* **Steps to use EJB in SOA application (4 mark)**

**Ans :**

The steps we will go through are:

1. Create the EJB that is to be invoked from the Composite application through EJB Binding. Make sure the remote interface is created as well.

2. Deploy this EJB (in this simple example to the same WebLogic Managed Server that is running the SOA Suite)

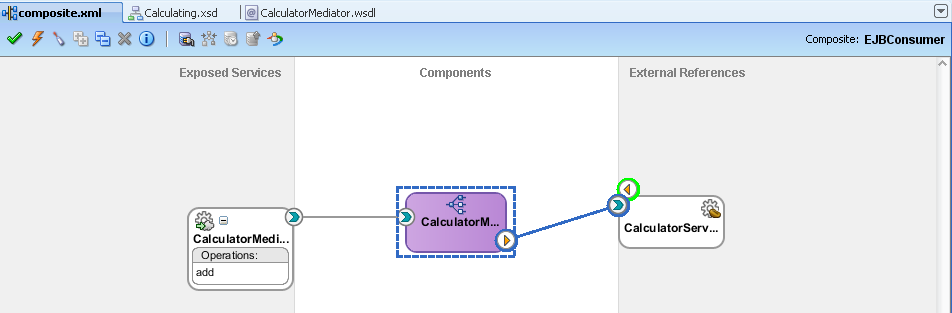
3. Create a new SOA Composite Application

4. Copy the remote interface to the SCA-INF/src directory (or copy a JAR containing that interface to SCA-INF/lib)

5. Create an EJB Binding Reference. Configure it to invoke the EJB created and deployed above; specify the Java Interface that describes the interaction

6. Create a Mediator that is wired to the EJB Binding Reference; create mappings from the input and output of the Mediator to the XML structure derived by JDeveloper from the Java Interface definition

7. Finish the composite application; from here on it is ‘business as usual’ since none of the other components in the applications are aware that the Mediator talks to something ‘special’ like an EJB.

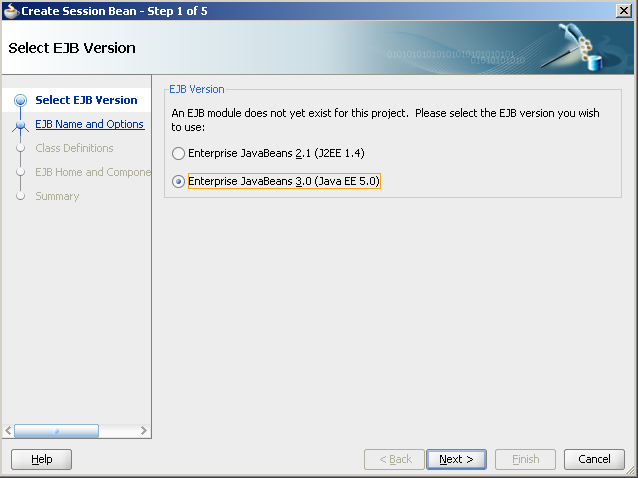
[](https://technology.amis.nl/wp-content/uploads/images/wireMediatorToEJB.png)

### 1. Create the EJB

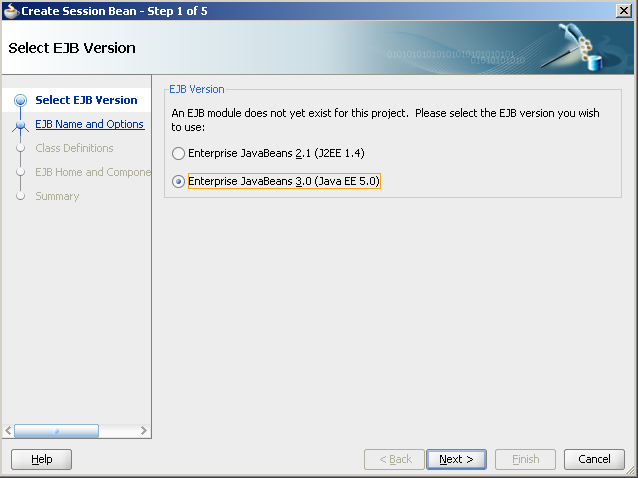
Open JDeveloper. Create a new (generic) application. Call the application CalculatorApplication and the default Project Calculator. I set the application’s package prefix to nl.amis.calculator (though you can choose anything you like).

Open the New Gallery, select category EJB and click on item Session Bean

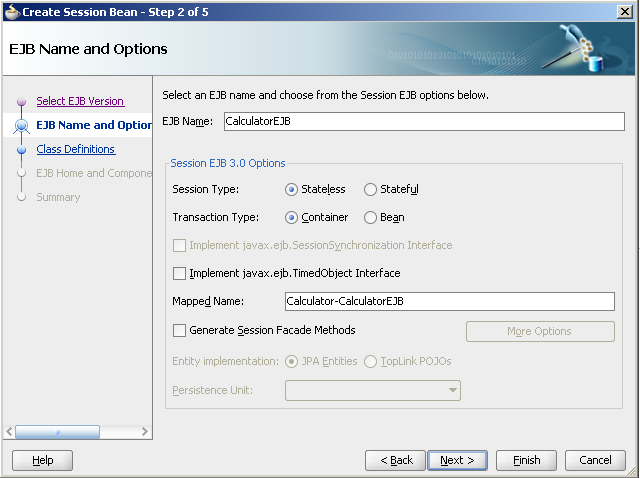
.

