps API[5]. This idea was however discarded since the structure of location information from LDAP was inconsistent between different users and faculties.

The most important service provided by LDAP used by *Umume-rest* is the possibility to search for first- and surnames to retrieve their unique CAS-username. This username is used to authenticate users, see the following section.

2.2 Central Authentication Serive

Every employee and student at Umeå university can get a CAS-user and password by physically visiting a reception and identifying themselves. This CAS-user enables access to different services around campus, for example accessing the wireless network, scientific articles and checking results from finished courses. More information about CAS can be found at `http://www.it.umu.se/vara-tjanster/centralt-anvandarnamn/cas/` (in Swedish).

The CAS used by *Umume-rest* is located at `https://cas.umu.se/` and is used by *Umume-rest* to authenticate requests to add or change information about your own user. This means that no extra username or password is required for users or *Umume-rest* and users can only edit their own information.

3 Compilation and deployment

All files needed to use set up the web service are located at:
`dit06ajn~/edu/soa/umume`/umume-rest and the files for the web site are at:
`dit06ajn~/edu/soa/umume`/umume-website

3.1 Servlet container

This project has been developed with *Apache Tomcat 6.0.20*[6] as servlet container but any Java servlet container should do.

3.2 Web service - umume-rest

This section explains how to understand, compile and use *umume-rest*. This catalog contains the following sub directories:

[1] `http://twitter.com/`
[2] `http://java.sun.com/`
[3] `http://www.springframework.org/`
[4] `https://jersey.dev.java.net/`
[5] `http://code.google.com/apis/maps/`
[6] http://tomcat.apache.org/

- `src` contains the source code.
- `src/main/resources/` contains the SQLite database and the configuration files for standard behaviour of the compiled system, see section **??**.
- `target` will, after a successful compilation, contain all the compiled sources as well as configuration files used by this web service.
- `lib` contains all requires third-party libraries needed by the *Umume web service*.
- `conf` contains the build-properties file and the jar-file for Ivy.

And these are the files in the catalog:

- `build.xml` contains settings for building this web service.
- `ivy.xml` contains all Ivy dependencies.
- `ivysettings.xml` contains the Ivy settings.

3.2.1 Compilation

The following commands will require the software tool *Apache Ant*[7]. More details about what happens using *ant* in this project is found in the file *build.xml*[8].

To deploy *umume-rest* issue the following command:

```
salt:./umume-rest> ant deploy
```

This will create a directory `target` if it does not already exists and compile/move source-code and configuration files to that directory. Then a war file is generated and moved to the servlet container (remember to configure the build properties in `conf/build.properties`)

The root-directory for class-files when using *umume-rest* is compiled to *target/classes/*, while the root-directory for test-code is compile to *target/test-classes/*.

To create portable *war*-file of the compiled sources issue the following command:

```
salt:./umume-rest> ant war
```

This will create *umume-rest.war* which can be deployed within any Java servlet container.

3.3 Web site - umume-website

This catalog contains the following relevant sub directories:

- `src` contains the source code.
- `war/WEB-INF/jsp/` contains all .jsp-files used for the web site design.
- `war/WEB-INF/classes/` will, after a successful compilation, contain all the compiled source files.
- `war/WEB-INF/lib` contains all requires third-party libraries needed by *umume-website*.
- `war/WEB-INF/tld/` contains Spring specific librarys.
- `war/css/` contains the css stylesheet.
- `war/images/` contains all images displayed on the web site.
- `war/js/` contains all javascript-files.

And these are the interesting files in *umume-website*:

- `build.xml` contains settings for building this web service.
- `ibuild.properties` contains the build properties.
- `iwar/WEB-INF/umume-servlet.xml` contains definitions of all views and beans used in *umume-website*.
- `iwar/WEB-INF/web.xml` contains definitions of filers, sevlets and taglibs used in *umume-website*.
- `iwar/WEB-INF/urlrewrite.xml` contains rules for URL rewriting.

