Distributed Systems – Assignment 2

Robin Guldener
ETH ID 11-930-369
robing@student.ethz.ch

Nico Previtali
ETH ID 11-926-433
pnico@student.ethz.ch

Lukas Bischofberger
ETH ID 11-915-907
lukasbi@student.ethz.ch

ABSTRACT

For the implementation of this assignment we made use of the two lab provided HTC desire number 25 and 37. We were able to implement all the assigments listed on the assignment sheet and to our best knowledge all services are working flawlessly. Our enhancements include the graphing of temperatures using the Google REST Charts API as well as multithreaded request management on the server.

1. INTRODUCTION

Web Services allow for the easy inclusion of services provided by any entity over the internet. One of the main advantages of web services are that they allow for a simple implementation of the service oriented architecture as they abstract and implement many of the common networking problems encountered in implementing such an architecture. Additionally web services have the advantage of making it dead simple to include functionality implemented by 3rd parties by simply calling their services with the required parameters.

RESTful Web Services use the HTTP verbs to specify the action to perform on an object identified by the service URL. The key advantage of this method over WS-* Web Services is their simplicity due to the implicit encoding of the action in the HTTP verb. However as the same time this is also one of their main drawbacks as they are somewhat more limited in terms of functionality and extensibility than WS-* Web Services.

WS-* Web Services describes a class of Web Services which make use of the WSDL protocol to advertise their capabilities to other services and usually implement the SOAP protocol for actual requests to the service (however, this is not a requirement, WSDL allows for any arbitrary request protocol to be specified). Their main advantage is the wide range of configuration and extensibility options as well as the automated service discovery. This flexibility comes with a price though and thus usually WS-* services are usually considerably more complex than their REST equivalents.

2. RESTFUL WEB SERVICES

In this assignment we retrieve the temperature of Spot 1 [3], which is obviously served by a RESTful Web Service. The application consists of a single Main Activity, which provides four buttons to invoke the RESTful Web Service. The first and second button will make HTTP requests using different libraries. The third and fourth button will make use of the `content negotitation` mechanism. The last button displays a visualization using cloud services explained in-depth in section Cloud Services. All HTTP requests are invoked asynchronously and are handled by the Main Activity using the AsyncTask class.

Since Android 3.0 attempts to perform a networking operation must not be handled on the GUI thread which is simultaneously the main thread of the application. This makes perfect sense because network operations include blocking methods. Blocking methods are those method which block the executing thread until the operation finished. One famous example of a blocking method is InputStream's `read()` method which blocks until all data from InputStream has been read completely. Networking operations in Java use a lot of InputStreams. Therefore it is necesserary to handle networking operations in a seperate thread to avoid blocking the main thread. If the main thread would be blocked, user input can not be made. For this reason Android provides an `AsyncTask` class.

```
1 // Our worker which performs networking operations
2 private class RESTWorker extends AsyncTask<String,
      Void, String> {
3   ...
4 }
```

Listing 1: Using AsyncTask

The AysncTask class provides several important methods, such as `doInBackground`, `onPostExecute`. `doInBackground` performs all operations asynchronous in background on a seperate thread. We use this method to do all the networking operations such as sending HTTP GET requests. One of the disadvantages of AsyncTask is that you cannot simply update GUI elements from within `doInBackrgound` since it is not synchronized with the main thread (= UI thread). For this reason, AsyncTask provides a method `onPostExecute` which is synchronized to the main thread. In this method one can easily update UI elements without having to worry about the synchronization.

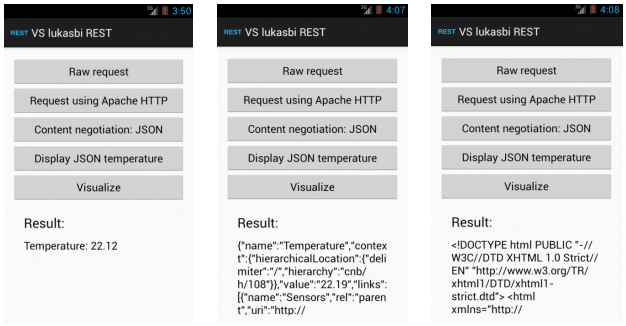
```
1   @Override
2     protected void onPostExecute(String result) {
3       super.onPostExecute(result);
4       // this method is synchronized with the user
            main thread (handled by Android), so no
            problems occur...
5       if (result != null) {
6         switch (mode) {
7           default:
8             // clear chart and write REST response
9             imgChart.setImageResource(R.drawable.
                  blank);
10            if (retrieveWithLabelAndWrite)
                  writeResult(result);
11            break;
12          case RESTWorker.MODE_CHART:
13            // clear REST response and display chart
14            writeResult("");
15            imgChart.setImageBitmap(bmp);
16            break;
17        }
18      }
19    }
```

Listing 2: Updating UI elements

3. WS-* WEB SERVICES

As a first task for the WS-* web services part we had to think about how we would implement invoking a WS-* web service only using the `java.net` library. The following steps would have to be pursued:

1. Get the web service's description by obtaining its WSDL file from a predefined address. This could be down-



(a) Extracted value from JSON repsonse (b) Raw JSON response (c) Raw HTML response

Figure 1: GUI of Main Activity showing different responses

loaded with an HTTP GET request utilizing the `URLConnection` class like thus:

```
// Assuming url is a String holding the url for the WSDL
URLConnection connection = new URL(url).openConnection();
InputStream stream = connection.getInputStream();
```

Listing 3: Getting data from a webserver using HTTP GET

2. Read the entire response from the input stream and parse the response using an XML library (or write your own). From the WSDL extract the URL of the web service you wish to call by looking at the `PortType` section and finding the `operation` which you would like to perform. Also be sure to check the appropriate input and output configuration as well as the actual transport protocol (usually SOAP) for your `operation` from the `binding` section

