Oracle® Banking Enterprise Originations

SOA Extensibility Guide

Release 2.10.0.0.0

F29510-01

April 2020

ORACLE®

Oracle Banking Enterprise Originations SOA Extensibility Guide, Release 2.10.0.0.0

F29510-01

Contents

Preface **8**
Audience 8
Documentation Accessibility 8
Related Documents 8
Conventions 8
1 About This Guide **11**
2 Objective and Scope **13**
2.1 Overview 13
2.2 Objective and Scope 13
2.2.1 Extensibility Objective 13
2.3 Complementary Artefacts 13
2.4 Out of Scope 14
3 Overview of Use Cases **15**
3.1 Extensibility Use Cases 15
3.1.1 SOA Customization 15
4 Workflow Task Routing and Assignment Rules **17**
4.1 Workflow Task Routing and Assignment Rules 17
4.2 Origination Human Tasks 17
4.3 Origination Approval Human Tasks 17
5 Workflow Analytics **21**
5.1 Architecture 21
5.2 OBP Workflow Analytics Extension Points 21
5.2.1 Introduction of new human task 21
5.2.2 Addition of custom business indicator(s) to existing data object 22

5.2.3 Publishing analytics data to system other than Oracle BAM 22
5.2.4 Publishing new DataObject from OBP Domain 22
6 SOA Customizations 25
6.1 Customization Layer 25
6.2 Customization Class 26
6.3 Enabling Application for Seeded Customization 27
6.4 SOA Customization Example Use Cases 28
6.4.1 Add a Partner Link to an Existing Process 28
6.4.2 Add a Human Task to an Existing Process 46

List of Figures

Figure 3–1 SOA Customization 15

Figure 4–1 SM500 page 18

Figure 4–2 SM500 page with appropriate Datatype 18

Figure 4–3 To Fetch Attribute Value 19

Figure 4–4 Custom Attributes 20

Figure 4–5 Redeploy process 20

Figure 5–1 Architecture Diagram 21

Figure 6–1 Add an entry for new Customization Layer 26

Figure 6–2 Create Customization Class 27

Figure 6–3 Enabling Application for Seeded Customization 28

Figure 6–4 Select SOA Project 29

Figure 6–5 Enter SOA Project Name 29

Figure 6–6 Configure SOA Settings 30

Figure 6–7 Create Mediator 31

Figure 6–8 Select Target Type 32

Figure 6–9 Request Transformation Map to create new mapper file 33

Figure 6–10 Mapping Input and Output string 33

Figure 6–11 Select Deployment Action 34

Figure 6–12 Deploy Configuration Settings 35

Figure 6–13 Select Deployment Server 35

Figure 6–14 Select Target SOA Server 36

Figure 6–15 Select SOA Domain 37

Figure 6–16 Test Web Service 38

Figure 6–17 Customization of SOA Application - Flow 39

Figure 6–18 Customization of SOA Application - Notify Customer 40
Figure 6–19 Add Partner Link Component 41
Figure 6–20 Add Invoke Component 42
Figure 6–21 Edit Copy Rules Variable 43
Figure 6–22 Add Assign Components - Reply 44
Figure 6–23 Design View of the BPEL Process 44
Figure 6–24 Test Customized Composite - Flow 45
Figure 6–25 Test Customized Composite - invokeEchoService 46
Figure 6–26 Select SOA Project 47
Figure 6–27 Create SOA Project Name 47
Figure 6–28 Configure SOA Settings 48
Figure 6–29 Configure BPEL Process Settings 48
Figure 6–30 Enter Human Task Name 49
Figure 6–31 Create Human Task - General Tab 49
Figure 6–32 Add Human Task Parameter 50
Figure 6–33 Create Human Task - Data Tab 50
Figure 6–34 Add Participant Type Details 51
Figure 6–35 Create Human Task - Assignment Tab 51
Figure 6–36 Select Human Task Parameters 52
Figure 6–37 Create Human Task - Delete Condition 52
Figure 6–38 Create Human Task - Expression Builder 53
Figure 6–39 Create Human Task - Copy Rules 53
Figure 6–40 Create Human Task - BPEL Process 54
Figure 6–41 Select Human Task Form 55
Figure 6–42 Select Human Task Form Deployment Action 56
Figure 6–43 Select Human Task Form - Weblogic Options 56

Figure 6–44 Add Customization Scope to SOA Application 57
Figure 6–45 Add Partner Link Component 58
Figure 6–46 Add Invoke Component 59
Figure 6–47 Add Receive Component using BPEL functions 60
Figure 6–48 Add Assign Component 61
Figure 6–49 Deploy and Test Customized SOA Composite - My Tasks Tab 62
Figure 6–50 Deploy and Test Customized SOA Composite - Flow 62
Figure 6–51 Deploy and Test Customized SOA Composite - Invoke Input 63
Figure 6–52 Deploy and Test Customized SOA Composite - Receive Output 63

Preface

This guide explains customization and extension of Oracle Banking Enterprise Originations.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This guide is intended for the users of Oracle Banking Enterprise Originations.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/us/corporate/accessibility/index.html.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/us/corporate/accessibility/support/index.html#info or visit http://www.oracle.com/us/corporate/accessibility/support/index.html#trs if you are hearing impaired.

Related Documents

For more information, see the following documentation:

- For installation and configuration information, see the Oracle Banking Enterprise Originations Localization Installation Guide - Silent Installation guide.
- For a comprehensive overview of security, see the Oracle Banking Enterprise Originations Security Guide.
- For the complete list of licensed products and the third-party licenses included with the license, see the Oracle Banking Enterprise Originations Licensing Guide.
- For information related to setting up a bank or a branch, and other operational and administrative functions, see the Oracle Banking Enterprise Originations Administrator Guide.
- For information on the functionality and features, see the respective Oracle Banking Enterprise Originations Functional Overview documents.
- For recommendations of secure usage of extensible components, see the Oracle Banking Enterprise Originations Secure Development Guide.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 About This Guide

This guide is applicable for the following products:

- Oracle Banking Platform
- Oracle Banking Enterprise Originations

References to Oracle Banking Platform or OBP in this guide apply to all the above mentioned products.

2 Objective and Scope

This chapter defines the objective and scope of this document.

2.1 Overview

Oracle Banking Platform (OBP) is designed to help banks respond strategically to today's business challenges, while also transforming their business models and processes to reduce operating costs and improve productivity across both front and back offices. It is a one-stop solution for a bank that seeks to leverage Oracle Fusion experience for its core banking operations, across its retail and corporate offerings.

OBP provides a unified yet scalable IT solution for a bank to manage its data and end-to-end business operations with an enriched user experience. It comprises pre-integrated enterprise applications leveraging and relying on the underlying Oracle Technology Stack to help reduce in-house integration and testing efforts.

2.2 Objective and Scope

Most product development can be accomplished through highly flexible system parameters and business rules. Further competitive differentiation can be achieved through IT configuration and extension support. In OBP, additional business logic required for certain services is not always a part of the core product functionality but could be a client requirement. For these purposes, extension points and customization support have been provided in the application code which can be implemented by the bank and / or by partners, wherein the existing business logic can be added with or overridden by customized business logic. This way the time consuming activity of custom coding to enable region specific, site specific or bank specific customizations can be minimized.

2.2.1 Extensibility Objective

The broad guiding principles with respect to providing extensibility in OBP are summarized below:

- Strategic intent for enabling customers and partners to extend the application.
- Internal development uses the same principles for client specific customizations.
- Localization packs
- Extensions by Oracle Consultants, Oracle Partners, Banks or Bank Partners.
- Extensions through the addition of new functionality or modification of existing functionality.
- Planned focus on this area of the application. Hence, separate budgets specifically for this.
- Standards based - OBP leverages standard tools and technology
- Leverage large development pool for standards based technology.
- Developer tool sets provided as part of JDeveloper and Eclipse for productivity.

2.3 Complementary Artefacts

The document is a developer's extensibility guide and does not intend to work as a replacement of the functional or technical specification, which would be the primary resource covering the following:

- OBP Zen training course
- OBP installation and configuration
- OBP parameterization as part of implementation
- Functional solution and product user guide

References to plugin indicate the eclipse based OBP development plugin for relevant version of OBP being extended. The plugin is not a product GA artefact and is a means to assist development. Hence, the same is not covered under product support.

2.4 Out of Scope

The scope of extensibility does not intend to suggest that OBP is forward compatible.

3 Overview of Use Cases

The use cases that are covered in this document shall enable the developer in applying the discipline of extensibility to OBP. While the overall support for customizations is complete in most respects, the same is not a replacement for implementing a disciplined, thoughtful and well-designed approach towards implementing extensions and customizations to the product.

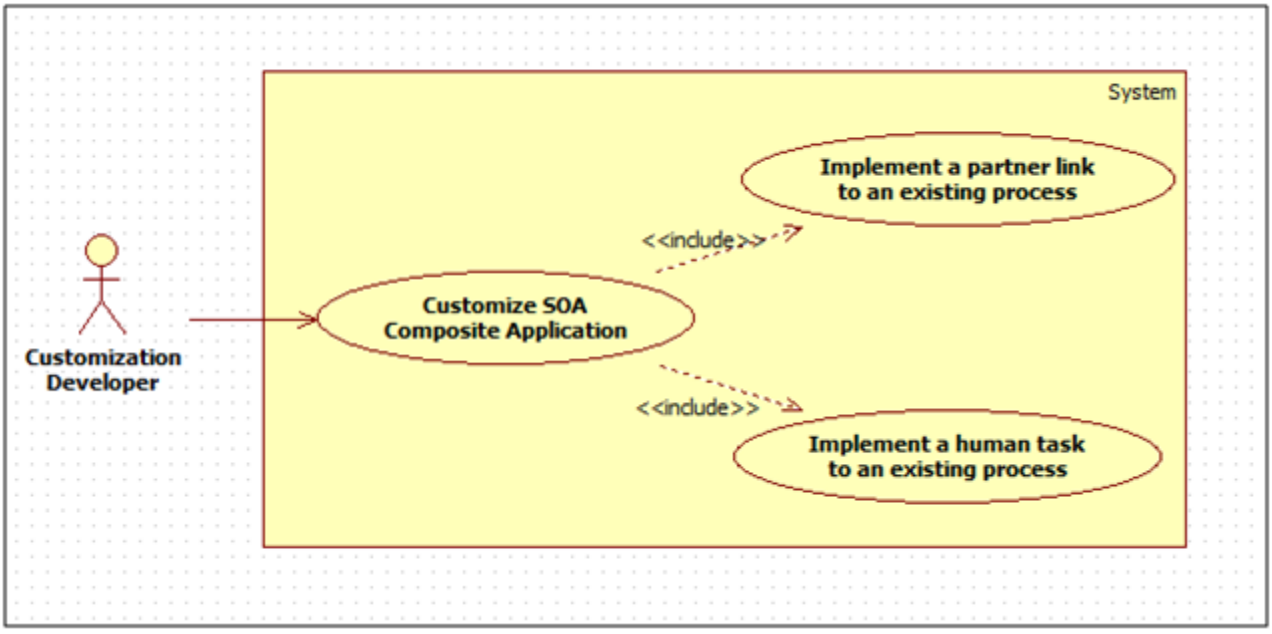
3.1 Extensibility Use Cases

This section gives an overview of the extensibility topics and customization use cases to be covered in this document. Each of these topics is detailed in the further sections.

3.1.1 SOA Customization

OBP Application provides the feature for customizing SOA composite applications based on the additional requirements which may vary from client to client. It includes implementing the partner link to an existing process or implementing human tasks or sub processes which can be hooked into an existing product process.

Figure 3–1 SOA Customization



4 Workflow Task Routing and Assignment Rules

This chapter covers the extensibility aspects of business rules used in SOA.

4.1 Workflow Task Routing and Assignment Rules

As part of overall origination workflow, different human tasks are created. Each human task can be routed to specific assignees. Such task assignment can be configured using a business rules. All such business rules are logically grouped together into different BPEL composites based on the functionality. Specific list of facts are provided out of the box for rule configuration. However, more facts can be made available for rule configuration by customizing specific application services which return the list of facts.

Below sections provide the information about services which can be extended to include more custom facts.

4.2 Origination Human Tasks

Custom facts can be added to the response of below application service using post extensions:

- com.ofss.fc.appx.origination.service.core.OriginationBusinessIndicatorApplicationServiceSpi

Key names used to populate the dictionary object of the service response are made available automatically as custom facts for configuration of business rule. These custom attributes can be accessed in the rule configuration as below:

- getCustomTextAttribute(customAttributes,"<Custom Attribute Name>")

4.3 Origination Approval Human Tasks

Approval workflow routing rules can be extended to include existing or entirely new attributes.

This is demonstrated using an example:

For structure solution approvals, routing to be setup based on whether the task includes a secured product.

Use case:

Table 4–1 Use Case

Service-id	com.ofss.fc.appx.origination.service.lending.core.application.LendingApplicationServiceSpi.confirmStructureSolution
Approval Process	com.ofss.fc.approval.lending_confirmstructuresolution
Custom attribute	LendingStructureSolution_IsTaskIncludesSecuredProduct

Broad steps to use custom attributes in routing rule:

- Step 1: Create custom attribute.
- Step 2: Configure adapter name that will be used to retrieve value of the custom attribute.
- Step 3: Develop the adapter.
- Step 4: Modify routing rule to include the custom adapter.

