Practical 5

Theme

- deploying a RESTful server using Eclipse and Tomcat
- using an XML schema to define and validate representations of objects in a RESTful service
- modifying a RESTful client-server chat system written in Java using JAX-RS and Jersey

Key concepts: RESTful remote interface, XSD XML schema, XML validation, Java servlet, servlet container, Eclipse schema editor, Eclipse Java EE perspective

5.1. Start up and essential configuration

a) *Log-in to Ubuntu and start a terminal.*

b) *Download and extract source code archives* `lab5-server.zip` *and* `lab5-client.zip` *within your* `CS3250-DS-1011` *folder.*

Beware, the extractor program sometimes creates an extra folder named after the zip file, so you may get `CS3250-DS-1011/lab5-server/lab5-server`. In that case, the right-most `lab5-server` should be used as the project folder in Eclipse.

c) *Start Eclipse 3.6 for JavaEE by typing* `eeclipse.sh &` *into the terminal or pressing* `Alt-F2` *and typing* `eeclipse.sh` *into the pop-up.*

Make sure you are using a workspace folder inside `CS3250-DS-1011` such as `CS3250-DS-1011/workspace`.

d) *Tell Eclipse about your Tomcat installation:*

(i) switch Eclipse to the Java EE perspective using the widget near the top right of the main Eclipse window (you may need to go via Other...)

(ii) select the Servers tab near the bottom of the workspace

(iii) right-click inside the panel within the Servers tab and select New > Server

(iv) in the dialogue that appears select Apache > Tomcat v6.0 server and click Next

(v) click Browse and navigate to the `/usr/local/tomcat` folder and press `Enter`

(vi) click Finish

5.2. JAX-RS server deployment using Tomcat.

a) *Open the given* `lab5-server` *code as a dynamic web project in Eclipse:*

(i) In the Java EE perspective, select File > New > Dynamic Web Project. The New Dynamic Web Project dialogue box will open.

(ii) In the Project name text field put `lab5-server` (**the name is important!**).

(iii) Untick the Use default location box and click Browse to navigate to the location of the decompressed source code eg. `CS3250-DS-1011/lab5-server`.

(iv) Ensure that the target runtime selected is Apache Tomcat v6.0.
If it is not available, it means that something went wrong with task 5.1.(d).

(v) Click Finish.

(vi) In a file manager (in *nautilus* or in the terminal) move the folder `rest` from folder `lab5-server/src2` into the folder `lab5-server/src`.

(vii) In Eclipse refresh the project lab5-sever by selecting it in the Project Explorer and then pressing `F5`.

The project should now appear with an error indicator. The errors will disappear after the following steps.

b) *Add the Jersey library to the project lab5-server:*

(i) Right-click on the project in the Project Explorer and select Build Path>Configure Build Path....

(ii) In the Libraries tab click on Add Library... button.

(iii) Click on User Library, press Next and click on User Libraries...

(iv) Click on New..., type `Jersey` and press OK.

(v) Click on Add JARs... and browse to `/usr/local/jersey` and select all three jar files present in this folder and press OK and then again OK.

(vi) Tick the box next to Jersey and click Finish and then OK.

There are still errors remaining in the project but fewer than before.

c) *Deploy the `lab5-server` project on the Tomcat server.*

Drag and drop the project name from the Project Explorer view onto the Tomcat server in the Servers view.

Alternatively, right-click on the Tomcat server in the Servers view and select Add and Remove Projects.... Then ensure that the `lab5-server` project is moved to the right hand side of the dialogue and click Finish.

d) *Start up the Tomcat server.*

(i) Right-click on the Tomcat server in the Servers view and select Debug. You will see the server's log messages in the Console view for several seconds.
Tip: When server runs in debugging mode, you can set breakpoints in the server Java code and trace the server as well as the client.

(ii) Wait until Eclipse shows the Servers view again.

e) *Download and deploy the `Message.xsd` schema on the server and test it.*

Copy the downloaded file `Message.xsd` to the `WebContent` folder. This will make it available at the URL:

```
http://localhost:8080/lab5-server/Message.xsd
```

You can test the web server by clicking on the above URL. This should open the schema in Firefox. If it does not, something is wrong with the deployment; try eg restarting the server and checking that the project name is `lab5-server`.

f) *Generate the `ChatMessage` class from the schema in file `Message.xsd`.*

Unfortunately, Eclipse does not seem to offer an easy way of doing this.

Therefore, switch to a terminal and type the following commands (adjusting directory names according to where you placed the project):

```
cd
cd CS3250-DS-1011/lab5-server/src
xjc ../WebContent/Message.xsd
```

g) *Refresh the Eclipse view of the project to discover the newly generated class.*
Right-click on the project name in the Project Explorer and select Refresh.
There should no longer be errors in the project.

h) *Add the Jersey* `ServletAdaptor` *to the project deployment descriptor.*

(i) In the Project Explorer, right-click on the project lab5-server and click Properties.
(ii) Select the Project Facets item, tick JAX-RS and click on the text Further configuration required near the bottom of the dialog.
(iii) Fill the field JAX-RS servlet class name with the following:

```
com.sun.jersey.server.impl.container.servlet.ServletAdaptor
```

i) *Make the Jersey library available to Tomcat:*

(i) Open the project lab5-server Properties dialog and select Deployment Assembly.
(ii) Click Add..., select Class Path Container and click Finish.
(iii) Select User Library, click Next, tick Jersey and press Finish, followed by OK.

