The Open University

T320 E-business technologies: foundations and practice

Block 3 Part 2 Activity 2: Generating a client from WSDL

Prepared for the course team by Neil Simpkins

Introduction	1
WSDL for client access	2
Static versus dynamic WSDL	3
OU demo services	3
Creating a dynamic web project	4
Adding a simple client	6
More clients	12

Introduction

In this activity I shall illustrate how you can quickly and easily generate a client so that you can access a web service.

You have in fact already completed this exercise, at least in the main part, when you created the 'Hello' web service and generated a client to test that web service at the same time. However, here there will be a couple of slight differences. Firstly, you will not be producing a web service and a client, just a client for a web service. Secondly, you will explicitly use a WSDL description of the web service on which to base the client. This was also the case 'behind the scenes' when you deployed and tested the 'Hello' service.

The client will be generated within Eclipse, which uses the WSDL description of a service to determine how a request to the service should be formulated and what response is expected. The client itself takes the form of a simple set of web pages, just as before when you tested the 'Hello' web service.

It's important to recognise that the WSDL document that Eclipse uses to generate the client might be located in an Eclipse project locally on your machine, or it might be hosted on a machine somewhere on the Internet.

Copyright © 2008 The Open University.
This document is made available under the Creative Commons Attribution - No Derivative Works 3.0 Unported Licence (http://creativecommons.org/licenses/by-nd/3.0/)

WSDL for client access

A WSDL service description provides all the information that is required to use a web service. The T320 version of Eclipse incorporates the Web Tools Platform (WTP), which provides support for generating a client based on the information inside a WSDL.

The WSDL that describes a service includes a range of crucial information, such as the required content of a request and also the location of the web service. This location, called the endpoint reference (EPR), is included in the service description. The endpoint reference is thus a key item in the service description. If we move the web service then the EPR in all descriptions of the web service needs to be changed.

In Figure 1 I have depicted the realisation of the 'Hello' web service after completing the practical activity you carried out in Part 1. The WSDL is held locally in your workspace and will have a local EPR for the web service implementation, such as `http://localhost:8080/Hello/services/Hello`.

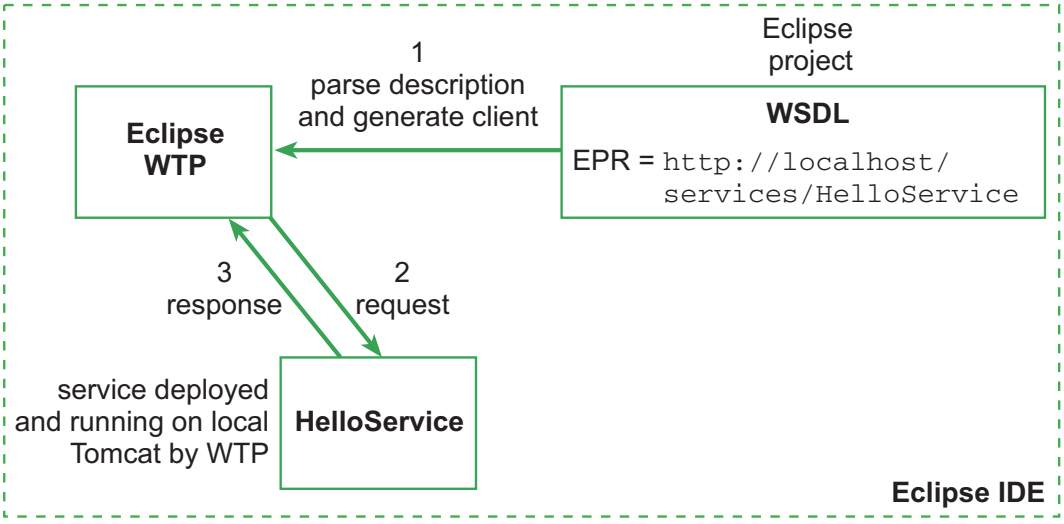

Figure 1 Overview of local access to 'Hello' service with Eclipse

When you used Eclipse to deploy the web service and generate a client, it performed a lot of work behind the scenes. It first deployed the web service to the server, started the service running and set up a proxy so that it could pass requests to the service and receive any response. Then it generated the WSDL description of the service and put this into the 'WebContent/wsdl' directory of the project.

To generate the client web pages, Eclipse parses the WSDL document. Based on that, it can determine the input requirements for the web service that are used to create the 'Inputs' form etc. The WSDL also tells Eclipse to send requests to the local Tomcat server. This configuration is illustrated by the Eclipse WTP, project and local service depicted in Figure 1; this is, of course, exactly the setup you employed in completing the Part 1 practical activity.

If the web service is deployed to a remote server then Eclipse has to be given the location and will download the WSDL for processing; in this respect the two scenarios are not much different. The web service itself can also be situated on a remote server, as specified by the EPR in the service WSDL. This configuration is illustrated in Figure 2.

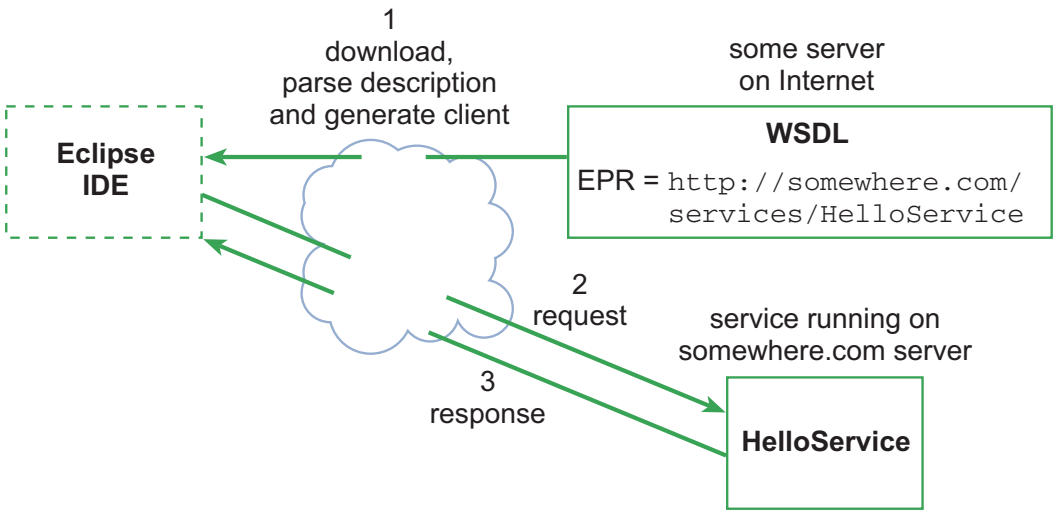

Figure 2 Overview of remote access to 'Hello' service with Eclipse

It is, of course, equally possible for a WSDL document held in a local Eclipse project to specify a remote EPR. Also, rather obviously, the WSDL description of a service and the service itself may reside on a single remote server. The 'physical distribution' aspect has no real impact on the mode of operation of the web service.