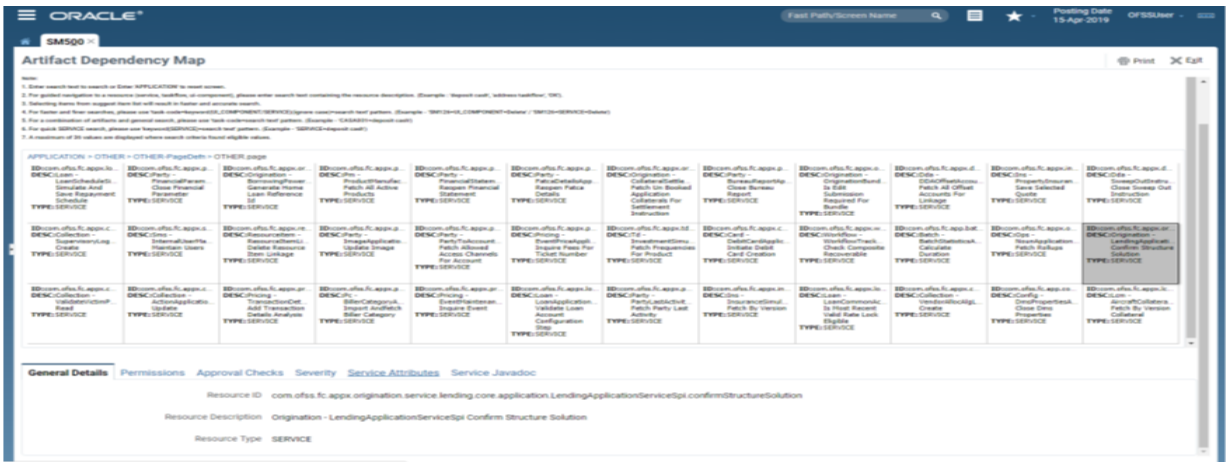
Step 1: Create the new attribute using Artifact Dependency Map (Fast Path: SM500) page

This configuration ensures that OBP fetches the name for the custom attribute as part of its attribute evaluation framework when confirm-structure-solution is executed.

Step 1.A: Search for below service ID in Artifact Dependency Map (Fast Path: SM500) page

'com.ofss.fc.appx.origination.service.lending.core.application.LendingApplicationServiceSpi.confirmStructureSolution'

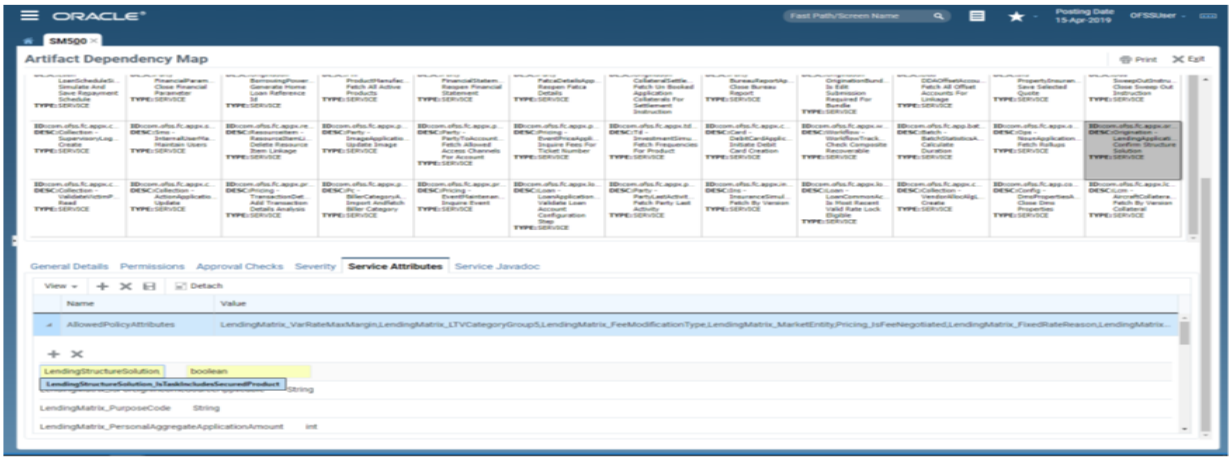
Figure 4–1 SM500 page



Step 1.B: Add attribute name 'LendingStructureSolution_IsTaskIncludesSecuredProduct'

With appropriate datatype in 'AllowedPolicyAttributes' under 'Service Attributes' tab in SM500 page as shown below:

Figure 4–2 SM500 page with appropriate Datatype



Step 2: Define the attribute in Constraint Attribute config with associated adapter to fetch attribute value

This configuration step ensures that the attribute retrieval framework executes the appropriate adapter to fetch the value of the custom attribute.

In flx_fw_config_all_b, insert the mapping of attribute and it's associated adapter.

Example:

Use the below insert statement to map attribute and the associated adapter developed to supply attribute value.

Figure 4–3 To Fetch Attribute Value

```
Insert into FLX_FW_CONFIG_ALL_B
(PROP_ID,CATEGORY_ID,PROP_VALUE,FACTORY_SHIPPED_FLAG,PROP_COMMENTS,SUMM
ARY_TEXT,CREATED_BY,CREATION_DATE,LAST_UPDATED_BY,LAST_UPDATED_DATE,OBJE
CT_STATUS_FLAG,OBJECT_VERSION_NUMBER) values
('LendingStructureSolution_IsTaskIncludesSecuredProduct','ConstraintAttributeHelper','<Custom-
adapter-name-as-per-extension-naming-standards>','N','<description>','<create-user>',SYSDATE,
<create-user>'),'A',1);
```

The category_id 'ConstraintAttributeHelper' (or an appropriate override) in Preferences.xml will be used to retrieve the adapter added.

Example:

<Preference name="ConstraintAttributeHelper"
PreferencesProvider="com.ofss.fc.infra.config.impl.DBBasedPropertyProvider"

parent="jdbcpreference"

propertyFileName="select prop_id, prop_value from flx_fw_config_all_b where category_id = 'ConstraintAttributeHelper'" syncTimeInterval="1800000" />

Step 3: Add derivation logic to set the attribute value in Adapter that is mapped in config

Develop the adapter mentioned in Step 2, method name to be named appropriately.

Example:

If adapter-name in Step 2 = 'FacilityApprovalDataMatrixAdapter' the method name will be

'getIsTaskIncludesSecuredProduct'

Step 4: Using custom fact in business rule

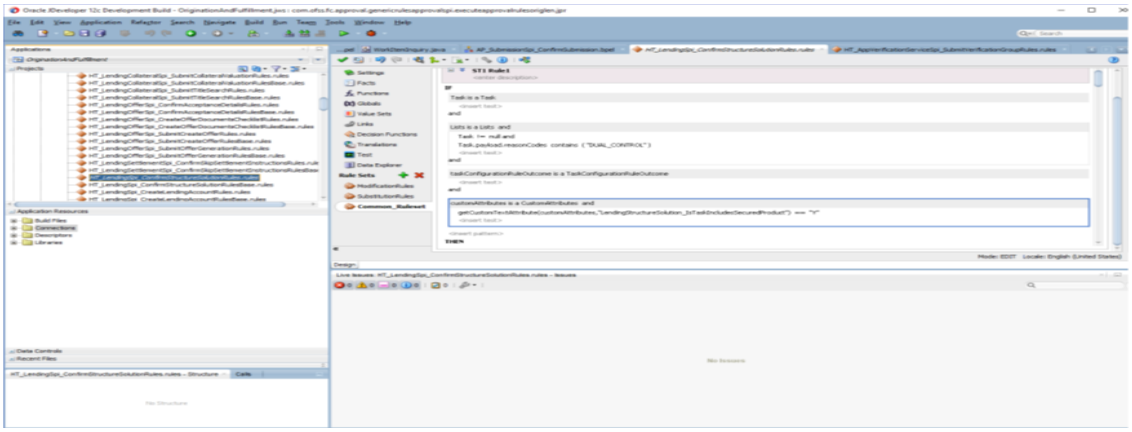
a. Extract the rules process corresponding to the approvals process:OriginationAndFulfillment\process\com.ofss.fc.approval.genericrulesapprovalspi.executeapprovalrulesoriglen\SOA\oracle\rules\HT_LendingSpi_ConfirmStructureSolutionRules.rules
b. Open .rules file (HT_LendingSpi_ConfirmStructureSolutionRules.rules).
c. In ruleset, modify rule with the desired condition using customAttributes (see example and screenshot below)

Figure 4–4 Custom Attributes

```
customAttributes is a CustomAttributes and
getCustomTextAttribute(customAttributes,"LendingStructureSolution_IsTaskIncludesSecuredProduct")=="Y"
```

d. Redeploy process

Figure 4–5 Redeploy process



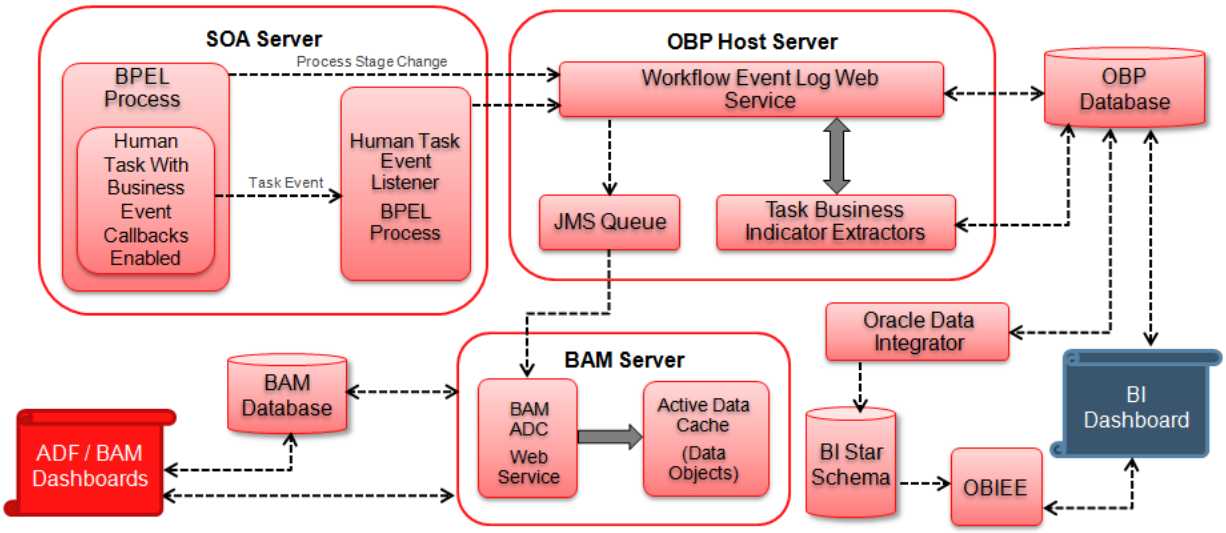
5 Workflow Analytics

This chapter explains the workflow analytics of Oracle Banking Platform (OBP).

5.1 Architecture

The following architecture diagram depicts the instrumentation approach for BPEL processes as well as human workflows. It also illustrates the approach to collect business indicators and the mechanism to publish them to the analytics system.

Figure 5–1 Architecture Diagram



5.2 OBP Workflow Analytics Extension Points

This section explains the extension points for OBP workflow analytics.

5.2.1 Introduction of new human task

The following steps can be performed for introducing new human task.

1. Enable business events in .task file.
2. Attribute "task Key" to be added in task payload. This is hash (#) separated string of all business key identifiers for the task. For example, SubmissionId#ApplicationId.
3. Write business indicator extractor class for new human task by implementing com.ofss.fc.app.adapter.workflow.task.ITaskBIExtractor.
4. Override WorkflowTaskConfiguration preference in config/preferences.xml and add constant for task specific business indicator extractor.

BIEXTRACTOR$NewHumanTaskName = "<Fully qualified path of task specific business indicator extractor class>"

5. Create new DTO extending com.ofss.fc.app.workflow.dto.bam.task.TaskMetricDTO and populate the same in business indicator extractor class. For example, NewHumanTaskMetricDTO.
6. Create a data object class corresponding to NewHumanTaskMetricDTO, for example, NewHumanTaskMetricDO by extending abstract class com.ofss.fc.integration.analytics.bam.dataobject.framework.BAMDataObject.
7. Override "BAMDataObjectAssemblers" preference in config/preferences.xml and add constant for new data object assembler.

   NewHumanTaskMetricDTO = "<Fully qualified path of data object assembler class>"

8. Create data object on BAM server in the Administration tab of BAM Composer.

5.2.2 Addition of custom business indicator(s) to existing data object

The following steps can be performed to add custom business indicators to existing OriginationTaskMetricDO data object.

1. Create new data object class, for example, CustomOriginationTaskMetricDO by extending existing com.ofss.fc.integration.analytics.bam.dataobject.banking.origination.OriginationTaskMetricDO and add new custom attributes.
2. Extend the existing implementation of task business indicator extractor class to populate dictionary object of OriginationTaskMetricDTO for new custom attributes.
3. Extend the existing implementation of data object assembler to populate and return instance of CustomOriginationTaskMetricDO.
4. Modify the definition of OriginationTaskMetricDO data object on BAM server using BAM architect tool to add custom business indicator attributes.

5.2.3 Publishing analytics data to system other than Oracle BAM

The following steps can be performed to publish analytics data to system other than Oracle BAM.

1. Write custom implementation class CustomRealtimeAnalyticsDataPublisher by extending existing class com.ofss.fc.app.adapter.impl.integration.analytics.RealtimeAnalyticsDataPublisher.
2. Write custom implementation class CustomAnalyticsDataPublisherFactory by implementing the interface com.ofss.fc.app.adapter.integration.analytics.IAnalyticsDataPublisherFactory and return the instance of CustomRealtimeAnalyticsDataPublisher class.
3. Override "AdapterFactories" preference in config/preferences.xml and override the constant value for "ANALYTICS_DATA_PUBLISHER_FACTORY".

   ANALYTICS_DATA_PUBLISHER_FACTORY = "<Fully qualified path of CustomAnalyticsDataPublisherFactory class>"

5.2.4 Publishing new DataObject from OBP Domain

Any object from OBP Domain which extends DataTransferObject can be published to BAM server by following simple steps explained below.

