5.4 DEVELOPING A WEB SERVICE

Use Web services tools to discover, create, and publish Web services that are created from Java beans, enterprise beans, and WSDL files.

Creating a Web Service from WSDL

Start from WSDL to build the web service to implement a web service that is already defined either by a standard or an existing instance of the service. In either case, the WSDL already exists. The JAX-WS import tool processes the existing WSDL document, either from a local copy on disk or by retrieving it from a network address or URL. When developing a web service starting from an existing WSDL, the process is actually simpler than starting from

Java. This is because the policy assertions needed to enable various WSIT technologies are already embedded in the WSDL file.

To create a web service from WSDL, create the following source files: WSDL File, Web Service Implementation File, custom-server.xml , web.xml, sun-jaxws.xml, build.xml, build.properties

The following files are standard files required for JAX-WS. custom-server.xml, sun-jaxws.xml and web.xml

The build.xml and build.properties files are standard in any build environment.

WSDL File

```
<wsp:Policywsu:Id="AddNumbers_policy">
<wsp:ExactlyOne> <wsp:All> <wsrm:RMAssertion>
<wsrm:InactivityTimeout Milliseconds="600000"/>
<wsrm:AcknowledgementInterval Milliseconds="200"/>
</wsrm:RMAssertion> </wsp:All> </wsp:ExactlyOne> </wsp:Policy>
```

Web Service Implementation File

AddNumbersImpl.java

```
package fromwsdl.server; import javax.jws.WebService;
import javax.jws.WebMethod;
@WebService (endpointInterface= "fromwsdl.server.AddNumbersPortType")
public class AddNumbersImpl{
  @WebMethod(action="addNumbers")
  public intaddNumbers (int number1, int number2)
```

```
        throw new AddNumbersFault_Exception(message, fault); }
      return number1 + number2; }
    public void oneWayInt(int number) {
  System.out.println("Service received: " + number); } }
```

Publishing a Web service

The Web service, also known as the business service, describes a Web service's endpoint and where its WSDL file resides. The WSDL file lists the operations that service provides.

Prerequisites: Register with a registry, Launch the Web Services Explorer, Add the registry to the Web Services Explorer, Create a Web service, Deploy the Web service, Publish a Business Entity.

Web services are published using two different publication formats: simple and advanced.

Publish a business service using the simple option

- In the Web Services Explorer, select UDDI Main and select the registry to which the users want to publish the business entity.
- In the Actions pane toolbar, click the Publish icon Picture of the Publish icon.
- Select Service and select the Simple radio button.
- Enter the publish URL, your user ID, password, WSDL URL, service name, and service description in the respective fields.
- Click Go to publish your business entity.

Ensure that you select the service document, since the service element is the basis for the Business Service that you will publish. The Web Services Explorer is automatically updated with your published Web service. The registry contains pointers to the URL of the WSDL service document of the Web service. Businesses can now discover and integrate with your Web service.

Publish a Web service using the advanced option:

In the Web Services Explorer, select UDDI Main and select the registry to which the users want to publish the business entity.

- In the Actions pane toolbar, click the Publish icon Picture of the Publish icon.
- Select Service and select the Advanced radio button.
- Enter the publish URL, your user ID, password, and WSDL URL in the respective fields.

- Click Get or Find to associate the service with a business entity.
- Click Get or Find to associate a specific service interface with the service.
- Click Add to create service names.
- Click Add to create service descriptions.
- Click Add to create categories. Enter your service categories. Select a category type from the drop down list.
- Click Browse to open the Categories pane.
- Navigate through the hierarchical taxonomy and select the appropriate classification for your business service, then exit the Categories pane.
- Click Go to publish your business entity.

Testing a web service

The loosely coupled nature of web services and non-existence of a User interface present a challenge to the developers and testers alike. Following are some of the challenges that web service testers have to face: Scalability and Security, Absence of User Interface, Distributed across network and Testing the service

Types of testing

As with traditional applications, there are different sorts of testing that are needed to be carried out in case of web services.

- **Proof of concept Testing**

  Web service is a new concept and because of this we need to make sure that the architecture that we have chosen for our application is a correct one.

- **Functional testing**

  Web service is designed to solve a business problem. It has a predefined function to perform. This type of testing validates whether the service performs the intended function correctly, does it handle the exception conditions gracefully and does it handle the boundary value conditions.

- **Regression testing**

  Regression testing aims to ensure that the web service is still working across builds or releases. This sort of testing needs to be carried out during each release; hence it is an ideal candidate for automation.

- **Load testing**

  Load or stress testing is a test of the performance of the web service when many simultaneous users are accessing the system. The response of the web service must be

consistent and also its performance must not degrade with the increase in the number of users.

- **Unit testing**

  Unit test cases must be written before the application is developed. As and when the application is built, test cases are applied on the code. Hence the functionality is verified as and when we develop the web service.

- **Basic testing**

  The main aim of this testing is to test whether the web service is accessible and can be invoked properly. Main focus in this phase should be to carry out the following procedures.

  - Get the WSDL file and test whether it is well-formed and in compliance with the WSDL specifications published by W3C
  - Using this WSDL file generate the client side stubs that handle the interaction with the web service.
  - Test the web service functionality that is whether the webs ervice responds to the requests submitted to it correctly.
  - Invoke the sample invoker by passing it the parameters required by the web service. Check the response of the web service from a functionality point of view.
  - The sample invoker calls the client stubs which further call the web service
  - The stub constructs the SOAP message from the parameters passed to it and passes this message to the service. This message can be monitored by a Sniffer program like TCP Monitor.
  - If there are any security checks, like username and password we need to test their effectiveness. The intent of this step should be to break in the system and gain unauthorized access.

- **Testing SOA**

  As organizations create a web service interface to their systems and overcome security issues, they will be able to exchange data with business entities such as customers, suppliers and partners in a more uninhibited and loosely coupled manner. For testing such collaborating web services we need to focus on the following:

  - In a system where web services interact with each other, we need to test the 'publish', 'find' and 'bind' capabilities of the constituent web services.
  - A particular SOAP message may typically have a designated recipient, but may also have one or more intermediaries along the message route that take actions based upon the instructions provided to them in the header of the SOAP message. Web services testing must verify the proper functionality of these intermediaries also.

- **Interoperability testing**

  In the loosely coupled environment of a service-oriented architecture, separate sources don't need to know the details of each other's working, but they need to have enough common ground for reliably exchanging messages without error or misunderstanding. Standardized specifications help in creating such a common ground, but differences in implementation may still cause problems in the communication. Interoperability is when services can interact with each other without encountering such problems.