3.3.1 Compilation

The following commands will require the software tool *Apache Ant*[9]. More details about what happens using *ant* in this project is found in the file *build.xml*[10].

To deploy *umume-website* issue the following command:

```
salt:./umume-website> ant deploy
```

This will create a directory `war/WEB-INF/classes` if it does not already exists and compile/move source-code and configuration files to that directory. Then the catalog is copied the the pre-defined servlet container (`conf/build.properties`).

To create portable *war*-file of the compiled sources issue the following command:

```
salt:./umume-rest> ant deploywar
```

This will create *umume.war* which can be deployed within any Java Servlet container.

4 Usage

This description explains how to communicate with the web sercive and how to navigate on the web site.

[7] http://ant.apache.org/
[8] http://ant.apache.org/manual/using.html
[9] http://ant.apache.org/
[10] http://ant.apache.org/manual/using.html

4.1 Web service - umume-rest

There are two different resources in the service. Both allows GET requests and and one of them also allows authorised users to add or change information with PUT.

- GET `/users/\{uid\}` returns information about a user in either *application/xml* or *application/json* depending on which the user requested.
- PUT `/users/\{uid\}?ticket=\{CAS-ticket\}&s` updates information about the specified user if the request contains a CAS-ticket and a service-URL that validates. The format of the request is *application/xml*. For more information se 1
- GET `/search/\{searchString\}` returns a list of users matching the provided search string in either *application/xml*, *application/json* or *application/javascript* depending on which the user requested.

CodeSnippet 1 PUT request example

```
<person ref="http://mega.cs.umu.se:8080/umume-rest/users
    <description>My own description</description>
    <latitude>0.0</latitude>
    <longitude>0.0</longitude>
    <twitterName>MyTwitterName</twitterName>
</person>
```

4.2 WADL

See complete WADL in Appendix A.2.

4.3 Web site - umume-website

The following images describes how to navigate through the *umume-website*.

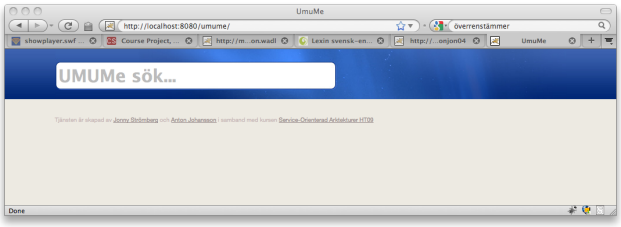

Figure 1: Start page.

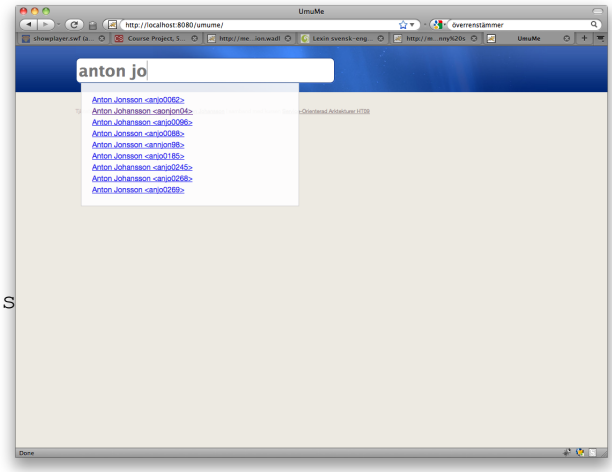

Figure 2: Search for persons.