1. Identify/Write the new CustomDataTransferObject that extends existing DataTransferObject which needs to be published to BAM and has all the requisite attributes which need to be presented in BAM.
2. Write the new CustomDataObject class by extending existing class BAMDataObject and adding desired flat attributes which are not complex types of OBP Domain. (Use Calendar for dates and BigDecimal for amounts). Also implement methods keyFieldsAsString() and getDataObjectPath() methods so as to return identical corresponding values as maintained for this DataObject in BAM.
3. Create new CustomDataObject in BAM through the Administration tab of BAM Composer, so as the names of the attributes are identical to those kept in CustomDataObject in OBP Domain.
4. Create new CustomDataObjectAssembler class by implementing existing IDataObjectAssembler and overriding to BAMDataObject() method to return the CustomDataObject assembled from source CustomDataTransferObject.
5. Identify the preference for BAMDataObjectAssemblers from config/preferences.xml and in the preference, add corresponding entry for the new CustomDataTransferObject as key=value pair. Here, key is fully qualified class name string with dots replaced by underscores and value corresponds to the fully qualified name string of the CustomDataObjectAssembler class.
6. For example, public final String com_example_CustomDataTransferObject = "com.example.assembler. CustomDataObjectAssembler";
7. Identify the events which will publish the CustomDataObject in the system flow inside OBP Domain. On such events, assemble the DataObject and invoke publish of existing BAMDataPublisher.

6 SOA Customizations

OBP provides the functionality for customizing the SOA composite applications. The steps to customize a SOA composite application are similar to those of customizing an ADF View Controller application with a few differences. The similarities and differences would be apparent in the examples demonstrated in the following sections.

The following section provides details about the SOA Components Customization. The detailed documentation for customizing and extending the SOA Components is also available at the Oracle website:

http://docs.oracle.com/cd/E25178_01/fusionapps.1111/e16691/ext_soaedit.htm

6.1 Customization Layer

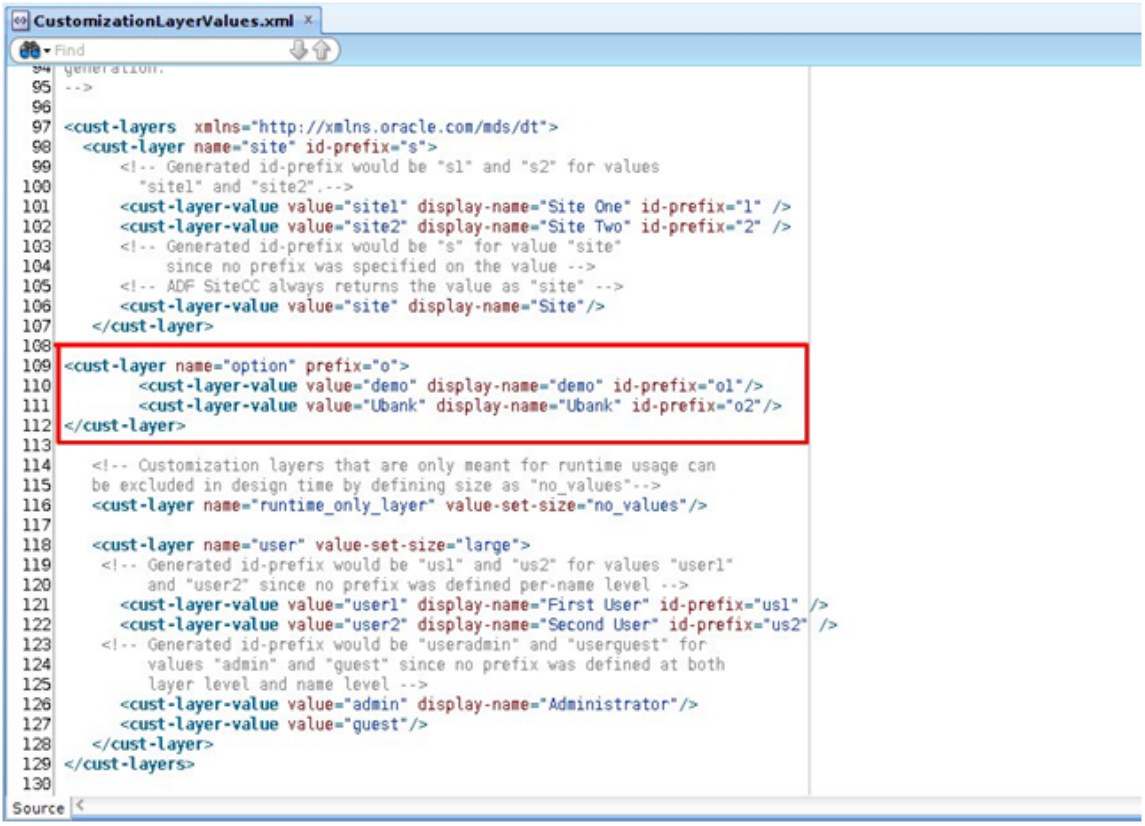
To customize an application, you must specify the customization layers and their values in the *CustomizationLayerValues.xml* file, so that they are recognized by JDeveloper.

You need to create a customization layer with name option and values demo and another bank name.

To create this customization layer, follow these steps:

1. From the main menu, choose the **File** -> Open option.
2. Locate and open the file *CustomizationLayerValues.xml* which is found in the <*JDEVELOPER_HOME*>*/jdeveloper/jdev* directory.
3. In the XML editor, add the entry for a new customization layer and values as shown in the following image.

Figure 6–1 Add an entry for new Customization Layer



4. Save and close the file.

6.2 Customization Class

Before customizing an application, a *customization* class needs to be created which is the interface that the *Oracle Meta-data Services* framework uses to define which customization layer should be applied to the application's base meta-data.

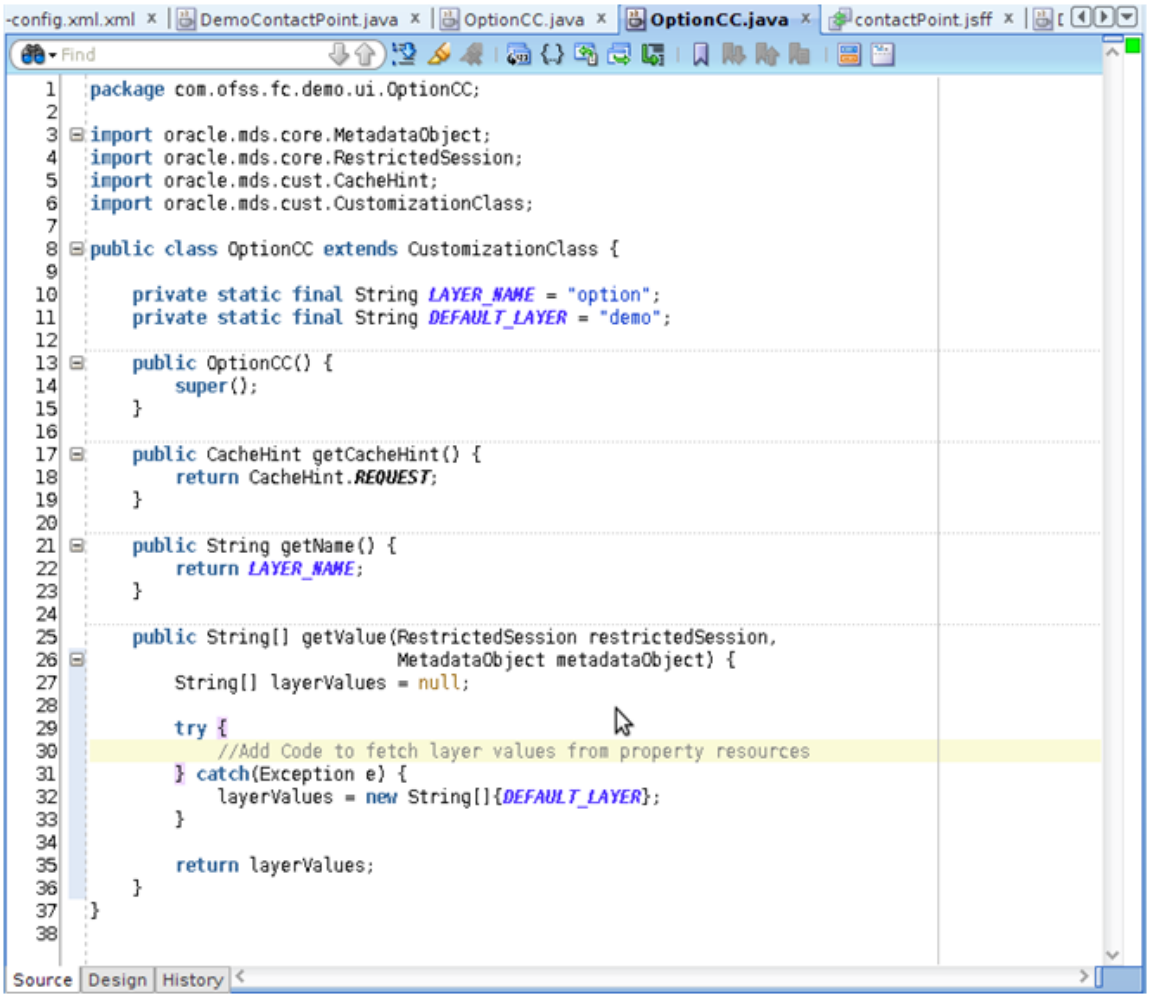
To create a customization class, follow these steps:

1. From the main menu, choose **File** -> **New**.
2. Create a generic project and give a name *(com.ofss.fc.demo.ui.OptionCC)* to the project.
3. Go to **Project Properties** for this project and add the required **MDS** libraries in the classpath of the project.
4. Create the customization class in this project. The customization class **must** extend the *oracle.mds.cust.CustomizationClass* abstract class.
5. Implement the following abstract methods of the *CustomizationClass* as follows:
    - **getCacheHint()** - This method will return the information about whether the customization layer is applicable to all users, a set of users, a specific HTTP request or a single user.

- **getName()** - This method will return the name of the customization layer.
- **getValue()** - This method will return the customization layer value at runtime.

The screenshot below depicts a sample implementation of the above methods.

Figure 6–2 Create Customization Class



6. Build this class and deploy the project as a JAR file *(com.ofss.fc.demo.ui.OptionCC.jar)*. This JAR file should only contain the customization class.
7. Place this JAR file in the location *<JDEVELOPER_HOME>/jdeveloper/jdev/lib/patches* so that the customization class is available in the classpath of Jdeveloper.

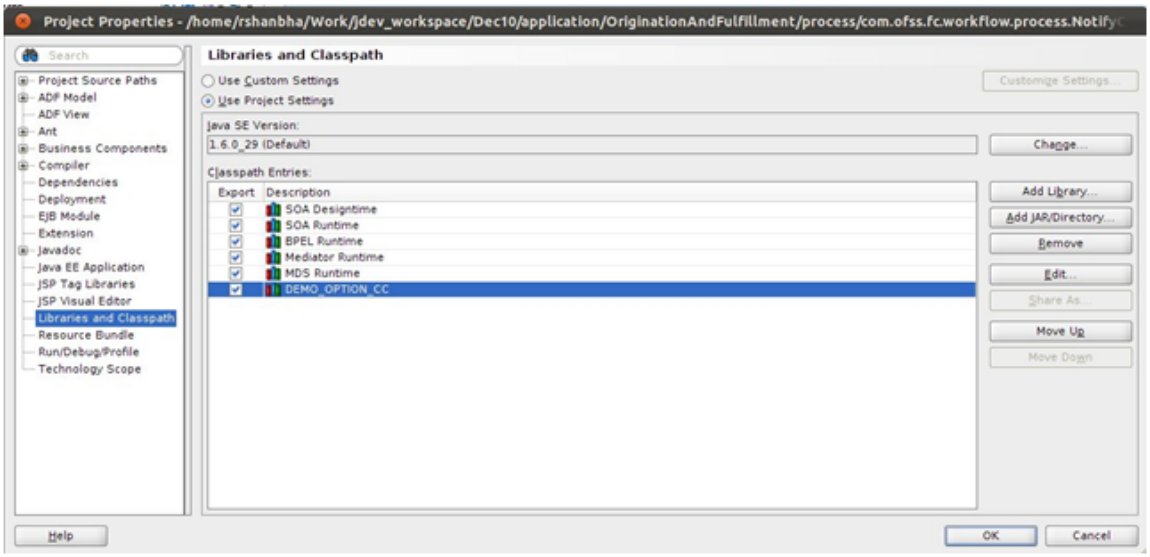
6.3 Enabling Application for Seeded Customization

Seeded customization of an application is the process of taking a generalized application and making modifications to suit the needs of a particular group. The generalized application first needs to be enabled for seeded customization before any customizations can be done on the application.

To enable seeded customization for the application, follow these steps:

1. Go to the **Project Properties** of the application's project.
2. In the *ADF View* section, check the *Enable Seeded Customizations* option.
3. In the *Libraries* and *Classpath* section, add the previously deployed com.ofss.fc.demo.ui.OptionCC.jar which contains the customization class.

Figure 6–3 Enabling Application for Seeded Customization



4. In the **Application Resources** tab, open the *adf-config.xml* present in the *Descriptors/ADF META-INF* folder.
5. In the list of *Customization Classes*, remove all the entries and add the *com.ofss.fc.demo.ui.OptionCC.OptionCC* class to this list. The sections below will elaborate in detail the actual customization of a SOA process with examples.

6.4 SOA Customization Example Use Cases

This section describes the examples use cases of SOA customization.

6.4.1 Add a Partner Link to an Existing Process

In this example of SOA customization, we will be adding a Partner Link call to an Echo Service to an existing SOA process. The Echo Service will take a string input and respond with the same string as output.

