Distributed Systems – Assignment 2

Frederik Rothenberger
frothenb
frothenb@student.ethz.ch

Simon Wehrli
siwehrli
siwehrli@student.ethz.ch

ABSTRACT
This report describes the two Android apps programmed by Simon Wehrli and Frederik Rothenberger. The rest-App demonstrates the different approaches to get sensor data from the *SunSpot* server. It also uses a Web Service itself for drawing charts. In the second app (ws) we implemented the same basic functionality as in Task 1 with WS-* architecture using SOAP and working with WSDL. Finally, the third app (server) shows the server side of a Web Service with appliance of the RESTful architecture.

INTRODUCTION
All apps are divided into two subcomponents. A graphical user interface let the user choose his preferred settings. A second subcomponent is the actual communication to servers (e.g. Web Services) resp. starting a server on the device. This logic is time intensive and hence moved out to an asynchronous task, which update view components via a callback method. We first present the different components with a focus on important design decisions and then discuss the problems encountered during the development process.

TASK 1&4

VS_siwehrli_rest (App 1)

Main Activity
This activity holds all content produced during solving task 1. The implementation is really straight forward. Three methods implement the different approaches to get basically the same result: the current value of the temperature sensor on spot 1. All versions use the URI *http://vslab.inf.ethz.ch:8081/sunspots/Spot1/sensors/temperature* to send the request to, but the request is made by

- hand, meaning http lines are built up with a *StringBuilder*
- the apache library, using a *HttpGet* object
- the apache library, but additionally setting the http header "Accept" to "application/json". The XML parsing is then done via a JSON object

Chart Activity
This activity holds all content produced during solving task 4. The design is slightly more complicated here. Since we decided to show a time to temperature line chart periodically updated every second, we have a lot of traffic which may be delayed due to network delay and the android OS, which may interrupt our application process. This let us choose to do the creation of the diagram in an asynchronous task (*AsyncTask*), which fetches the actual data from the *SunSpot* server, add the data point to the data set, uses the Google Chart Tool to create the updated line chart and publishes it via a callback method in a *ImageView*.

We solved the timing problem by making the time points on the x-axis adaptive to the time points when data was actually measured. This means basically to store the time when retrieving temperature values from the server.

TASK 2

On how to invoke a WS-* web service (in 8 easy steps)
- download the WSDL file from the lookup server
- download scheme *.xsd* file (if one is described in the WSDL file)
- parse the files in order to get the following information:
  - the name of the method we want to invoke
  - the namespace of the method
  - the URL of the server providing this service
  - the arguments needed (including their type)
  - the type of the return value (if any)
- build a soap object containing the arguments, the namespace and the method name
- wrap the soap object into a soap envelope
- send the envelope to the server URL (e.g. over http)
- receive response
- parse the response to get out the return value

The last step concludes the WS-* method invocation.

VS_siwehrli_ws (App 2)

Main Activity
The main activity of this Android app implements the following required functionalities:

- Sending a Soap request in order to receive a temperature value
- Displaying said temperature value on screen
- Displaying the raw XML response which was received via http describing the Soap object which contains the result value to the previously sent Soap request
- Displaying the raw Soap response on screen

In order to do this, the activity consists of one button and six *TextViews* (three to display the temperature, raw XML and raw Soap response, the other three to display static text describing the content of the former three). The button triggers the WS-* service invocation and leads to an update of the currently displayed values. If the WS-* service invocation fails for some reason, the

TextViews change content accordingly and display an error message.

As suggested in the task description, we used the kSOAP2 library for Android to handle all the Soap interaction with the server. Since this uses the network and therefore a lot of time, we decided to implement the networking part in an asynchronous task. The implementation of the asynchronous task is done via extending the standard Android *AsyncTask* and overwriting the *doInBackground* method.

TASK 5

VS_siwehrli_server (App 3)

Main Activity

The main activity is the only component of this app implementing a server which provides information about the phone's sensors and lets you vibrate the phone and play a sound remotely. It consists of a single *TextView* which serves as a server log, aggregating information about all actions taken. More specifically, every creation of an *AsyncTask* as well as information on every incoming request gets displayed in this *TextView*. Additionally the html sent out to clients gets logged there too.

This service is based on RESTful architecture and knows three states (no action / sound / vibrate) represented by the following URLs: "phone_ip:8081/", "phone_ip:8081/sound" and "phone_ip:8081/vibrate", where *phone_ip* is a placeholder for the phone's IP address.

As represented in the URLs above, the server responds to requests to the port 8081. All requests are served with a html containing a table with sensor information and two URLs which describe the vibrate and the play sound request. Depending on whether the request contained the keywords "vibrate" or "sound" the phone either vibrates, plays a sound or takes no action.

In order to keep response times short and allow multiple clients to be handled, every incoming connection starts a new *AyncTask* (again inherited from the Android *AsyncTask* with an overwritten *doInBackground* method, as in App 2). In this *AsyncTask* the request gets parsed, activating actuators as requested, and the html table for the response is constructed. This table is generated dynamically based on the number of sensors and filled in with the respective values.

Because the server needs to be available for new connections at all times, the task of listening to new requests and accepting them is also done in a separate *AsyncTask* dedicated to only that.

GENERAL DIFFICULTIES

In task 4 we had some problems with the Google Chart API (there may be a reason that it's deprecated). Even though it's powerful, it has its limitations and the request gets difficult to read if you customize your chart to a further extend. We may have reached the point when we prefer thinking in objects and do not translate this down to a string.

Another source of minor problems was the implementation of the http protocol by hand. Specifically, the exact format concerning the number and position of all the required newline and carriage return symbols were initially not taken seriously enough, giving rise to a rather unnecessary debugging session.

WHO DID WHAT

Frederik coded Task 2 and 5 while Simon did Task 1 and 4. Task 3 was answered together. All important algorithms were discussed and evaluated as a team.

CONCLUSION

We know had much fewer problems to get started with coding than in the first assignment and we could concentrate on the interesting questions. And it made more fun due to that fact!