Static versus dynamic WSDL

The WSDL file that was generated when you created the 'Hello' web service was produced by Axis software that is part of the WTP embedded in the T320 version of Eclipse. This software examines the Java code implementing the web service as a basis for determining the required input data, likely content returned, etc. and subsequently creating the WSDL document.

There is also a server-side set of Axis software tools that have the same WSDL generation capability. So if a web service is hosted on a remote server that is running this Axis server-side software, the WSDL can be generated 'on demand' by Axis. (Of course, you have no control over any of the options that you selected when generating the 'Hello' web service – such as the encoding style, which will be investigated in Part 6.) This dynamic approach to describing a service is not a general principle behind web services, but it is useful to appreciate this possibility because it means we don't have to actually deploy an independent WSDL.

There are various reasons that you might want to have a separate, static WSDL; for example, so that you can tailor the options you require or so that you can have more than one WSDL in different locations describing your service.

In the following sections I am going to show you how to use Eclipse to generate just a simple client for your local 'Hello' web service if you wish, and how to do so for a remotely hosted version of the same service. In this second case the web service in question is running on top of the Axis server platform and has a WSDL document that is generated by an HTTP request. For Eclipse when generating a client there is no real difference between a local WSDL description, a remote static description or a remote dynamically generated description.

OU demo services

At the OU there is a small set of toy web services. These can be accessed using a client in the same way as you tested the 'Hello' web service using Eclipse. In fact, one of the web services hosted is a copy of the 'Hello' service.

Here you will use a copy of the 'Hello' web service that is hosted at the OU as the service to be accessed by the client. You can just as easily access other services on the Internet or your local implementation of the 'Hello' service in the same way.

When you go to this location, if you do not already have a log-in session running then you will be redirected to a log-in page and asked to log in to the University network. Log in with the same username (OUCU) and password that you use to access the T320 course web site.

Creating a dynamic web project

Start your Block 3 Eclipse and make sure you have a Tomcat server configured for use (see the guide *Configuring an Application Server in Eclipse* if you are unsure how to do this). Then create a new dynamic project using File > New > Project… (Figure 3).

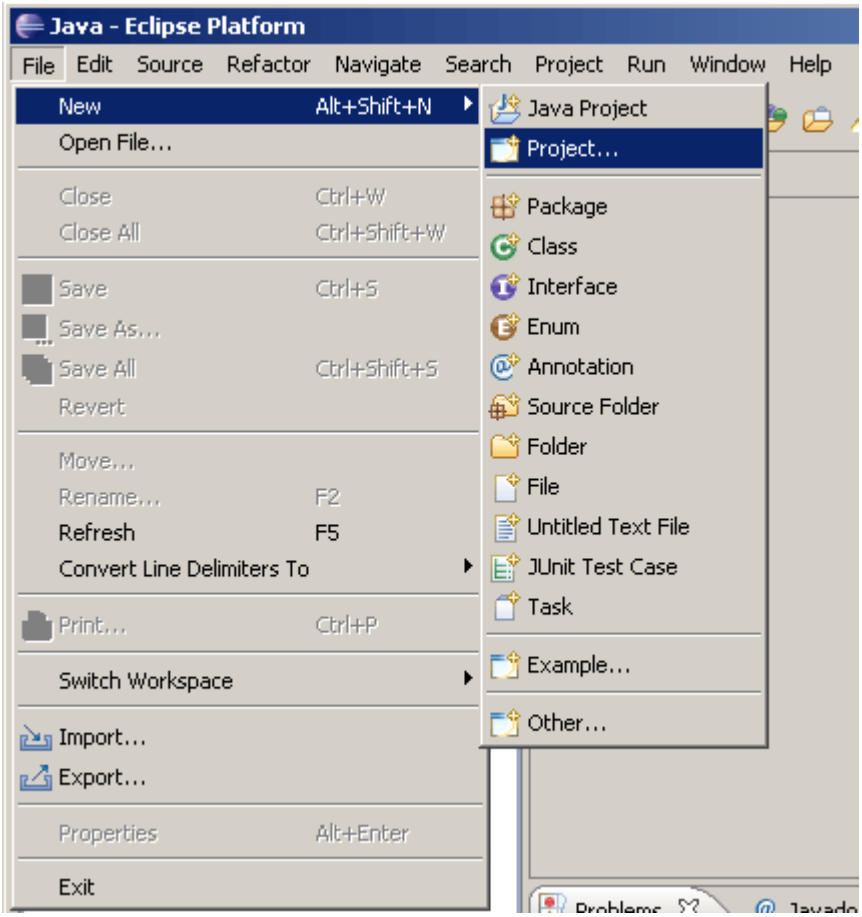

Figure 3 Selecting a new project

Then, in the New Project dialogue box, expand the 'Web' option and select 'Dynamic Web Project' (Figure 4) before clicking the 'Next' button.

Then give your project a name, such as 'HelloServiceClient', and click 'Finish' (Figure 5).

