Assignment 2

Start:	15 October 2010
End:	29 October 2010

In this assignment you will learn to develop distributed Web applications, called Web Services[1], using two different paradigms: **REST** and **WS-***.

The 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, is an implementation of the REST principles. Distributed RESTful applications can be developed using the HTTP protocol using as a universal interface for interacting with services (called "resources") on the Web. Such applications make use of HTTP verbs (GET, POST, PUT, DELETE, etc.) and mechanisms (e.g., URLs, HTTP Content Negotiation). Furthermore, REST defines how to serve different formats (e.g., HTML, JSON, XML) for a given resource depending on the clients needs.

WS-* services, sometimes called "Big Web services", describe a set of XML-based standards (e.g., WSDL, SOAP, 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.

In this assignment you will implement a mobile phone application that gathers data from services provided over the Web by Wireless Sensor Nodes (SunSPOTs[3]). The SunSPOTs expose their services (e.g., temperature, light, etc.) through two different interfaces: REST based and WS*- based. Figure 1 illustrates the setup. All subtasks should be accessible through separate buttons.

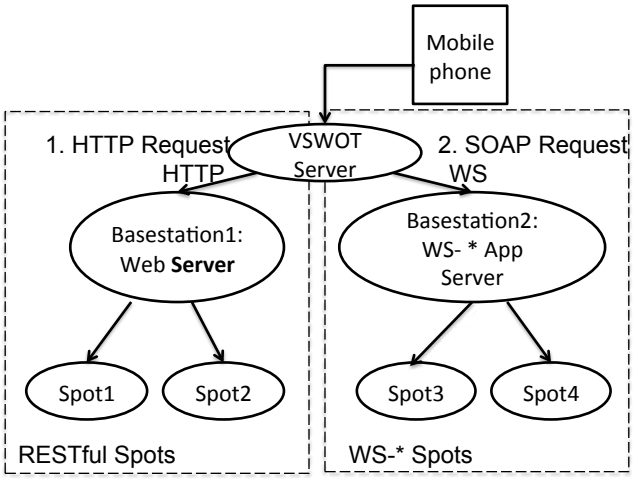

Figure 1: The mobile phone accesses the sensor values through HTTP or SOAP WS-* requests.

1 Experimenting with RESTful Web Services

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

[1] `http://en.wikipedia.org/wiki/Web_service`
[2] `http://www.w3.org/Protocols/rfc2616/rfc2616.html`
[3] `http://sunspotworld.com/`

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, acceleration). You will write an Android application that requests the temperature measurement from *Spot1* and displays the value on the screen. Implement each subtask in a new function and provide an additional button to perform the action.

What to do

- RESTful Web services can be invoked using the HTTP protocol. Several different libraries can be used to create HTTP calls. In this assignment, we start by making a "raw" request without the use of any external library. Open a TCP connection to *vswot.inf.ethz.ch* (port 8081) and make an HTTP GET request to obtain the temperature information of *Spot1*. Display the raw response on the screen.
  Hint: Use the `Socket` class in the `java.net` library. 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`. For the HTTP protocol have a look at `http://www.elektronik-kompendium.de/sites/net/0902231.htm` or google for other howtos.

- Now use the Apache HTTP Client library (`import org.apache.http.*`, etc., or another HTTP Client/REST library of your choice) when placing an HTTP request to get the temperature resource from *Spot1*. Display the raw response on the screen.

- You probably noticed that, so far, we get HTML responses back from the SunSPOTs. As mentioned before, a RESTful architecture can offer several representations of the same resource. To get another representation 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 return a JSON file 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 translated to JavaScript objects. Display the raw JSON response.

- Parse the retrieved JSON response to extract the temperature value. Display this value only on the screen.
  Hint: You can find several libraries on the Web to parse JSON for many languages[4]. Android already includes `org.json.*` so you can directly instantiate and work with these JSON classes (i.e., `org.json.JSONArray` and `org.json.JSONArray`).

2 Experimenting with WS-* Web Services

Spot3 and *Spot4* offer a WS-* Web Service interface. Start by opening your browser and navigating to: `http://vswot.inf.ethz.ch:8080/WebServicesApp/SunSpotsWSService`. 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.
Hint: Try to navigate to the "schema" specified in the WSDL file.

