CRUD Operations Using JSF, Web Services and JSF

Skill Level: Intermediate

Kamesh Pandrangi (kamesh.pandrangi@in.ibm.com)
System Engineer
IBM

30 Oct 2009

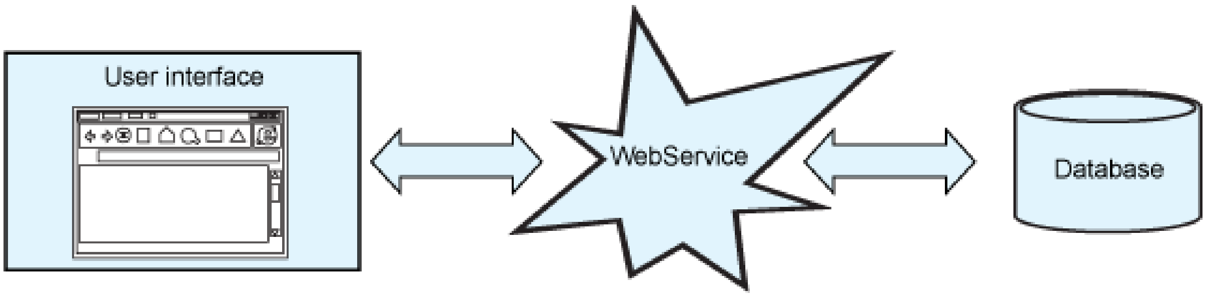
This article explains the use case of adding, updating, searching and deleting using JSF, Web Services and OJB in WebSphere Integration Developer version 6.1. The article discusses the steps in developing each layer including the integration among these layers.

Introduction

With the introduction of Web services as a place holder for core business logic based on SOA platform enabled the enterprise application development in an independent layer structure thus making it easy for development and maintenance as well.

In this article we will discuss the development of each layer and the interaction among layers; resulting in an end-to-end enterprise solution. CRUD operations being the basic operations for any kind of enterprise business, addressing these areas gives an overview for the designing and developing the solutions based on SOA. To answer the technical challenges and show the capabilities JSF is used for developing the User Interface layer and OJB for handling the data at data layer.

Figure 1. Web service



Java Server interface

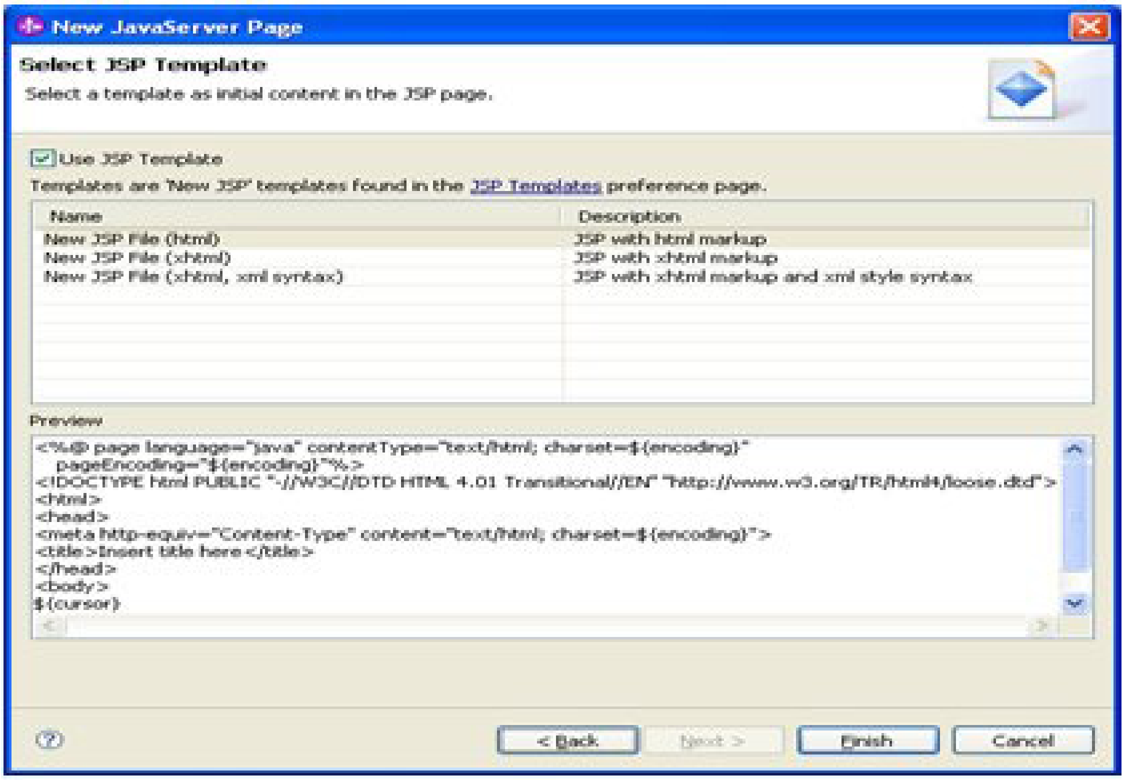
The user interface (UI) layer is developed using Java Server Faces Technology. It includes:

- A set of APIs for representing the UI components and managing their state, handling events and input validation, converting values, defining page navigation, and supporting internationalization and accessibility
- A default set of UI components
- Two JavaServer Pages (JSP) custom tag libraries for expressing a JavaServer Faces interface within a JSP page
- A server-side event model
- State management
- Managed Beans (JavaBeans created with dependency injection)
- Unified Expression Language for both JSP 2.0 and JSF 1.2

Creating a JSF project is similar to the creating a normal web project in WebSphere Integration Developer (WID) 6.1. The user friendly features of WID enable the user to easily create Web projects. Below, we begin by creating a Dynamic Web project. The following steps begins the process of building the JSF project in WID 6.1.

- Right-click on Project Explorer > Select New > Select Dynamic Web Project
- Specify the Project name and EAR name
- Check the required Project Facet like JSTL, Basic Facet Support that enable us to use the features of built in functionality
- Specify the Context Root, Content Directory & Java Source Directory
- Create a new JSF page by Right clicking on Web Content > New > JSP
- Specify the JSF page name

Figure 2. Importing JSP templates



Select JSP Template

Select a template as initial content in the JSP page.

[✓] Use JSP Template

Templates are 'New JSP' templates found in the JSP Templates preference page.

Name	Description
New JSP File (html)	JSP with html markup
New JSP File (xhtml)	JSP with xhtml markup
New JSP File (xhtml, xml syntax)	JSP with xhtml markup and xml style syntax

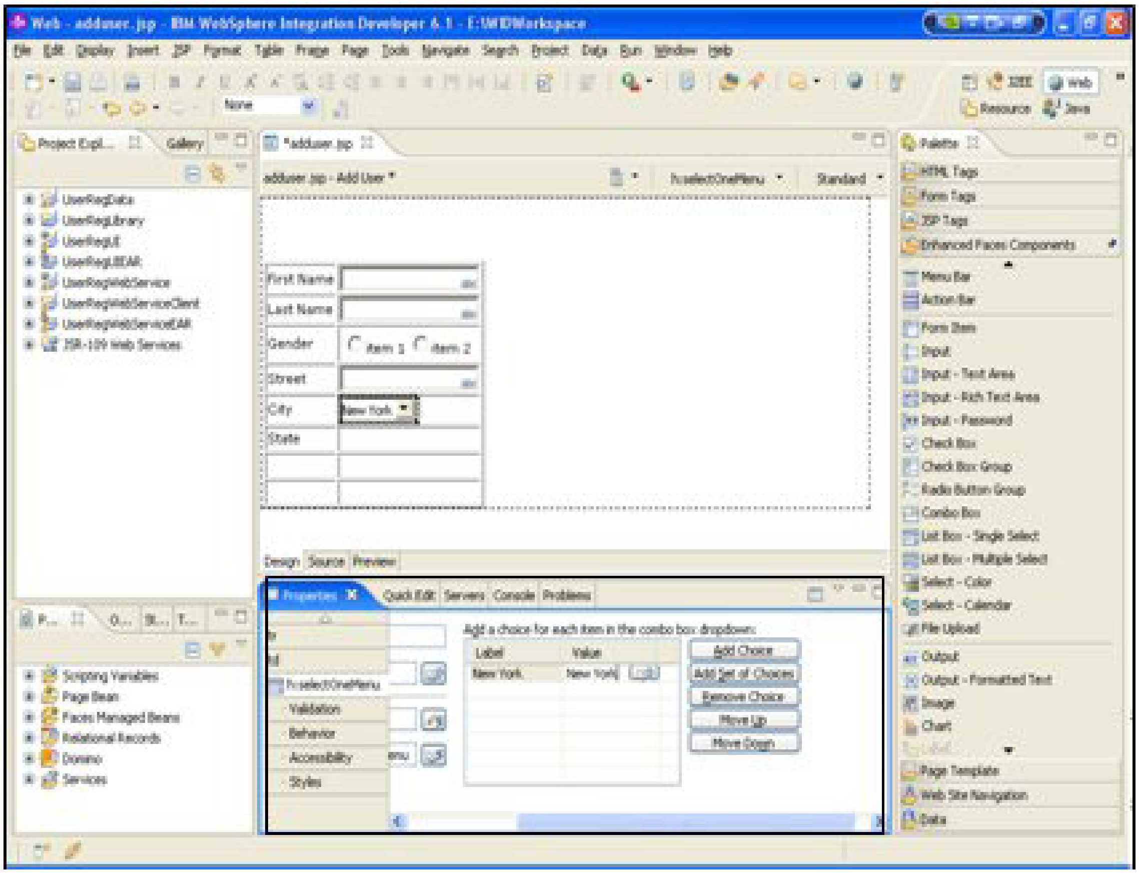
Preview

```
<%@ page language="java" contentType="text/html; charset=${encoding}"
    pageEncoding="${encoding}"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=${encoding}">
<title>Insert title here</title>
</head>
<body>
${cursor}
```

Open the page in design view and add the required components from the Enhanced Faces Components. These enhanced Faces Components provide a rich a set of widgets that are normally used in developing applications such as a drop down list, selection list, data table, and panels. The biggest advantage with JSF is addressing the many user interface components in basic form and connecting them with the relevant operations in the JavaBean. To make a good look and feel we will build the components in a table to view the alignment and position of fields in a consumable way.