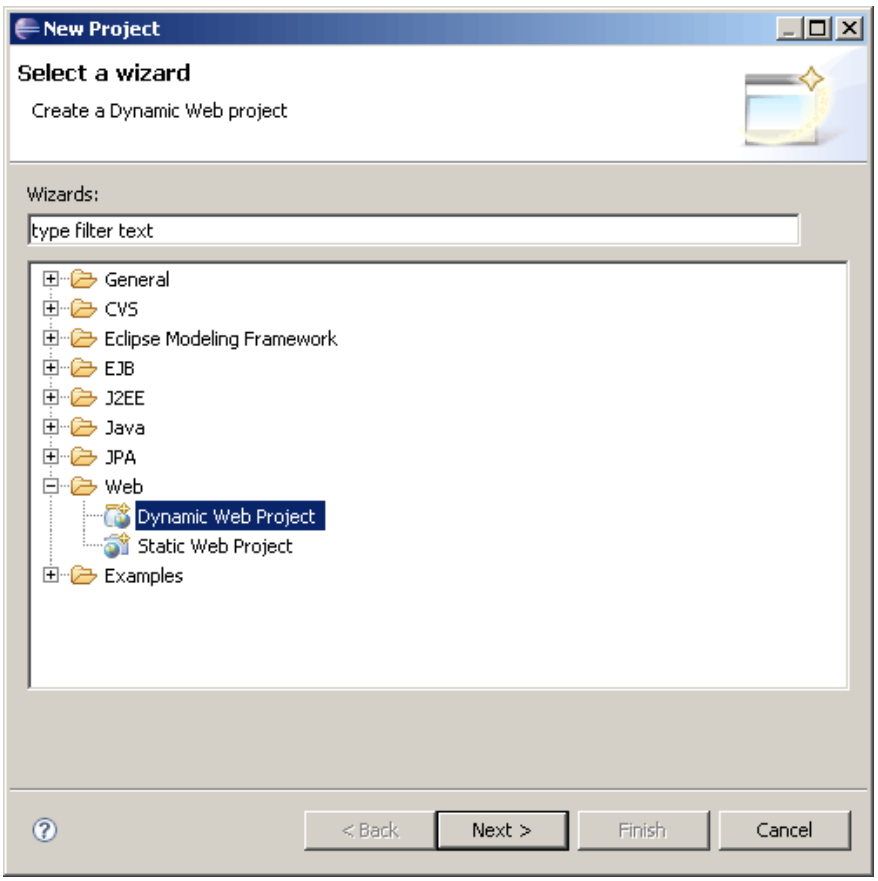

Figure 4 Dynamic web project selection

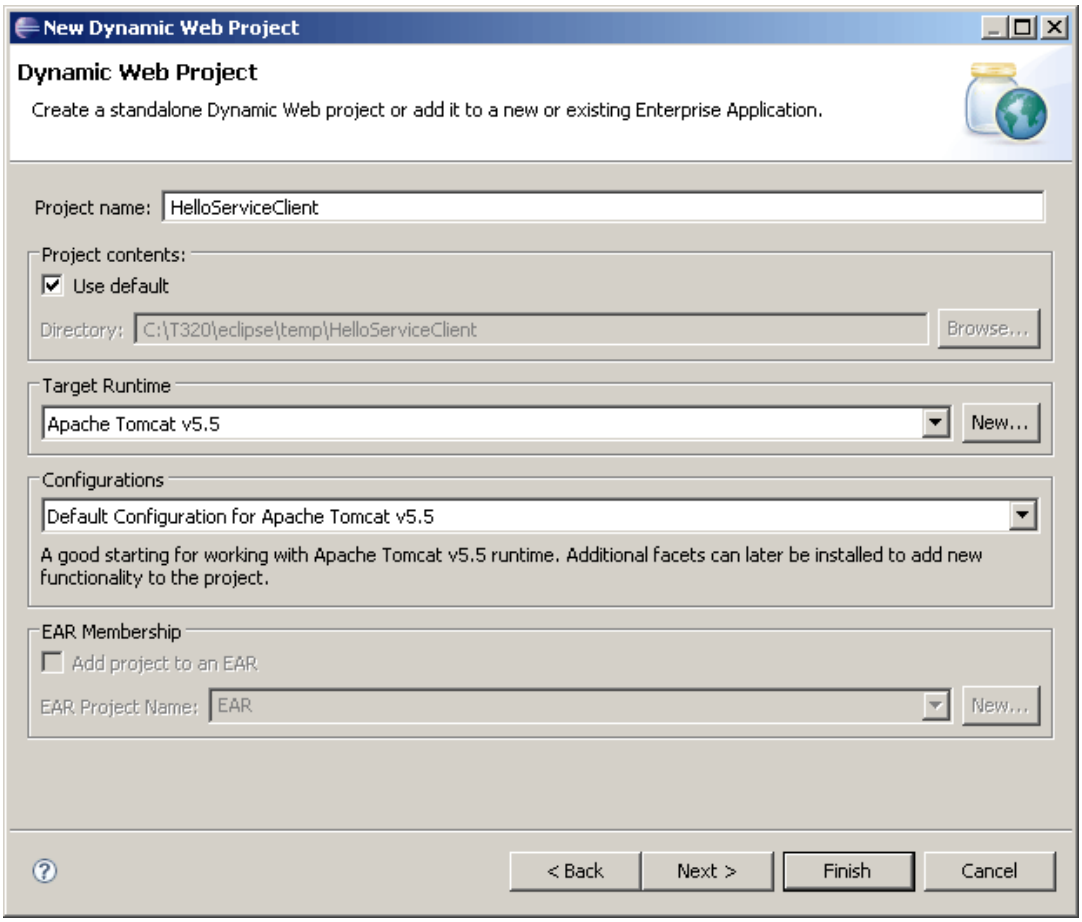

Figure 5 Naming the new project

As Eclipse builds the new project, you will most likely be asked if you wish to open the project in the Java EE perspective (Figure 6). You should click on 'Yes' when this is offered.

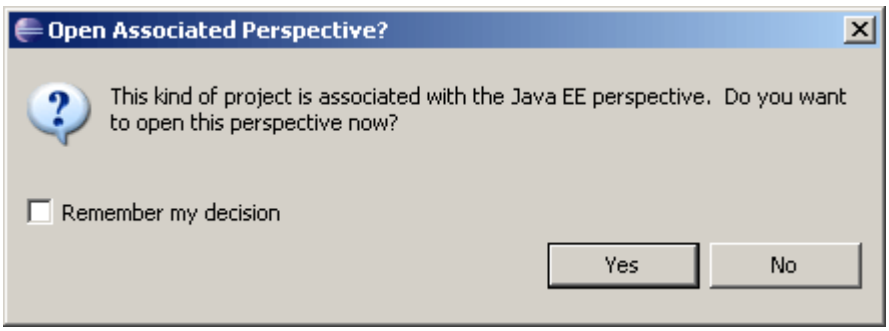

Figure 6 Dialogue option to open Java EE perspective

After a time, the newly created project will be displayed in Eclipse (Figure 7).