[4] `http://json.org`

What to do

- **(report)** WS-* Web services can be invoked by first getting their WSDL and then sending SOAP messages to the Web service "end-point". A WSDL contains a description of these SOAP messages and end-points. Describe step by step how you would proceed to invoke a Web service using the `java.net` library only, as you did for RESTful Web Services. You do not need to implement it. Include this description in the report you submit.

- Create a **new** application and use a WS-* library to make a WS-* call from the Android device. Implement the subtasks with a button and function each. Use the kSOAP2 library patched to work on Android[5] (or another WS-* library of your choice) to get the temperature data from *Spot3*. First, display the raw SOAP response on the screen.
  Hint: Check out `http://code.google.com/p/ksoap2-android/wiki/Links` for help.

- When using WS-* services, messages are always represented using XML "over the wire" and then "unmarshalled" on the client side into platform specific objects (SOAPObjects in the case of kSOAP2). Find out how to get the XML containing the temperature and simply display it as raw XML on your Android phone.

- Parse the retrieved SOAPObject (or XML response) to extract the temperature value. Display this value on the screen. Feel free to parse the value using a library.

3 Assessing Web Service Technologies

Imagine that you are an IT consultant who needs to evaluate these two technologies, REST and WS-*, for working with embedded devices.

What to do

Answer the questions at `https://spreadsheets.google.com/viewform?hl=en&pli=1&formkey=dFQ4WGM2Tkt2T0s0dWlCOXAwVXE0MUE6MQ#gid=0`. This part of the assignment is *to be done individually*, so do not forget to state your ETH login name at the beginning of the form.

4 Cloud Services

Use the Google Visualization APIs to display the sensor measurements you retrieved with your phone. You can read more about the specific display methods at: `http://code.google.com/apis/visualization/documentation/gallery.html`.

What to do

Pick your favorite visualization scheme. Send the measured values to the service and display the visualization image on the phone's screen. You may choose between REST and WS-* APIs for accessing the sensors.

[5] `http://code.google.com/p/ksoap2-android/`

5 Your Phone as a Server (+report)

Similar to the SunSPOT sensor node, your phone can also act as a sensing device. **Reuse** your implementation in Assignment 1 and provide two sensors and two actuators through a REST server running on your mobile phone.

What to do

- Implement the server in a similar manner to the `Socket` client in Task 1, only now make use of `ServerSocket`. Wait for and `accept` incoming connections and then handle the requests according to the HTTP protocol. You will have to pick a port greater than 1024 (e.g., 8081).
  Hint: You should reuse your `Service` from Assignment 1. Also, your server does not have to be RFC-compliant. Just implement a minimal version that works with your browser (e.g., header options can be ignored).
- Run the Android 2.2 Wi-Fi Hotspot app and connect your laptop to the phone. Use your laptop's browser to test the functionality of your Android Web server.
- Describe your chosen architecture and control flow to fulfill client requests in the report, even if you might not be able to get the server running. The report has to be done individually.

6 Enhancements (+report)

Pick at least one of the following and describe the corresponding code in the report:

- Implement a WS-* request by hand in a similar way to the REST request using raw sockets.
- Improve the Web server in Task 5 with multi-threading and additional services. The server should be able to accept and answer requests form several clients at once.
- Instead of using the Google Visualization API implement local visualization on your phone.

Deliverables

The following deliverables have to be submitted by 9:00am on October 29:

1. **code.zip** You should create a zip file containing the Eclipse projects created in this assignment.
2. **report.[txt,pdf,doc,docx]** Document explaining the architecture and control flow of your Web service invocation, Android Web Server, and enhancements. Feel free to make use of diagrams. The text should not exceed one page.
3. **form for task 3** Technology-related questions at `https://spreadsheets.google.com/viewform?hl=en&pli=1&formkey=dFQ4WGM2Tkt2T0s0dWlCOXAwVXE0MUE6MQ`.
4. **feedback form** Tell us your experience with learning the REST and WS-* technologies: `https://spreadsheets1.google.com/viewform?hl=en&formkey=dGQzOVpGZGRBRW9iemNkb0hQbDNqZlE6MQ`.

Marks

The distribution of marks is as follows (partial solutions will be marked individually):

- Tasks 1-3 and report: 4.0
- Tasks 1-5 and report: 5.0
- All tasks and report: up to 6.0

Submission

Code and report must be uploaded at `http://www.vs.inf.ethz.ch/edu/vs/submissions/`. 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 9:00am on October 29.