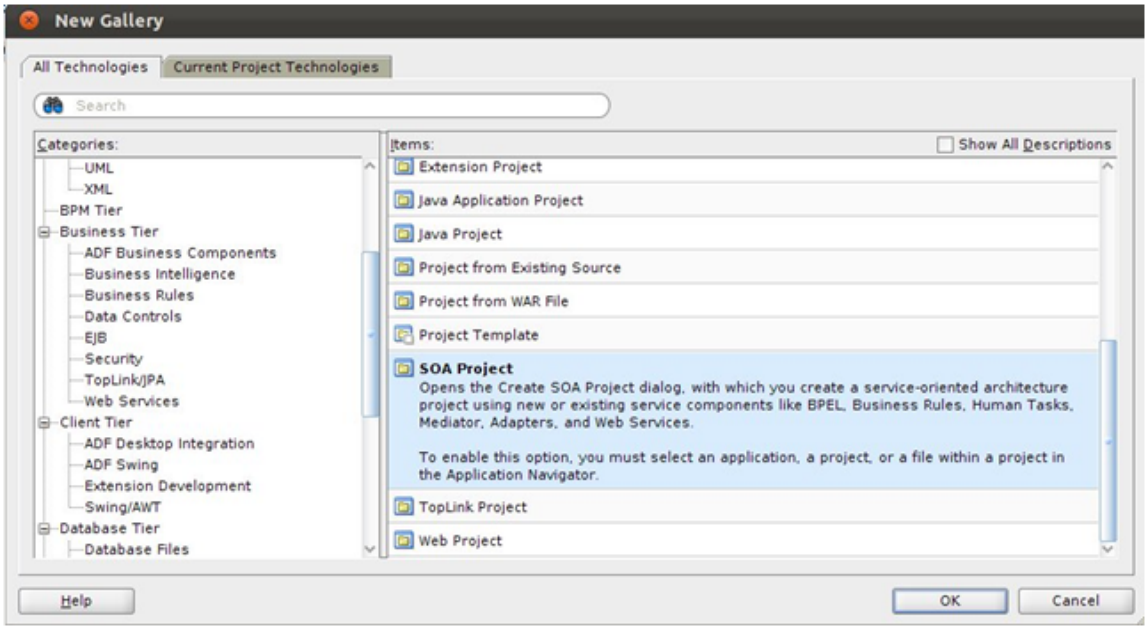
The following section will explain how to create a SOA project and process with the example of Echo Service.

Step 1 Create SOA Project

You will need to create a SOA project to contain the Echo Service process. To create the SOA project, follow these steps:

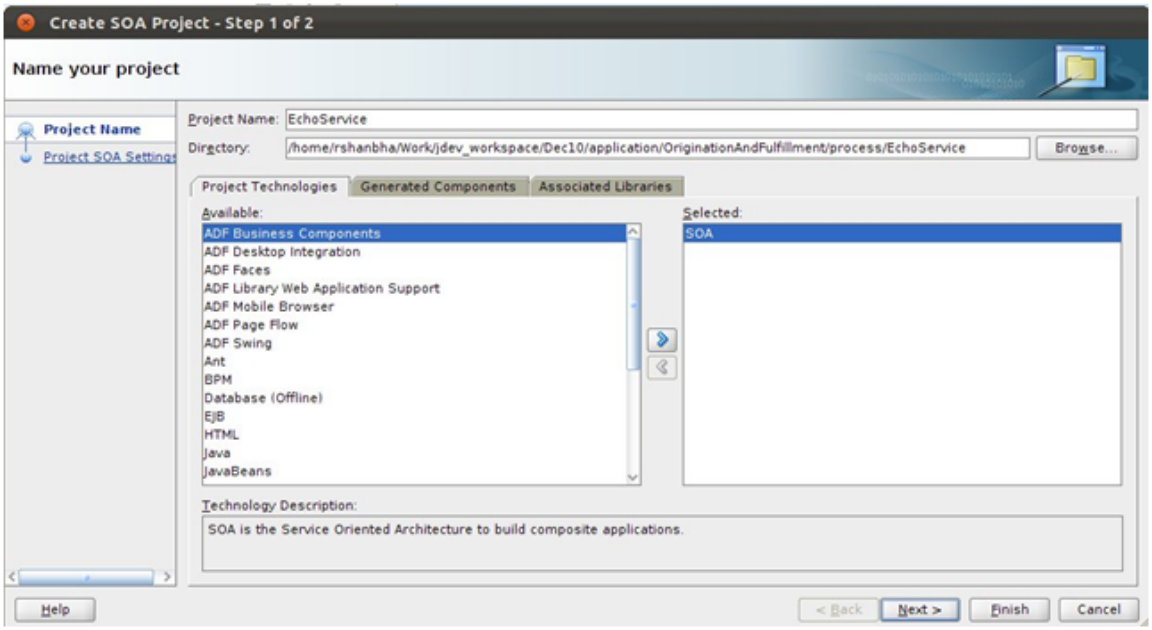
1. In the Main Menu, go to **File** -> **New**.
2. In the Project Gallery that opens, select *SOA Project* and click **OK**.

Figure 6–4 Select SOA Project



3. In the **Create SOA Project** wizard, enter appropriate project name (EchoService) and location for the project.
4. Click **Next**.

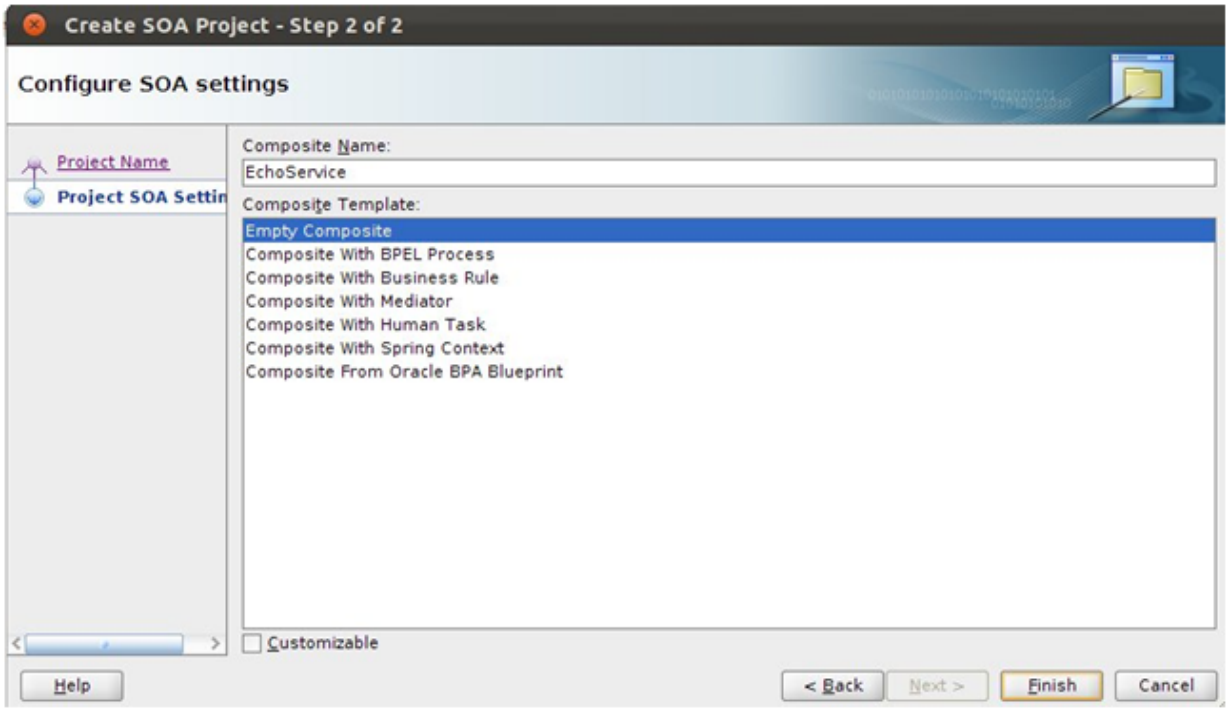
Figure 6–5 Enter SOA Project Name



5. In the next dialogue of the wizard, enter appropriate name (EchoService) for the SOA composite.

6. Select *Empty Composite* from the drop-down menu.
7. Click **Finish**.

Figure 6–6 Configure SOA Settings



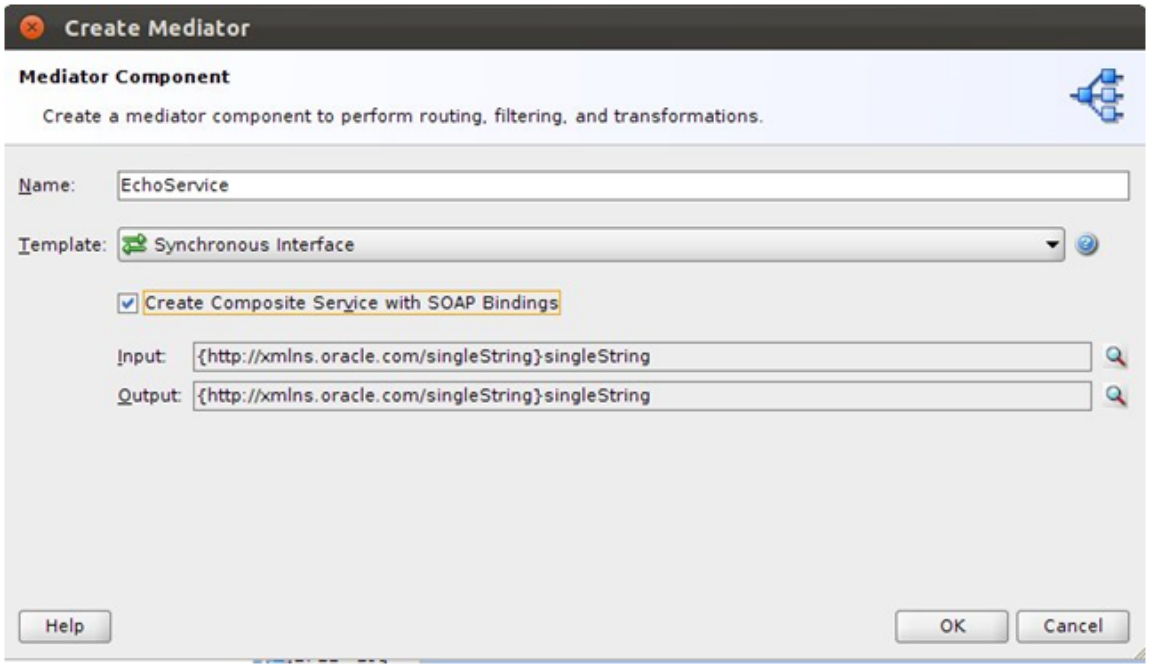
Step 2 Add Mediator Component

You will need to add a *Mediator* component to the BPEL process to process the input to the SOA process and generate an output.

To add the *Mediator*, follow these steps:

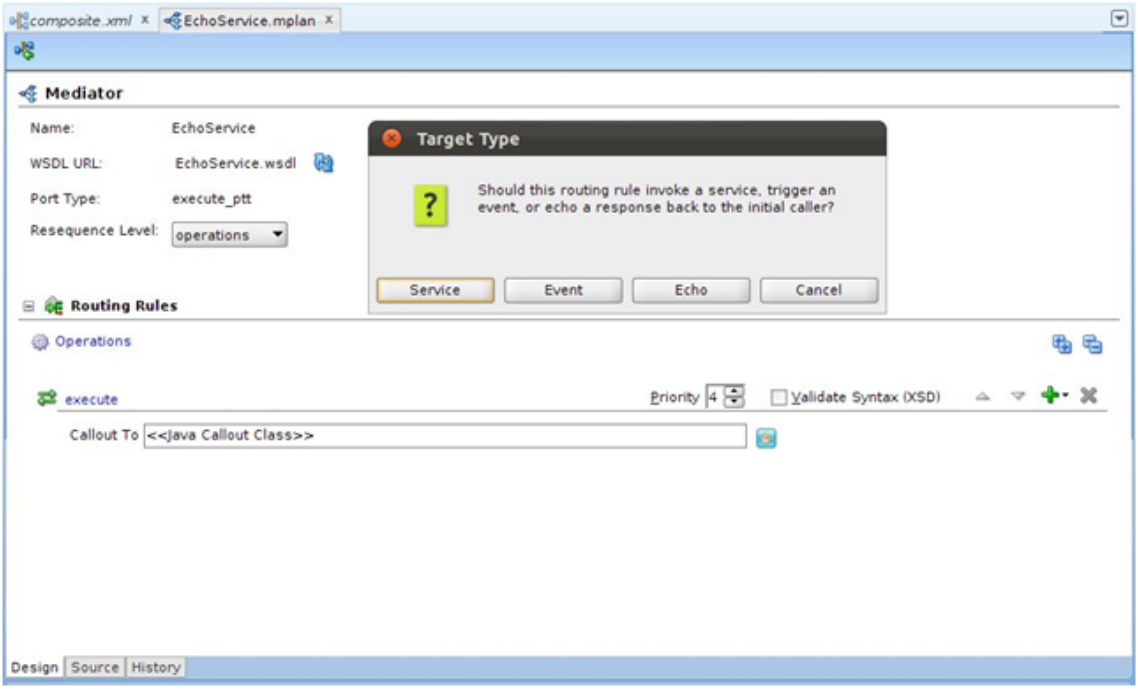
1. From the **Project Navigator** tab, select and open *EchoService.bpel* in the *Design* mode.
2. From the **Component Palette** tab, in *SOA Components* section, select the *Mediator* component.
3. Drag and drop it onto the *bpel* process.
4. In the **Create Mediator** dialogue that opens, enter appropriate name (EchoService).
5. From the **Templates** drop-down, select *Synchronous* Interface.
6. Check the *Create Composite Service with SOAP Bindings* option.
7. Click **OK**.

Figure 6–7 Create Mediator



8. An *EchoService.mplan* file will be created. Open this file in **Design** mode.
9. In the *Routing Rules* section, click the icon for *Add*.
10. Select *Static Routing Rule* from pop-up menu.
11. In the **Target Type** dialogue that opens, click **Echo**.