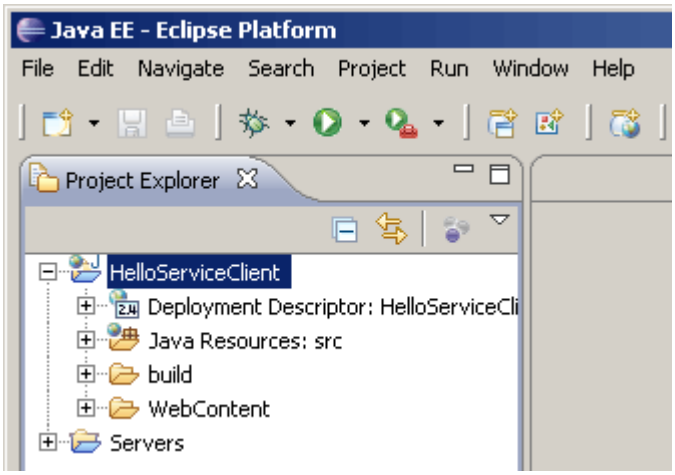

Figure 7 Newly created project in the Project Explorer

Adding a simple client

Next you need to add a web service client to the project. To do this, select the project in the Project Explorer and right-click on it. Then select New > Other… as shown in Figure 8.

You should then be presented with the dialogue box shown in Figure 9; here, you should open the 'Web Services' folder and select the 'Web Service Client' option before clicking on the 'Next' button.

