Service-Centric Cloud Technologies - Practical Study #8

Implementing Web Services with JAX-WS

Part I of this exercise focuses on the familiarisation with NetBeans tools for building Web Services with JAXB from the POJO model.

In Part II you should attempt to use JAXB to bind a Java client to an existing Service to demonstrate the advantage of using SOA for application integration without the need for a pre-defined agreement on communication mechanisms.

Part I Working with XML Schemas in NetBeans

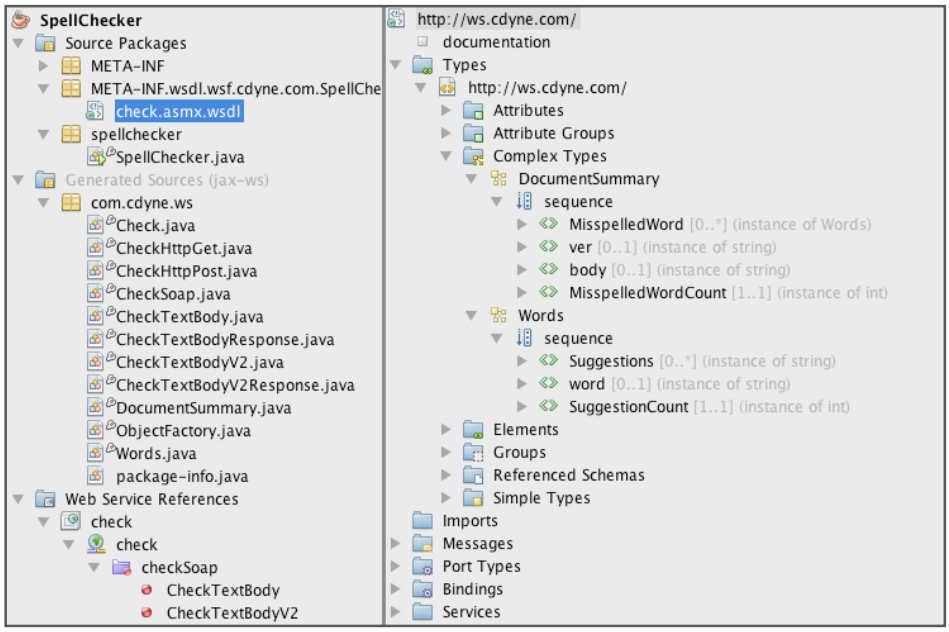
1. Follow the Netbeans tutorial: http://netbeans.org/kb/docs/websvc/jax-ws.html in order to learn how to:
   - Design and Create the Web Service
     notes: *Always select **Glassfish** as your EE server; do not use Apache Tomcat*
     Make sure that the option: ☑ Implement Web services as Stateless Session Bean *is **checked**.*
   - Deploy and Test the Web Service
   - Consuming the Web Service in a:
     a. Java class in a Java SE Application
     b. Servlet in a web application
     c. JSP page in a web application a Java class in a Java SE Application

   It is essential that you patiently follow all the tutorial steps.
2. Find the CalculatorWS WSDL file. Identify the WSDL file elements relevant to building your web service in terms of:
   - Service binding: Location and Binding Format
   - Invoked operations, their parameters, and returned values
   - How do the WSDL XML types map into Java Types

Part II Using JAX-WS to bind to a public web service

1. Create a new Java Application (call it SpellCheckWS *) that will use a public web service to perform a spell check for any given text.
2. Right-click the SpellCheckWS node and choose New > Web Service Client. In the "WSDL URL" field, enter the URL to the public web service: http://wsf.cdyne.com/SpellChecker/check.asmx?wsdl . Accept the defaults and the package name and Finish.
3. Open the WSDL file "check.asmx.wsdl" under the 'META-INF.wsdl' package as shown in the image below. By examining the XML types in the WSDL file, you can identify the data structure used by the SpellChecker web service. It has a major complex type (**DocumentSummary**) that contains a List (**MisspelledWord**) of **Words** type. In turn the complex **Words** type contains a sequence of each misspelled word (**word**) and a list of corrections (**Suggestions**).
4. Now examine some of the equivalent Java classes (see image below) that are automatically generated to process the XML type above. For instance you should be able to clearly relate the declarations in the **Words.java** class to the structure of the **Words** complex XML element.

** For a complete account of the spell checker web service, visit: https://netbeans.org/kb/docs/websvc/client.html*



5. Consume the external Web Service by dragging its **checkTextBody** from the **Web Service References** folder into the main java file **SpellChecker.java**. A static method allowing you to call the web service method will be created (*checkTextBody(java.lang.String bodyText,…*).
6. The code segment below calls the web service to spell-check and uses the XML java objects to correct the result. Copy the code into the **main(**String[] args**)** method body and run the file.

```
//call the web service to correct text
DocumentSummary myDoc = checkTextBody("dificulty speling", "any");
//get the list of misspelled words objects
List<Words> misspWords = myDoc.getMisspelledWord();
Iterator itr = misspWords.iterator();
//for each misspelled words object (Words type)
while(itr.hasNext()) {
    //get and print the mispelled word
    Words wrongWord = (Words)itr.next();
    System.out.println("Misspelled Word: " + wrongWord.getWord());
    //get and print a list of Suggestions for correct spelling
    List<String> suggestions = wrongWord.getSuggestions();
    Iterator itr2 = suggestions.iterator();
    while(itr2.hasNext()) {
        System.out.println(itr2.next());
    }
}
```

Make an effort to understand how the XML types in '3.' above translate into the Java objects used in the illustrated code. Pay special attention to how unbound XML elements translate into Java Lists:

XML: <s:element minOccurs="0" maxOccurs="unbounded" name="MisspelledWord" type="tns:Words" />
Java: List<Words> misspelledWord;

7. To further exercise the consumption of web services from a GUI perform the following:
   - Create a graphical editor that utilises the SpellCheck Web Service. Initially the editor can consist of a simple text box to enter text and another text box to display the misspelled words with suggestions for corrections.

** For a complete account of the spell checker web service, visit: https://netbeans.org/kb/docs/websvc/client.html*

- A more sophisticated version should allow the user to graphically (by click of a button) exchange the misspelled word(s) by a corresponding correction word.

Part III Modelling application integration using Web Services

Create another *provider* web service that will *feed text* passages to the spellcheck web service created above. Ideally the *provider* web service should capture the text passage from a public web location such as a news site, RSS, web blog, Wikipedia, etc.

A script demonstrated basic retrieval of text information from a web resource is provided below.

```
//script demonstrating how to read the content of a URL resource
//be ware that all the html content will be retrieved, i.e. including the tags: <h1>, <img>, etc.
import java.io.*;
import java.net.*;

public class OpenStreamTest {
   public static void main(String[] args) {
      try {
        URL compDept = new URL("http://www.ntu.ac.uk/sat/about/academic_teams/comp_tech.html");

        BufferedReader ins = new BufferedReader(new InputStreamReader(compDept.openStream()));
        String inString;

        while ((inString = ins.readLine()) != null) {
          System.out.println(inString);
        }
        ins.close();
      } catch (MalformedURLException me) {
          System.out.println("MalformedURLException: " + me);
      } catch (IOException ioe) {
          System.out.println("IOException: " + ioe);
      }
   }
}
```