- Click on the HTML Tags > Select Table > Specify the number of rows and columns
- Add the required labels for the form fields
- Click on Enhanced Faces Components > Select Input > Drag into required cell
- Repeat the process to add other components

Figure 3. Properties of JSF components

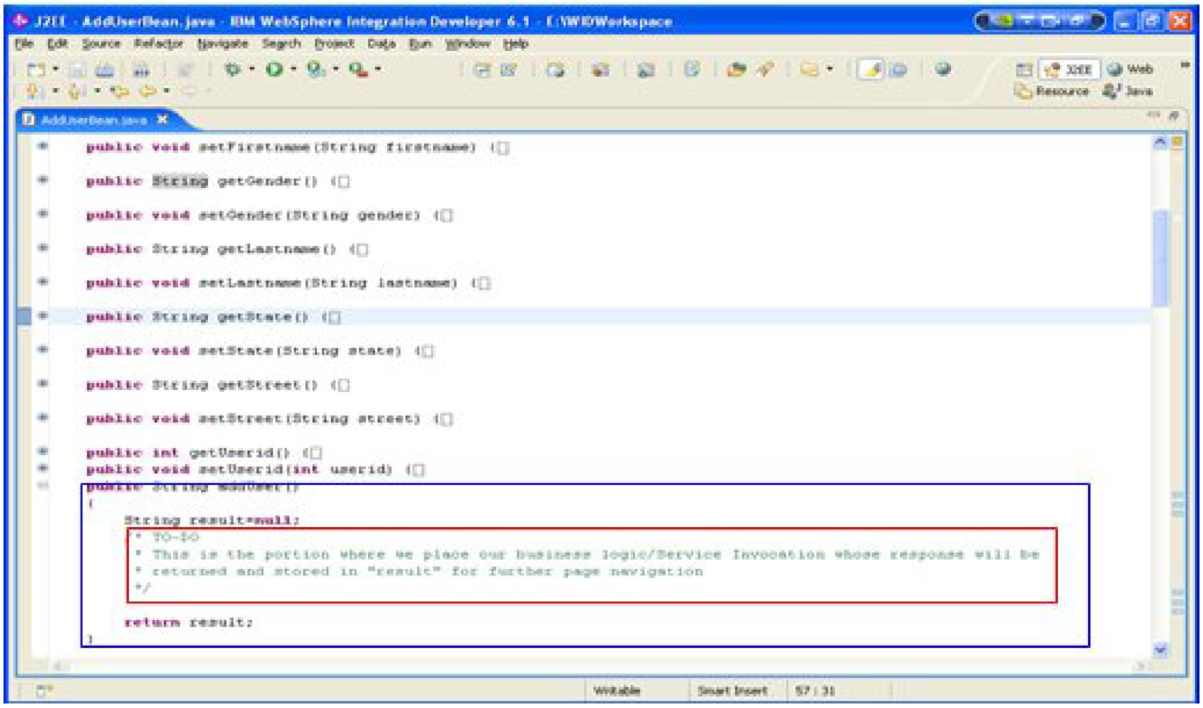


Dpecify the properties of a component by clicking on the Properties tab.

Once we are finish with designing the page using the widgets and other components. We then look into the next part of mapping the components to the relevant properties at the JavaBean. Within the JSF we call it as managed bean that holds the properties and provide operations on them, the following steps describe how to start up with the managed bean.

- Create a managed bean with required properties
- Select the property > right-click > Generate Getters & Setters
- Create a method and specify the required operation (either the actual business logic or Service Invocation), Figure 4 shows how we work with managed beans.

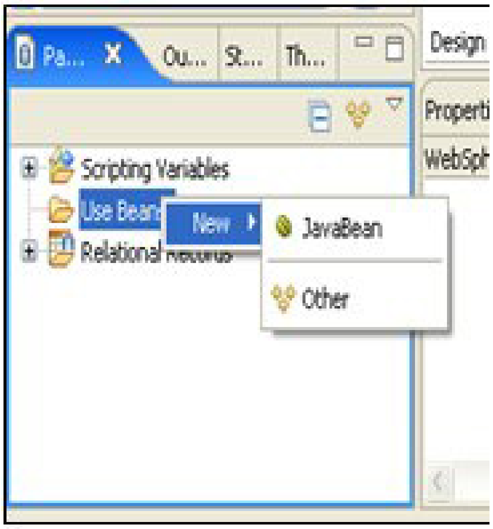
Figure 4. Managed Bean properties and operations



```
public void setFirstname(String firstname) {
public String getGender() {
public void setGender(String gender) {
public String getLastname() {
public void setLastname(String lastname) {
public String getState() {
public void setState(String state) {
public String getStreet() {
public void setStreet(String street) {
public int getUserid() {
public void setUserid(int userid) {
public String addUser()
{
    String result=null;
    /* TO-DO
     * This is the portion where we place our business logic/Service Invocation whose response will be
     * returned and stored in "result" for further page navigation
     */
    return result;
}
```

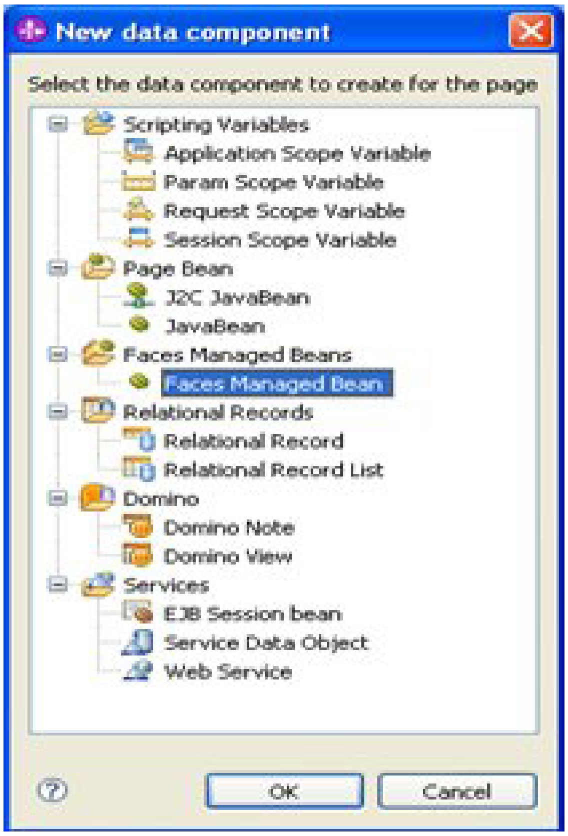
J2EE - AddUserBean.java - IBM WebSphere Integration Developer 6.1 - E:\WIDWorkspace
File Edit Source Refactor Navigate Search Project Data Run Window Help
AddUserBean.java
J2EE
Web
Resource
Java
Writable
Smart Insert
57 : 31

Figure 5. Page data



In Page Data right click on Use Bean > New > JavaBean.

Figure 6. Data component

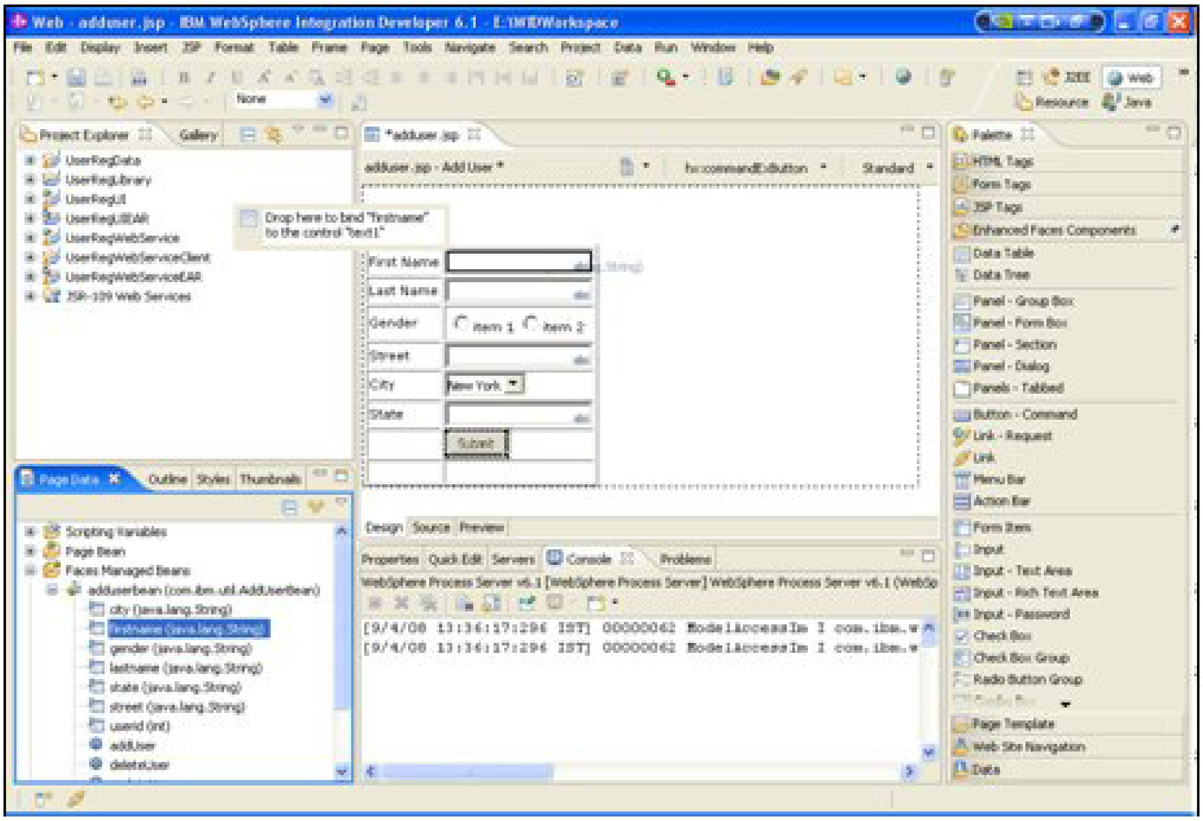


Selection of the faces managed bean

Once the managed bean is created with the required properties and method innovations, we specify the name for the managed bean with its corresponding class. Depending on the functionality and usage of the class we make this bean as reusable by specifying its scope. These are reflected in *faces-config.xml*.

