The Open University

T320 E-business technologies: foundations and practice

Block 3 Part 3 Activity 3: Deploying a web service

Prepared for the course team by Neil Simpkins

Introduction

In this activity I shall briefly show you how to use your Axis2 server account at the University to deploy your 'Hello' web service to the server.

Before you do that, you need to prepare the 'Hello' web service that you created and tested in Eclipse, ready to be deployed. This takes the form of simply wrapping the various files of the service up into a single 'archive' file.

Axis itself is a web application on the server side. This means that it is installed where Tomcat expects to find any web applications. So inside the 'webapps' directory on the server there is a web application archive file 'axis2.war' that Tomcat will deploy, creating an 'axis2' directory to hold the application's files. Current versions of Tomcat can do this without being restarted, which is termed 'hot deployment'.

As the 'Hello' web service is a Java-based application, it would be possible to produce a web application archive (WAR) file and then drop this into the 'webapps' directory. This would be appropriate if you had written code to handle all messaging to and from the service yourself, but you used Axis to handle all these aspects. So, you need to deploy the 'Hello' service as an Axis web service. Just like Tomcat, Axis has a directory where it expects to find services that are deployed as Axis archives with an 'aar' file extension (Figure 1).

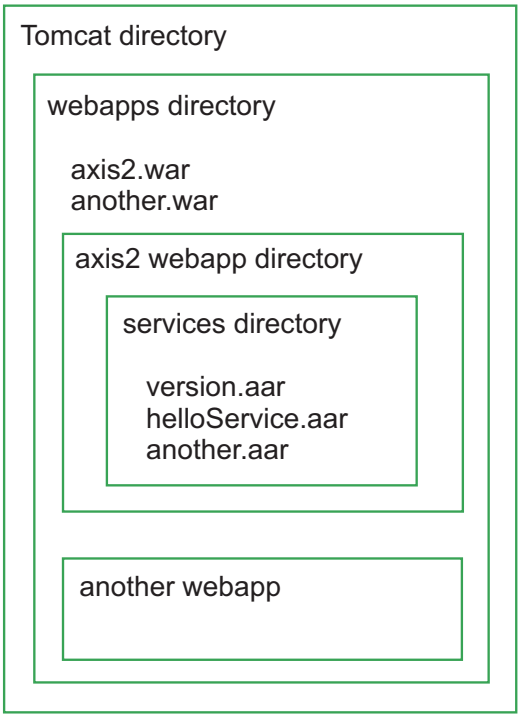

Figure 1 Directory structure outline of Tomcat and Axis2 containers

Creating an archive

There are a range of approaches that we can take to both packaging up and deploying the 'Hello' service. The process is remarkably simple but we will use some tools to help.

The files of the 'Hello' service need to be packaged up into a single archive called, say, 'helloService.aar'. The structure of the archive, in terms of directories and where the different files are placed, has to follow some conventions. However, Axis provides a tool to generate an archive, so you can use this to create an archive of the 'Hello' web service without any need to know how to structure the archive. This tool is described by the Apache Foundation at:

http://ws.apache.org/axis2/tools/1_3/eclipse/servicearchiver-plugin.html

Start Eclipse, and ensure that the 'Hello' web service project is within the workspace and listed in the Package Explorer view. Select File > New > Other… so that the 'Select a wizard' pane appears (Figure 2). Expand the 'Axis2 Wizards' folder, select the 'Axis2 Service Archiver' and click the 'Next' button.

You will then see the dialogue box shown in Figure 3, which requires you to locate the compiled Java classes for the web service.

