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

Concisely state (i) which Android device you used, (ii) which tasks you completed and which are working correctly or limited, and (iii) what your specific enhancements are.

1. INTRODUCTION

Use the introduction for background information on the assignment. See your assignment sheet for specific questions on the topic that you have to answer in this section. Use references such as books [3], papers and theses [5], or specifications [4] whenever available. Web sites for documentation [1], tutorials, etc. are a special case. In a thesis, you would put them as footnotes. At this stage, however, you will only have a few "real references," so we put the Web sites into the bibliography. Cite every source you used throughout the assignment.

For Assignment 2, please give a short overview of Web Services, especially their benefits. Compare RESTful against WS-* Web Services by listing some advantages and disadvantages of both concepts.

2. RESTFUL WEB SERVICES

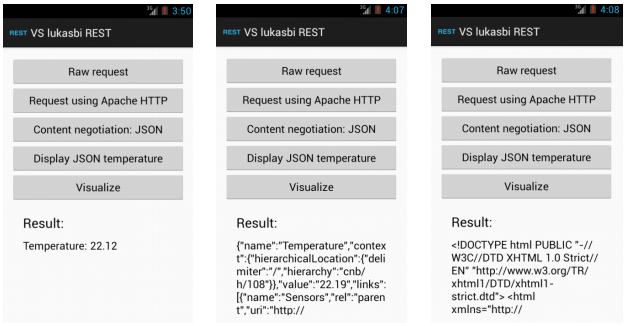
- Describe shortly, how you designed your application to implement this Task. Which Android core elements did you use (e.g., Activity, Service, AsyncTask, Intent)?
- Explain why it is beneficial (or even required, since Android 3.0), to off-load networking tasks to AsyncTask. **Hint:** Explain what *blocking*-methods are in the context of handling I/O Streams.
- Explain how you make use of the different methods provided by AsyncTask, such as `doInBackground`, `onPreExecute`, `onProgressUpdate`, `publishProgress` or `onPostExecute`.

In this assignment we use the temperature of Spot 1 [2], which is obviously a RESTful Web Service. The application consists of a single Activity. This Activity 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 asynchronous using the AsyncTask class.

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



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

Figure 1: GUI of Main Activity showing different responses

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

Listing 1: 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 2: 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 3: 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!

- Describe shortly, how you designed your application to implement this Task. Did you reuse elements from Task 1?
- What are the roles of WSDL files and SOAP requests in WS-* Web Services?
- Explain why SOAP messages are exchanged in XML format and what unmarshalling to platform specific objects means.

4. CLOUD SERVICES

- Which diagram type did you choose? You can show a screen shot and describe your custom functionalities, if you have implemented any.

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.
- 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();
  }
}
```

- 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 kind of music 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. ENHANCEMENTS

This is usually the final part of an assignment. Here you are free to describe what you did, however, stick to a concise, scientific writing style.

7. CONCLUSION

Give an overall conclusion that summarizes the main challenges you encountered and your lessons learned.

8. REFERENCES

[1] Services: Sending Notifications to the User. `http://developer.android.com/guide/components/services.html#Notifications`. Accessed on 29 Aug 2013.

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

[3] E. Burnette. *Hello, Android: introducing Google's mobile development platform*. Pragmatic Bookshelf, 3 edition, 2010.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[5] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, UC Irvine, 2000.