This new bean definition is completed within th bottom-left panel called *Page Data* where we select the Faces Managed Bean, once you refresh the Faces Managed Beans in Page Data, you can view the bean with its available variables and methods. Using Websphere Integration Developer directly map the bean properties to the fields that are designed and developed in JSF page. Figure 7 shows the drag and drop functionality provided in WID.

Figure 7. Drag & Drop bean properties



```
[9/4/08 13:36:17:296 IST] 00000062 ModelAccessIm I com.ibm.w
[9/4/08 13:36:17:296 IST] 00000062 ModelAccessIm I com.ibm.w
```

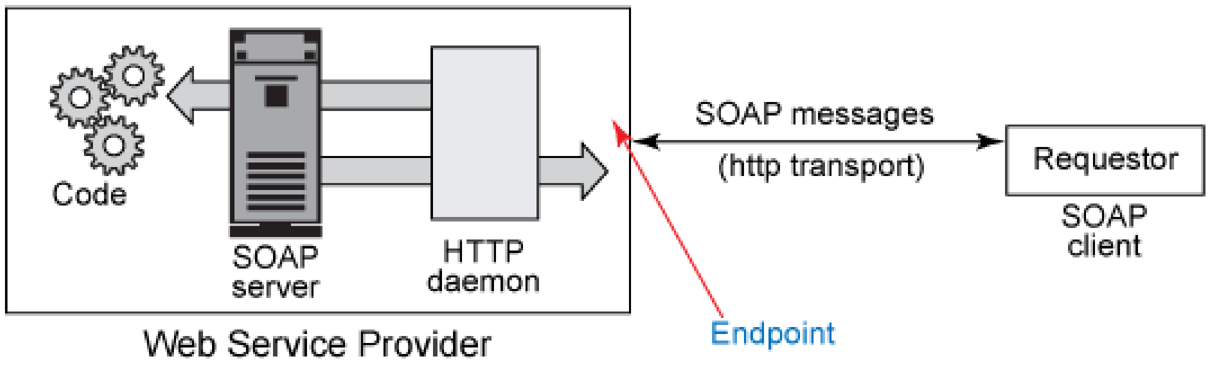
Similarly you can also map the properties and operations for other operations of Retrieve, Update and Delete. Once the four operations are complete deploy the EAR onto the server and test it using the test clients that will push the data from User Interface to the next layer.

- Map the variables to the corresponding field in Design View
- Map the method to be triggered on Submitting the form to the Submit button

Using Web services

Web services form an interoperable layer that connects the user interface and database layer.

Figure 8. Overview of Web services



Services are defined using six major elements:

1. **Types**, which provides data type definitions used to describe the messages exchanged.
2. **Message**, which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
3. **PortType**, which is a set of abstract operations. Each operation refers to an input message and output messages.
4. **Binding**, which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
5. **Port**, which specifies an address for a binding, thus defining a single communication endpoint.
6. **Service**, which is used to aggregate a set of related ports.

The Web service creates an interoperable layer that connects the user interace and the database layer. This enables us to store the business logic and define rules as and when and re-deploy only this layer whenever changes come. It provides us the flexible way to upgrade the UI and back end layer without disturbing core business logic.

We follow three major steps in creating a Web service here:

1. Creating a library that stores the Data Objects and WSDL's
2. A Web project to store the generated java skeletons
3. A java project to store the generated client

Creating the library

The creation of a library and storing the created WSDL helps us to share the library with other projects and provides a common place for managing them. This library is also used to store the data objects that are used across the project. So at any point of time any updates on these data objects can be made at a single place to avoid the conflict in other projects.

Depending on the type of operations and the data used in the project the data objects are defined. In the day to day business operations the data is wrapped in the meaningful form of objects that are transferred in XML format. We have many XML parsers that will wrap and unwrap these complex objects and serve the business need. Since these data objects are defined in a common place like library and imported by interface, they can be updated as an when required without changing the actual wsdl definition.

The following steps provides guidance with the creation of library and Data objects:

- Select Library and click Next
- Create a new XSD by Right clicking on Web Content > New > Other > XML > XMLSchema
- Right click on newly created XSD and open with Business Object Editor as shown in Figure 9
- Specify the name and type for the element in Properties Tab

At this point we can group the object of objects thus creating a complex object as a whole, where each object can contain the primitive data types like integer, string or a Boolean value. The basic structure of a complex object can be depicted in figures 9 and 10.

Figure 9. Working with the Business Object Editor

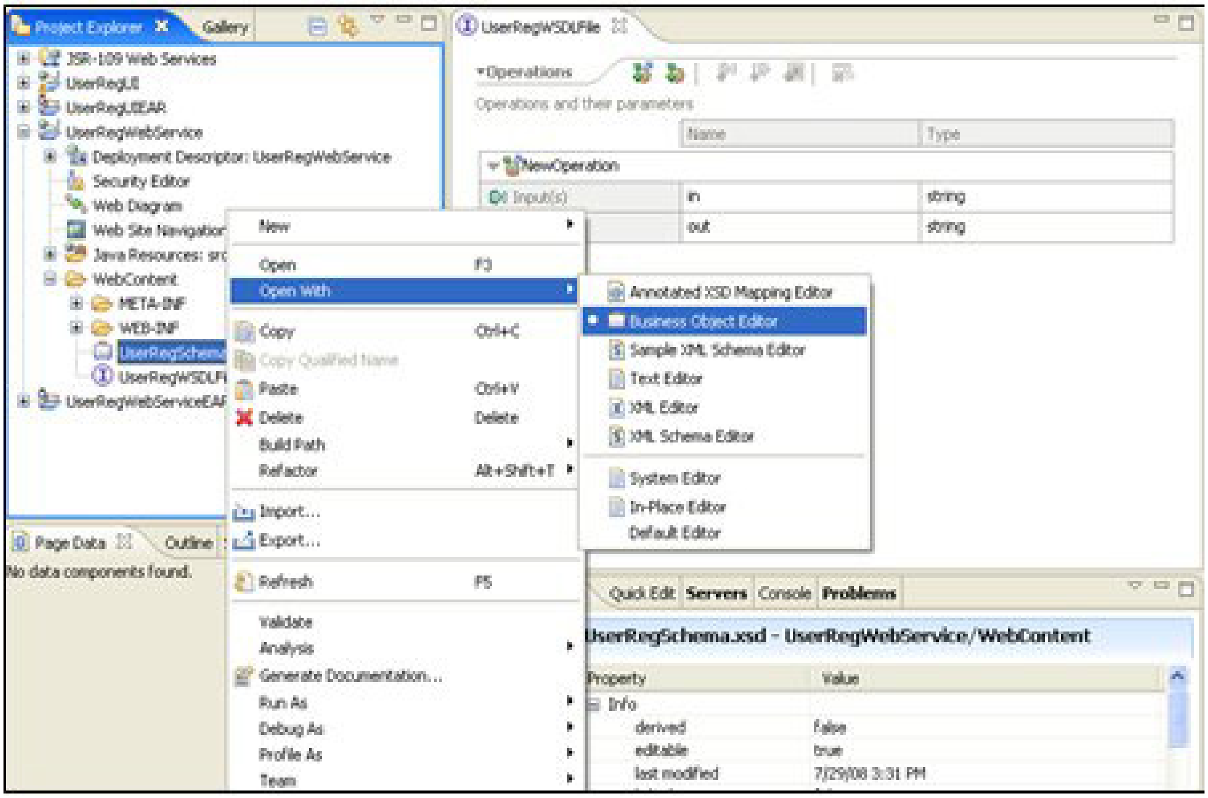
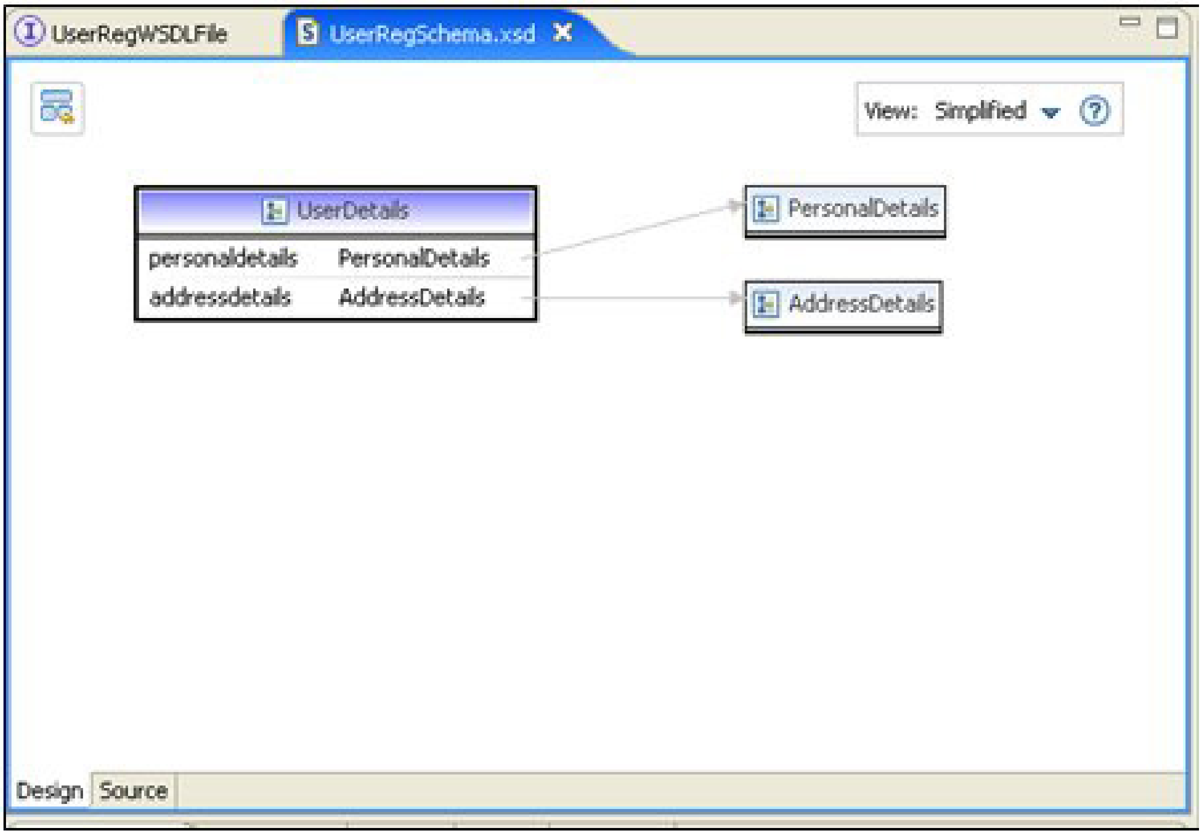