3. Prepare the actual request, here we will use SOAP. For SOAP this means creating the request XML by defining the SOAP envelope and specifying the header and body information required by the `operation` that is to be queried/called. In this example we a re calling one of the web services provided by the VSLAB and request a list of all discovered SunSPOTs:

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:getDiscoveredSpots xmlns:ns2="http://webservices.vslecture.vs.inf.ethz.ch/"/>
    </S:Body>
</S:Envelope>
```

Listing 4: SOAP Request

4. Now that the request is ready it has to be actually transferred to the endpoint. For this we establish a new `URLConnection` with the endpoint and send our SOAP request using the HTTP POST keyword to the endpoint like this:

```
// Assuming url is a String holding the url for the web service endpoint & soap is the SOAP XML request string
URLConnection connection = new URL(url).openConnection();
connection.setDoOutput(true); // Triggers POST.
connection.setRequestProperty("Content-Type", "application/xml+soap;charset=" + charset);
OutputStream output = connection.getOutputStream();
try {
     output.write(soap.getBytes(charset));
} finally {
     try { output.close(); } catch (IOException logOrIgnore) {}
}
InputStream response = connection.getInputStream();
```

Listing 5: POST-ing a SOAP request to the endpoint

5. Finally we parse the XML from the response to obtain the data we asked for. Done!

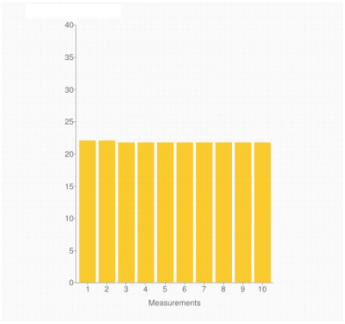


Figure 2: Temperature chart

For the second part of the WS-* Web Services we implemented a simple Android application that downloads the latest temperature from Spot3 using SOAP requests. Our application consists of only two classes, one subclass of Activity, which implements our MainActivity, and one subclass of AsyncTask, which perfoms all the required networking in the background. This second class was largely recycled from Task 1 and adapted to fit the use of the WS-* Web Services.

To be able to perform requests against a Web Service one must first know which services and actions exist. This part is covered by WSDL files and we retrieved the WSDL file corresponding to the Web Service for this task from the lecture website. From the WSDL we were able to determine the service endpoint as well as the action name and the parameters to pass in. With this information we were able to actually build the requests required, in this case SOAP requests. The actual creation of the SOAP request is performed by the ksoap2-androidlibrary[2] which takes care of forming the appropriate XML strings.

The main advantage of SOAP using XML instead of a binary format is that it is completely platform independent and allows for the encoding of arbitrary objects. The process of transforming these objects back into platform specific objects (ie. Java objects if the final destination platform is Java) is commonly referred to as unmarshalling.

4. CLOUD SERVICES

The measured temperatures of Spot 1 are visualized with a bar chart (cf. Figure 2(a)). The chart is generated by sending a HTTP POST request to the Google Chart Tools Image cloud service [1]. The main obstacle we encountered was to scale the data to make it conform with the scaling of the axes, but once this hurdle was overcome the implementation of this visualization method proofed straight forward thanks to the provided cloud service. It was certainly a lot less laborious than the implementation of the visualization in assignment 1.

5. YOUR PHONE AS A SERVER

- The implementation of the server relies on an Activity to start and stop the server and some background threads to handle network action and multitasking. The idea was to let the communication with the sensors run in a separate service, but we didn't manage to easily bind the sensor and actuation service to the ServerHelperThread. So we decided to place the acquiring of the sensor data and the actuations in the same thread which answers the client's request.

- Java code for accepting multiple connections: Each time the socket is bound to an incoming connection, it gets assigned to a helper thread which computes the response. The socket is then free again for a new connection.

```
while(!Thread.currentThread().isInterrupted())
    {
  try {
    socket = serverSocket.accept();

    //start a new thread to handle connection
    ServerHelperThread sth = new
        ServerHelperThread(socket, mContext);
    new Thread(sth).start();
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Listing 6: Delegating connection response handling to ServerHelperThread

- We implemented both sensing and actuation. For the sensing there is the possibility to see the list of the sensors and also to see the details of a chosen specific sensor.
  Actuation was implemented for vibrating the phone for a specified amount of time (by the slider) and for starting some sound on the phone.
  We implemented both sensing and actuation by the GET method. The GET method is easier to implement and the data is easier to acquire. Secondly we didn't manage to safely get the data of the POST method, apparently the browser didn't send an end of the header and the InputStreamReader was waiting for more data, so we stuck with the GET method.

6. CONCLUSION

One of our key insights from this assignment was that working with REST services proofed much more convenient than working with full-scale WS-* services. Such an outcome was expected however due to the characteristics of both methods as explained in the introduction. Furthermore we were pleasantly surprised by the ease of use even WS-* services provided when used in combination with a library like ksoap2[2]. On the side of the webserver the key challenge was to access the sensor data and to analyze the issue with the never ending POST requests, however, in the end both problems could be resolved by slightly adapting our design and we are confident about our solutions. Overall we feel that we have acquired a good experience working with Android networking and look forward to apply this knowledge in future assignments.

7. REFERENCES

[1] Google Chart Tools Image. `https://developers.google.com/chart/image/`. Accessed on 26 Oct 2013.

[2] ksoap2-android library.

[3] Temperature of Spot 1. `http://vslab.inf.ethz.ch:8081/sunspots/Spot1/sensors/temperature`. Accessed on 26 Oct 2013.