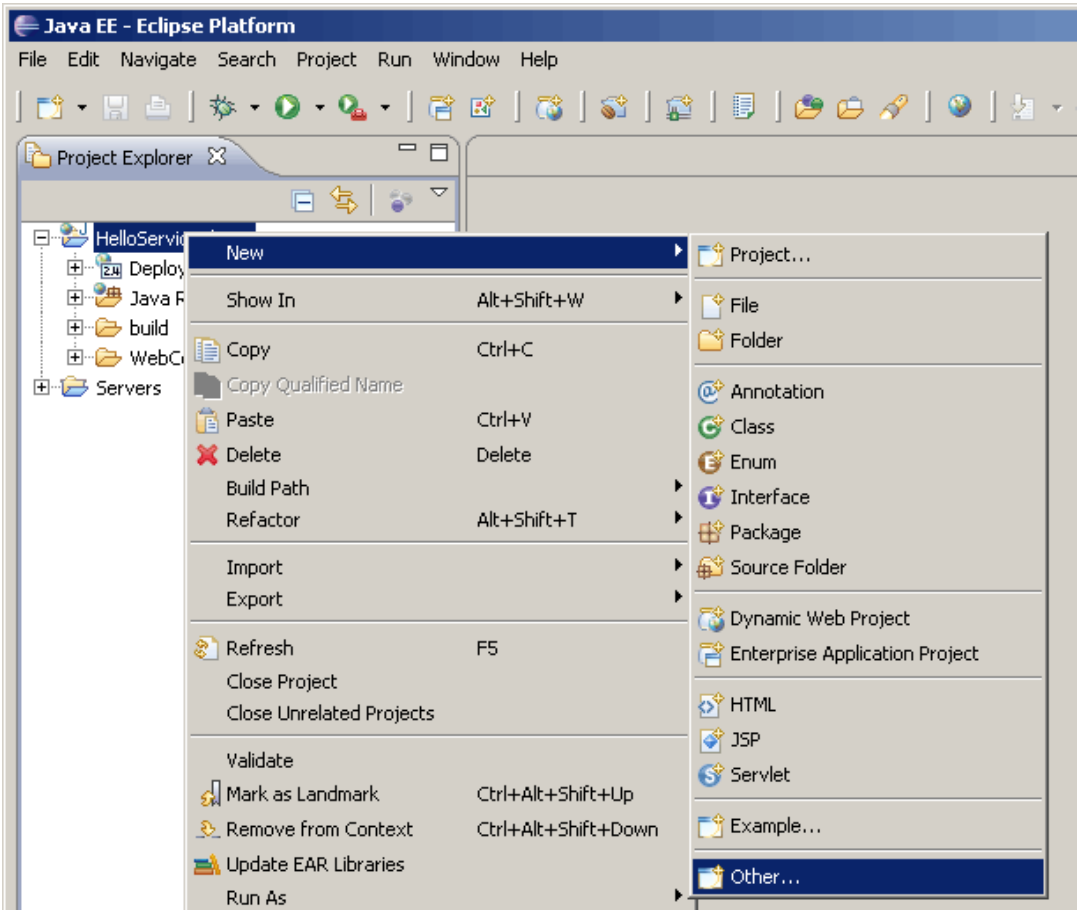

Figure 8 Selecting New > Other… to add a client to the project

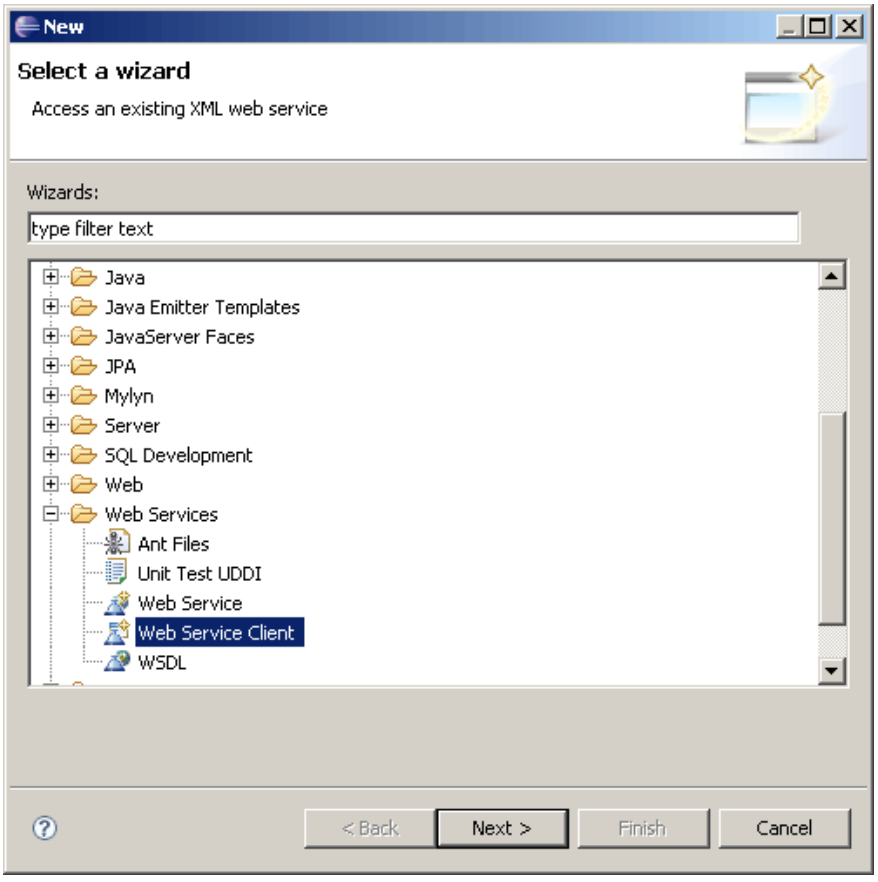

Figure 9 Selecting a wizard to generate a web service client

You will then be presented with the dialogue box shown in Figure 10. You may well recognise this as including much of the lower section of the dialogue box shown in Figure 16 of the *Implementing a simple web service* activity that you completed in Part 1 of the block. There are a few differences, the most significant being that here there is a need to specify a 'Service definition'.

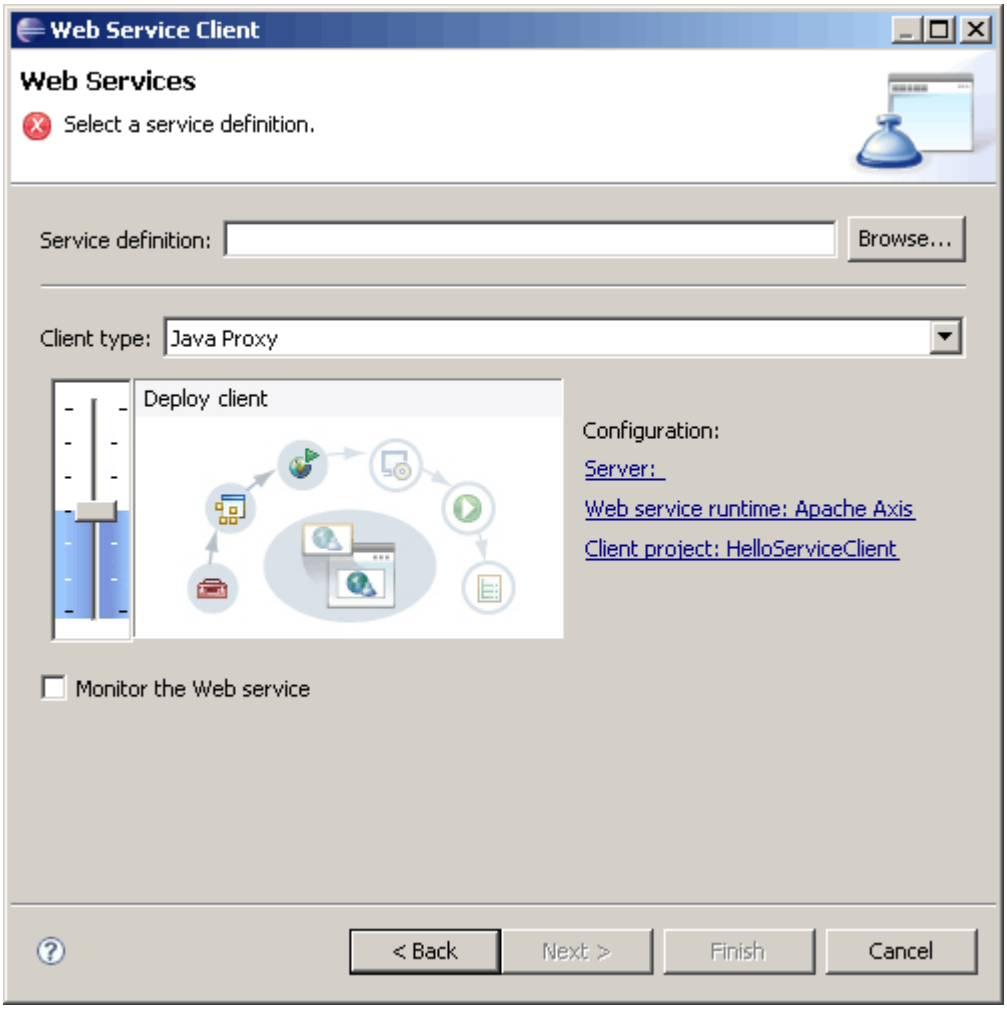

Figure 10 Initial client configuration dialogue

Click on the 'Browse' button at the top of the box. This will take you to the dialogue box shown in Figure 11. Here the topmost text box has the label 'Enter a URI to a WSDL, WSIL[1] or HTML document'. A URI can be entered, which might be the location of a file on your machine or, for example, an HTTP URL. So there is the flexibility to use a WSDL document that resides on your machine or one that can be acquired from the Internet over HTTP (or FTP etc.).

When you created the 'Hello' web service, there was at this stage a slightly different text box to complete that requested a 'service implementation'. This you filled with the name of the Java class that implemented the web service logic. Eclipse (and the WTP) then used that code to create the WSDL, which it placed inside your project and then used to generate the client. If you have the 'Hello' project handy (and deployed to your local Tomcat server) then you could point this new client at that local 'Hello' web service.

[1] Web Services Inspection Language (WSIL), a joint effort by Microsoft and IBM, is a different approach to describing web services; see, for example, **http://www.ibm.com/developerworks/library/specification/ws-wsilspec/**.

The chief difference between WSDL and WSIL appears to be that WSIL lists groups of web services and their endpoints in an XML format rather than describing a single service.

Instead of that, here I am going to create a client that uses a remote implementation of the 'Hello' web service over the Internet.