Figure 10. Structure of Complex Business Object

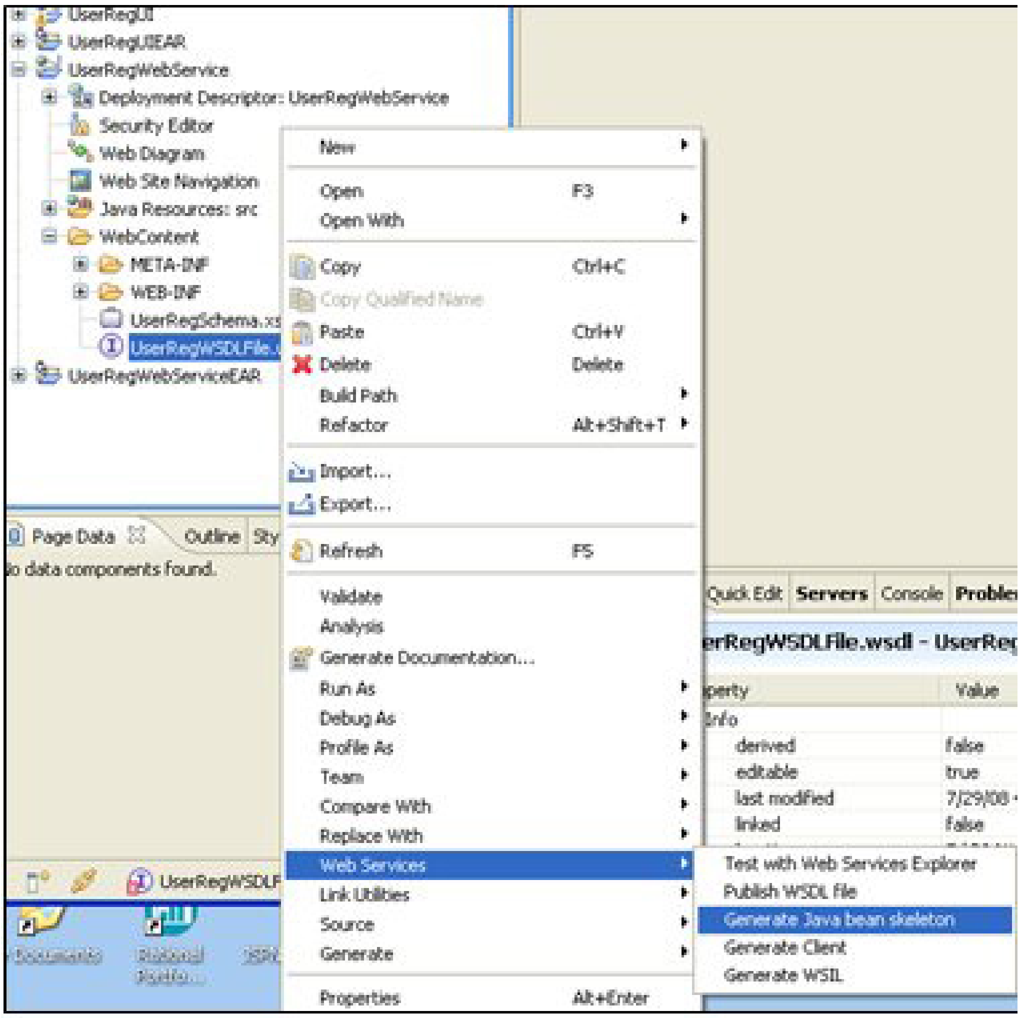


Once we are ready with the data objects, we can create a wsdl that takes these data objects as inputs and outputs depending on the operation type. To use these data objects we need to import them to our wsdl specifying the corresponding namespace.

Generating the JavaBean

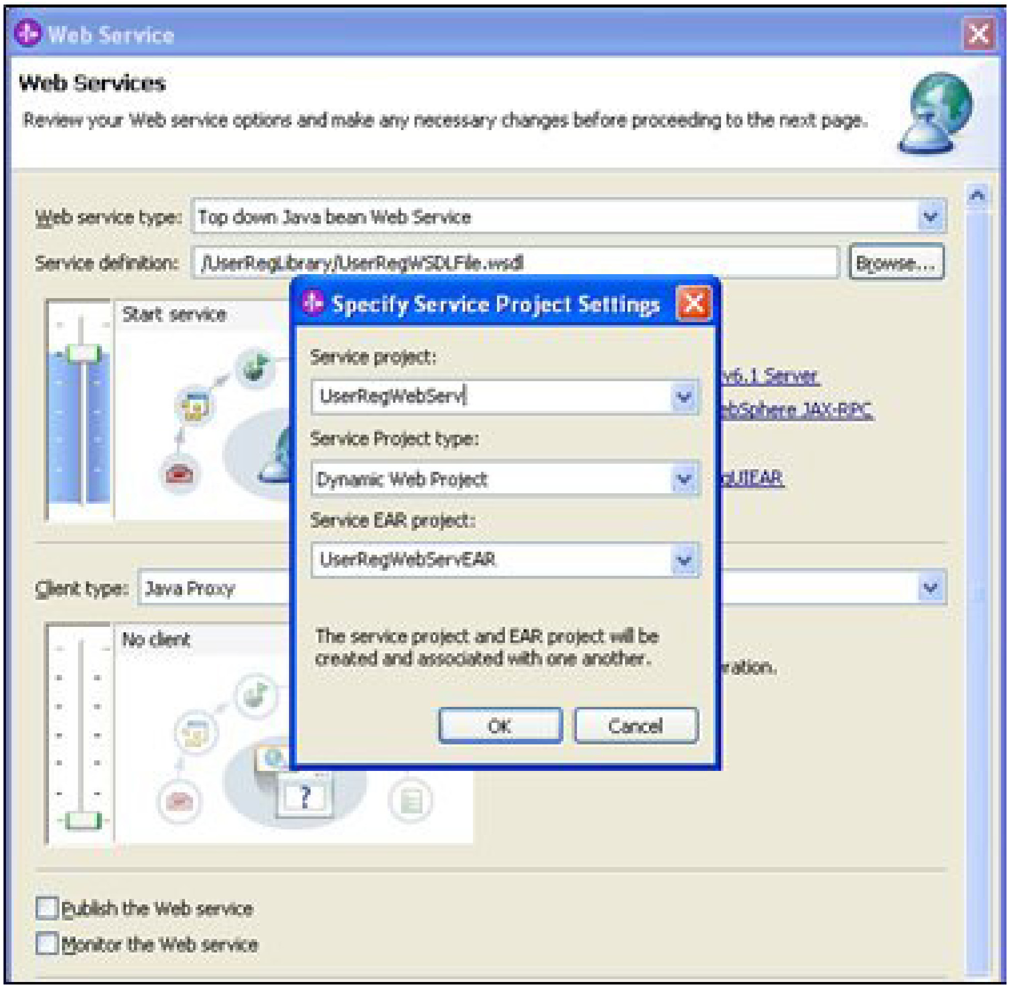
The WSDL's created earlier define the operations performed with input and output parameters. The actual core business logic is defined by implementing those operations, this can be achieved by generating the Java Bean shell. Figure 11 demonstrate how to create the Java Bean shell.