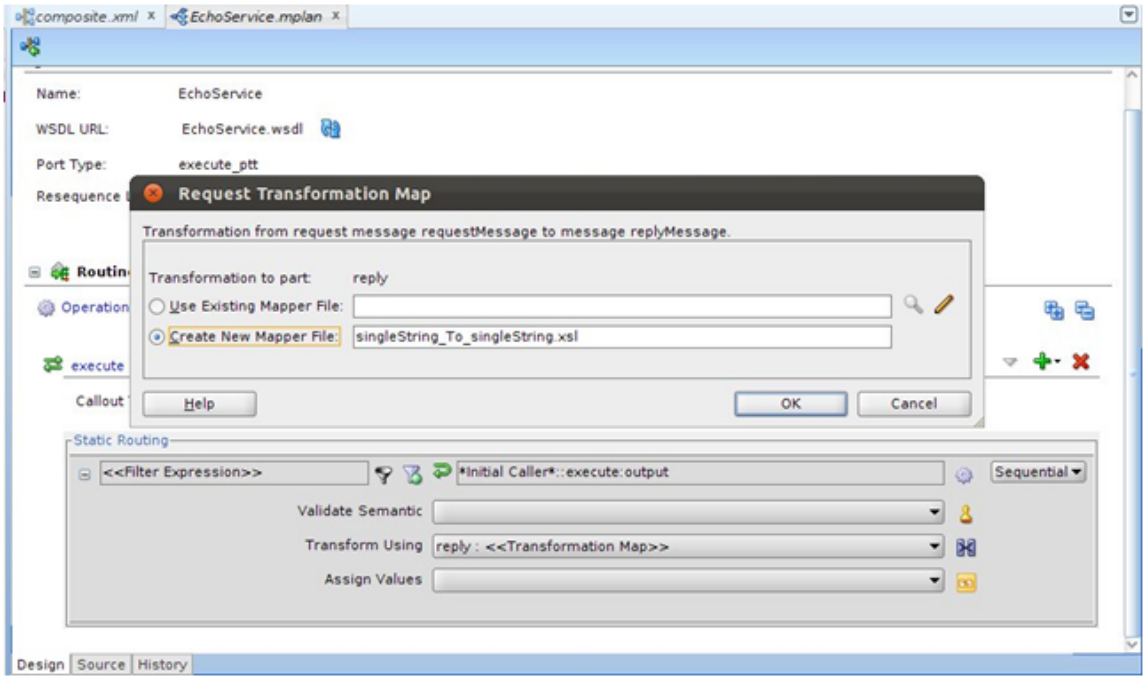
Figure 6–8 Select Target Type



A *Static Routing* section will be added to the screen.

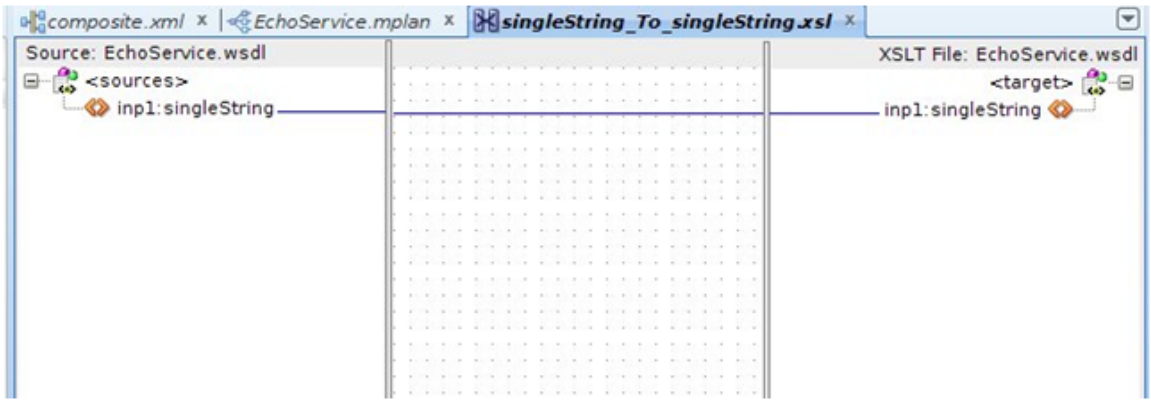
12. Click the icon next to the *Transform Using* drop-down.
13. In the **Request Transformation Map** dialogue that opens, select the option **Create New Mapper File**.
14. Click **OK**.

Figure 6–9 Request Transformation Map to create new mapper file



15. This will create a *singleString_to_singleString.xsl* file. Open this file in **Design** mode. You will see the input parameters in tree format on the left hand side and the output parameters on the right hand side of the screen.
16. In this case, the input and output contain a single *string*.
17. Select the input string from the left hand side and drag and drop it to the output string on the right hand side. This will create a mapping between input and output parameters.

Figure 6–10 Mapping Input and Output string



18. Save all files and build the project.

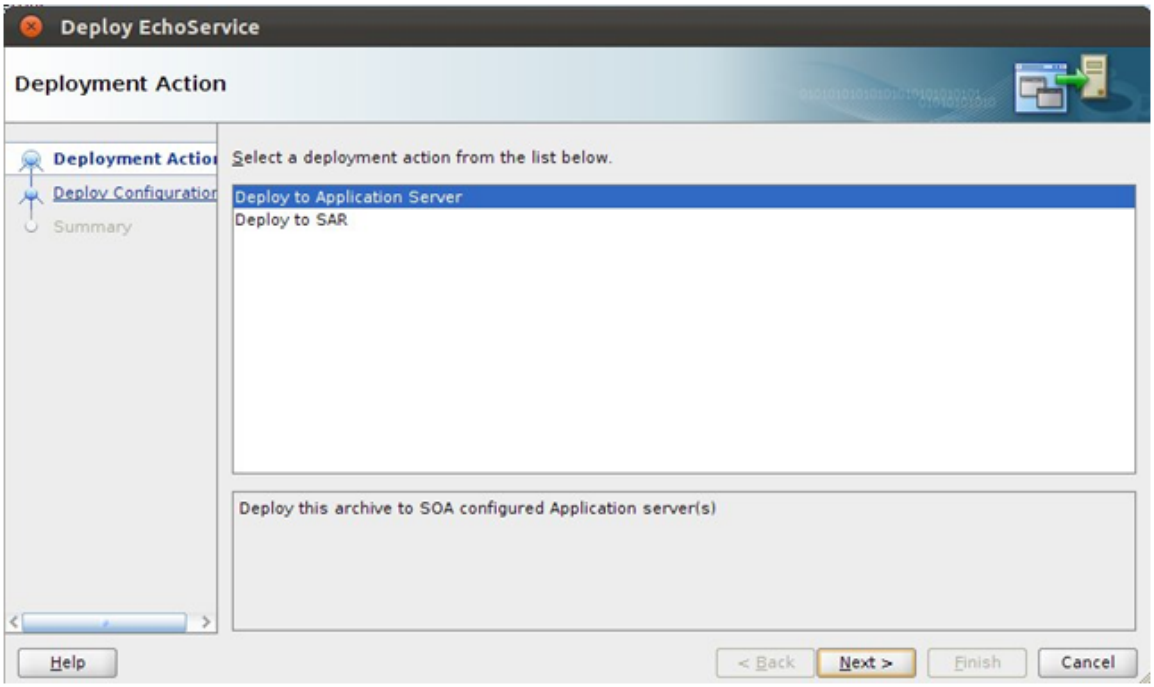
Step 3 Deploy Project to SOA Server

You will need to deploy this project to a SOA Server. From the *Admin* team, get details of the SOA Server and configure it in your JDeveloper.

After adding the SOA Server to your JDeveloper, follow these steps to deploy the *EchoService* composite to the server:

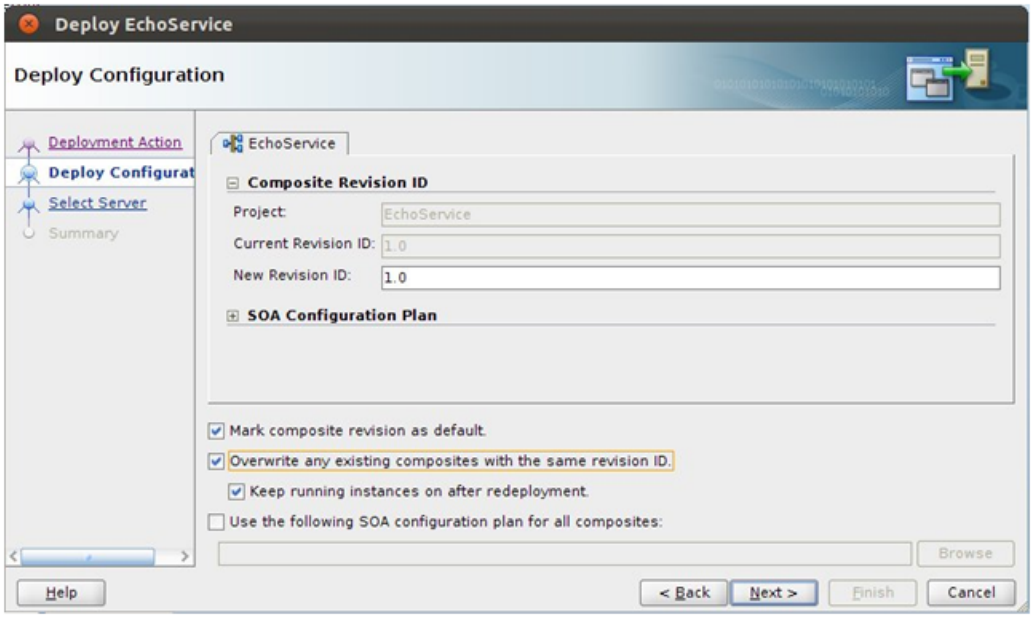
1. In the **Project Navigator** tab, right click the project and select *Deploy*.
2. In the **Deploy EchoService** dialogue that opens, select *Deploy* to *Application Server* from the list.
3. Click **Next**.

Figure 6–11 Select Deployment Action



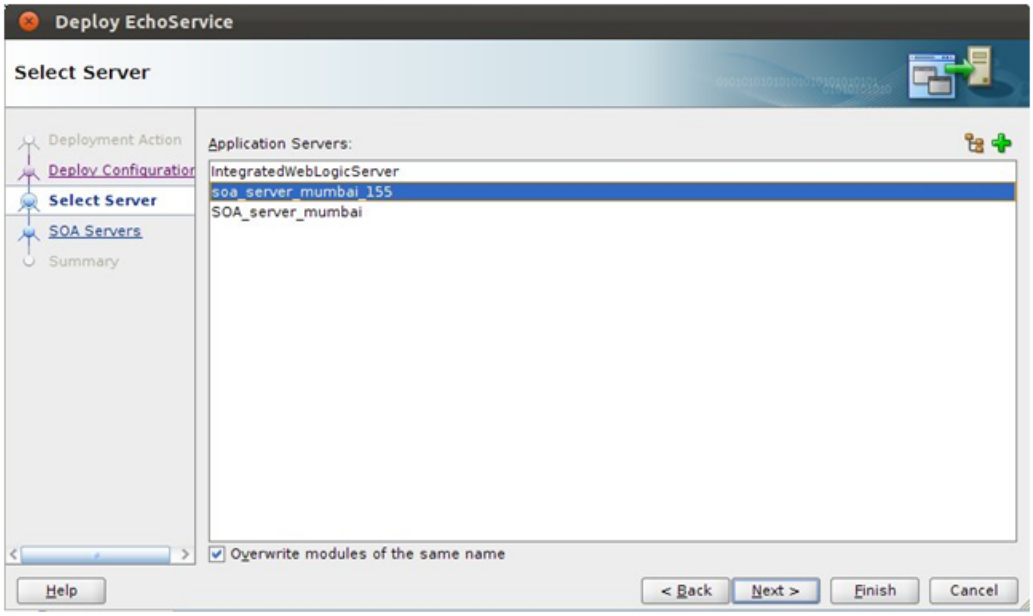
4. In the Deploy Configuration dialogue, check the option *Overwrite any existing composites with the same revision ID*.

Figure 6–12 Deploy Configuration Settings



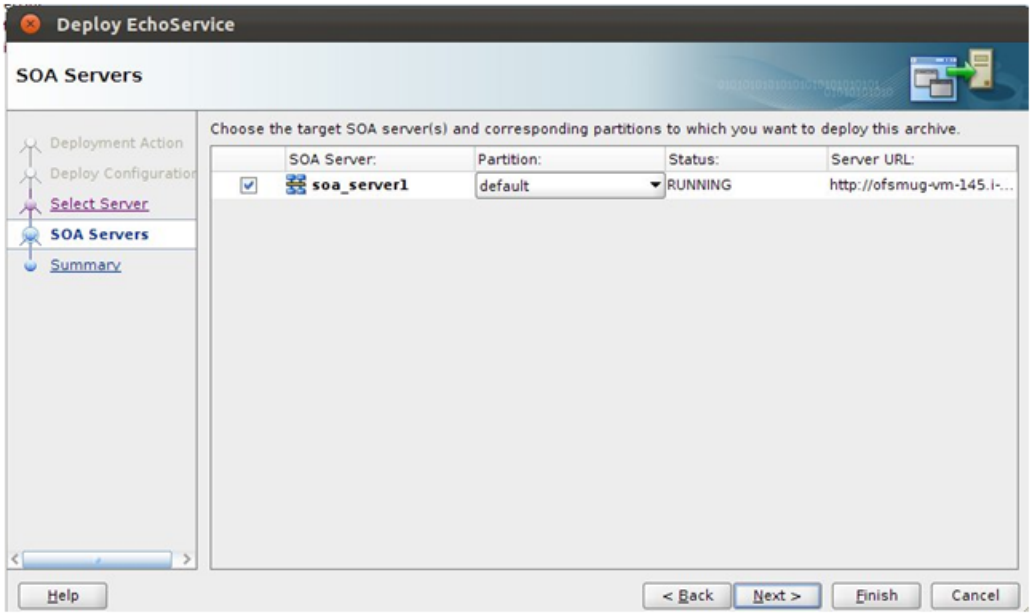
5. Click **Next**.
6. From the list of Application Servers, select the appropriate SOA Server.