[](https://technology.amis.nl/wp-content/uploads/images/newSBSelectEJBVersion.png)

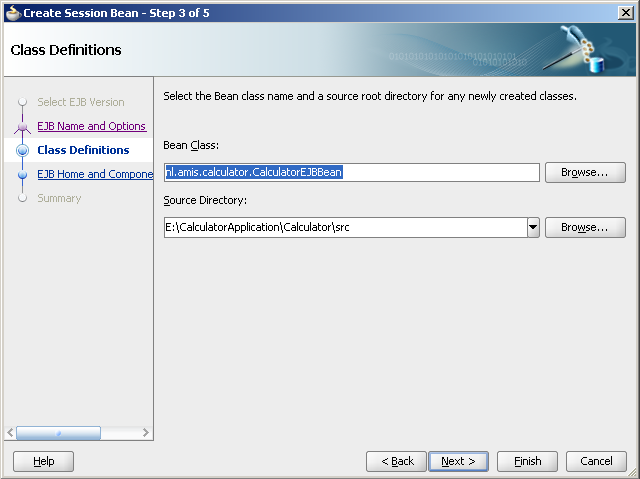
The Create Session Bean–––– dialog appears. On the first tab, select Enterise JavaBeans 3.0 as the EJB version.

[](https://technology.amis.nl/wp-content/uploads/images/newSBSelectEJBVersion.png)

On the EJB Name and options page, set the name to CalculatorEJB and the mapped name to Calculator-CalculatorEJB. This mapped name is used later on to locate the EJB when it is deployed.

[](https://technology.amis.nl/wp-content/uploads/images/newSBName.png)

On the next page, Class Definitions, accept the defaults.

[](https://technology.amis.nl/wp-content/uploads/images/newSBClassDefs.png)  
Press Next.

On the page EJB Home and Component interface, check the checkbox for Remote Interface and set the name of the Interface to Calculator. Uncheck the checkbox for Local Interface.

Press Finish.

(note: the wizard is not adding an awful lot of value over simply creating the Interface and Class yourself, it is just a matter of a few annotations; so feel free to completely skip the wizard and create the Java objects manually).

Add the following method signature to the Calculator interface definition:

public int add(int operand1, int operand2);

(heavy stuff going on here)

and provide the implementation of this method in the CalculatorEJB class:

public int add(int operand1, int operand2) {

return operand1 + operand2;

}

The interface and class with appropriate annotations now becomes:

package nl.amis.calculator;

import javax.ejb.Remote;

@Remote

public interface Calculator {

public int add(int operand1, int operand2);

}

and

package nl.amis.calculator;

import javax.ejb.Remote;

import javax.ejb.Stateless;

@Stateless(name = "CalculatorEJB", mappedName = "Calculator-CalculatorEJB")

@Remote

public class CalculatorEJBBean implements Calculator {

public int add(int operand1, int operand2) {

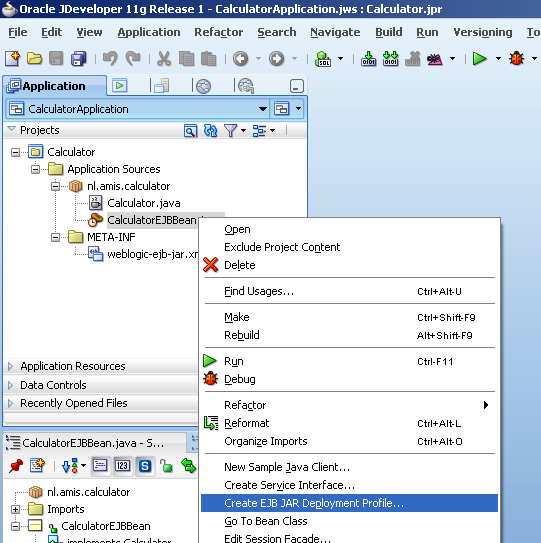
return operand1 + operand2;

}

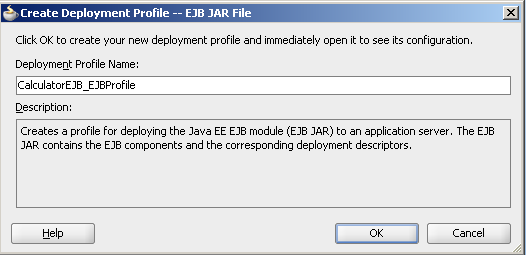
}

### 2. Deploy this EJB

The EJB in all its simplicity is complete – or at least ready for first deployment and testing. Right click the CalculatorEJB class and from the context menu select the option Create EJB JAR Deployment Profile.