j) *Check that the servlet has been deployed using a browser.*
The browser should display a blank screen without any error messages:

```
http://localhost:8080/lab5-server/jaxrs/topics/
```

5.3. Completing and using a Java client

a) *in Eclipse, create a new ordinary Java project (**NOT** a Dynamic Web Project) from existing sources in the folder* `lab5-client` *that was created when you unzipped the archive* `lab5-client.zip` *earlier.*
When prompted whether to switch to the Java perspective, answer No.

b) *Move the* `rest` *folder from folder* `lab5-client/src2` *into folder* `lab5-client/src` *and refresh the project in Eclipse.*

c) *Add the Jersey library to project* lab5-client.
This is done the same way as for the server project (see task 5.2.(b)) except that the Jersey library is already available to be ticked.

d) *Generate the* `ChatMessage` *class from the schema in file* `Message.xsd`.
As with the server, switch to the terminal and type the following commands (adjusting directory names according to where you placed the project):

```
cd
cd CS3250-DS-1011/lab5-client-java/src
xjc http://localhost:8080/lab5-server/Message.xsd
```

There should be no more errors in the project lab5-client at this point.

e) *In Eclipse, Execute* `TestSend` *twice and then* `TestFetch` *once, observing the console output. If you see a Tomcat error, scroll to the far right for the actual message.*

f) *Repeat task 5.2.(j) to see the two topics added to the server by the* `TestSend` *client program.*

g) *View the messages using a Web browser.*
Use URLs such as the following:

```
http://localhost:8080/lab5-server/jaxrs/topics/sport/messages/1
```

5.4. Modifying the server so that it understands message threads

a) *Open the* `Message.xsd` *under* `WebContent` *in the Eclipse* `lab5-server` *project, switch to the* **source** *view and observe its contents.*

b) *Switch the schema viewer to the* **design** *view and double-click on the* `ChatMessage` *element.*
You should see a diagram with two boxes connected by a line. This is a graphical view of the definition of the `ChatMessage` element in the schema.

c) *Add an optional* `parentMsg` *sub-element inside the* `ChatMessage` *element.*

(i) select the "all" symbol that groups the four elements together
(ii) select Add Element from the right-click menu
(iii) change the name of the element to `replying` (first select the name by clicking on it, possibly twice)
(iv) change the type of the element to **`int`** using the pop-up menu
(v) set the multiplicity to [0..1] using the right-click menu

d) *Repeat tasks 5.2.(e) and (f) and 5.3.(b) to update the server's* `ChatMessage` *class according to the new schema.*
Notice that this automatically redeploys the project on Tomcat — all topics and messages will disappear when repeating task 5.2.(j).

e) *Add a new method to the server proxy class to be able to send a reply message.*
The method should be called `newReply` and it should differ from `newMessage` only in having one extra **`int`** parameter, which is assigned to the newly added `parentMsg` field of a `ChatMessage`.

f) *Add an instruction to program* `TestSend` *to send a reply to one of the messages:*
Add the following lines to the `TestSend` program on line 23 (ie just after sending the first message to topic1):

```
        sendReply(server, topic1, "hooray!",1);
        sendReply(server, topic1, "boo!",1);
```

and also write the method `sendReply` as a clone of the existing method `sendMessage` with the additional `parentMsg` parameter.

> **Reflection.** This completes the modification of the client in order to make use of the new ability of the XML serialisable message objects to specify threading information. The server will accept and store the thread information but will not offer this information any advanced manner — if the clients want to reconstruct a thread, they need to download all messages and link them using their `parentMsg` fields. The following tasks improve the way server can provide information about message threads.

g) *Add a method* `getReplies` *to the class* `Messages`.

The method should look as follows except TODO should be replaced by an appropriate condition:

```
public List<Integer> getRepliesToMsg(int msgNo)
{
    List<Integer> result = new LinkedList<Integer>();

    synchronized(messages)
    {
        for (ChatMessage msg : messages)
        {
            Integer parentMsg = msg.getParentMsg();
            if ( TODO ) // exercise - beware parentMsg can be null
            {
                result.add(msg.getSerialNumber());
            }
        }
    }
    return result;
}
```

h) *Annotate and modify the above method so that the server can use it to answer queries such as:*

`http://localhost:8080/lab5-server/jaxrs/topics/sport/messages/1/replies`

The response should be a single line of text with the numbers of the children messages listed one after another separated by a single space (eg "4 6 10").

You can test your method by opening the above URL in Firefox. If you get an exception, you can trace the server by setting breakpoints in it and reloading the browser's view.

i) *(Optional) Add a method* `getReplies` *the Java client's proxy class and test it.*

The method should have the following form:

```
public List<Integer> getReplies(String topic, int msgNo)
throws UniformInterfaceException
{
    TODO // exercise - mostly similar to getTopics and getMessage
}
```

Add a call to this method in `TestFetch` and run some tests with the help of `TestSend`.