Figure 11. Structure of Complex Business Object



Right click on WSDL > Select Web Services > Generate Java bean skeleton

Figure 12. Specify Service Project Settings


Web Service

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Web service type: Top down Java bean Web Service

Service definition: /UserRegLibrary/UserRegWSDLFile.wsdl

Client type: Java Proxy

Specify Service Project Settings

Service project: UserRegWebServ

Service Project type: Dynamic Web Project

Service EAR project: UserRegWebServEAR

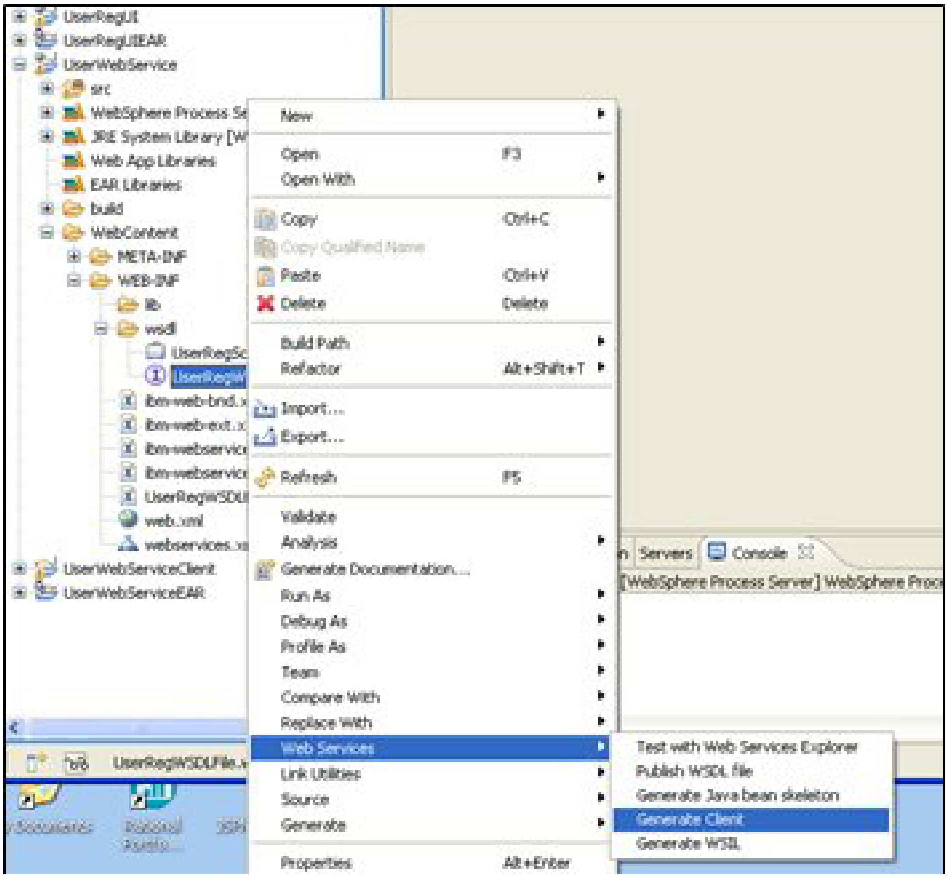
The service project and EAR project will be created and associated with one another.

Specify the Service Project Name with its project type and EAR name.

Generating client

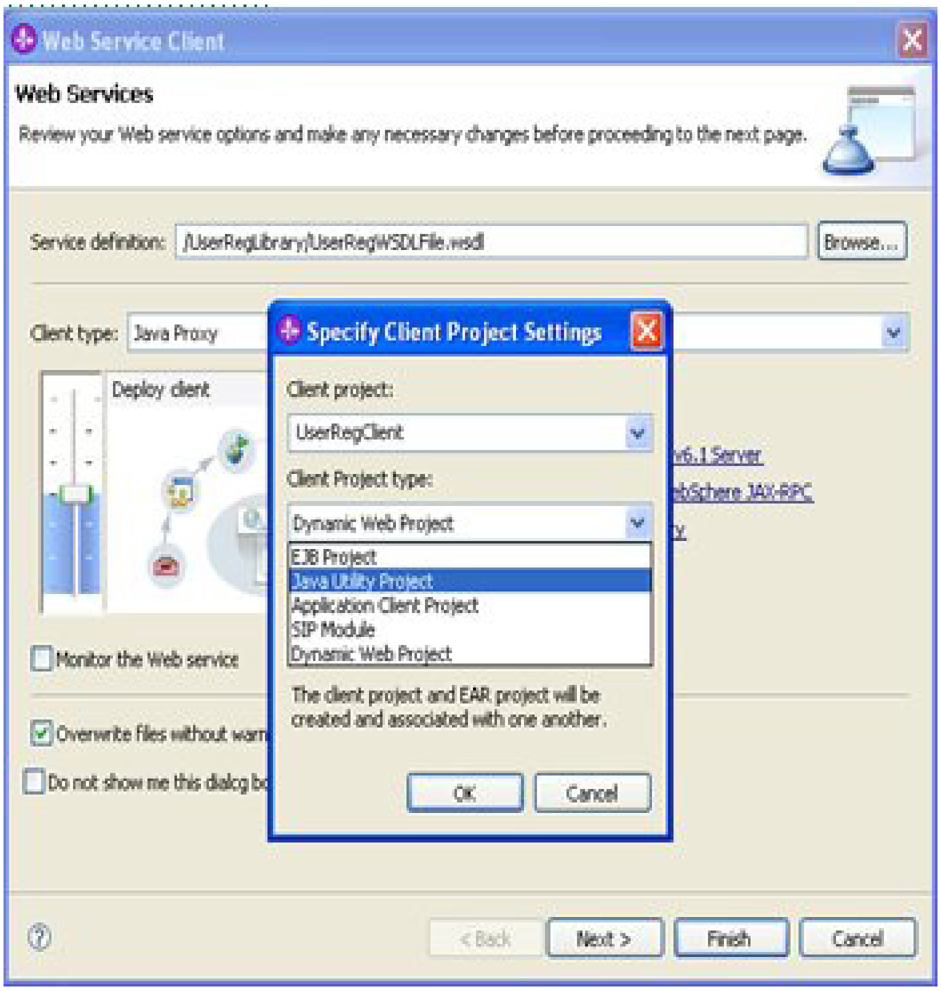
To check the implemented classes functionality we generate client that actually make a call to the web service and confirm us that everything is fine. In real time we actually give the wsdl to the user from which they generate the client and invoke for the required functionality. Figure 13 shows how to create the client.

Figure 13. Generating the Client



Right click on WSDL > Select Web services > Generate Client

Figure 14. Specify Client Project Settings


Web Service Client
Web Services
Review your Web service options and make any necessary changes before proceeding to the next page.
Service definition: /UserRegLibrary/UserRegWSDLFile.wsdl
Browse...
Client type: Java Proxy
Deploy client
Monitor the Web service
Overwrite files without warn
Do not show me this dialog bo
Specify Client Project Settings
Client project:
UserRegClient
Client Project type:
Dynamic Web Project
EJB Project
Java Utility Project
Application Client Project
SIP Module
Dynamic Web Project
The client project and EAR project will be created and associated with one another.
OK
Cancel
v6.1 Server
bSphere JAX-RPC
< Back
Next >
Finish
Cancel
Click on Client Project > Specify the Client Project Name

ObJectRelationalBridge (OJB)

Data being the major component in any project should be manipulated and stored in a right way at right time. Delay in response time or dirty read/write will degrade the performance of the system and keeping the system in an inconsistent state as a whole. In the current tutorial the DB layer sits at the end storing the data that is passed through UI layer and service layer.

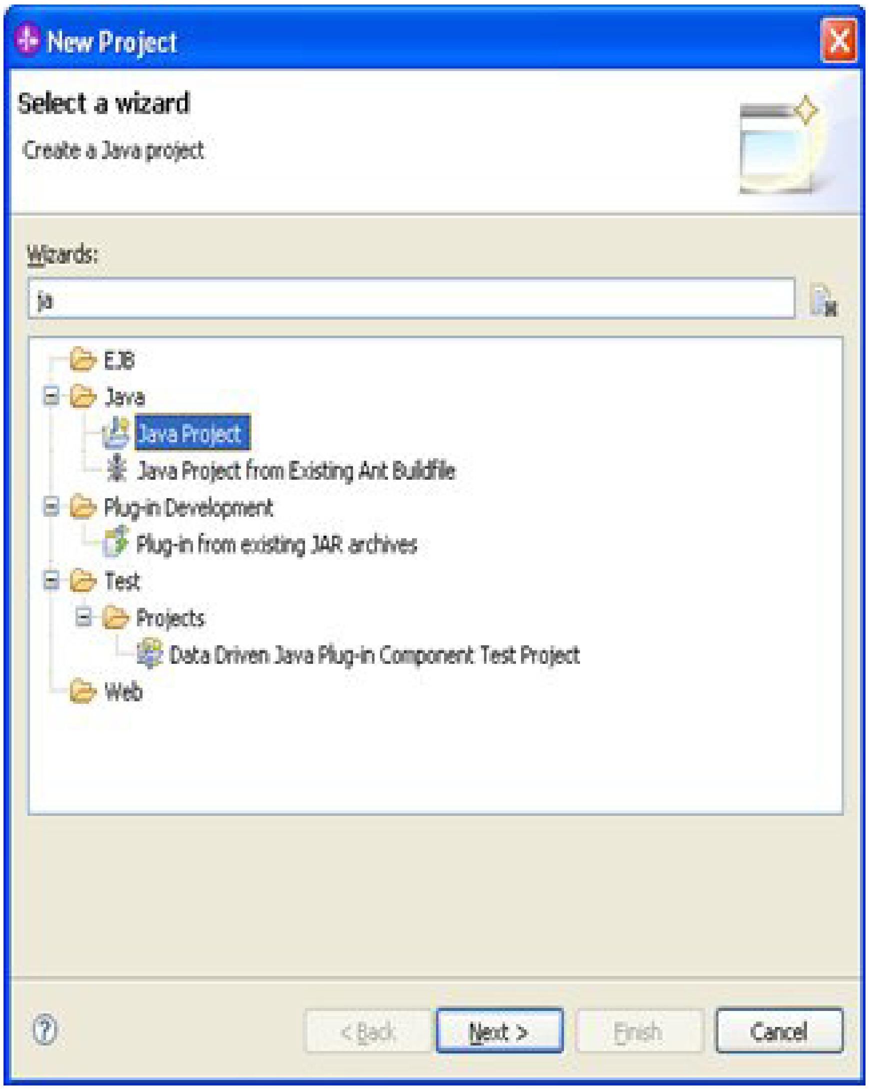
With the introduction of ORM tools the data manipulation and maintenance became easy that can be directly mapped with the java, OJB is one among them that is designed developed by Apache. It allows persistence for java objects against relational databases.

The following are some key features of OJB:

- Provides flexibility through support to multiple persistence APIs
- The applications that are designed and developed using OJB are scalable for future enhancements.
- It support JNDI lookups and Bean Managed Persistence
- The XML based ORM helps to manipulate during run-time
- It provides a flexible configuration and plug-in mechanism that allows to select from set of predefined components.
- The open source feature with 100% pure Java orientation facilitates rapid development.