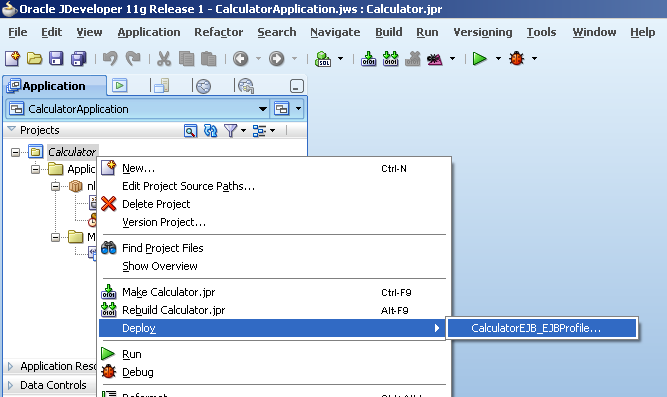
[](https://technology.amis.nl/wp-content/uploads/images/createDeploymentProfile.png)

Specify the name for the deployment profile as CalculatorEJB\_EJBProfile – or any other name you fancy.

[](https://technology.amis.nl/wp-content/uploads/images/createDeploymentProfile2.png)

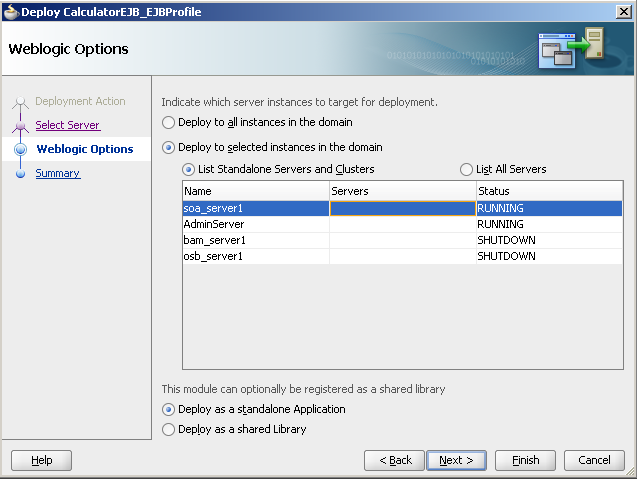
Click OK and click OK again (twice) – accepting all defaults.

To actually deploy the EJB, right click the project in the navigator, click on the option Deploy and select the CalculatorEJB\_EJBProfile from the submenu.

[](https://technology.amis.nl/wp-content/uploads/images/deployEJB.png)

The Deployment Wizard appears. Select Deploy to Application Server on the first page.

Select your Application Server connection to the WebLogic domain running the SOA Suite on the second page. On page three, select Managed Server soa\_server1 as the deployment target. Make sure the radio button Deploy as standalone Application has been selected.

[](https://technology.amis.nl/wp-content/uploads/images/deployEJB4.png)

Press Finish to commence deployment, or first inspect the summary on the next page and press Finish there.

Deployment is performed – preceded by compilation, build and package to EAR file.

### 2b. Test the EJB using a simple client application

If you like, you could test the EJB that was just deployed. Right click the CalculatorEJB Session Bean and select the menu item New Sample Java Client.

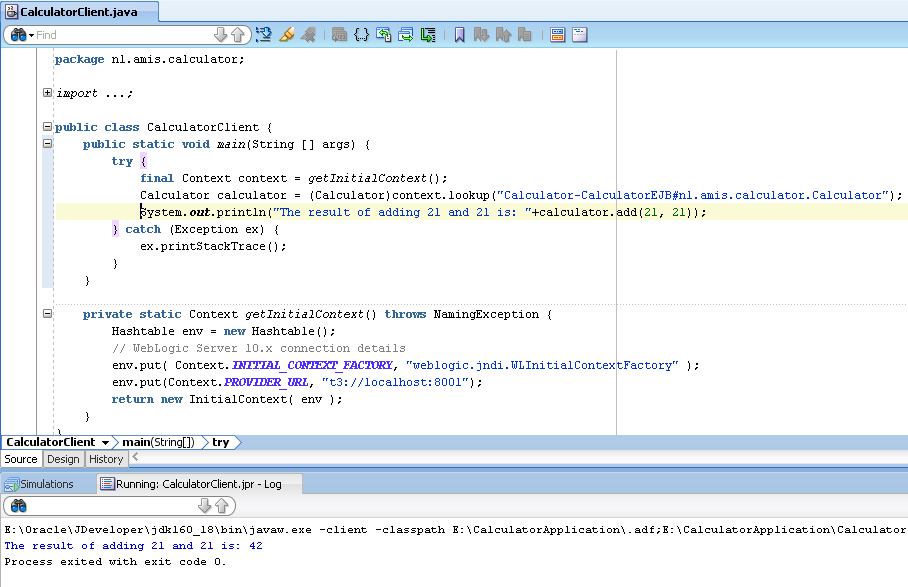
Click on the button New Project in the popup that appears.  
Specify a project name, for example CalculatorClient. Press OK.