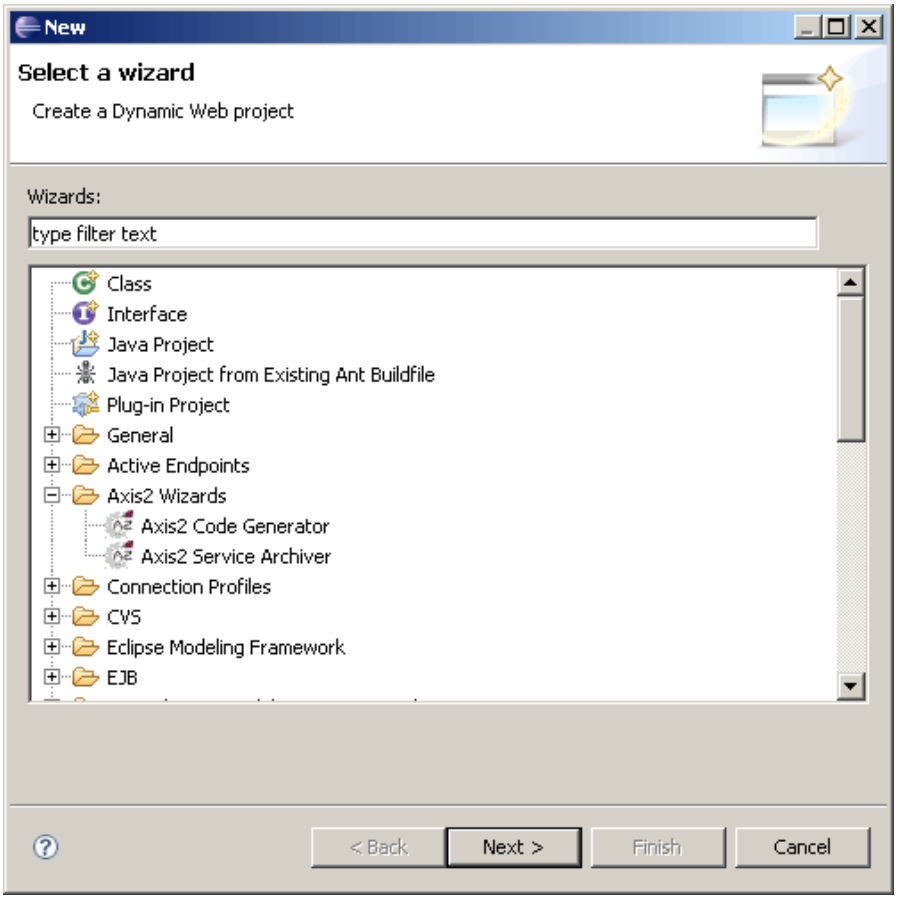

Figure 2 Axis2 plug-ins listed in Eclipse wizard selection dialogue box

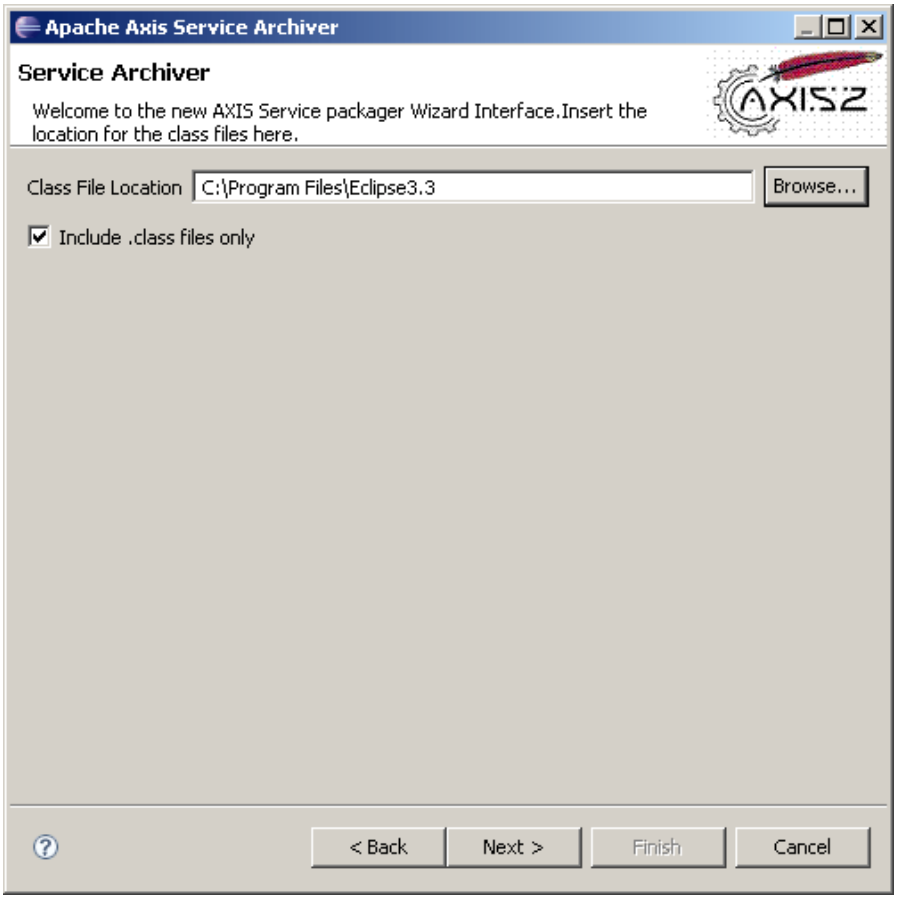

Figure 3 Locating the web service Java classes

Unless you know the exact location and can type it in, click the 'Browse…' button and then browse to the location of your workspace (which will be called 'workspaceBlock3' unless you changed the installation). Inside the workspace, browse to the 'Hello' project, then open up the 'build' folder and select the 'classes' sub directory (Figure 4). This is the location in which Eclipse will automatically have put your compiled Java class files. Click 'OK'.

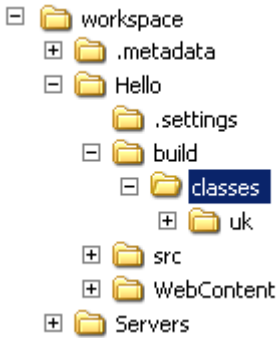

Figure 4 Location of class files in the 'Hello' project

Take some care to locate the Hello project. When you generate the test client earlier another project will have been created, usually named 'HelloClient' which should not be used here.

The project includes only a single class file inside the classes directory, so you can leave the 'Include .class files only' box checked (as shown in Figure 3) and click 'Next'.

The next dialogue box gives you the option of including a WSDL file for the web service in the archive. If a WSDL file is included then this will be available from the server later and can be used to access the service. If no WSDL is included then the service might be described elsewhere, by a WSDL file or in a UDDI perhaps.

In fact, there is a problem in deploying the WSDL from Eclipse to a web server elsewhere. The WSDL includes a reference to the location of the service, which will have the form:

```
<wsdl:port binding="impl:HelloSoapBinding" name="Hello">
    <wsdlsoap:address location="http://localhost:8080/Hello/services/Hello"/>
</wsdl:port>
```

This, of course, will not be the correct location once the service is deployed to another server machine. The WSDL could be edited, but rather than do that select 'Skip WSDL' and click 'Next'.

Now you can add any external (Java) libraries that were used to support the service to the archive (Figure 5). It is common practice to use third-party library code whenever possible, to avoid 'reinventing the wheel'. In the simple 'Hello' service you didn't need to use any, so just click 'Next'.

This will take you forward to the dialogue box shown in Figure 6. A 'service.xml' file is used by Axis to determine a range of service properties, such as the name of the service implementation class and the operations that the service can perform. You could write the 'service.xml' file by hand but it's simpler to let Axis generate it, so tick the 'Generate the service xml automatically' box and then click the 'Next' button.

