TC30 Coursework 3

Deadline: 09.00, Friday 18 November 2011

Introduction

This exercise aims to give you some practical experience of writing clients in Java to consume Web Services.

Useful Resources

- Web Services examples: available on TC30 area on the VLE `http://www.leeds.ac.uk/vle`
- Documentation for Apache Axis, under `http://ws.apache.org/axis`
- servlets and JSP examples if you choose to develop the full, Web-based user interface to your client: available on TC30 area on the VLE `http://www.leeds.ac.uk/vle`

Preparation

- Review the material covered in Lectures 9–11.

The Task

1. Your main task is to build a Web Services client that integrates two Web services. These Web services are listed at `http://www.xmethods.net` or `http://www.webservicex.net` (but could be somewhere else). Feel free to investigate the use of Web services provided by Yahoo `http://developer.yahoo.com` and Google, for example `http://code.google.com/apis/maps/documentation/webservices/`.

   Example 1: In the *QuoteOfTheDay* example available on `http://www.xmethods.net`, the service returns the name of an author which the integrated client should then use to do a search on the author and output the top ten links for that author.

   Example 2: In the *Shakespeare* example (also available on `http://www.xmethods.net`), the client would need to get the partial text of a speech from the user and pass it to the service. It will then need to extract the name of the play or name of the speaker from the result and use these data to construct another query to the second Web service.

   There is a full service list at `www.xmethods.net` and `http://www.webservicex.net`. As you need to choose your own service, look at the services that you may find interesting, taking note of the implementation platform and examining the WSDL for each service.

   Note that the order of difficulty varies. Some services are easy to handle because results are returned as simple strings of plain text; others are slightly more challenging because the returned string is XML rather than plain text (albeit very simple XML), therefore requires parsing. More marks are available if you tackle original and challenging services.

2. Now choose one of the services. You are advised to choose an easy to handle service if you are not a confident Java programmer. Save a copy of the WSDL document for the service you have chosen, as you will be needing it later.

   Question 0. What is the name of the services you have chosen? What do they do? What is the name of the publisher?

3. Before starting on any coding, if you have not set up Axis during the Lab. session on Web Services then make sure your environment is correctly setup.

4. Before implementing a class to encapsulate a chosen Web service, you should generate client-side stubs for the service from its WSDL document. This is done by running Axis' `WSDL2Java` tool, via the shell script `wsdl2java` already provided for you. For example, if you have saved the WSDL to a file `service.wsdl`, then the following is required:

```
./wsdl2java service.wsdl
```

After running the script, you will see that a hierarchy of subdirectories has been created, reflecting the package within which the stubs and helper classes reside (depending on the service you use you may need to move into a subdirectory before you find the `.java` files). Have a look at these files and make that sure you understand their purpose. (The documentation provided with Axis will prove useful here, particularly the section of the User Guide on `WSDL2Java`.)

> **Question 1.** How does this approach compare with the steps required to implement a Java RMI client? How does it differ from the example of a Web Services client discussed in lectures and developed using JAX-RPC and JAX-WS?

5. Move back up into the top directory and compile the stubs with a command like

   ```
   javac /path/to/java/files/*.java
   ```

   Then package the stub classes as a single JAR file with a command something like this:

   ```
   jar cvf stubs.jar /path/to/java/files/*.class
   ```

   You will need to submit this JAR file with your source code. You should also make sure that it is in your CLASSPATH when you try to compile or run the consumer of your chosen Web Service.

6. To implement the consumer of a chosen Web Service, note the following points about implementation:

   - There are three steps involved in making a method call to a service exposed via statically-generated stubs:
     (a) Create an instance of the 'Locator' class
     (b) Get an instance of the 'Soap' class by calling appropriate method (e.g., `get...Soap`)
     (c) Invoke the desired method of the 'Soap' object

     For further details, see the Axis User Guide—especially the section covering WSDL2Java. The `main` method of this class can contain code to test the service.

7. The final step is to implement a class that combines the two Web services and provides an appropriate user interface for the collection of input data and presentation of results. There are two options here:

   - **Option 1 [4 marks]:** Make the application console-based, with input data coming from the command line if required and results being written to a file as a static page of HTML.
   - **Option 2 [10 marks]:** Implement the application as a servlet or using JSP, as appropriate. The HTML will be generated dynamically in this case. If you are following this option, you will need to package your application in the conventional way, as a WAR file.

   Option 2 is a significantly greater challenge!

8. Answer the following questions:

> **Question 2.** Consider the respective platforms upon which the service providers and service consumer are implemented. What does this tell you about how Web Services differ from more conventional distributed object technology?

> **Question 3.** What are the factors affecting the performance of your application? In what ways, and at what cost, could performance be improved? (You may assume that you have the freedom to reimplement the service providers as well as your service consumer, using a different hardware and software platform if necessary.

Submission

Submit your answers to Questions 0–3 on paper in the normal way, via the CSO postbox. Remember to attach a completed coursework header sheet. To submit your code for the exercise, create a Zip or tar archive of the files which make up your system and submit the file using SIS. Remember to include a descriptive `README` file as well as any additional libraries.

Be sure to include in your submission all the files that you have generated for your client. If you have organised your files into a directory hierarchy, then please package this as a single Zip or tar archive and submit that file using SIS. If your application is packaged as a WAR file, please include this file in your submission.

Important: Full instructions on how to install, compile and run your client should be provided as part of your written submission *and* in the file named `README` that you submit along with your code.

Distribution of Marks

Answer to Question 1	4
Answer to Question 2	4
Answer to Question 3	4
Web Service 1	6
Web Service 2	6
Integration / user interface	10
Successful execution	3
Coding style / comments	3
	40