A CalculatorClient class is generated and subsequently opened in the Java Source editor. You need to make a change in the getInitialContext method. The PROVIDER\_URL property is defined using port name 7001. However, since we selected the managed server soa\_server1 – and not the default AdminServer, you may need to use a different port number. In my case the port number is 8001.

Next, add the following line of Java code to do at least something with the Calculator object the JNDI lookup retrieves for us:

System.out.println("The result of adding 21 and 21 is: "+calculator.add(21, 21));

Run the CalculatorClient and see the result coming in. This demonstrates that the EJB as deployed is accessible and doing it (trivial) job.

[](https://technology.amis.nl/wp-content/uploads/images/testEJBwithSampleClient.png)

### 3. Create a new SOA Composite Application

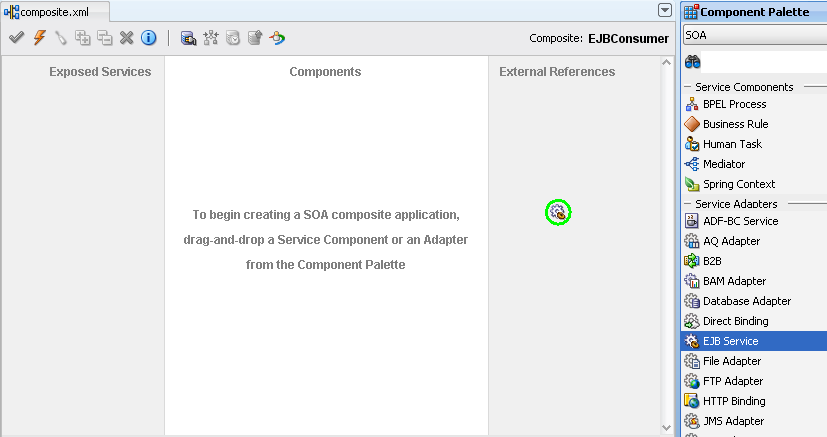
Create a new SOA application. In my case it is called EJBConsumer, as is the project. The template used is simply Empty Composite.

### 4. Make the Java Interface Calculator available to the Composite

Copy the remote interface Calculator to the SCA-INF/src directory (or copy a JAR containing that interface to SCA-INF/lib). Note: remove the @Remote annotation and the import of javax.ejb.Remote because otherwise the Java interface won’t compile.

### 5. Create an EJB Binding Reference.

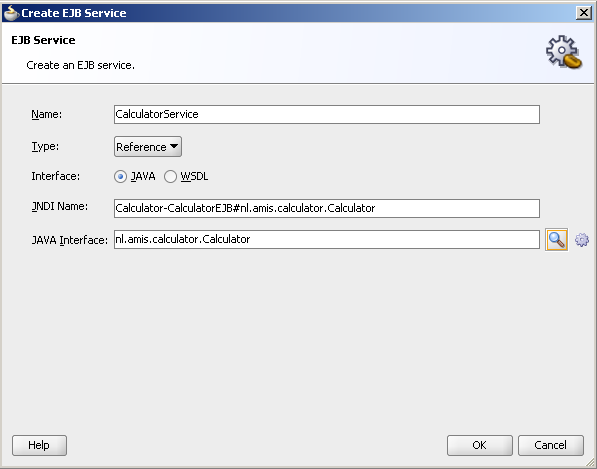
Open the composite editor. Drag the EJB Service from the Service Adapters list on the Component Palette to the References lane.

[](https://technology.amis.nl/wp-content/uploads/images/dragEJBBinding.png)

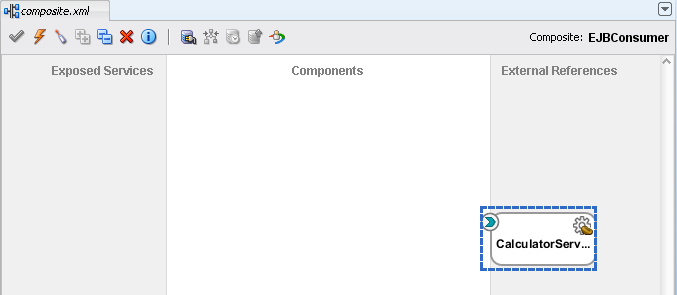
The Create EJB Service dialog appears. Enter a name – for example CalculatorService. Type is Reference, Interface is Java (instead of WSDL). JNDI Name is Calculator-

CalculatorEJB#nl.amis.calculator.Calculator, composed of the Mapped Name of the EJB and the interface describing the interaction. (note: this mapped name is used in the JNDI Lookup in the EJB Sample Client we created above in step 2b; you can copy it from there).

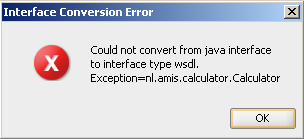
Browse for the Java Interface – nl.amis.calculator.Calculator. Press OK to complete the EJB Service definition.