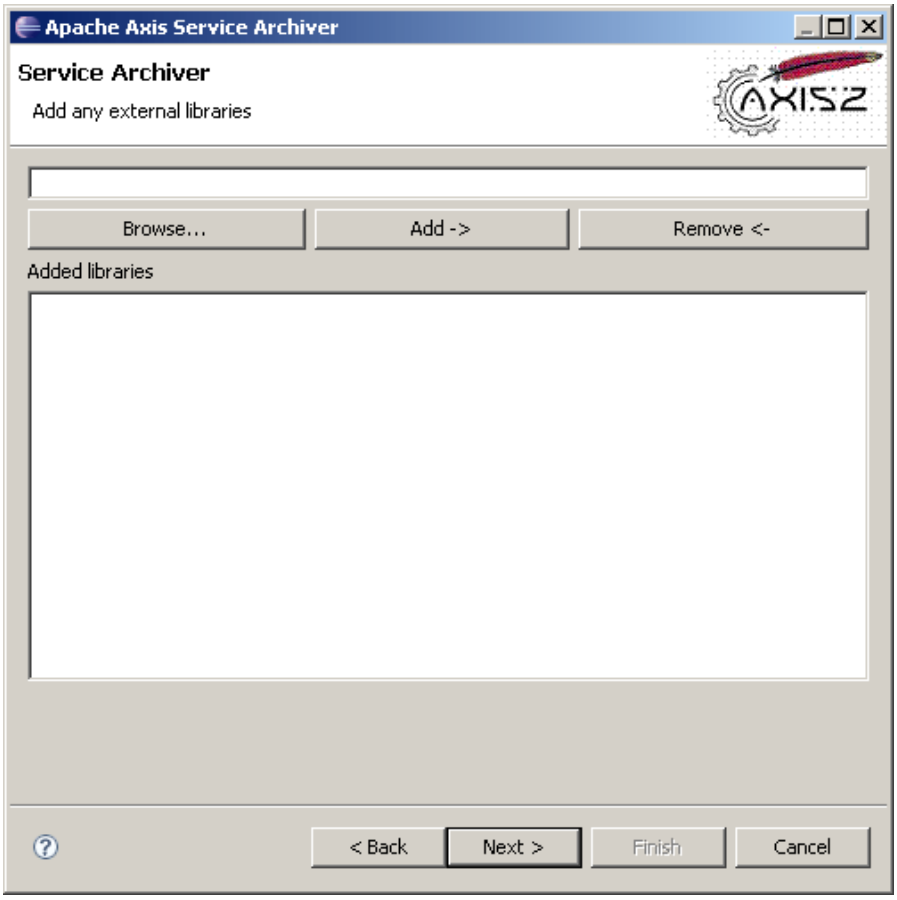

Figure 5 Adding any external libraries

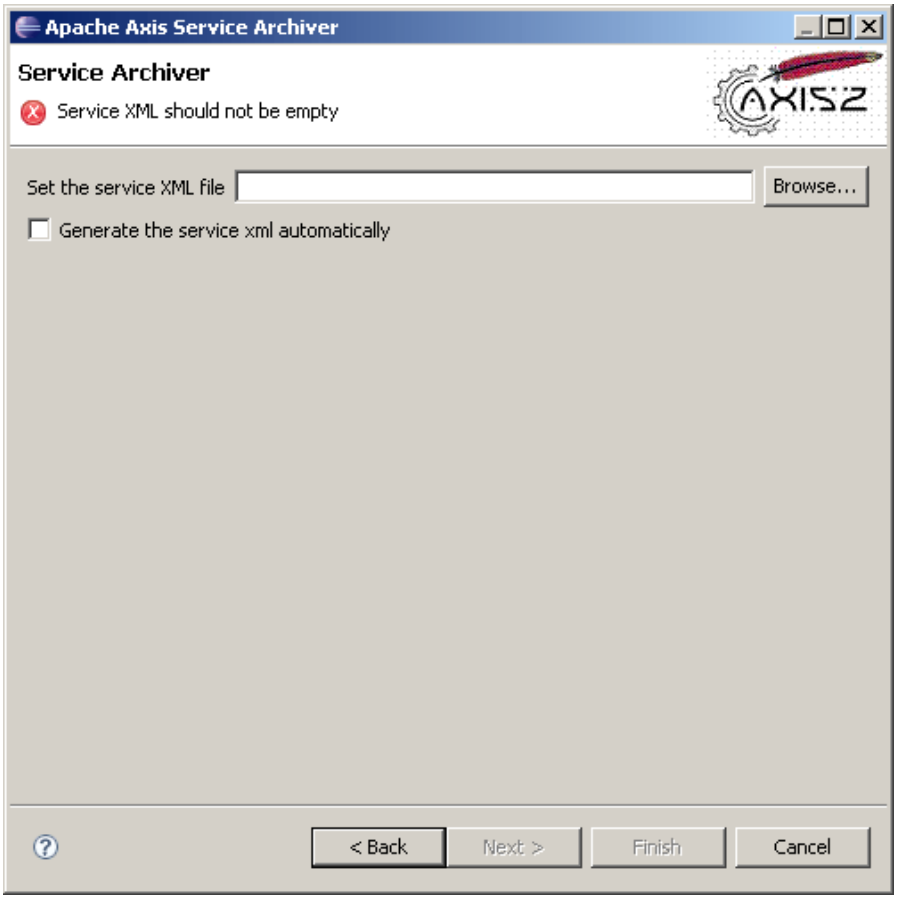

Figure 6 Selection or generation of service XML file

The next dialogue box (Figure 7) seeks to identify the code that is being used and establish what needs to be written into the 'service.xml' file. The default service name is given as 'MyService', which you should change to something more meaningful, perhaps 'HelloService'.

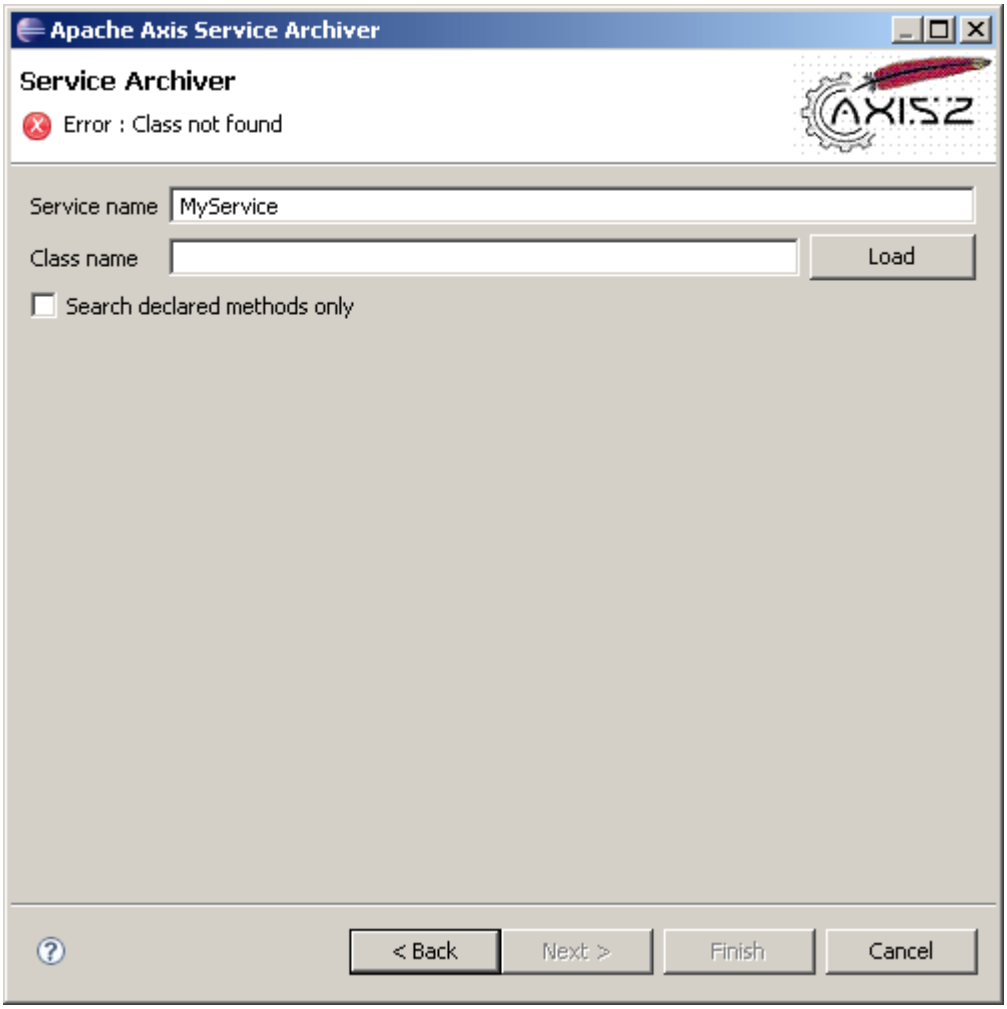

Figure 7 Service class selection dialogue box

The class name for the service is 'Hello', but this needs to be qualified with the package name used when creating the service. Your package name, as I described earlier, should include your OUCU. Type your package and class name into the 'Class name' box, which should be something like:

```
uk.ac.open.t320.<OUCU>.Hello
```

where '<OUCU>' is your own OUCU and click on the 'Load' button. You will see that a list of methods is given (Figure 8). These are the methods that have been 'inherited' by the code you wrote for the 'Hello' class.