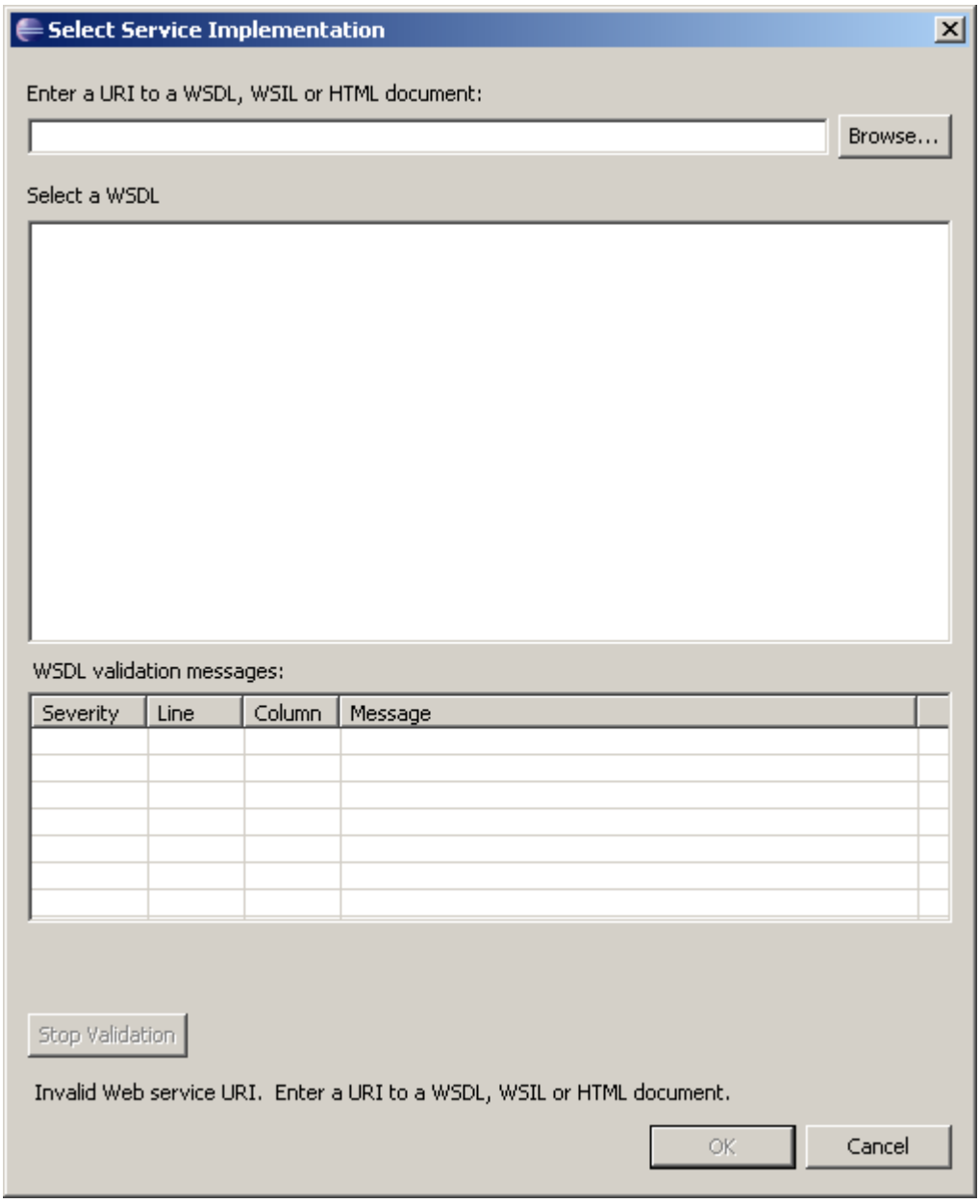

Figure 11 Select Service Implementation dialogue box

I shall look at the sample web service hosted at the University at the end of this booklet. For now, you should note that the URL for the EPR of the 'Hello' web service at the OU is:

http://t320webservices.open.ac.uk/t320/services/HelloService

Axis will generate the WSDL document describing a web service if we postfix the EPR with a parameter 'wsdl'. So using a '?' to append this to the URL, we can acquire the WSDL for the 'Hello' service using:

http://t320webservices.open.ac.uk/t320/services/HelloService?wsdl

You can view the WSDL document by copying this URL into a web browser (Figure 12).

Now enter the URL for the WSDL in the Eclipse Select Service Implementation dialogue box. When you have done this, you should see the URL listed in the list of WSDLs under the text 'Select a WSDL' (Figure 13).