[](https://technology.amis.nl/wp-content/uploads/images/createEJBService.png)

The composite editor appears with the EJB Binding Reference as expected.

[](https://technology.amis.nl/wp-content/uploads/images/compositeWithEJBBinding.png)

However: the Create EJB Service dialog is flawed! Even though everything seems to be correct, it is not so. When you try later on to wire a Mediator or another Service Component to this Reference, you will run into an Interface Conversion Error such as “Could not convert from java interface to interface type wsdl.Exception=nl.amis.calculator.Calculator”

[](https://technology.amis.nl/wp-content/uploads/images/interfaceConversionError.png)

The problem is that the dialog upon completion did not write an attribute to the Reference definition. It created:

&lt;reference name="CalculatorService"&gt;

&lt;interface.java interface="nl.amis.calculator.Calculator"/&gt;

&lt;binding.ejb uri="Calculator-CalculatorEJB#nl.amis.calculator.Calculator"

ejb-version="EJB3"/&gt;

&lt;/reference&gt;

and it should have created:

&lt;reference name="CalculatorService"&gt;

&lt;interface.java interface="nl.amis.calculator.Calculator"/&gt;

&lt;binding.ejb uri="Calculator-CalculatorEJB#nl.amis.calculator.Calculator"

javaInterface="nl.amis.calculator.Calculator"

ejb-version="EJB3"/&gt;

&lt;/reference&gt;

Go into the source view of the composite editor and add the atribute javaInterface:

javaInterface="nl.amis.calculator.Calculator"

### 6. Create a Mediator that is wired to the EJB Binding Reference

First, create an XSD document – called Calculating.xsd – with the following element definition:

&lt;schema attributeFormDefault="unqualified"

elementFormDefault="qualified"

targetNamespace="http://nl.amis.calculator"

xmlns="http://www.w3.org/2001/XMLSchema"&gt;

&lt;element name="calculatorParameter"&gt;

&lt;complexType&gt;

&lt;sequence&gt;

&lt;element name="operand1" type="int" /&gt;

&lt;element name="operand2" type="int" /&gt;

&lt;/sequence&gt;

&lt;/complexType&gt;

&lt;/element&gt;

&lt;element name="calculatorResult"&gt;

&lt;complexType&gt;

&lt;sequence&gt;

&lt;element name="outcome" type="int" /&gt;

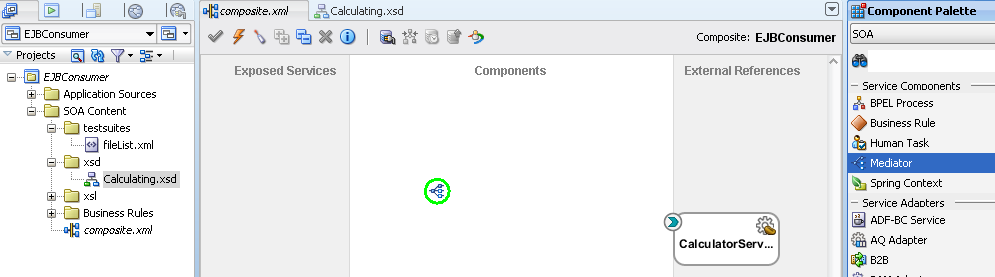
&lt;/sequence&gt;

&lt;/complexType&gt;

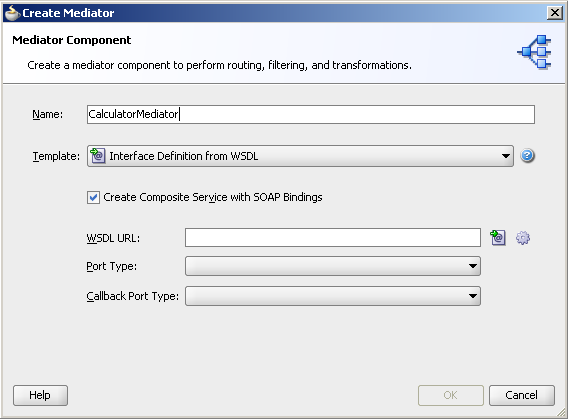
&lt;/element&gt;

&lt;/schema&gt;

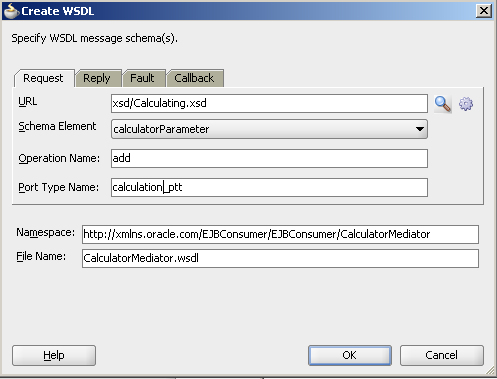
Next, drag a Mediator to the composite and drop it in the central area.