You can ignore the inherited methods and list just the methods that you have coded by checking the 'Search declared methods only' box (Figure 9)

Leave the 'helloName' method box ticked and click 'Next'.

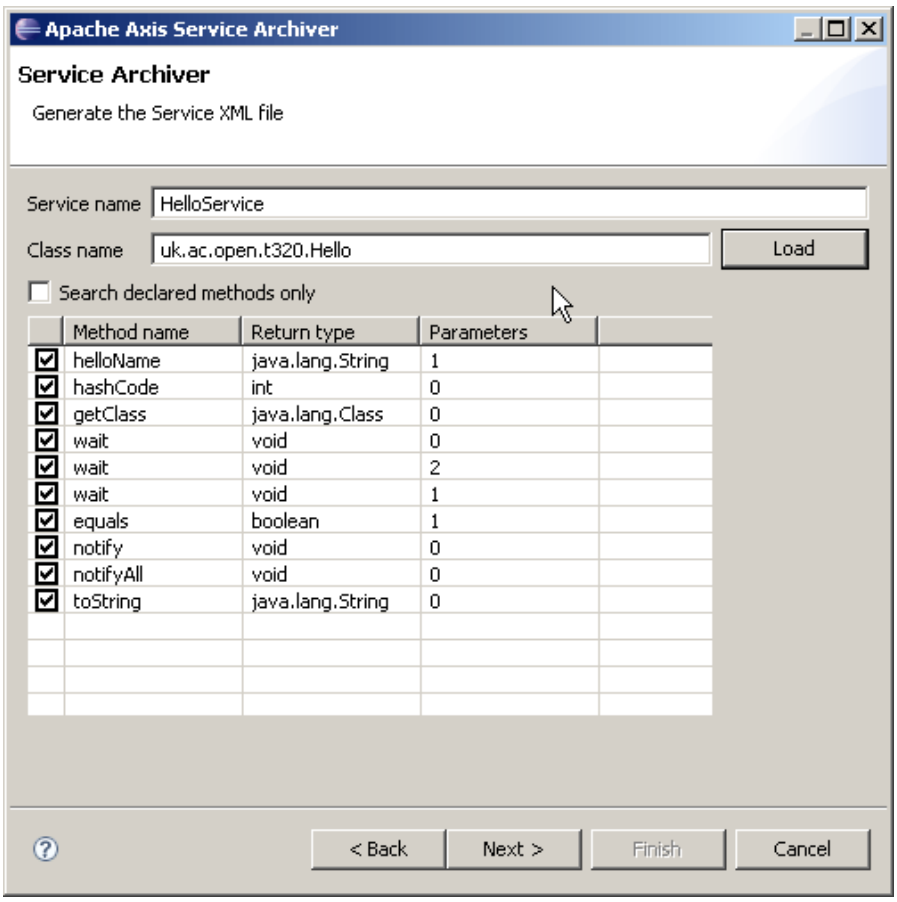

Figure 8 Service class selection dialogue box showing potential methods

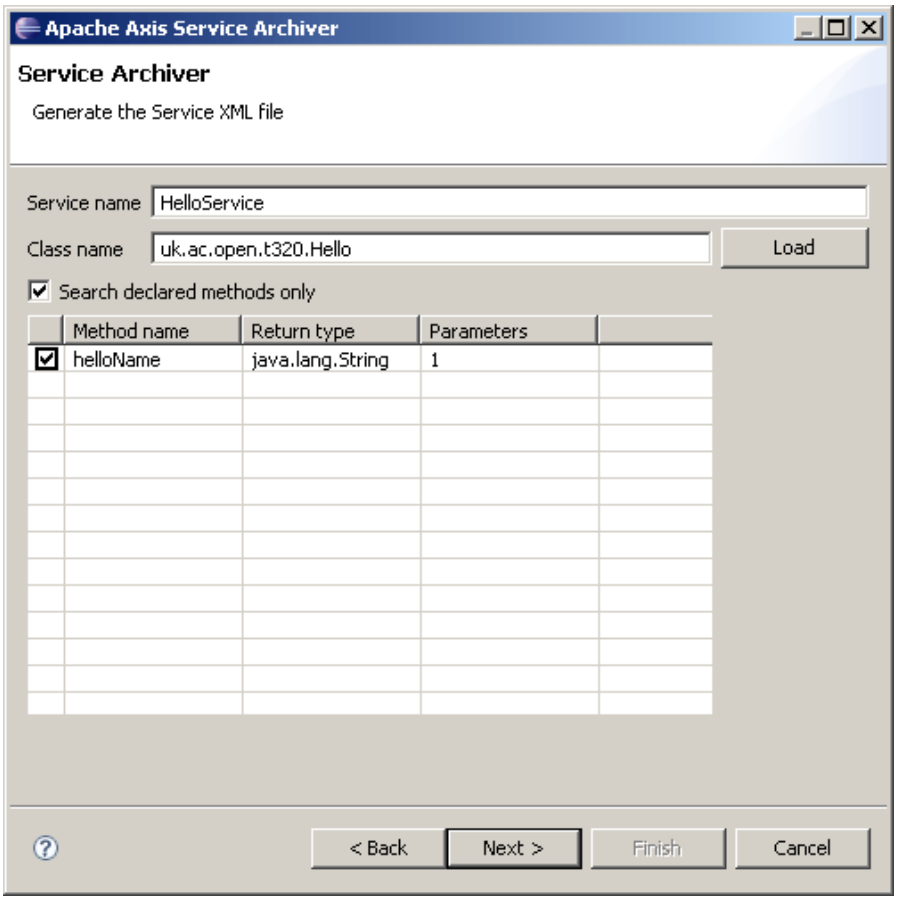

Figure 9 List of methods implemented directly by 'Hello' class

The next dialogue box (Figure 10) allows you to specify the output archive's filename and location. Browse to a suitable output location and name the archive something like 'helloService' (the `.aar` file extension will be added automatically). Then click 'Finish'.