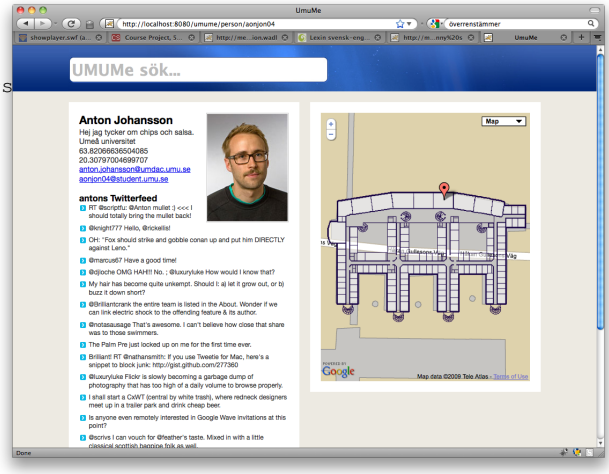

Figure 3: User profile, eventually with Twitter-feed and position at Google Maps.

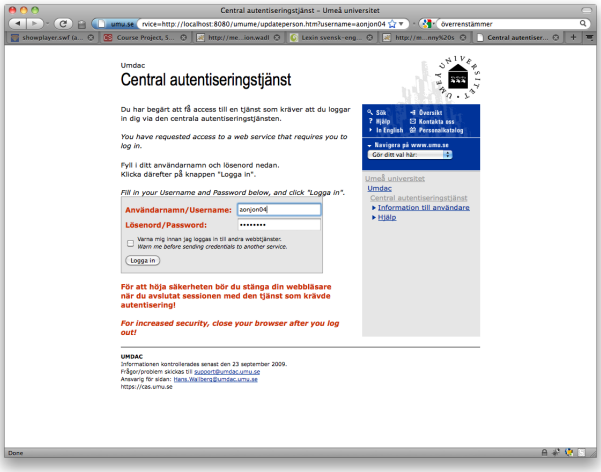

Figure 4: Click on ”uppdatera” at the bottom of the profile and then authorize CAS-user.

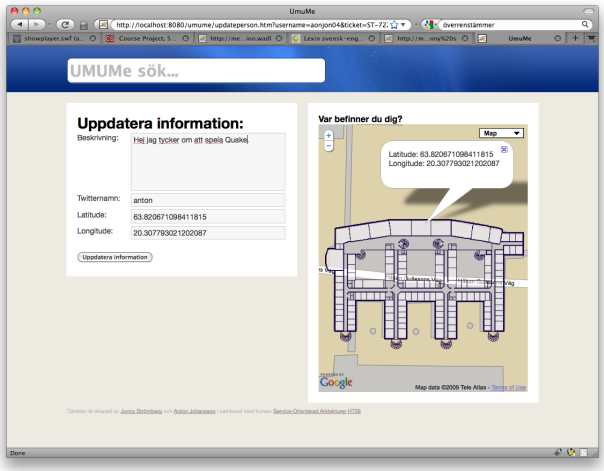

Figure 5: Update user information

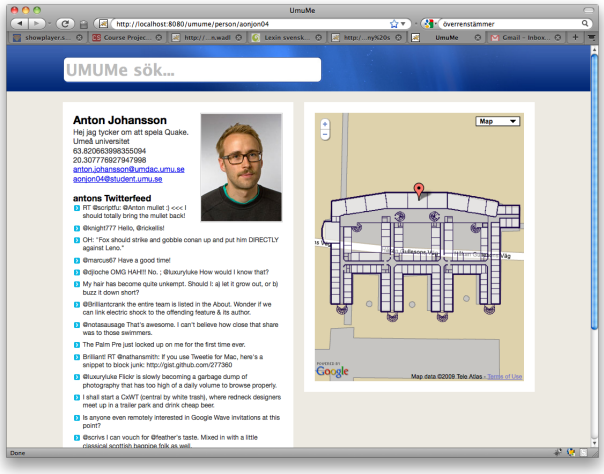

Figure 6: See updated information (if the authorized CAS-user was the same as the updated profile.