[](https://technology.amis.nl/wp-content/uploads/images/addMediator.png)

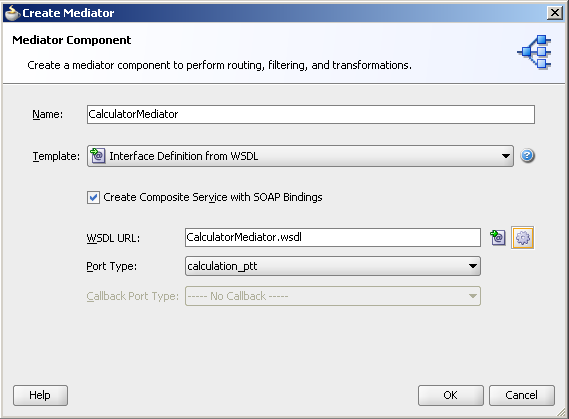
Set the name to CalculatorMediator.

[](https://technology.amis.nl/wp-content/uploads/images/addMediator2.png)

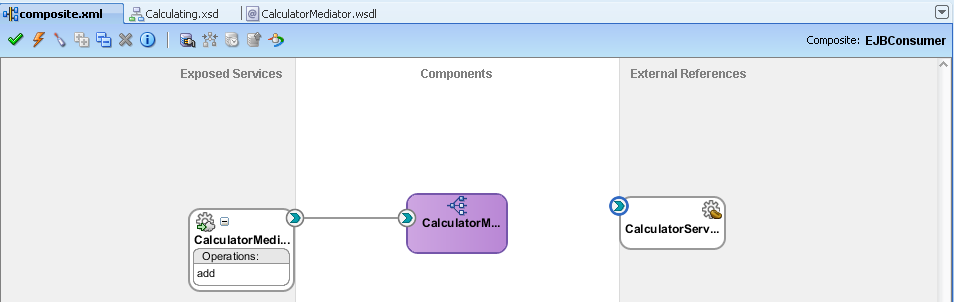
Select Template Interface Definition from WSDL and click on the generate icon after the WSDL URL field. The Request message is to be based on the calculatorParameter element in the Calculating.xsd schema definition. The Response message is based on the element calculatorResult.

[](https://technology.amis.nl/wp-content/uploads/images/addMediator2b.png)

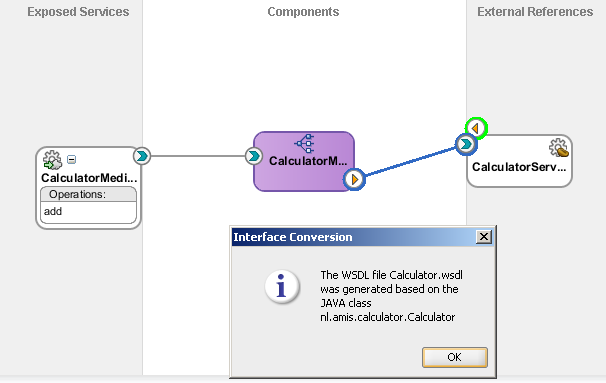
Press OK. The Mediator configuration is as follows:

[](https://technology.amis.nl/wp-content/uploads/images/addMediator3.png)

Press OK.

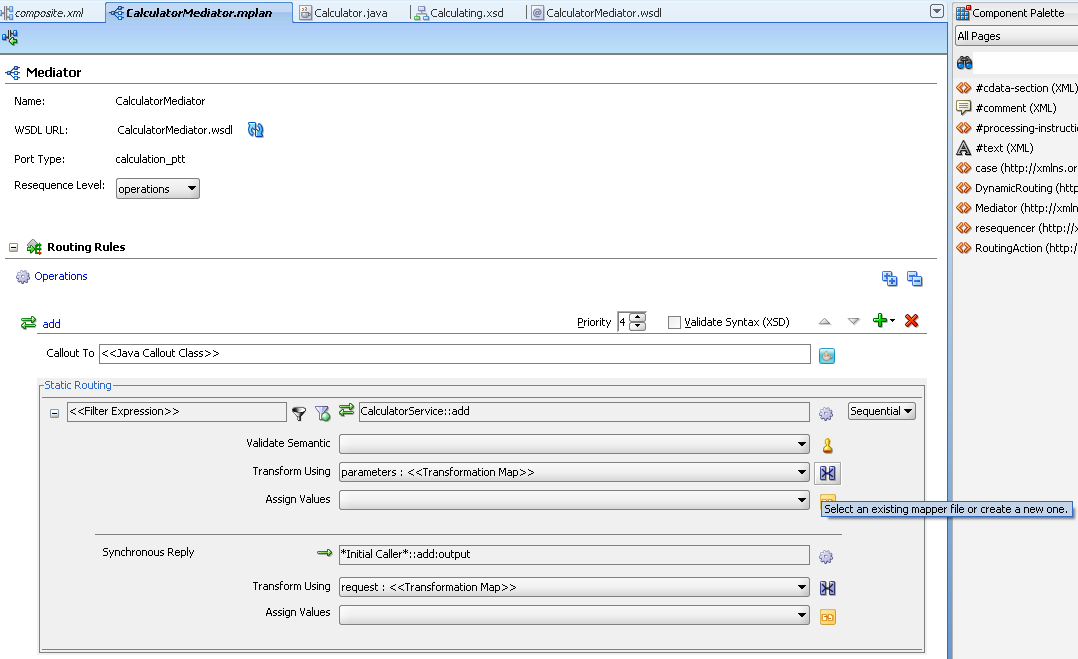
[](https://technology.amis.nl/wp-content/uploads/images/addMediator4.png)

Wire the Mediator to the CalculatorService EJB Binding.