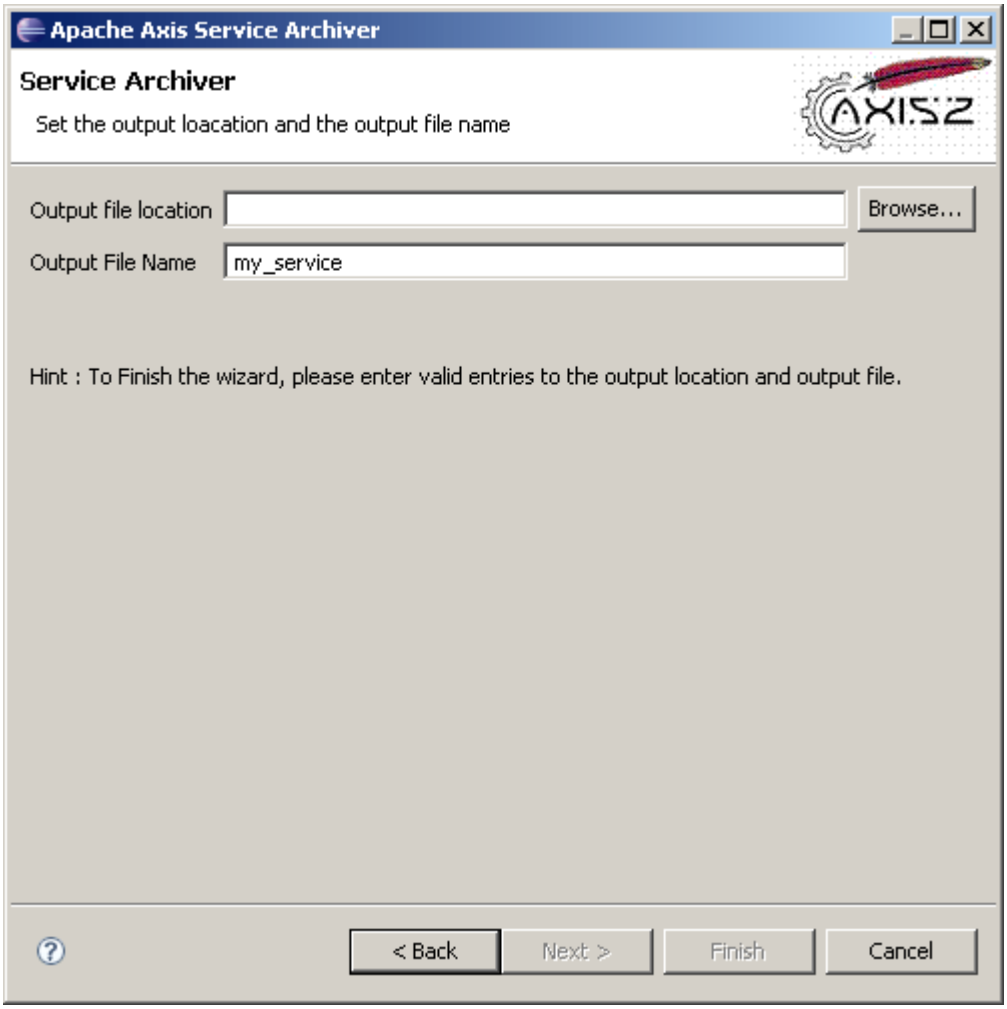

Figure 10 Setting the output file name and location

After a short time, you should receive a message confirming that the archive has been generated (Figure 11).

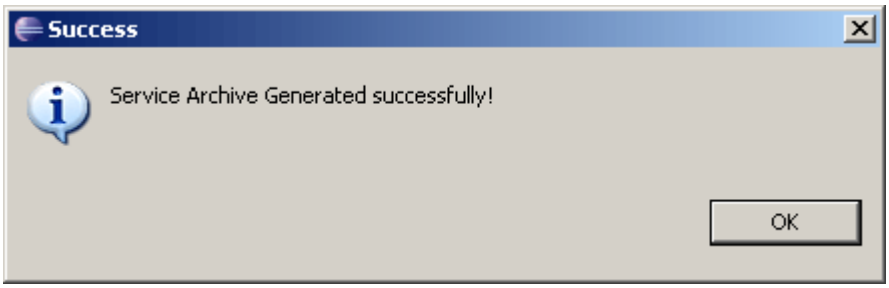

Figure 11 Confirmation of archive generation

The archive file can be opened using WinZip to reveal what has been included. In this case the archive contains only three files (Figure 12).

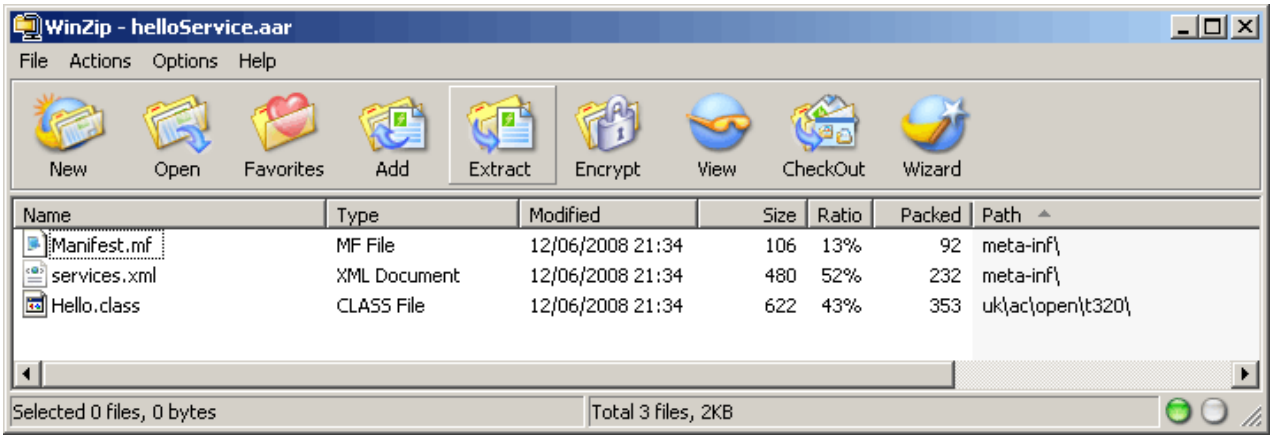

Figure 12 Files in the archive

The 'Hello.class' file is the Java file compiled and placed in a directory that is named after the package name you used for the code.