Figure 6–13 Select Deployment Server



7. Click **Next.**
8. Select the appropriate *Partition* of the SOA Server where the composite should be deployed.

Figure 6–14 Select Target SOA Server



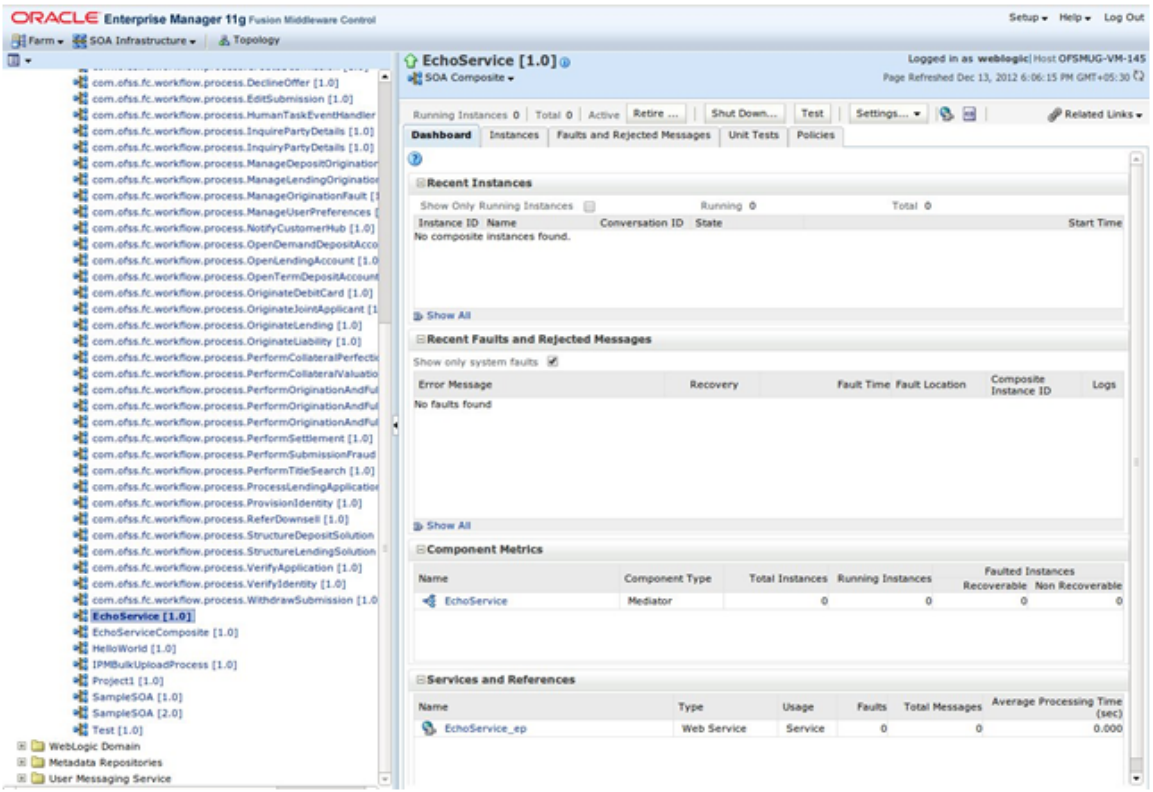
9. Click **Finish**.

Step 4 Test Echo Service

After deploying the *EchoService* composite to a SOA Server, you can test it through the EM console:

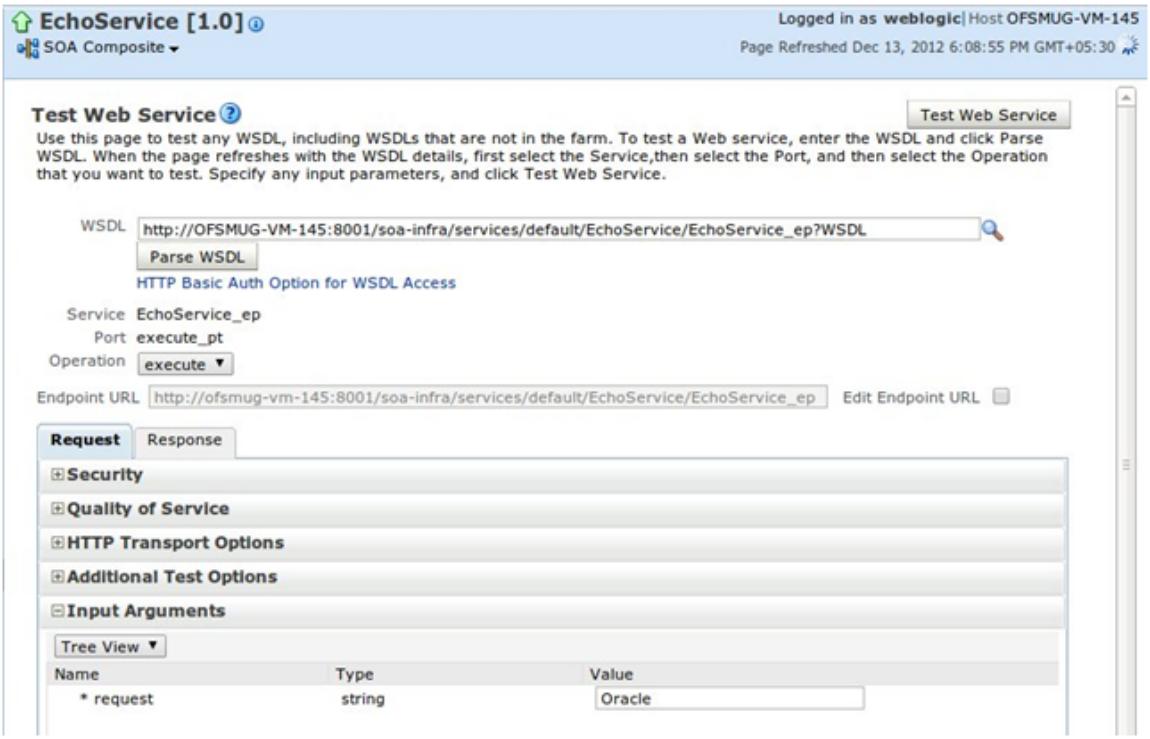
1. Log in to *em* console of the SOA Server to which the composite is deployed.
2. From the *SOA Domain* select the *EchoService* composite.

Figure 6–15 Select SOA Domain



3. On the right hand side panel, you can see the *Dashboard* which lists the instances of SOA requests to that composite and many other options.
4. Click the **Test** button to test the composite.

Figure 6–16 Test Web Service



5. In the *Input Arguments* section, enter input and click *Test Web Service*. You will be able to see the response in the *Response* section.

Step 5 Add Customizable Scope to SOA Application

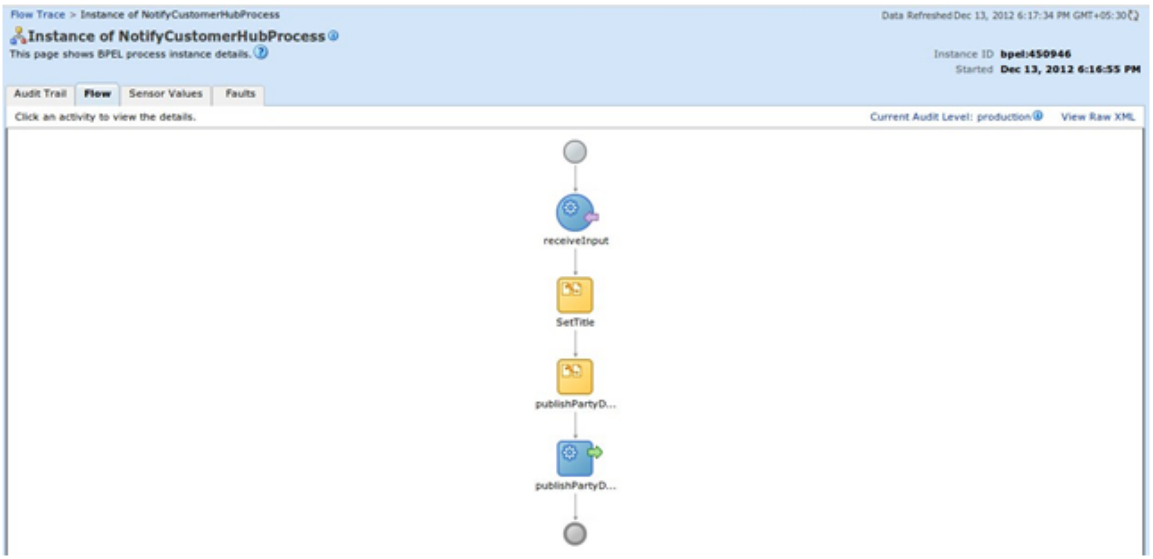
By default, a BPEL process in itself is not customizable. In addition to the steps followed to enable customizations in a SOA application, you will need to add a *Scope* component to the BPEL process and enable it for customizations.

To demonstrate customizations of a SOA process, we will be using the BPEL process *NotifyCustomerHubProcess* present in the composite *com.ofss.fc.workflow.process.NotifyCustomerHub*.

To see the flow of the *NotifyCustomerHubProcess* before customizations:

1. Deploy the composite to a SOA Server.
2. Log in to the *em* console and select the process from *SOA Domain*.
3. From the *Dashboard*, click **Test**.
4. Enter appropriate input and click *Test Web Service*.
5. From the *Dashboard*, click an *Instance* of the composite request.
6. Select the **Flow** tab to see the flow of the process.

Figure 6–17 Customization of SOA Application - Flow

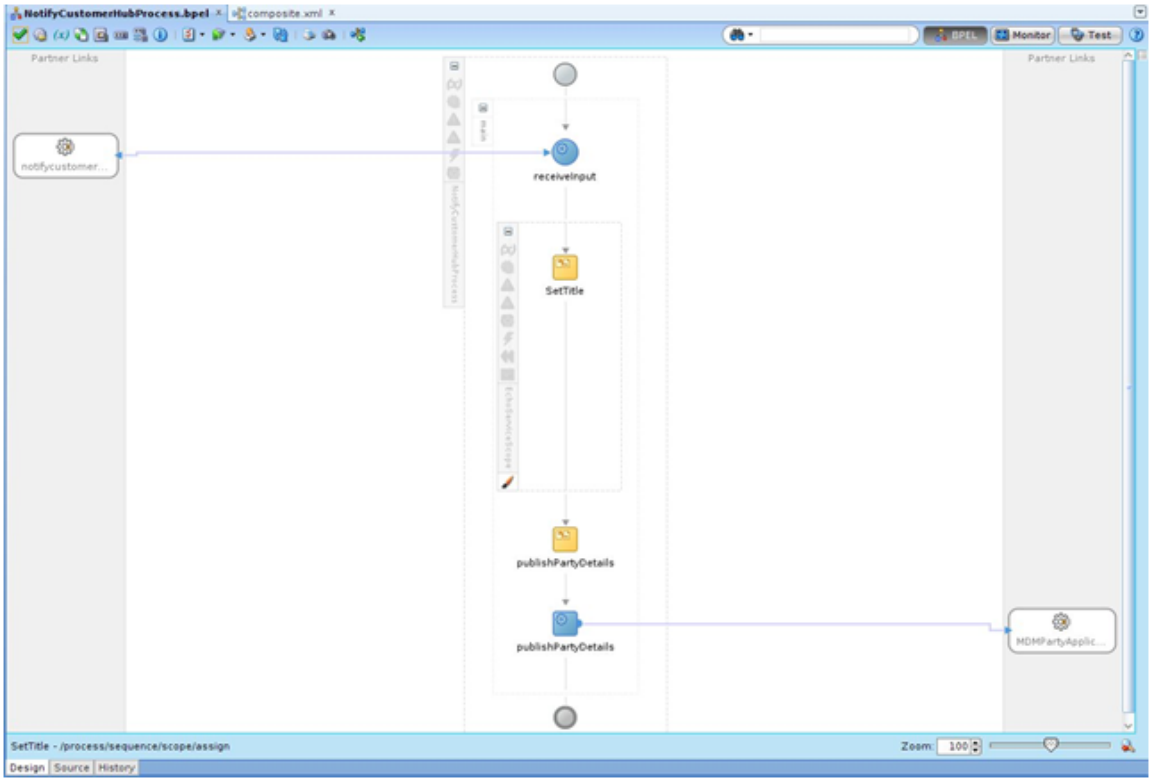


7. Open the SOA application which contains the base composite which will be customizing. The aforementioned process is present in the *OriginationAndFulfillment* application inside the *com.ofss.fc.workflow.NotifyCustomerHub project*.

To add a customizable scope to the BPEL process, follow these steps:

1. Open the *NotifyCustomerHubProcess.bpel* file in **Design** mode.
2. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the *Scope* component and drop it on to the BPEL process as shown in the figure.
3. Double-click the component and enter appropriate name (EchoServiceScope) for the component.
4. Drag and drop the existing *Assign* component labeled *setTitle* on to the newly added *EchoServiceScope* component.

Figure 6–18 Customization of SOA Application - Notify Customer



5. Right click the *Scope* component and select *Customizable* from the context menu.
6. Save all the changes and restart JDeveloper in *Customization Developer Role*.

Step 6 Customize the SOA Composite

After adding a *Customizable Scope* to the base composite, you can start performing customizations in JDeveloper's *Customization Developer Role*.

When you open the *NotifyCustomerHubProcess.bpel* file in *Design* mode, you will notice that all other components in the process, except the customizable *EchoServiceScope* component, are disabled. This means that your customizations are limited to that scope.

In the following sections, we will be adding a *Partner Link* call to the previously created *EchoService* BPEL process and other required components in the *customization* mode.