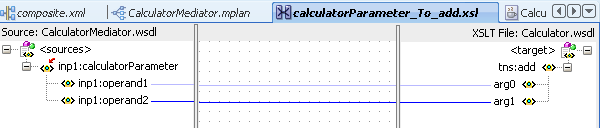
[](https://technology.amis.nl/wp-content/uploads/images/wireMediatorToEJB2.png)

JDeveloper creates the WSDL based on the Calculator Java interface.

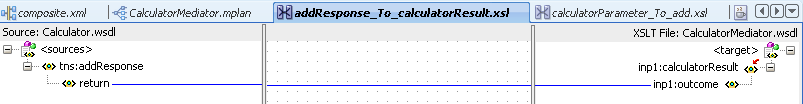
Click on the CalculatorMediator to bring up the editor and create mappings from the input and output of the Mediator to the XML structure derived by JDeveloper from the Java Interface definition.

[](https://technology.amis.nl/wp-content/uploads/images/mapMediatorToEJB.png)

create the mapping:

[](https://technology.amis.nl/wp-content/uploads/images/mapMediatorToEJB2.png)

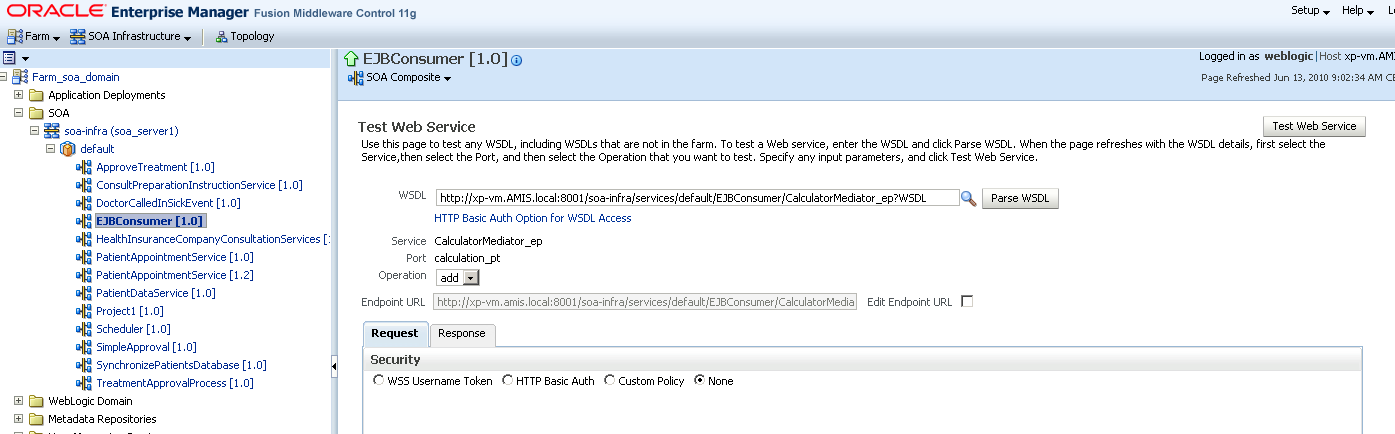
and

[](https://technology.amis.nl/wp-content/uploads/images/mapMediatorToEJB3.png)

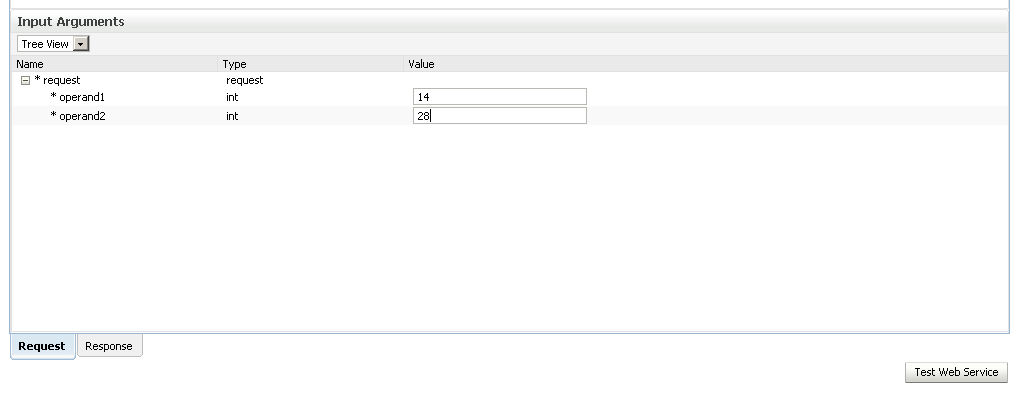
### 7.Deploy and Test the Composite application

Finish the composite application; from here on it is ‘business as usual’ since none of the other components in the applications are aware that the Mediator talks to something ‘special’ like an EJB.