A 'Manifest.mf' file describes an archive and can contain a range of additional configuration information (see http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html#JAR%20Manifest if you are interested in the details). The manifest in this case simply contains three lines, which give some version numbers such as the Java version being used:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Created-By: 1.5.0_06-b05 (Sun Microsystems Inc.)
```

The more important file is the 'services.xml' file, which contains an XML description of the 'Hello' service:

```
<service name="HelloService">
    <description>Please Type your service description here</description>
    <messageReceivers>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
         class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
        <messageReceiver mep=http://www.w3.org/2004/08/wsdl/in-out
         class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
    </messageReceivers>
    <parameter name="ServiceClass">uk.ac.open.t320.Hello</parameter>
</service>
```

The service description names the service as 'HelloService', contains a placeholder for a textual description of the service and specifies the 'ServiceClass' to be the 'Hello' Java class that you wrote, qualified with its package name.

The remaining chief elements are of the kind `<messageReceiver>`, which specify what Java class will handle messages to the service. You could write your own, but here Axis has suggested using some classes it provides for you. The first handles requests that have no response, the second handles request–response type messages. Both of these classes implement a message exchange pattern (MEP). MEPs are part of the WSDL specification (see http://www.w3.org/TR/2004/WD-wsdl20-extensions-20040803/).