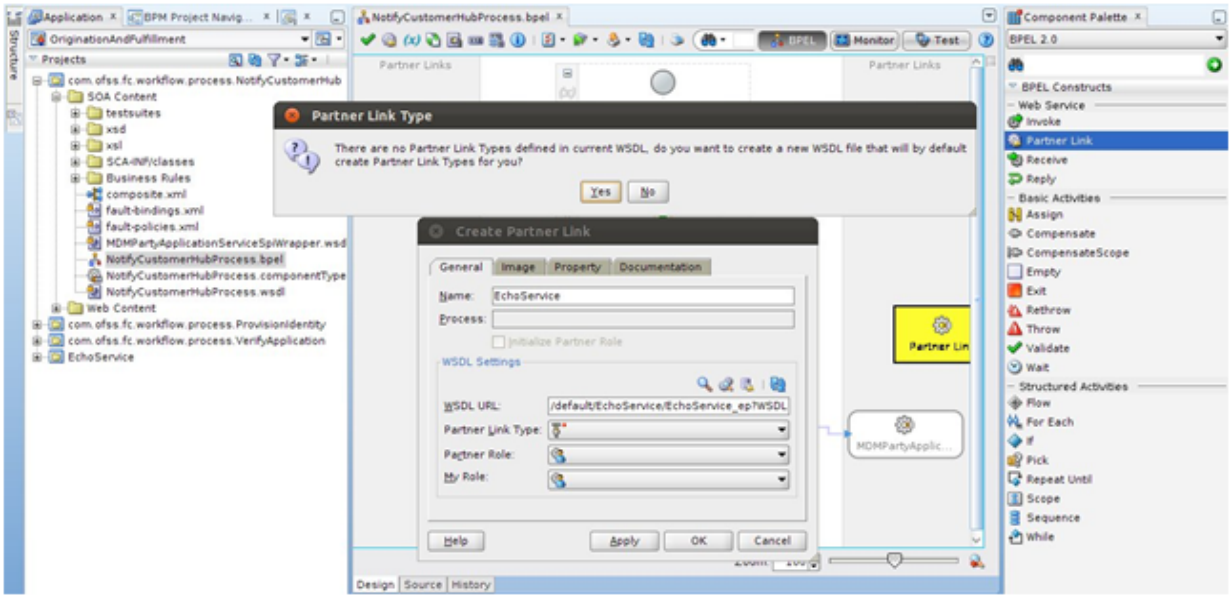
Step 7 Add Partner Link Component

To add a *Partner Link* to the BPEL process, follow these steps:

1. From the Project Navigator, open the NotifyCustomerHubProcess.bpel file in Design mode.
2. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the *Partner Link* component and drop it on to the *Partner Links* section of the BPEL process.
3. In the *Create Partner Link* dialogue that opens, enter appropriate name (EchoService) for the partner link.

4. In the *WSDL Settings* section of the dialogue, enter the URL for the previously created *EchoService* composite.
5. You will get alert notifying that there are no Partner Links defined in the current WSDL with an option to create a wrapper WSDL file with partner links defined for specified WSDL.
6. Click **Yes**. A new *EchoServiceWrapper.wsdl* file will be created which contains the partner links.
7. Select the newly defined partner link type and partner role in the *Partner Link Type* and **Partner Role** drop-down.
8. Select *Not Specified* option in the **My Role** drop-down.

Figure 6–19 Add Partner Link Component



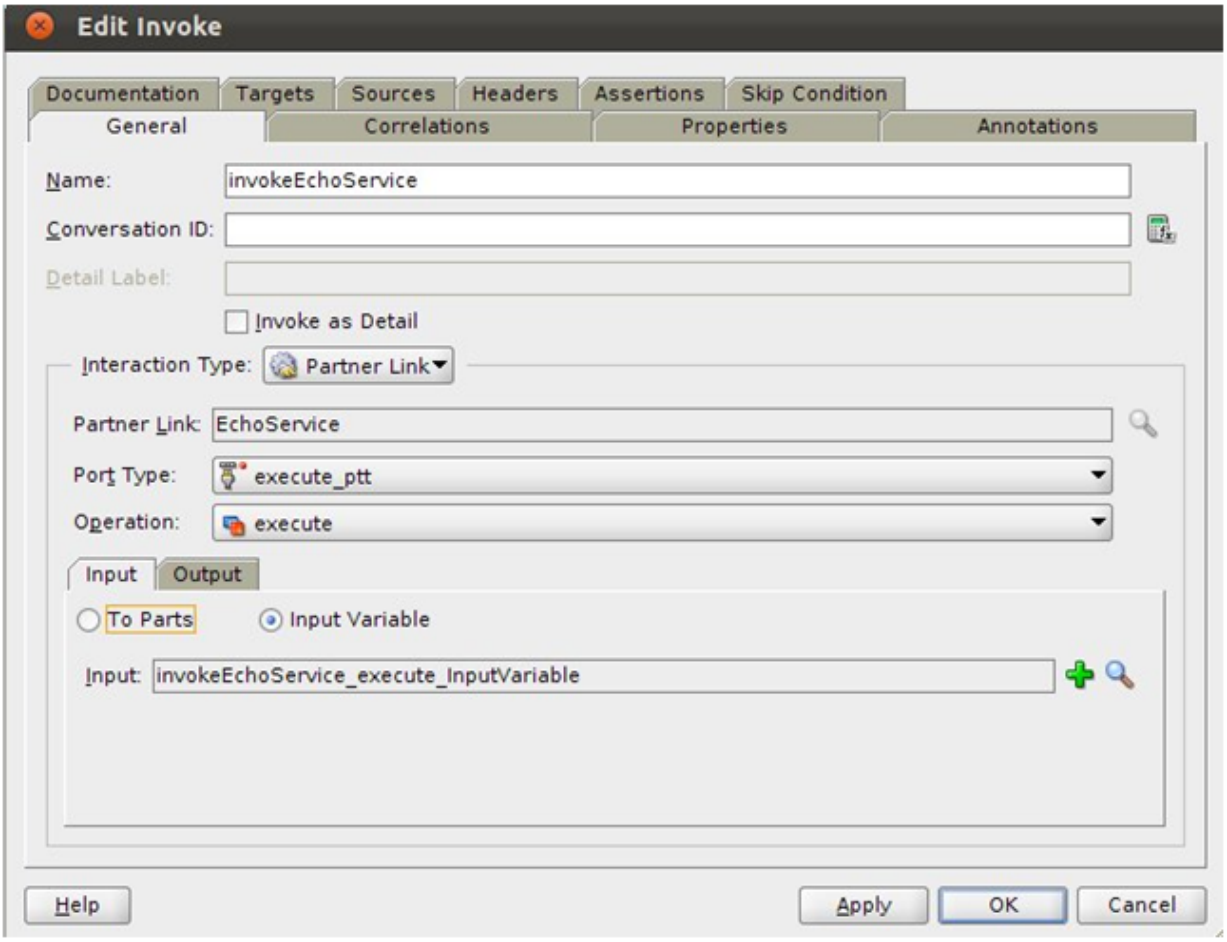
Step 8 Add Invoke Component

You will need to add an *Invoke* component to invoke the previously added partner link call to *EchoService*.

To add *Invoke component*, follow these steps:

1. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the *Invoke* component and drop it on the BPEL process inside the *EchoServiceScope* component.
2. Click the *Invoke* component and drag it to the previously added *EchoService* partner link.
3. Double-click the *Invoke* component.
4. In the *Edit Invoke* dialogue that opens, enter an appropriate name (invokeEchoService) for the component.
5. Click the icon for adding a new variable in the *Input Variable* and *Output Variable* sections.
6. Click **OK** to save the changes.

Figure 6–20 Add Invoke Component



Step 9 Add Assign Components

An *Assign* component is used to assign values to a variable. These values can be directly assigned from one variable to another or modified using BPEL functions available.

The *EchoService* accepts a single string as an input and gives a single string as an output. The *Input Variable* and *Output Variable* defined in the previously created *invokeEchoService* component will be used to hold the input value for the *EchoService* and the output returned respectively.

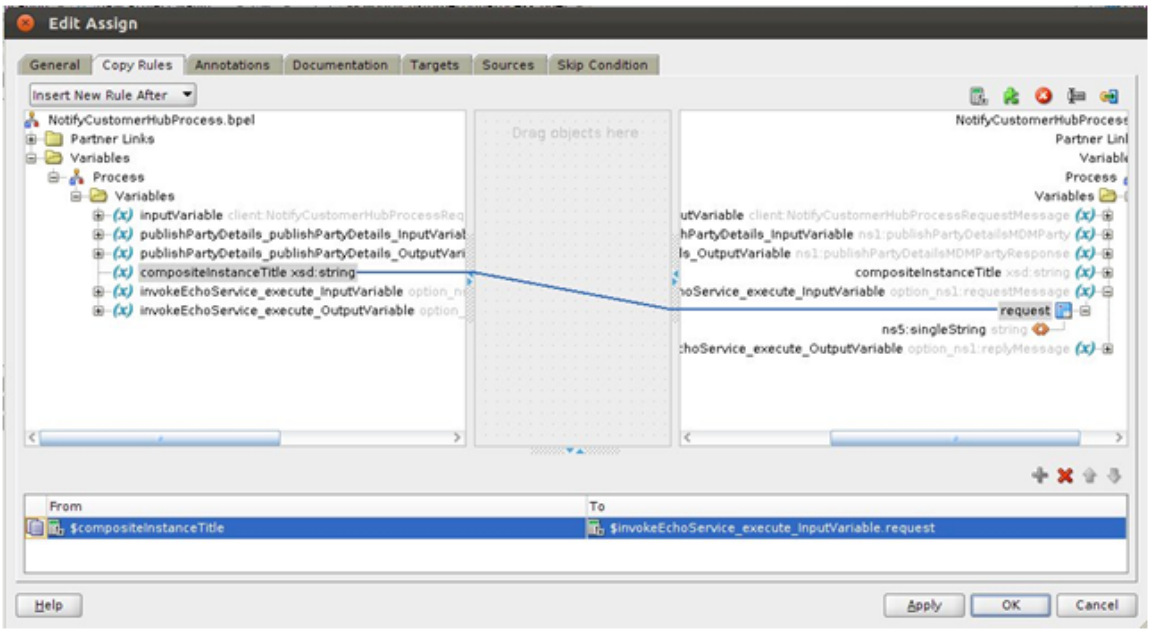
In our case, we will need to add two *Assign* components for following purposes:

- To populate the *Input Variable* of the *invokeEchoService* component with the value returned by the existing *setTitle* component.
- To populate the *setTitle* component with the value returned in the *Output Variable* of the *invokeEchoService* component.
- To add the *Assign* components

To add the *Assign* components, follow these steps:

1. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the *Assign* component and drop it on the BPEL process inside the *EchoServiceScope* component between the *setTitle* and *invokeEchoService* components.
2. Double-click the *Assign* component.
3. In the *Edit Assign* dialogue that opens, enter appropriate name (copyToEchoServiceInput) for the component.
4. In the **Copy Rules** tab, select the *compositeInstanceTitle* from the left hand side tree and drag it to the *invokeEchoService_inputVariable* on the right hand side screen as shown in the figure.
5. Click **OK**.

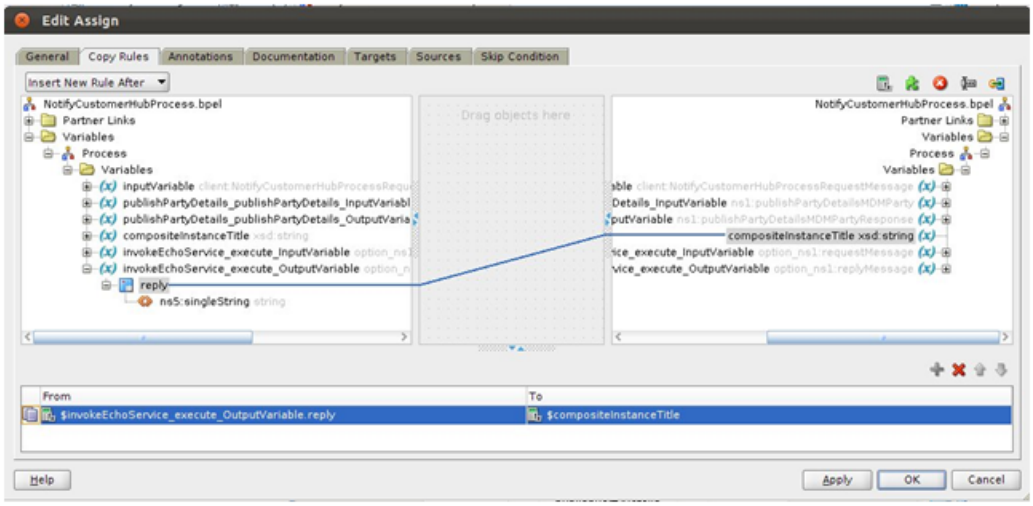
Figure 6–21 Edit Copy Rules Variable



6. Repeat the above steps for another *Assign* component labeled *copyFromEchoServiceOutput*. This component should be present after the *invokeEchoService* component.

   The *Copy Rules* for this component should be as shown in the figure below.

