Assignment 2

Start:	October 6, 2014
End:	October 20, 2014

Objectives

In this assignment you will learn how to develop distributed Web applications using the two different paradigms that you have seen in the lecture: *REST* and *WS-**. You will make use of these paradigms when implementing a mobile phone application that gathers data from services provided by wireless sensor nodes (SunSPOTs[1]) over the Web. The SunSPOTs expose their services (e.g., temperature sensor, ambient light sensor, etc.) through two different interfaces: REST-based and WS*-based (see Figure 1).

- **Representational State Transfer (REST)** is a style of software architecture for implementing Resource-Oriented Architectures (ROAs). HTTP (1.1)[2], the application protocol of the Web, represents an implementation of the REST principles. Distributed RESTful applications can be developed using the HTTP protocol as a universal interface for interacting with resources on the Web. Such applications make use of HTTP verbs (GET, POST, PUT, DELETE, etc.) and mechanisms (e.g., URIs and header options). Furthermore, REST defines how to serve different formats (e.g., HTML, JSON, or XML) for a given resource depending on the clients needs ("content negotiation").
- **WS-* services**, sometimes called "Big Web services," describe a set of XML-based standards (e.g., WSDL, SOAP, and UDDI) that can be used to implement Service-Oriented Architectures (SOAs). Rather than using HTTP as an application protocol, WS-* services use it as a transport protocol and define a number of additional layers to encapsulate distributed services.

This assignment comes predefined interfaces for the tasks 1 and 2. Besides this assignment sheet, you will find valuable hints on the implementation with the Javadoc comments given in the provided code.

With this assignment you can gain 10 points out of the total 45.

Task	Points
1	2
2	2
3	1
4	3
5	2
Total	10

1 Experimenting with RESTful Web Services (2 Points)

The SunSPOTs *Spot1* and *Spot2* deliver the sensor values through a RESTful interface. Open your browser and navigate to `http://vslab.inf.ethz.ch:8081/sunspots/`. From there you

[1] `http://sunspotworld.com/`
[2] `http://http://tools.ietf.org/html/rfc2616`

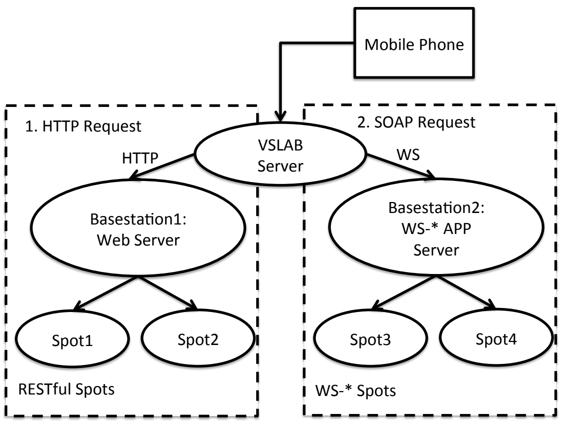

Figure 1: System setup for Assignment 2: Sensor nodes can be accessed either directly through HTTP or using SOAP messages.

can explore the HTML representation of the RESTful SunSPOTs. Browse and experiment with the SunSPOTs *Spot1* and *Spot2* and look at the sensor values they offer through this Web interface (e.g., temperature, light, and acceleration). You could also use the Spots' actuators to switch on or off their LEDs and choose colors. Your task is to write an Android application that requests the temperature measurement from *Spot1* and displays the value on the screen.

Activity

Create a new Android project called `vs-`*`nethz`*`-client` and set the application name to `VS` *`nethz`* `Client` (where *nethz* is the group leader's nETHZ account name).

a) Your Main Activity should provide three buttons for the user (one for each task). Each button should start a new Activity.

b) Create a new Activity for the REST client. This Activity has to implement the interface `ch.ethz.inf.vs.a2.sensor.SensorListener` so that the Activity can be informed upon reception of the temperature value.

Raw HTTP Request

RESTful Web services can be invoked using the HTTP protocol. In this assignment, we start by making a "raw" HTTP request.

a) Implement the interface `ch.ethz.inf.vs.a2.http.HttpSocket`. which opens a TCP connection to a destination with given host address and destination port, and sends a String representation of an HTTP request to that host.
Hints:

- Use the `Socket` class in the `java.net` package. To send and receive data use the `getInputStream()` and `getOutputStream()` methods of the `Socket` class and read and write to the corresponding `InputStream` and `OutputStream`.
- Remember to *flush* when using `PrintWriter`.

b) Implement the interface `ch.ethz.inf.vs.a2.http.Requester` and make use of your `HttpSocket` implementation from a).

c) Implement the interface `ch.ethz.inf.vs.a2.http.HttpRawRequest`, which generates the raw HTTP GET request, to obtain the temperature information of *Spot1*.
Hints:

- For the HTTP protocol have a look at `http://www.elektronik-kompendium.de/sites/net/0902231.htm`, search for other how-tos, or look directly into the standard (RFC 2616).
- Take care to correctly implement the HTTP protocol: HTTP headers are required to have a *carriage return and newline* at the end of each line. *println(...)*, however, uses the *line.separator* property, which is just a *newline* for Android.
- Use the provided unit tests to check your implementation.

d) Write a class RawHttpSensor that extends the abstract class `ch.ethz.inf.vs.a2.sensor.AbstractSensor`. This sensor retrieves the temperature value using the raw request. Make use of your `Requester` implementation in b), and your `HttpRawRequest` implementation in c).

e) Create a new RawHttpSensor object, register your Activity as a listener and invoke a request to retrieve a new temperature value from *Spot1*. Display the retrieved temperature value.

HTTP Library

Now we use the Apache HTTP Client library (`org.apache.http`) for sending requests and receiving responses.

a) Write a new implementation of the interface `ch.ethz.inf.vs.a2.http.Requester`, but this time make use of `org.apache.http.client.HttpClient`.

b) Write a class HtmlSensor that extends `ch.ethz.inf.vs.a2.sensor.AbstractSensor`. Use the `org.apache.http.client.methods.HttpGet` class to create your request and use your implementation from a) to retrieve the value.
Hint:

- The RawHttpSensor and the HtmlSensor share the same parseResponse(String response) method, since both have to parse the `text/html` representation of the response. Think of a good class hierarchy to avoid code duplication.

c) Create a new HtmlSensor object, register your Activity as a listener and invoke a request to retrieve a new temperature value from *Spot1*. Display the retrieved temperature value.

JSON Representation

So far, we only got HTML responses back from the SunSPOTs. As mentioned before, a RESTful service can offer several representations of the same resource. To get a different representation that is more appropriate for machine-to-machine communication than HTML, we use the HTTP content negotiation mechanism: Set the `Accept` header of your HTTP request to `application/json`. Instead of an HTML page, the Web server will now return a JSON object containing the temperature information. JSON is a lightweight version of XML, which is often used in Web mashups and RESTful interfaces because it can directly be evaluated as JavaScript.

a) Write a class JsonSensor that extends `ch.ethz.inf.vs.a2.sensor.AbstractSensor`. Make good reuse of your code above. You can parse the response using the JSON library that Android already provides (`org.json.*`).

b) Create a new JsonSensor object, register your Activity as a listener and invoke a request to retrieve a new temperature value from *Spot1*. Display the retrieved temperature value.

2 Experimenting with WS-* Web Services (2 Points)

While *Spot1* and *Spot2* offer a RESTful interface, *Spot3* and *Spot4* can be accessed using their WS-* Web Service interface. Open your browser and navigating to: `http://vslab.inf.ethz.ch:8080/SunSPOTWebServices/SunSPOTWebservice`. From there you can access the WSDL (Web Services Description Language) description of the offered functionality. Have a closer look at the WSDL interface and try to understand its content and what it provides as you did for the RESTful version. Finally, use the HTML interface at `http://vslab.inf.ethz.ch:8080/SunSPOTWebServices/SunSPOTWebservice?Tester` to test the Web Service in your browser.
Hint: To fully understand the WSDL description, also look at the *schema* specified in the WSDL file.

Activity

Create a new Activity for the SOAP client. This Activity has to implement the interface `ch.ethz.inf.vs.a2.sensor.SensorListener` so that the Activity can be informed upon reception of the temperature value. Remember to adjust your main Activity so that you can access this new Activity.

Manual SOAP Invocation

a) Write a class XmlSensor that extends `ch.ethz.inf.vs.a2.sensor.AbstractSensor`. Use the `org.apache.http.client.methods.HttpPost` class to create your SOAP request. You can use the XML templates provided by the Tester to create your requests.
Hint: You can use Netbeans and Wireshark to inspect an exemplary SOAP request.

b) For the implementation of the parseResponse(String response) method, make use of the XmlPullParser, which is also delivered by Android.

c) Create a new XmlSensor object, register your Activity as a listener and invoke a request to retrieve a new temperature value from *Spot3*. Display the retrieved temperature value.

SOAP Library

a) Write a class SoapSensor that extends `ch.ethz.inf.vs.a2.sensor.AbstractSensor`. Make use of the ksoap-2 library (`http://code.google.com/p/ksoap2-android/wiki/Links`) and use the XML template provided by the Tester, to figure out which parameters to set in the SOAPObject.

b) Create a new SoapSensor object, register your Activity as a listener and invoke a request to retrieve a new temperature value from *Spot3*. Display the retrieved temperature value.

3 Analyzing Your Network Traffic (1 Point)

Use the Android Emulator (`http://developer.android.com/tools/devices/emulator.html`) and Wireshark (`https://www.wireshark.org/`) to analyze your network traffic. For each sensor implementation (i.e. `RawHttpSensor`, `HtmlSensor`, `JsonSensor`, `XmlSensor`, `SoapSensor`) capture one pair of request and its response and save them in a text file with the filename `<SensorName>_Wireshark.txt`.