5 System description

5.1 Umume-rest

The RESTful web service uses the *JAX-RS API*[11] with the *Jersey*[12] implementation. This enables resources to be defined within normal java classes by merely adding annotations to specify at which path the resource is available and what HTTP-methods the resource is accessable by. Se an example in Code 2. This is basically what is used by *Umume-rest* to provide resources describing users. A java bean *PersonBean.java* is used to represent users. This bean is marshaled and unmarshaled by JAXB[13] into different mediatypes depending on what is requested. Our user resources can be fetched as XML or JSON[14] depending on which HTTP Accept header is sent by the request.

CodeSnippet 2 JAX-RS resource

```
// The resource is available at path /users/{uid}
@Path("/users/{uid}")
public class UsersResource {
    // The Java method will process HTTP GET requests
    @GET
    // It can return XML and JSON
    @Produces({"application/xml",
               "application/json"})
    public Person getUserXML(
                        @PathParam("uid") String uid) {
        //...
        return person;
    }
```

5.1.1 Update user resource

To update a user resource a valid CAS user is required. The request is made by a HTTP PUT request with new information about the referenced user, see example implementation in Code 3. This example consumes XML with updated information about the user resource.

To validate whether the request comes from the same user that requests the change a CAS ticket is required as a query string. This supplied ticket from CAS is validated at `https://cas.umu.se` using a CAS-client[15]. If the supplied ticket is valid a username can be extracted, and if this username is the same as the username of the resource to change it can be certain that the request comes from the person who is represented by the resource. Since a ticket is confidential information all this network traffic is tunneled through a secure connection using the HTTPS protocol. See Figure 8 for an overview of the communication.

CodeSnippet 3 Update resource

```
@PUT
@Consumes(MediaType.APPLICATION_XML)
public Response updateUser(
        @PathParam("uid") String uid,
        PersonBean pb,
        @QueryParam("ticket") String ticket,
        @QueryParam("service") String service) {
        // validate user, persist updated info
        return Response; //200 OK or errorcode
}
```

5.1.2 Persistance layer

5.2 Umume-website

Springframework Jersey Sökning JS Google Maps URL-filter

[11] `http://jcp.org/en/jsr/detail?id=311`
[12] https://jersey.dev.java.net/
[13] `https://jaxb.dev.java.net/`
[14] `http://www.json.org/`
[15] `http://www.jasig.org/cas`