Deploy the composite application in the normal way. Now we can test the composite application with EJB binding from the FWM EM Console:

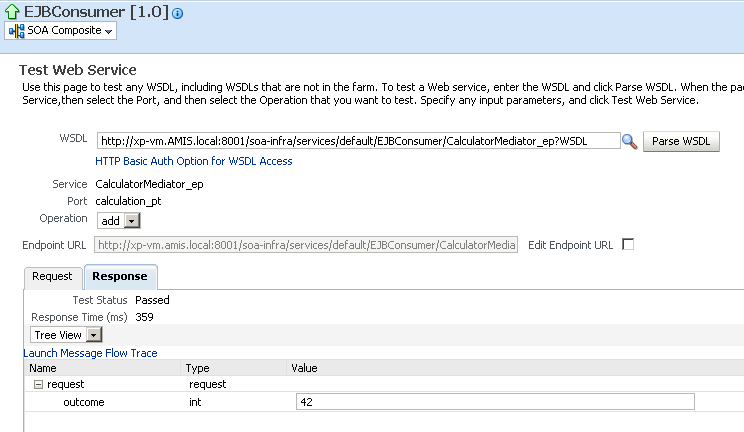
[](https://technology.amis.nl/wp-content/uploads/images/testCalc1.png)

Set input:

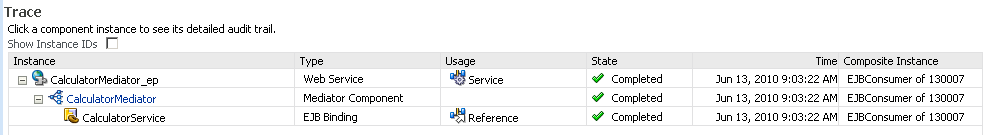
[](https://technology.amis.nl/wp-content/uploads/images/testCalc2.png)

and press the Test WebService button.

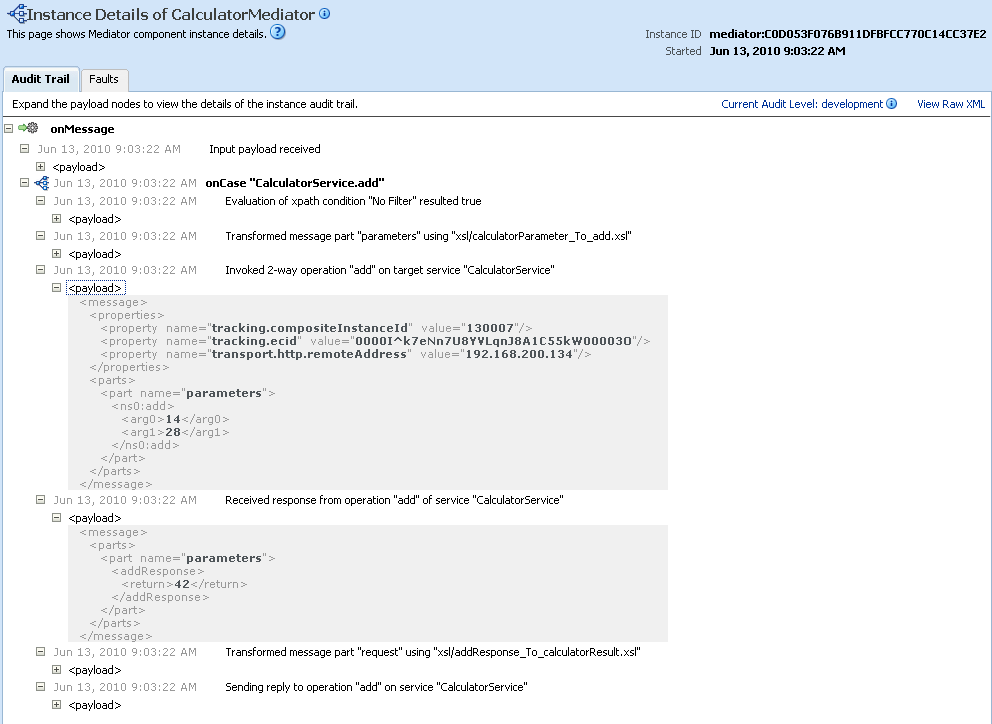
The response appears.:

[](https://technology.amis.nl/wp-content/uploads/images/testCalc3.png)

The flow trace:

[](https://technology.amis.nl/wp-content/uploads/images/flowtrace.png)

And the detailed audit overview for the Mediator:

[](https://technology.amis.nl/wp-content/uploads/images/mediatorAudit.png)