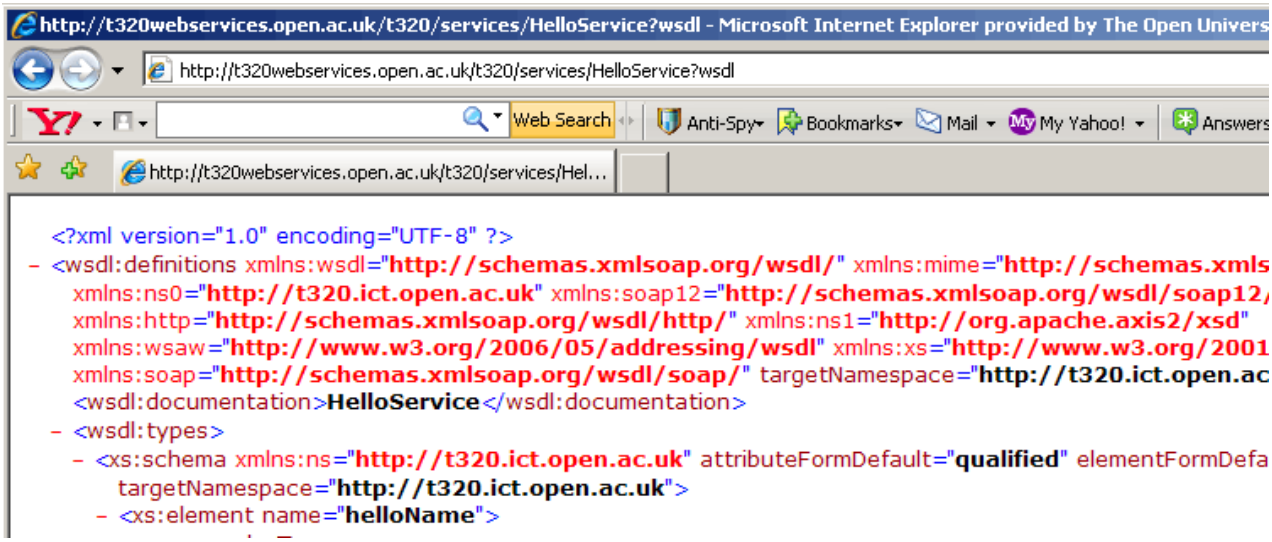

Figure 12 Hello service WSDL listed in a web browser

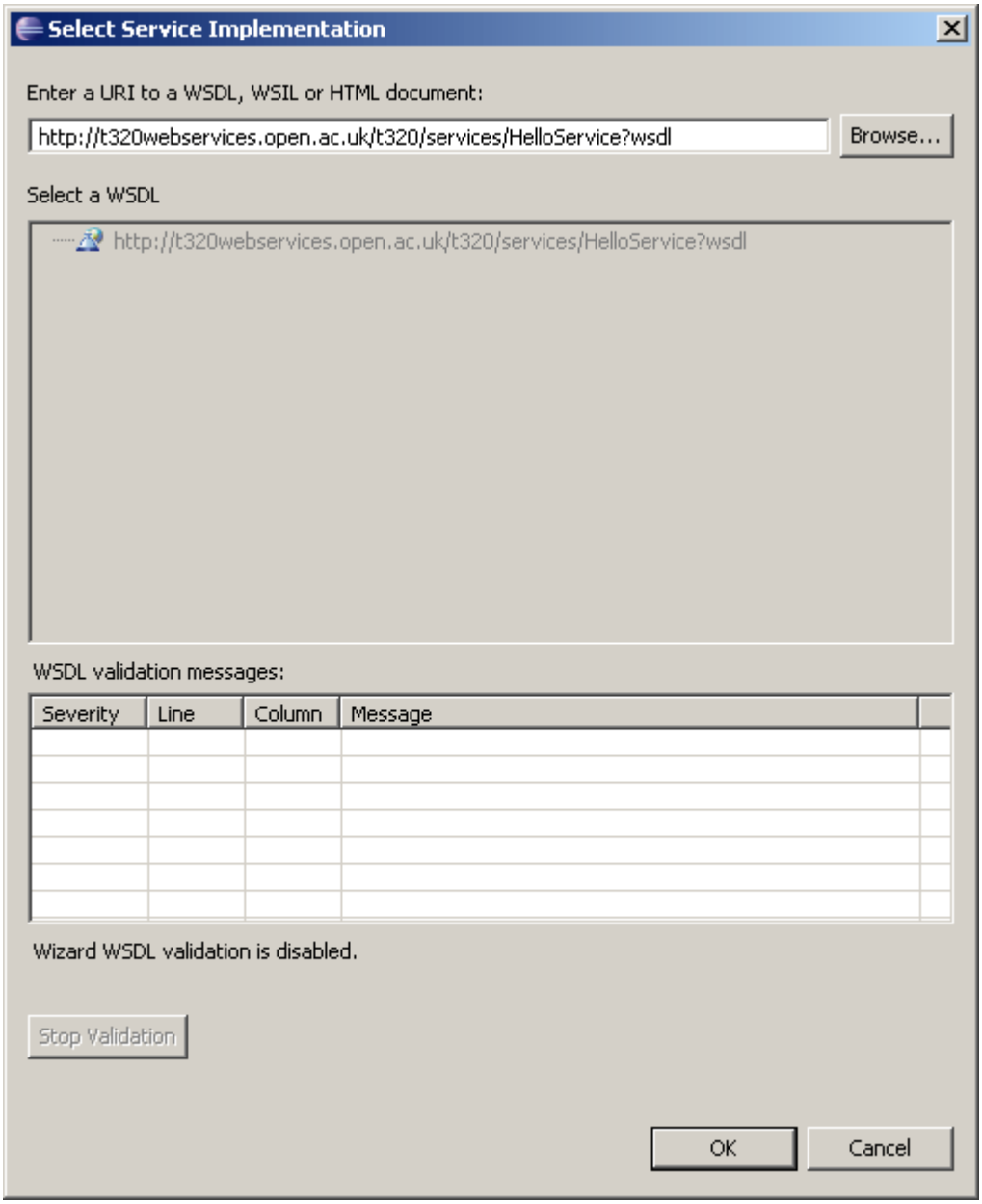

Figure 13 'Hello' service WSDL listed in dialogue box

At this point it is possible that you will receive a message in a pop-up dialogue box telling you that the WSDL cannot be retrieved. This generally means that the web service is not available. If this is the case, you should raise the matter in the course forums so that the services can be fixed by University staff.

If the WSDL is listed then click on the 'OK' button. This will take you back to the client configuration dialogue box (Figure 10).

Next raise the slider on the left of the dialogue box so that 'Test client' is displayed (i.e. you want to create a test client), and tick the 'Monitor the Web service' box if you wish to observe messages sent to and received from the web service (Figure 14). Then click the 'Finish' button.