There are many other configurations and options that can be placed in the 'services.xml' file, such as engaging a security module and other supporting facilities. I shall not cover these in any detail here.

The next step you will take is to upload the archive onto an OU server machine and then to place the archive in the correct position within a Tomcat and Axis installation.

Deploying an archive to Axis

Using a web browser go to the Axis2 home page (Figure 13).

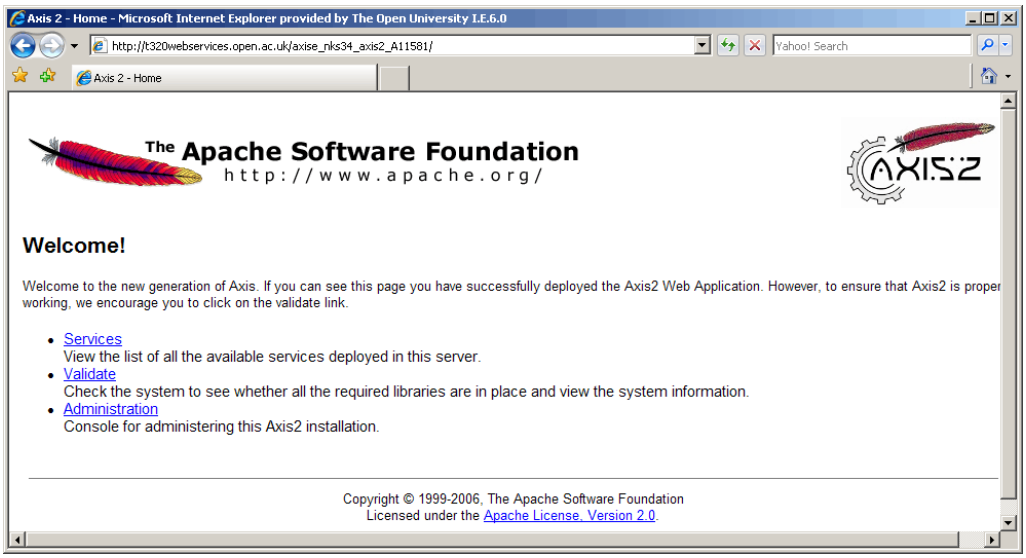

Figure 13 Axis2 home page

To upload a web service archive, you need to go to the administration console. Click on the 'Administration' link and you will be taken to the console page (Figure 14). Log in here with the Axis username and password, which by default are ‘Username’ of ‘admin’ and ‘Password’ of ‘axis2’. These values are found in the axis2.xml file of the installation.

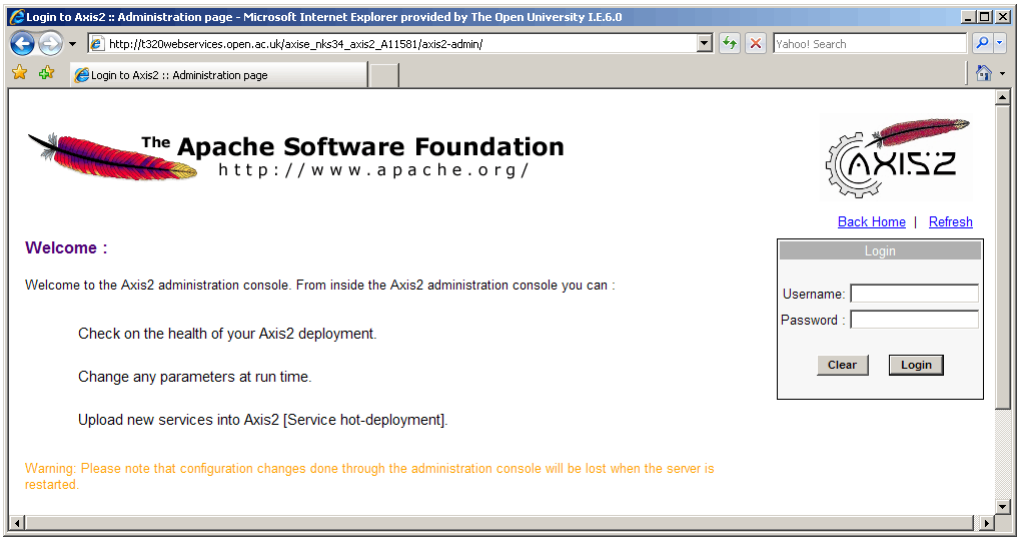

Figure 14 Axis2 administration console log-in page

You will then be shown the main administration page (Figure 15). This has many links for managing services and 'modules'. A module is a component that provides functions, such as security, for web services.

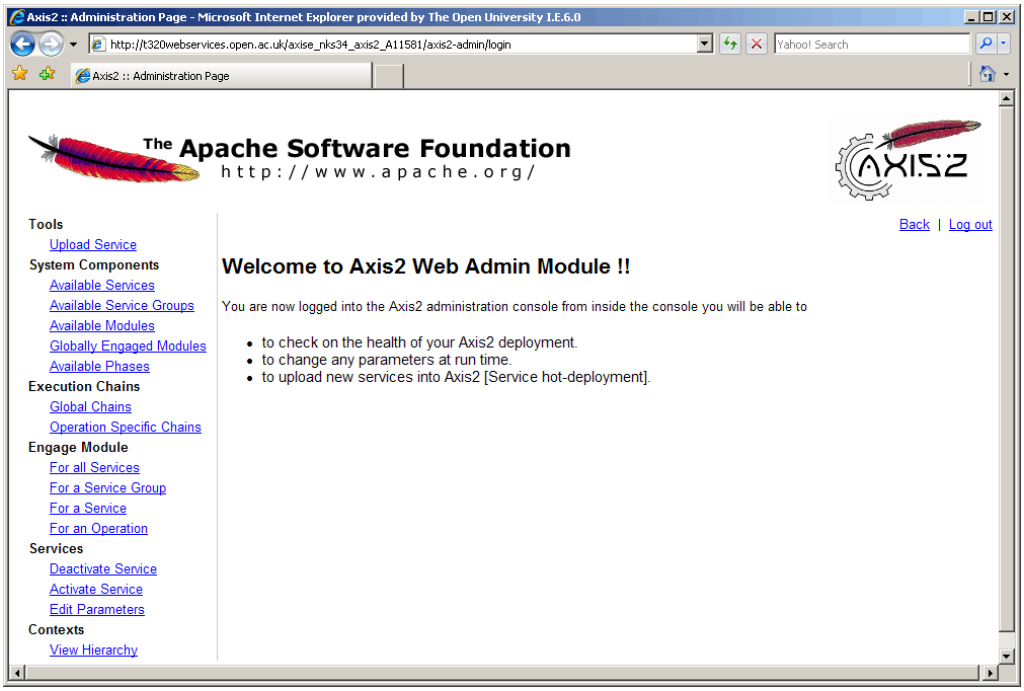

Figure 15 Axis2 administration page

First click on the 'Available Services' link (second from the top on the left-hand side). You will then see a list of the web services that are currently available, together with their status (Figure 16).

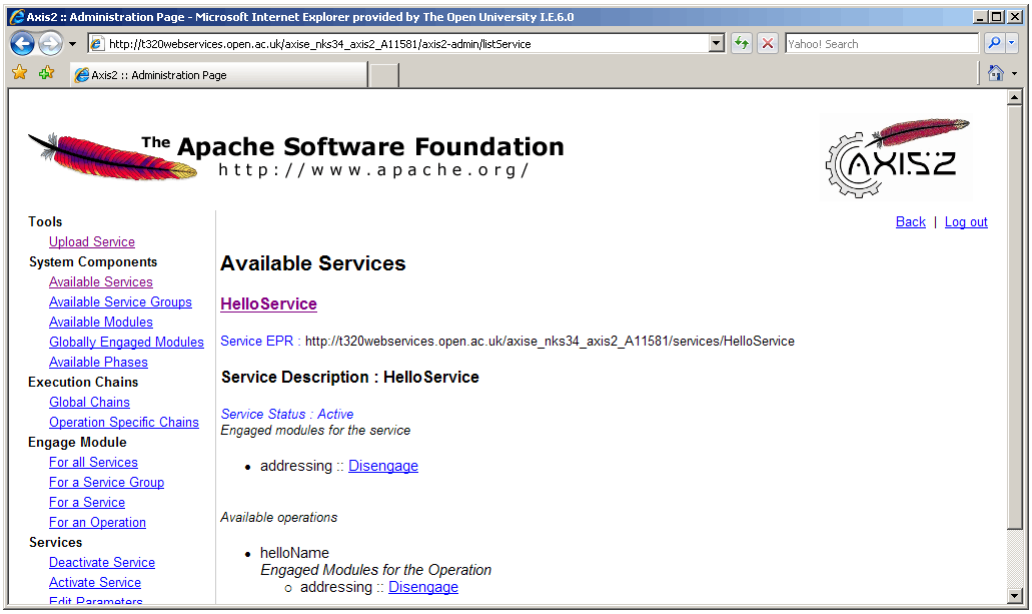

Figure 16 Available web services

Currently, the only web service that you will see in the list is 'Version'. As explained previously, the 'Version' web service is provided as an example web service and simply returns the version of Axis2 being used.