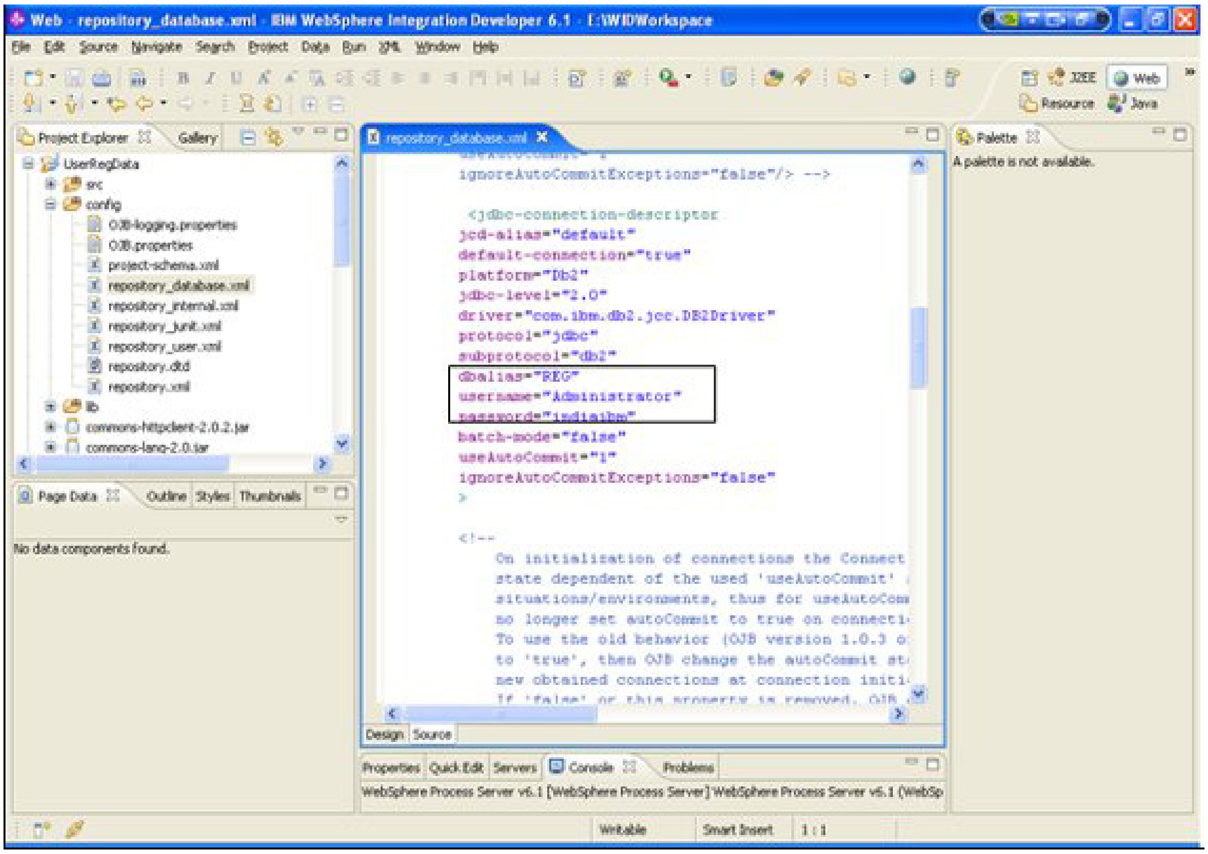
With the completion of the user inteface layer and services layer we move into the database layer which actually saves the data into the database. The annotations and xml configuration features provided by OJB enables us to map the table columns with the bean properties directly thus making the data manipulations easy. The built-in methods in OJB help us to store ,update and delete the data in a object model.

Figure 15. Creating New Java Project



To achieve this we create a Java Project with two main packages that store DAO's and Domain classes. Following the java naming conventions create the packages with appropriate names. The Domain package stores the classes that contain doclets mapping the properties to that of columns in the specific table, while the DAO package contains the real operational methods that stores the domain object. Import the required Jars into the lib folder. This include the necessary jars for the database support, in this article we use DB2 as database and import them accordingly.

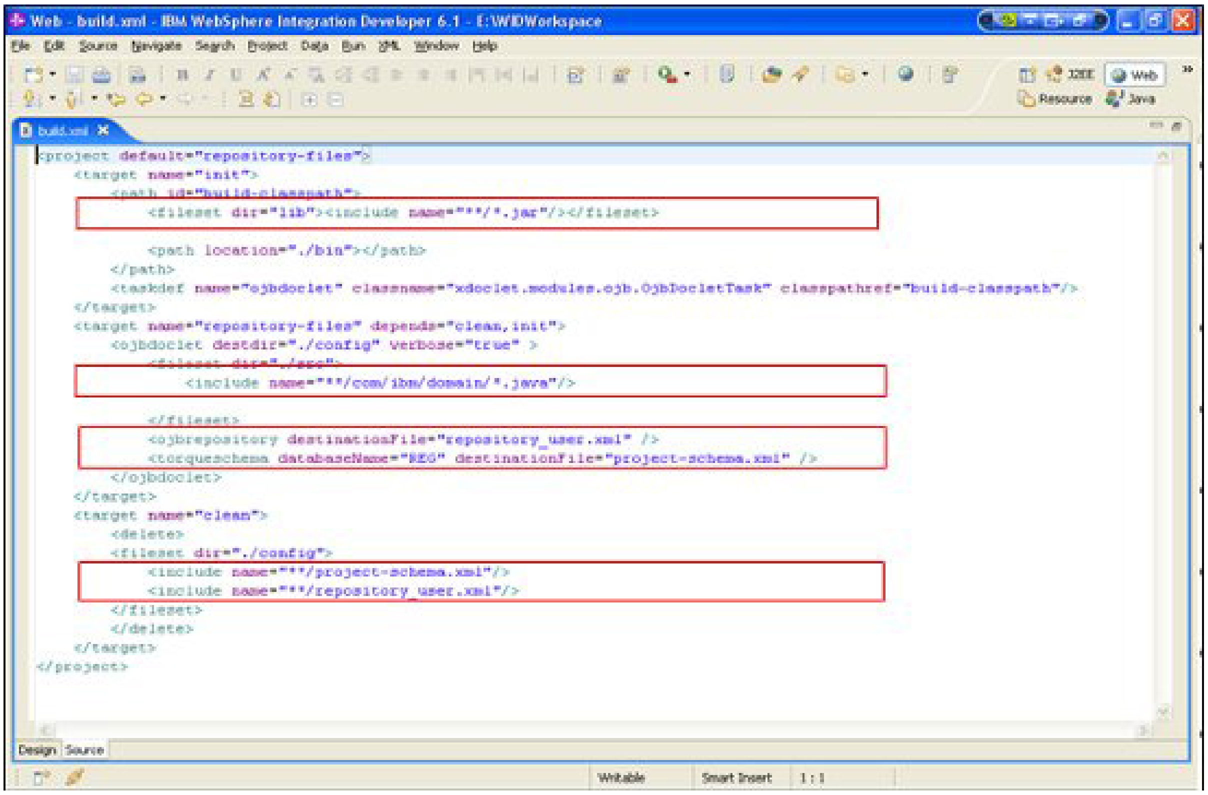
Figure 16. Configuring the repository_database.xml



```
ignoreAutoCommitExceptions="false"/> -->

 <jdbc-connection-descriptor
jcd-alias="default"
default-connection="true"
platform="Db2"
jdbc-level="2.0"
driver="com.ibm.db2.jcc.DB2Driver"
protocol="jdbc"
subprotocol="db2"
dbalias="REG"
username="Administrator"
password="indiaibm"
batch-mode="false"
useAutoCommit="1"
ignoreAutoCommitExceptions="false"
>

<!--
    On initialization of connections the Connect
    state dependent of the used 'useAutoCommit'
    situations/environments, thus for useAutoCom
    no longer set autoCommit to true on connecti
    To use the old behavior (OJB version 1.0.3 o
    to 'true', then OJB change the autoCommit st
    new obtained connections at connection initi
```

Once we are done with the basic packaging structure and importing of dependent jar files we moved ahead with configuring the XML files. The files are stored in the **config** folder, the main config file include *repository_database.xml* that is used to configure the database name and other credentials. The next file we deal is the *build.xml* that actually map the location of domain objects and classpath from where required jars are loaded. It also specifies the target location for files *project-schema.xml* and repository_user.xml. These two files will be generated and placed in config folder once we run the build.xml file. Figure 17 shows the typical *build.xml* file.