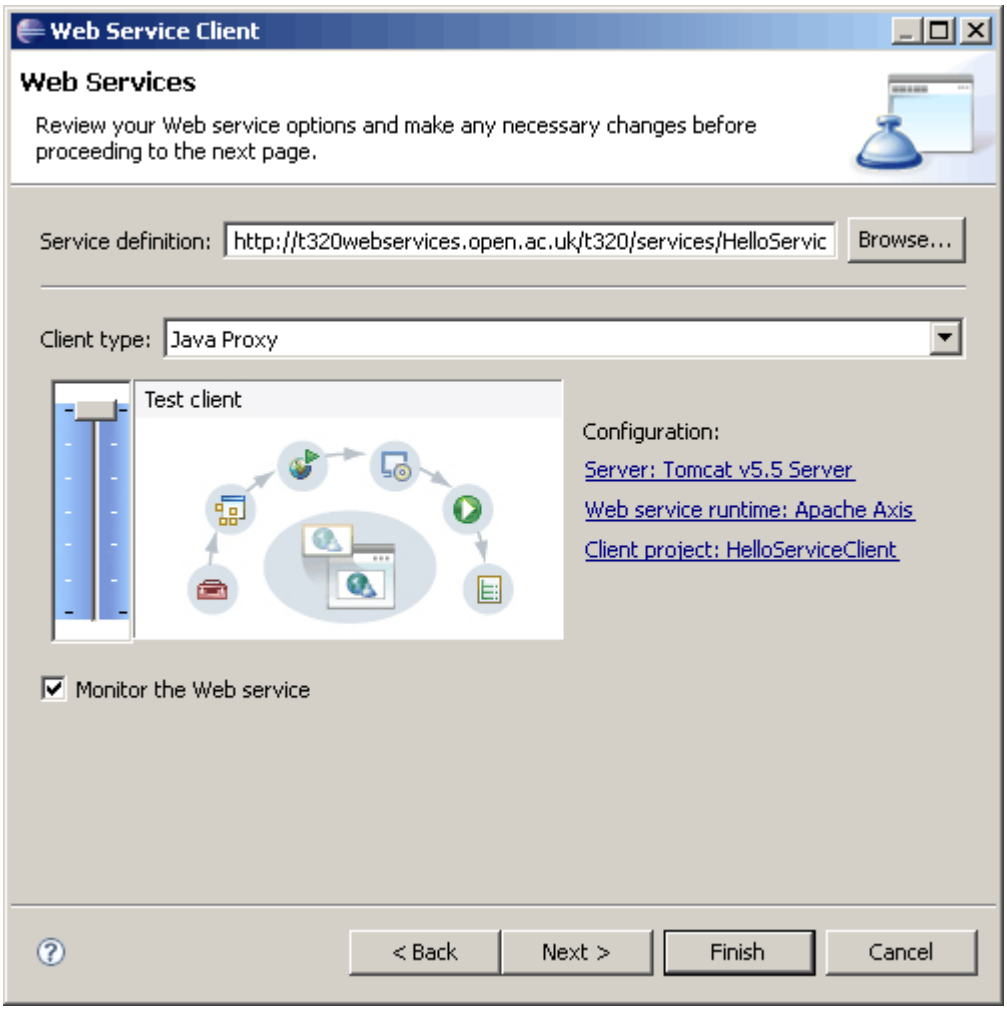

Figure 14 Completed client configuration dialogue box

After a time, the client will be generated and you will see that the now familiar web pages of the client are shown in Eclipse (Figure 15).

As before, you can use the web service by first clicking on the 'HelloName' method in the Methods panel, then entering a name in the text box that appears in the Inputs panel, and finally clicking the 'Invoke' button (Figure 16).

Everything in the client appears as it was when you created and tested the 'Hello' web service locally in Part 1. Of course, there is a real difference behind the scenes. The SOAP request to the web service is being sent across the Internet to the OU server 't320webservices.open.ac.uk', and the response is being sent back in the same way from the remote machine.

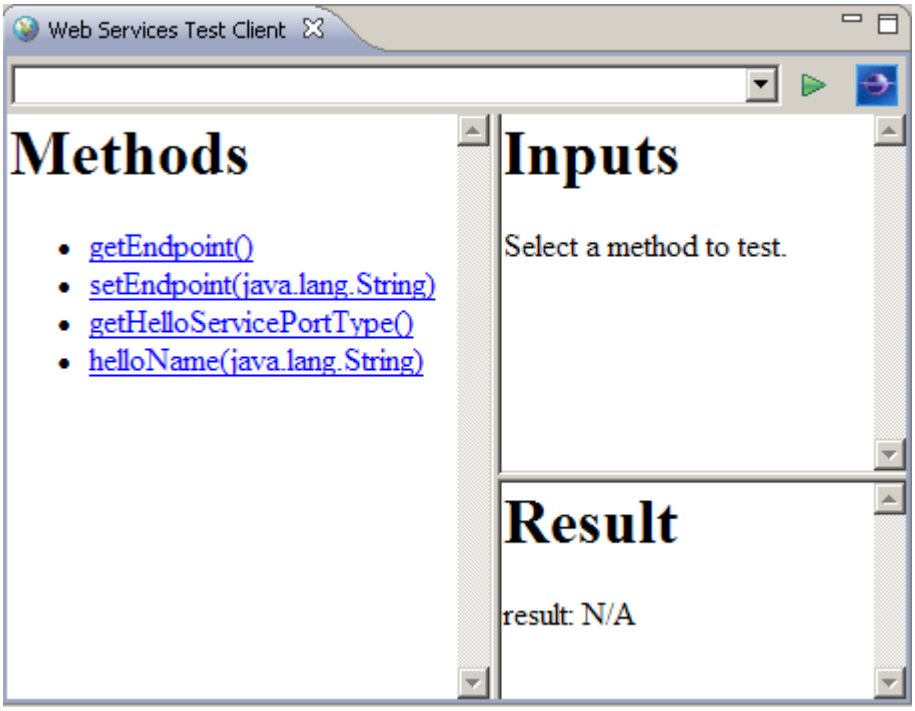

Figure 15 Initial client web pages in Eclipse

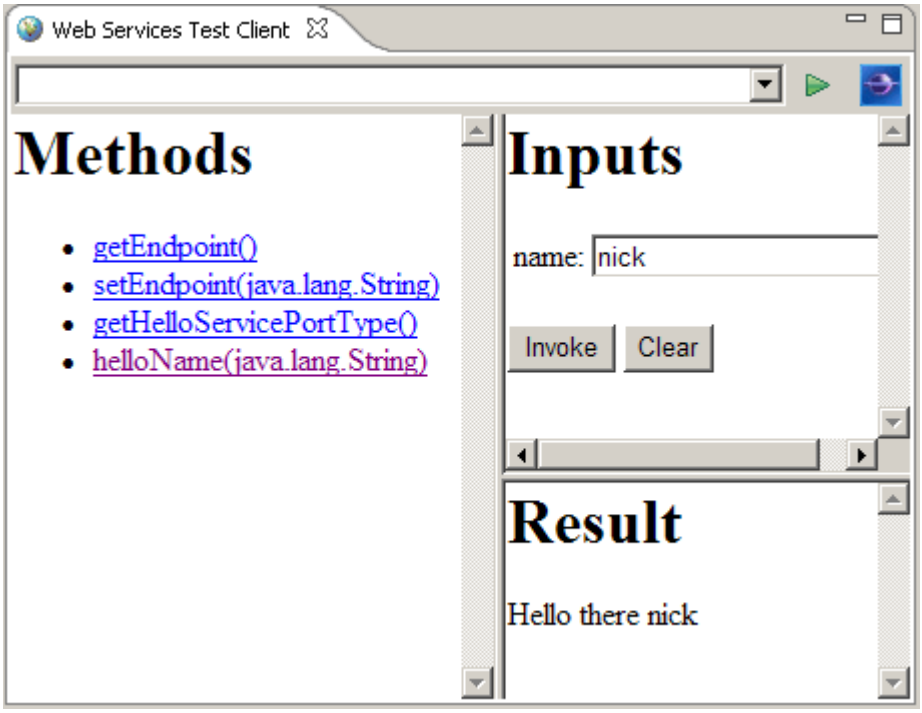

Figure 16 Client after invoking the 'Hello' service

More clients

You will find the 'Hello' service and others listed on the OU T320 Axis site at:

http://t320webservices.open.ac.uk/t320/services/listServices

You will find that you can access any of these services using the associated WSDL document that is generated by appending '?wsdl' to the EPR for the service.

Try out at least one other service to see how this works.