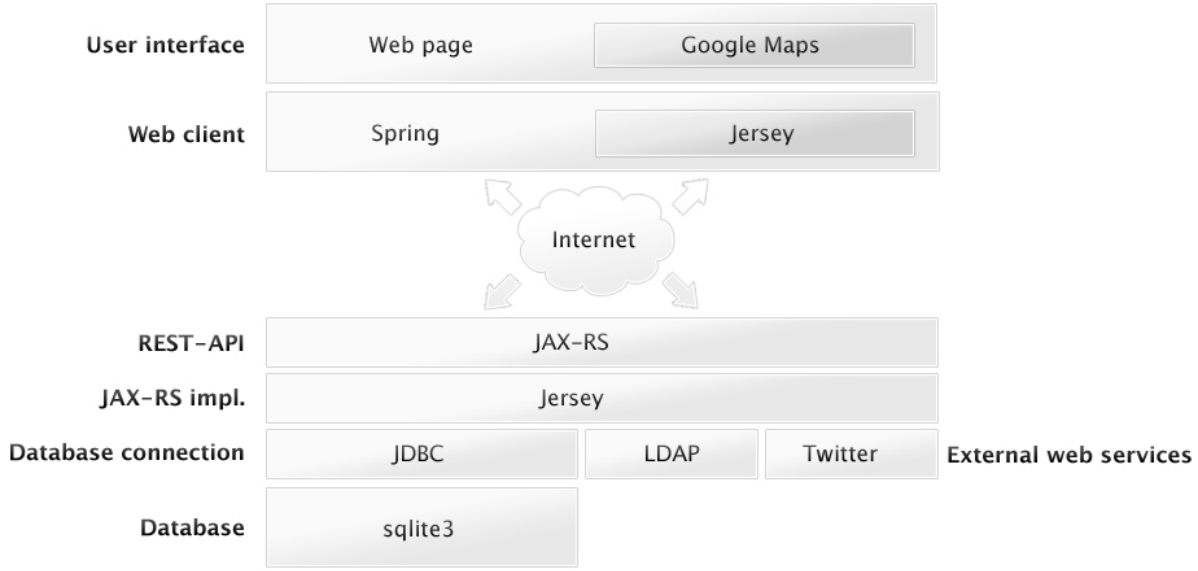

Figure 7: GCom stack

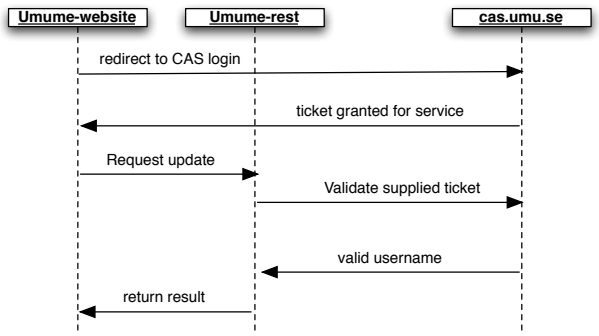

Figure 8: CAS authentication

5.2.1 Error handling

When sending messages a group member may detect that one of the receiving members has crashed. If the detecting member is a group leader it will directly send a groupchange message, otherwise it will send a membercrash message which when received by the group leader will result in a groupchange message multicasted to the group. A groupchange message contains information about the complete new group composition whereas a membercrash message only contains information about the members that have crashed.

When the member that has crashed is the group leader the exact same procedure is used except that when processes receive membercrash messages they will check if they are the new group leader, and if they are they will send a groupchange message to the group. Group members test if they are the new group leader by checking if they have the maximum value of *UUID*s in the current group. This is a variant of the *Bully alghoritm*, see page 482

	p1	p2	p3 - Sequencer (hold all messages)
Send order	one two		
			(Release messages in reverse order)
Receive order	one two	one two	one two

Table 1: Casual-Total ordering tests

6 Discussion

The following sections will discuss some of the problems and solutions encountered while implementing *GCom*.

6.1 No automatic failure detection

We have no automatic ”ping-function” in our system. That means that if for example a group member crashed, no one will realize this before they tries to send a message. This also implies that if the leader crashes and no messages is sent, there will be no way to connect to the group.

6.2 Security

Here there are great opportunities for development. For instance, theoreticly anyone could send I GroupChange-message and noone will know it the sender is the leader or not. It is the same way with all other messages types.

7 Scenarios

This section shows different scenarios that explains how the system works.

8 Tests

A Appendix

A.1 Project Specification

A.2 application.wadl

CodeSnippet 4 WADL

```
<application>
<doc jersey:generatedBy="Jersey: 1.0.3 04/16/2009 12:07 AM"/>
<resources base="http://mega.cs.umu.se:8080/umume-rest/">

<resource path="/search/{searchString}">
<param type="xs:string" style="template" name="searchString"/>
<method name="GET" id="searchForUsers">
<request>
<param type="xs:string" style="query" name="callback"/>
</request>

<response>
<representation mediaType="application/javascript"/>
</response>
</method>
<method name="GET" id="searchForUsers">
    <response>
    <representation mediaType="application/xml"/>
    <representation mediaType="application/json"/>
    </response>
</method>
</resource>

<resource path="/users/{uid}">
<param type="xs:string" style="template" name="uid"/>
<method name="GET" id="getUserXML">
<response>
<representation mediaType="application/xml"/>
<representation mediaType="application/json"/>
</response>
</method>
<method name="PUT" id="updateUser">
<request>
<param type="xs:string" style="query" name="ticket"/>
<param type="xs:string" style="query" name="service"/>
<representation mediaType="application/xml"/>
</request>
<response>
<representation mediaType="*/*"/>
</response>
</method>
</resource>

</resources>
</application>
```