Figure 17. Configuring the build.xml

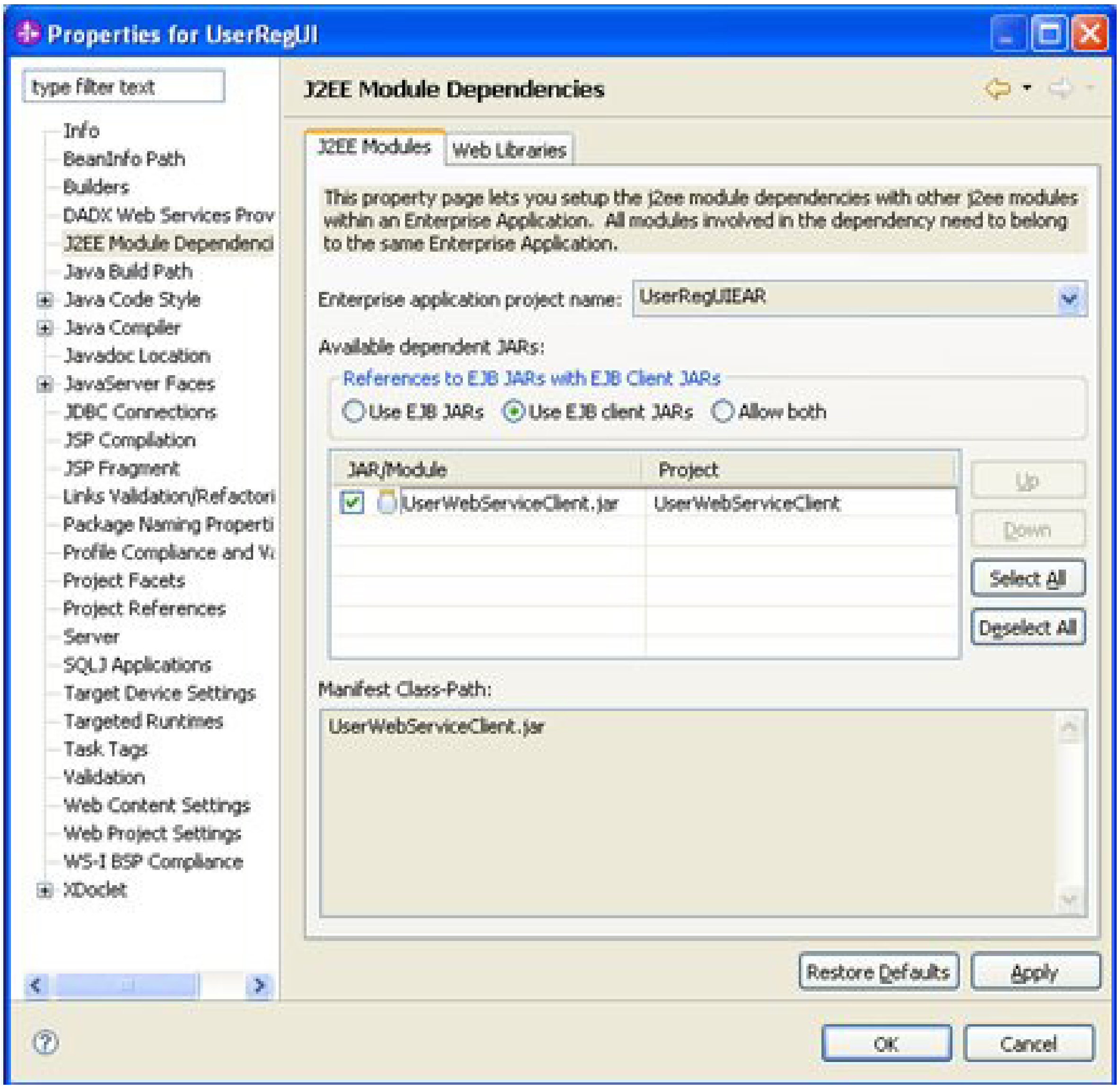


```
<project default="repository-files">
    <target name="init">
        <path id="build-classpath">
            <fileset dir="lib"><include name="**/*.jar"/></fileset>

            <path location="./bin"></path>
        </path>
        <taskdef name="ojbdoclet" classname="xdoclet.modules.ojb.OjbDocletTask" classpathref="build-classpath"/>
    </target>
    <target name="repository-files" depends="clean,init">
        <ojbdoclet destdir="./config" verbose="true" >
            <fileset dir="./src">
                <include name="**/com/ibm/domain/*.java"/>

            </fileset>
            <ojbrepository destinationFile="repository_user.xml" />
            <torqueschema databaseName="REG" destinationFile="project-schema.xml" />
        </ojbdoclet>
    </target>
    <target name="clean">
        <delete>
        <fileset dir="./config">
            <include name="**/project-schema.xml"/>
            <include name="**/repository_user.xml"/>
        </fileset>
        </delete>
    </target>
</project>
```

Integrating the user interface with services

Now we are ready with three individual layers and need to integrate them to make an end-to-end flow. Firstly we integrate the user interface with Web services. Since the user interface is a dynamic Web project, adding the service client jar makes the operations defined in Web service to be available to the user interface layer. This can be achieved in the following way, described below.

Figure 18. Adding web service client to UI



This property page lets you setup the j2ee module dependencies with other j2ee modules within an Enterprise Application. All modules involved in the dependency need to belong to the same Enterprise Application.

Enterprise application project name: UserRegUIEAR

Available dependent JARs:

References to EJB JARs with EJB Client JARs

Use EJB JARs [✓ Use EJB client JARs] Allow both

JAR/Module	Project
✓ UserWebServiceClient.jar	UserWebServiceClient

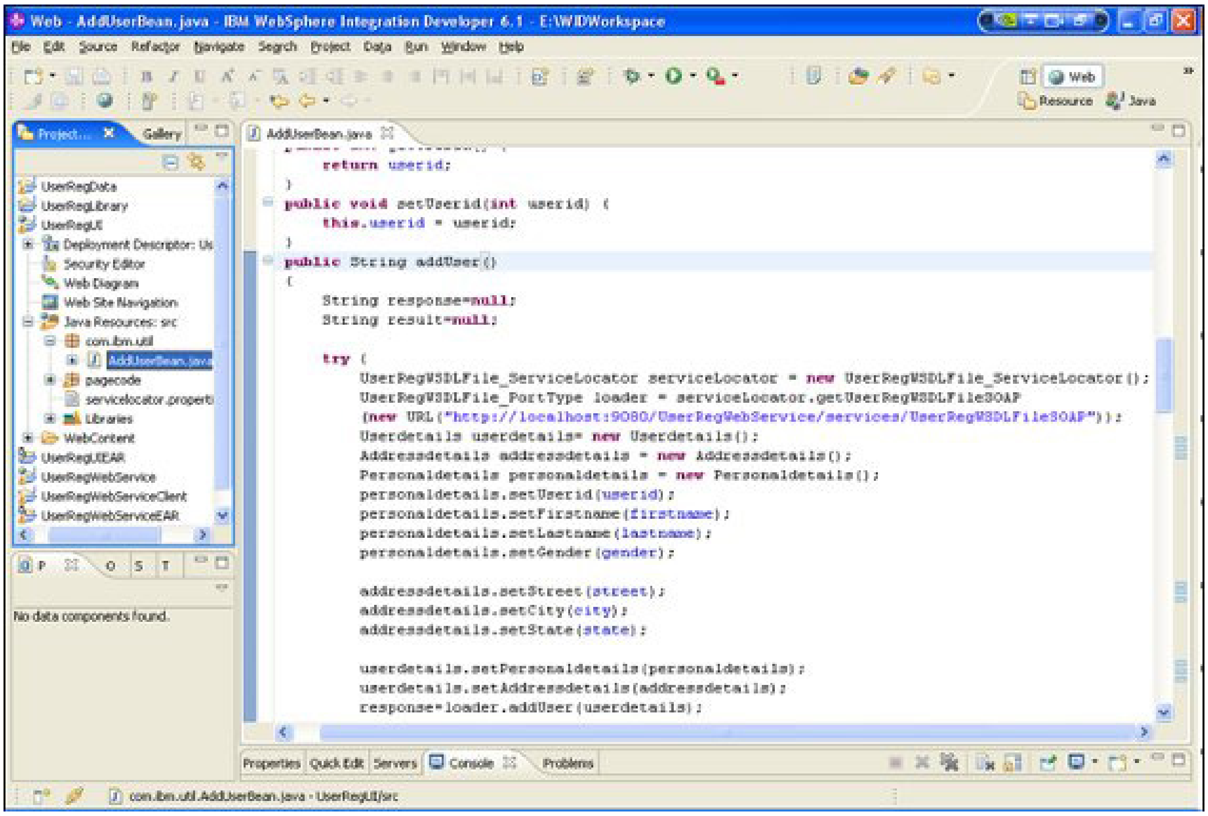
Manifest Class-Path:

UserWebServiceClient.jar

Right Click on UI Project > Properties > J2EE Module Dependencies > Check the Client JAR

Invoke the service using service locator and locate the endpoint with appropriate port type. If the server is listening in another port, we need to change accordingly. Depending on the object type at user interface layer we map them to objects that are defined at service layer. Figure 19 shows the sample code for invoking web service from a managed bean in user interface layer.