Now click on the 'Upload Service' link (the top link on the left-hand side), which will take you to the web page shown in Figure 17.

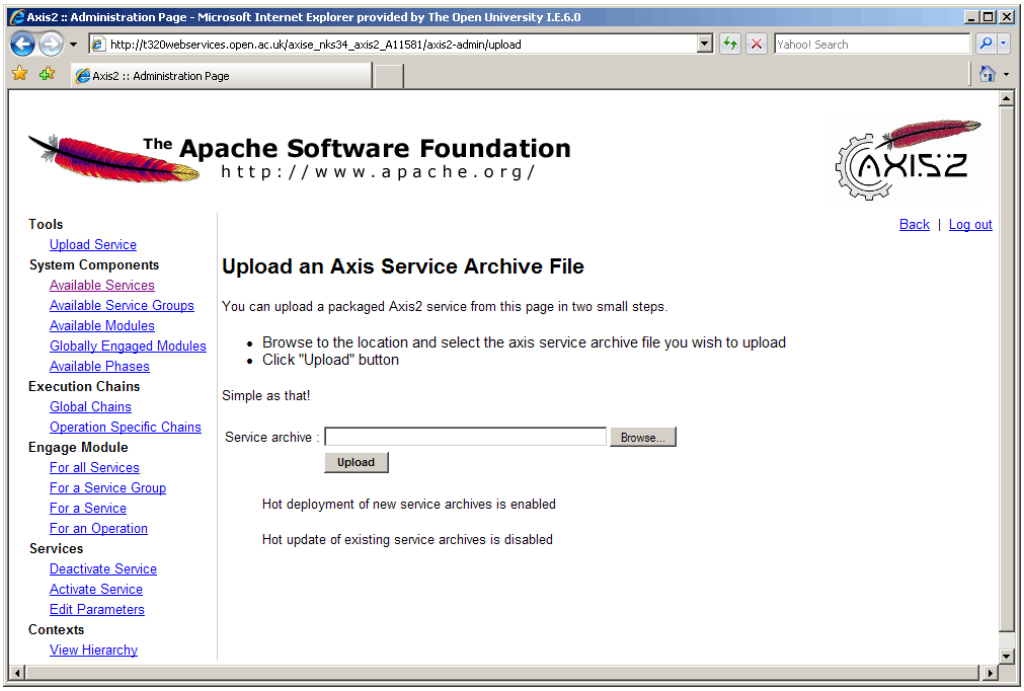

Figure 17 Upload service page

Click on the 'Browse…' button and navigate to the location you specified for the archive file earlier. Select the 'helloService.aar' file and then click 'Open'. Then, on the Axis2 upload page, click the 'Upload' button. You should see a green 'success' message appear above the 'Service archive' box (Figure 18).

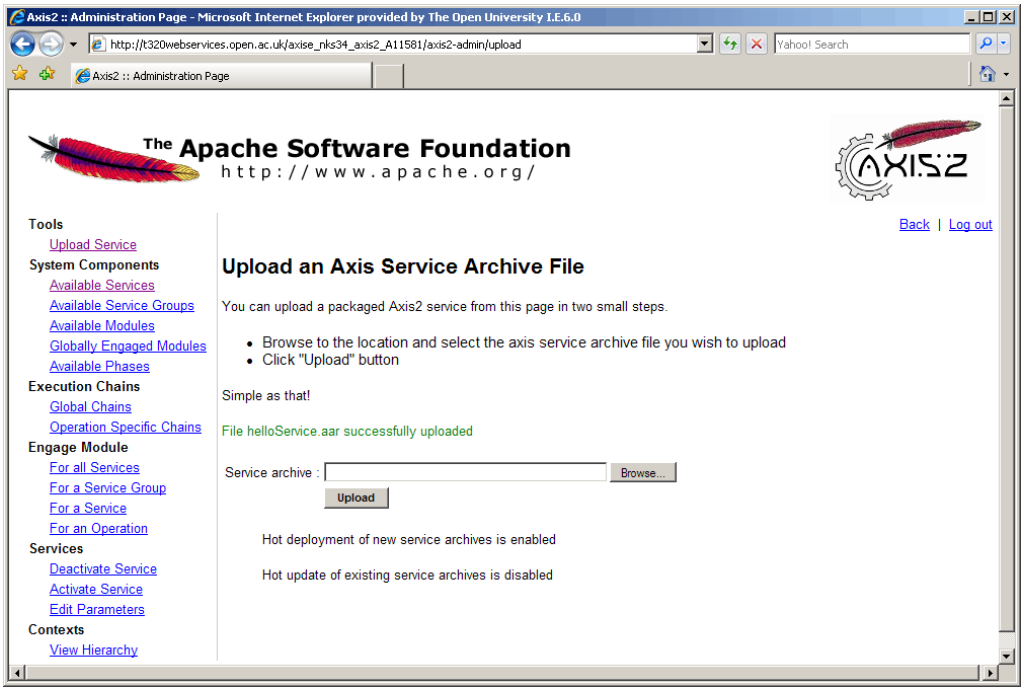

Figure 18 Archive file successfully uploaded

Now revisit the 'Available Services' link. You should find that 'HelloService' is now listed along with 'Version', although you may need to scroll down to see the 'HelloService' listing (Figure 19).

Figure 19 'HelloService' listed in services list

Now click on the 'HelloService' link on the 'Available Services' page. This will present you with a WSDL description of the web service (Figure 20).

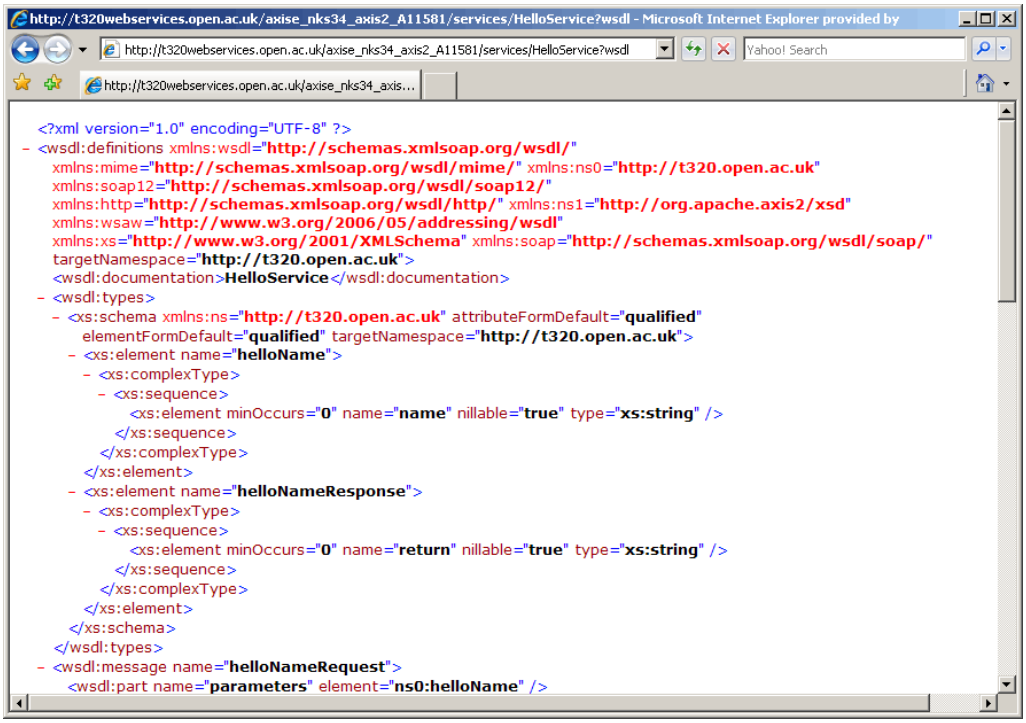

Figure 20 'HelloService' WSDL

This WSDL has been generated by Axis2 on the server and contains the correct address locations for the server (Figure 21).

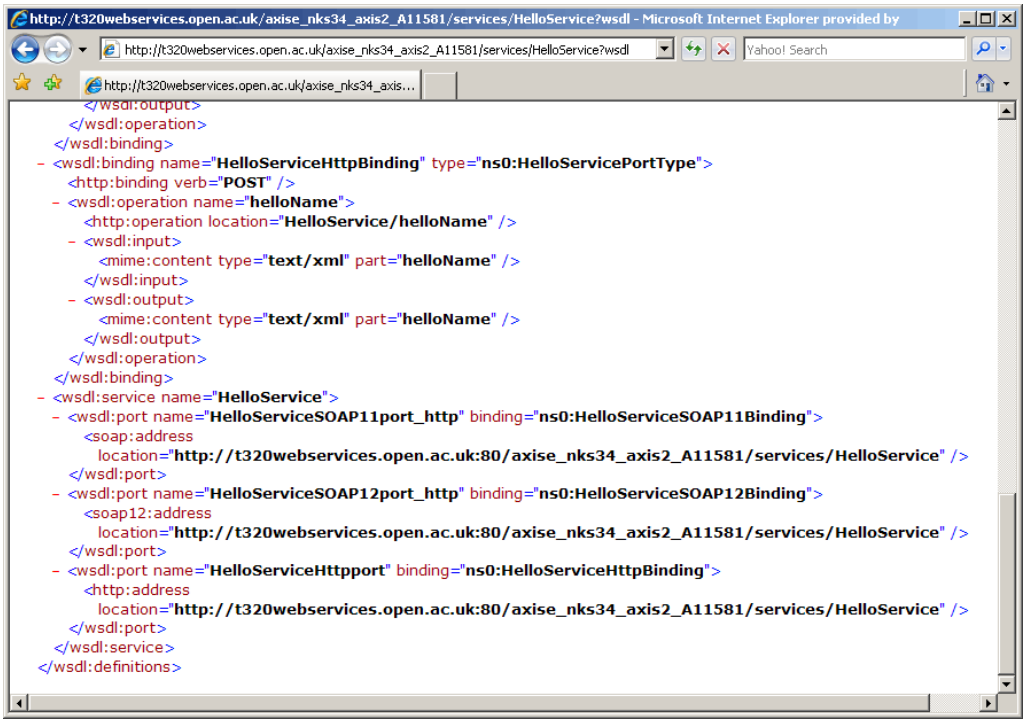

Figure 21 WSDL port and address descriptions

Now the service is deployed and the WSDL is being generated (you could have deployed a WSDL description as a static XML page instead).

Test the deployed service

Eclipse can be used to test the service. To do this, follow the steps in Part 2 Activity 2 (*Generating a client from WSDL*), which you used earlier to access a web service.

This should be very straightforward, but you might want to note the URL of your WSDL documents on the OU server now. In the example shown here it would be:

```
http://t320webservices.open.ac.uk/axise_nks34_axis2_A11581/
services/HelloService?wsdl
```