4 Your Phone as a Server (3 Points)

Similar to the SunSPOT sensor node, your phone can also provide RESTful Web Services for its sensing and actuation functionality. Your task is to make two sensors and two actuators available through a REST server running on your mobile phone.

a) Create a new Android project called `vs-`*`nethz`*`-server`. Set the application name to `VS` *`nethz`* `Server` and the package name to `ch.ethz.inf.vs.a2.server`.

b) Write a class that extends a Service such that your server can run in the background. The user should have the possibility to start and stop the server.

c) Use the `ServerSocket` to listen on a specified port, e.g, 8081 (you will have to pick a port greater than 1024). Wait for and `accept` incoming connections and then handle the requests, which works similar to your `Socket` client in Task 1.
Hints:
- First parse the received requests and differentiate the HTTP Method (e.g., GET or PUT).
- Second, determine the requested resource (e.g., sensor or vibration motor)
- Third, extract content that might have been transmitted (e.g., vibration pattern)
- Call a function that does the correct logic according to the request (e.g., getting the sensor value or turning on vibration). Reuse your implementation to access sensors and actuators from Assignment 1, Task 1.
- Formulate an HTTP response (take also care of handling errors such as unsupported requests, wrong formats, etc.)
- Your server does not have to be RFC-compliant. Implement a minimal version that works with your browser (e.g., most header options can be ignored).

d) Run the Android Wi-Fi Hotspot app and connect your laptop to the phone. Use your browser on the laptop to test the functionality of your Android Web server.

e) Add multi-threading to your Web server: The server should be able to accept and answer requests form several clients simultaneously.

5 Mini-Test (2 Points)

1. HTTP Protocol Version 1.1
   a) How is a *Request-Line* defined according to RFC 2616?
   b) How does a (minimal) request look like if you want to get the root document from the host *192.168.1.1* at port 8080?

c) Name two REST features that can be implemented using the HTTP header fields.

2. Network I/O

   (a) Name the two important classes to implement a client and a server, which communicate via TCP. Describe how these classes are used in the client and in the server implementation.

   (b) Java's library `java.io` provides classes and interfaces for stream-oriented I/O. In order to work with TCP connections, the class `Socket` provides methods to get an `InputStream` or an `OutputStream`. Use `InputStream` and `OutputStream` to explain *blocking* behavior. State, which of the methods from `InputStream` and `OutputStream` are showing blocking behavior.

3. Representational State Transfer
   Right or wrong? State for the following statements, whether they are correct or incorrect. If incorrect, please state why you think the statement is incorrect.

   a) REST is a protocol and can be used as an alternative to SOAP.

   b) Stateless means, the server stores client-context so the client does not have to send context information in each request.

   c) The HTTP methods POST, GET, PUT, DELETE correspond to the operations CREATE, READ, UPDATE, DELETE, to manipulate resources.

   d) REST defines JSON as data representation.

4. WS-* services

   a) Which document holds information about the definition of the SunSPOTWebservice? How can this document be retrieved?

   b) Where can the type definition of the elements `getSpot` and `getSpotResponse` be found? Give the element definitions for both `getSpot` and `getSpotResponse`.

   c) Imagine, the SunSPOTWebService would be implemented using SMTP as transport protocol. Where in the WSDL file would you declare the transport protocol? How does this affect the `soap:address` in the `service` definition?

5. Android Emulator Networking

   a) What IP address is assigned to an emulated device? Why is it the same address even if multiple emulated instances run on the same development machine?

   b) To whom does a call on an emulated instance to 127.0.0.1 refer?

   c) By which IP address can the development machine be reached from an emulated device?

   d) How can the development machine connect to a port on the emulated device?

Deliverables

The following deliverables have to be submitted by **9:00am, 20 October 2014**:

1. **code.zip** You should create a zip file containing the Eclipse projects created in this assignment. The projects should have been tested both on the mobile phone and on the emulator. The code must compile on our machines as well, so always use relative paths if you add external libraries to your project. Do not forget to include those libraries in the zip file. Please use UTF-8 encoding for your documents and avoid special characters like umlauts.

2. **answers.pdf** Your answers to the mini-test in **PDF** format.
3. **wireshark.zip** Create a zip archive that contains the Wireshark exports for all sensor types `<SensorName>_Wireshark.txt` from task 3.

Submission

The deliverables must be uploaded through:
`https://www.vs.inf.ethz.ch/edu/vs/submissions/`
The group leader can upload the files, and other group members have to verify in the online system that they agree with the submission. Use your nethz accounts to log in. The submission script will not allow you to submit any part of this exercise after the deadline. However, you can re-submit as many times as you like until the deadline.