Figure 19. Service Invocation from Managed Bean

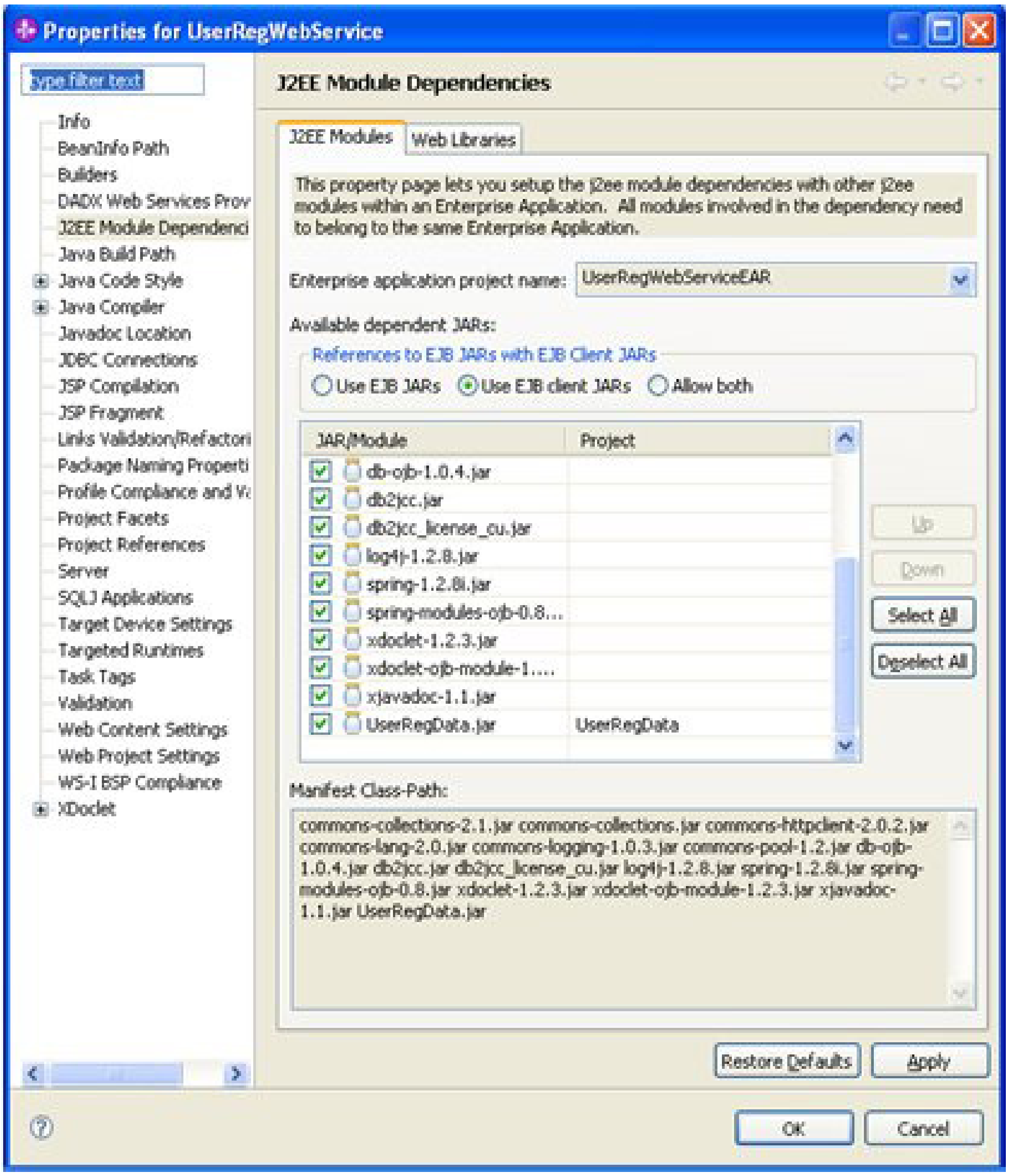


```
    return userid;
}
public void setUserid(int userid) {
    this.userid = userid;
}
public String addUser()
{
    String response=null;
    String result=null;

    try {
        UserRegWSDLFile_ServiceLocator serviceLocator = new UserRegWSDLFile_ServiceLocator();
        UserRegWSDLFile_PortType loader = serviceLocator.getUserRegWSDLFileSOAP
        (new URL("http://localhost:9080/UserRegWebService/services/UserRegWSDLFileSOAP"));
        Userdetails userdetails= new Userdetails();
        Addressdetails addressdetails = new Addressdetails();
        Personaldetails personaldetails = new Personaldetails();
        personaldetails.setUserid(userid);
        personaldetails.setFirstname(firstname);
        personaldetails.setLastname(lastname);
        personaldetails.setGender(gender);

        addressdetails.setStreet(street);
        addressdetails.setCity(city);
        addressdetails.setState(state);

        userdetails.setPersonaldetails(personaldetails);
        userdetails.setAddressdetails(addressdetails);
        response=loader.addUser(userdetails);
```

Integrating the services with the database layer

The integration of services with database layer includes adding the database java project as dependency jar in Web service project J2EE Module dependencies. Make certain that the required jars for the database project are already added in EAR that is generated for the Web service project. Figure 20 shows adding the data project as dependency jar to the Web service project.

Figure 21. Adding DB jar dependency



This property page lets you setup the j2ee module dependencies with other j2ee modules within an Enterprise Application. All modules involved in the dependency need to belong to the same Enterprise Application.

Enterprise application project name: UserRegWebServiceEAR

Available dependent JARs:

References to EJB JARs with EJB Client JARs

Use EJB JARs [✓ Use EJB client JARs] Allow both

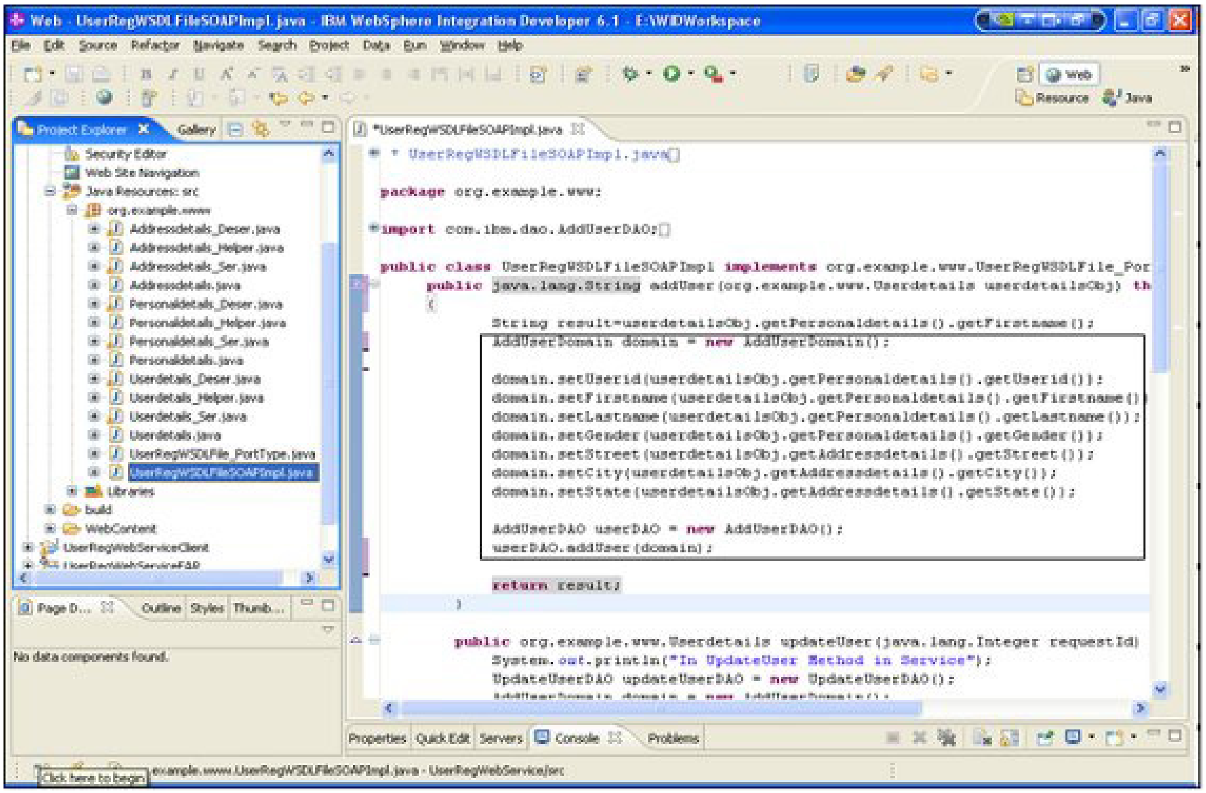
JAR/Module	Project
✓ db-ojb-1.0.4.jar	
✓ db2jcc.jar	
✓ db2jcc_license_cu.jar	
✓ log4j-1.2.8.jar	
✓ spring-1.2.8i.jar	
✓ spring-modules-ojb-0.8....	
✓ xdoclet-1.2.3.jar	
✓ xdoclet-ojb-module-1....	
✓ xjavadoc-1.1.jar	
✓ UserRegData.jar	UserRegData

Manifest Class-Path:

```
commons-collections-2.1.jar commons-collections.jar commons-httpclient-2.0.2.jar
commons-lang-2.0.jar commons-logging-1.0.3.jar commons-pool-1.2.jar db-ojb-
1.0.4.jar db2jcc.jar db2jcc_license_cu.jar log4j-1.2.8.jar spring-1.2.8i.jar spring-
modules-ojb-0.8.jar xdoclet-1.2.3.jar xdoclet-ojb-module-1.2.3.jar xjavadoc-
1.1.jar UserRegData.jar
```

Right Click on Web Service Project > Properties > J2EE Module Dependencies > DB JAR

Figure 21. Creating Domain and DAO objects



```
 * UserRegWSDLFileSOAPImpl.java

package org.example.www;

import com.ibm.dao.AddUserDAO;

public class UserRegWSDLFileSOAPImpl implements org.example.www.UserRegWSDLFile_Por
    public java.lang.String addUser(org.example.www.Userdetails userdetailsObj) th
    {
        String result=userdetailsObj.getPersonaldetails().getFirstname();
        AddUserDomain domain = new AddUserDomain();

        domain.setUserid(userdetailsObj.getPersonaldetails().getUserid());
        domain.setFirstname(userdetailsObj.getPersonaldetails().getFirstname()
        domain.setLastname(userdetailsObj.getPersonaldetails().getLastname());
        domain.setGender(userdetailsObj.getPersonaldetails().getGender());
        domain.setStreet(userdetailsObj.getAddressdetails().getStreet());
        domain.setCity(userdetailsObj.getAddressdetails().getCity());
        domain.setState(userdetailsObj.getAddressdetails().getState());

        AddUserDAO userDAO = new AddUserDAO();
        userDAO.addUser(domain);

        return result;
    }

    public org.example.www.Userdetails updateUser(java.lang.Integer requestId)
        System.out.println("In UpdateUser Method in Service");
        UpdateUserDAO updateUserDAO = new UpdateUserDAO();
```

- Create the Domain and DAO Objects
- Set the domain object using appropriate DAO

Conclusion

This article summarized the rapid application development on SOA platform using the services as core layer with cutting edge technologies like JSF and OJB. It focused on development of each layer and integrating with the other layers. This is an basic approach to start and visualise the SOA and identify the interopability of services that can glue any kind of technology together.

Resources

Learn

- For further reading visit the new to SOA and Web services page.
- In the SOA and Web services area on developerWorks, get the resources you need to advance your skills.
- "IBM WebSphere Developer Technical Journal: Web Services Architectures and Best Practices", (developerWorks, Oct 2003) book excerpt covers some of the architectural challenges posed by Web services, examines how to use (and not to use) Web Services, and describes some best practices in applying Web Services for solving tough architectural problems.
- Learn more about JavaServer Faces Technology.
- "Developing Web Applications with JavaServer Faces".
- Learn more about Apache ObJectRelationalBridge.
- View the OJB reference guide summary.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Download IBM product evaluation versions or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Kamesh Pandrangi
Kamesh has been working at IBM for over 3 years, exclusively on develping SOA solutions utilizing WebSphere software.