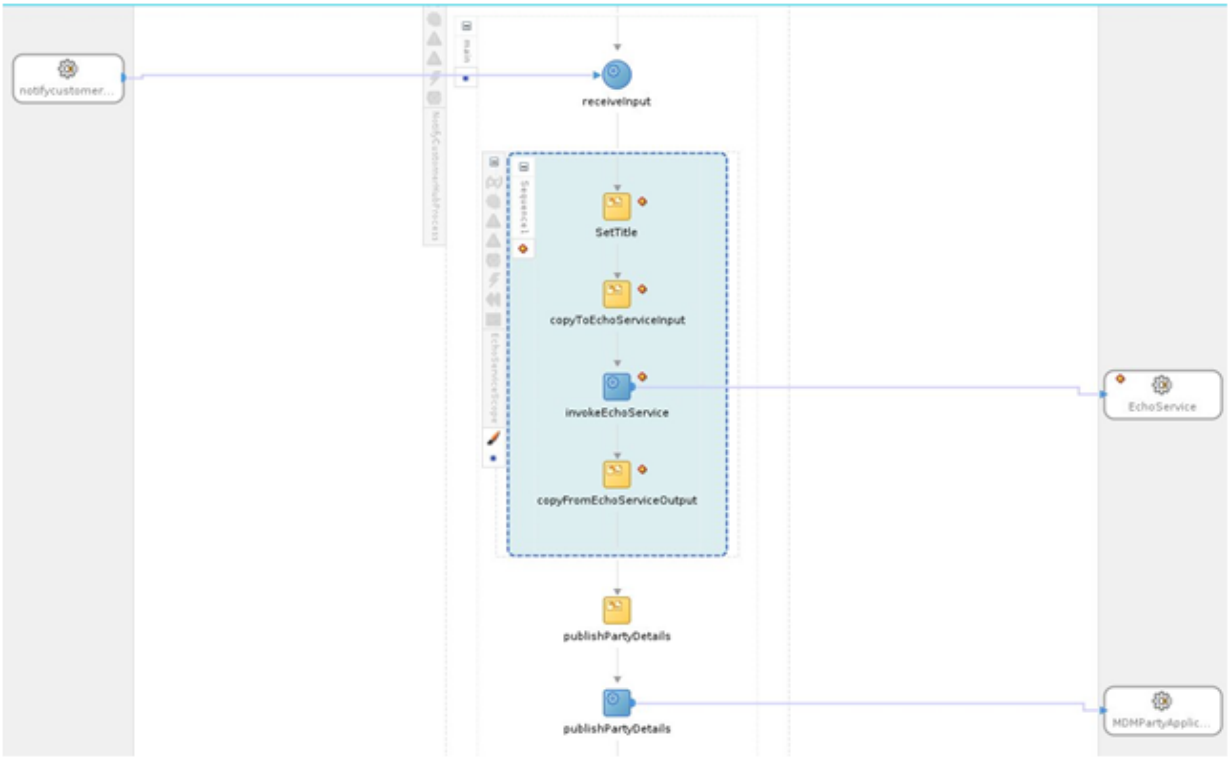
Figure 6–22 Add Assign Components - Reply



7. Save all the changes.

The **Design** view of the BPEL process should look as shown in the figure below:

Figure 6–23 Design View of the BPEL Process



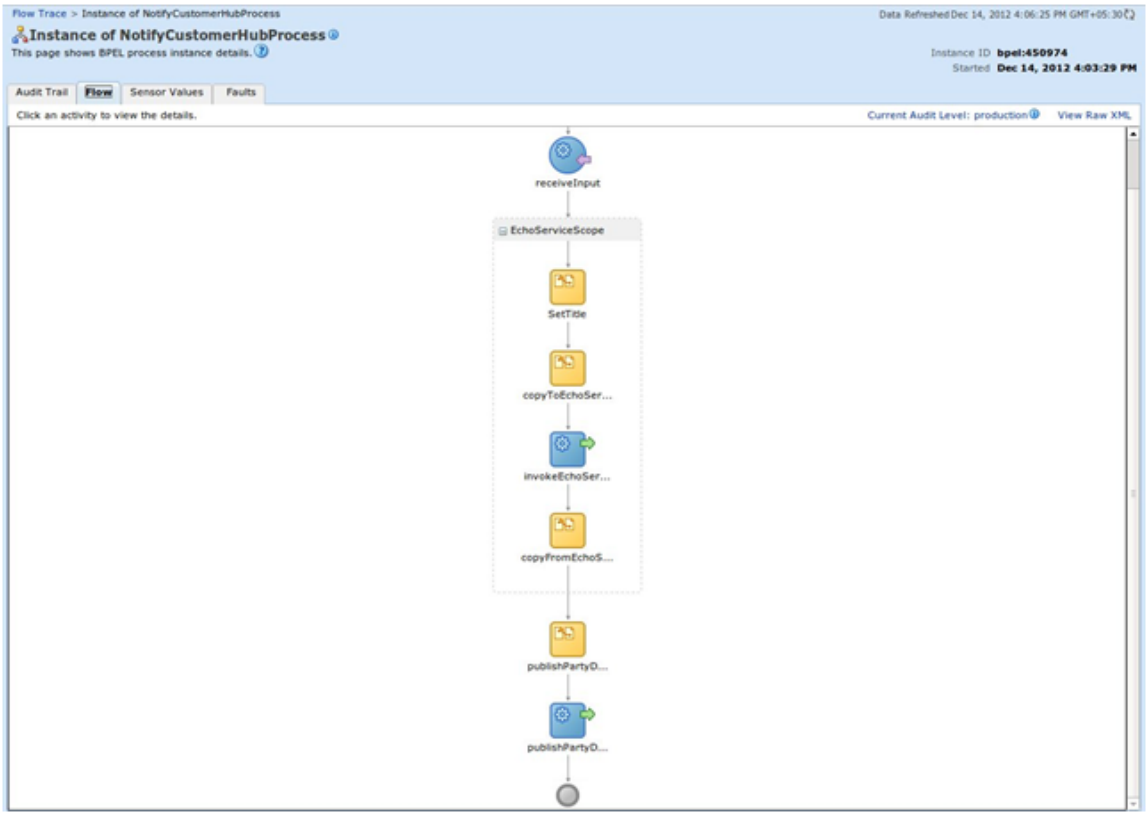
Step 10 Test the Customized Composite

After performing the customizations, build the project and deploy it to a SOA Server. You will need to include the *Customization Class JAR* in the runtime classpath of the deployed application.

To test the customized composite, follow these steps:

1. Log in to the *em* console and select the composite from the *SOA Domain*.
2. Click **Test** and enter appropriate input.
3. On the *Dashboard* panel, click the composite *Instance*. In the *Flow* panel of the screen, you will be able to see the flow of the customized composite.

Figure 6–24 Test Customized Composite - Flow



4. Click the *invokeEchoService* component from the flow to see the request and response XML for the invoke operation to the partner link.

Figure 6–25 Test Customized Composite - invokeEchoService

6.4.2 Add a Human Task to an Existing Process

This example demonstrates how to add a *Human Task* component mode.

In this example of SOA customization, we will be adding a *Human Task* to a BPEL process. Instead of adding the *Human Task* in customization mode, we will build a separate BPEL process with the human task and then customize the base composite to include a *Partner Link* call to that BPEL process.

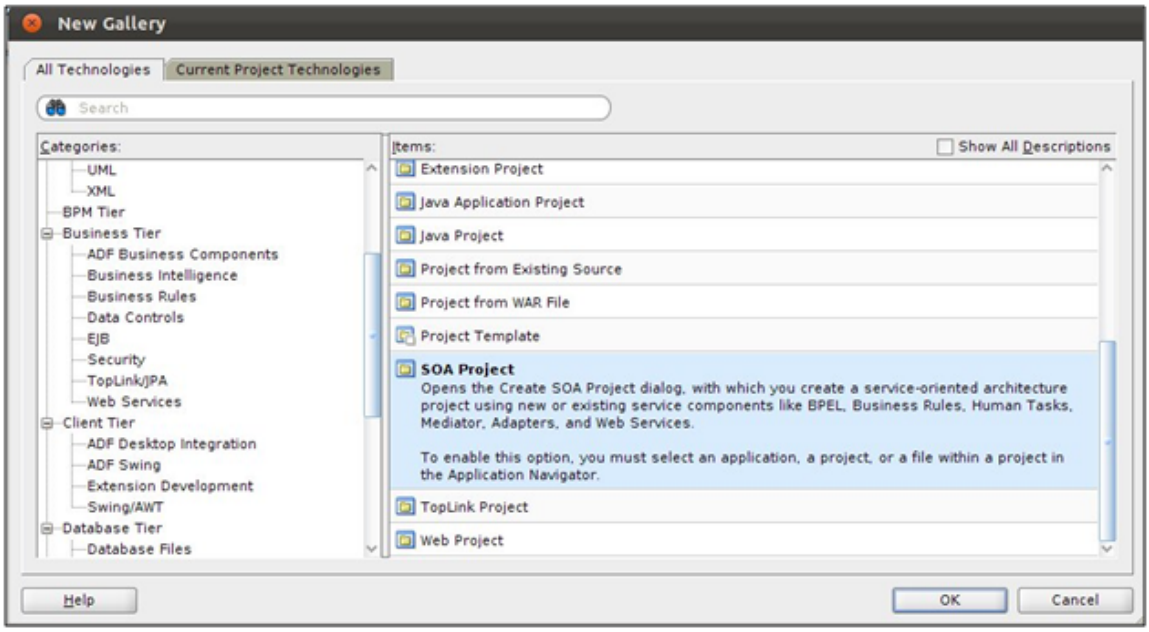
The following section will demonstrate how to create a BPEL process with *Human Task*. The human task will take the title as a *string* input and will have the outcomes *REJECT* and *APPROVE*. The BPEL process will invoke the human task passing the title as input. Based on the outcome of the human task, the title will be suitably modified and returned by the BPEL process.

Step 1 Create SOA Project

You will need to create a SOA project to contain the Echo Service process. To create the SOA project, follow these steps:

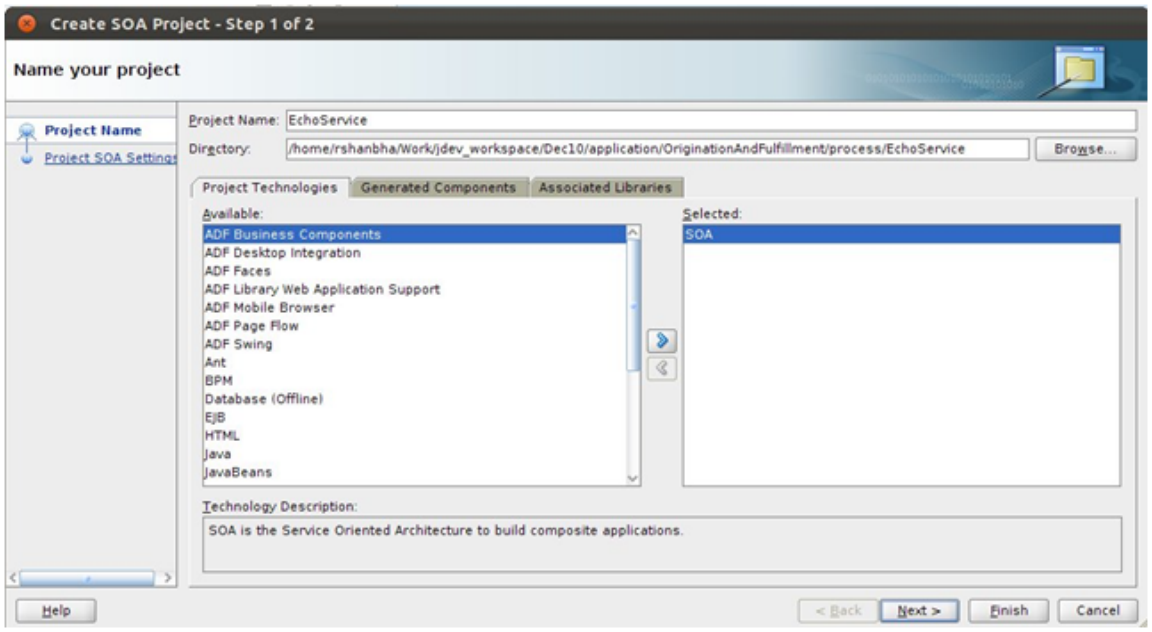
1. In the Main Menu, go to **File** -> **New**.
2. In the Project Gallery that opens, select *SOA Project*.
3. Click **Ok**.

Figure 6–26 Select SOA Project



4. In the **Create SOA Project** wizard, enter appropriate project name (TitleApproval) and location for the project and click **Next**.

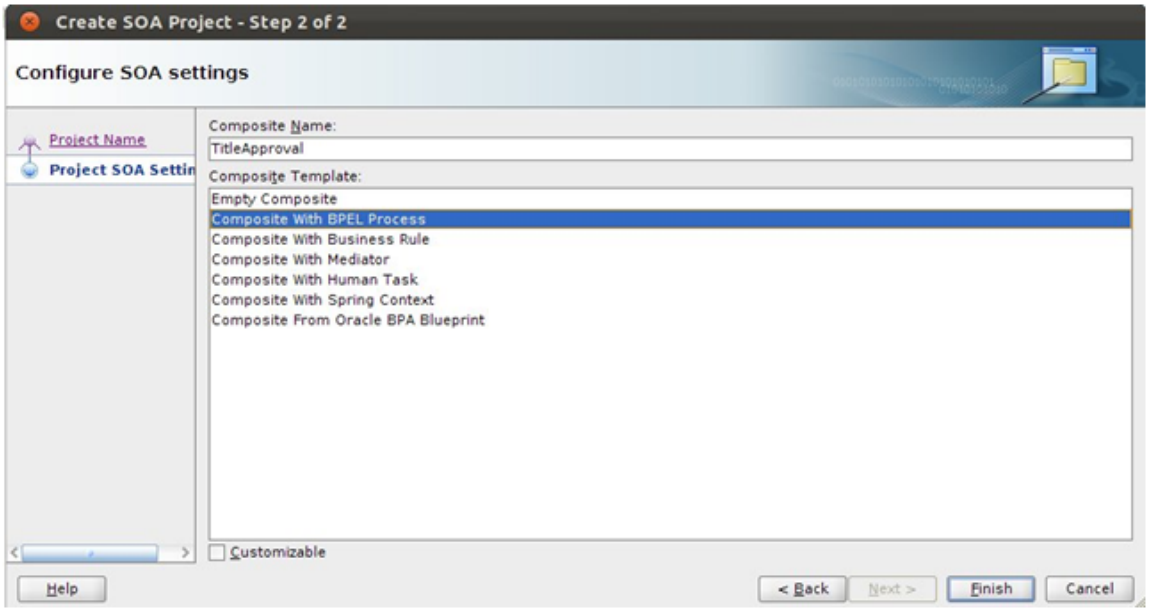
Figure 6–27 Create SOA Project Name



5. In the next dialogue of the wizard, enter appropriate name (TitleApproval) for the SOA composite.
6. Select *Composite With BPEL Process* from the drop-down menu.

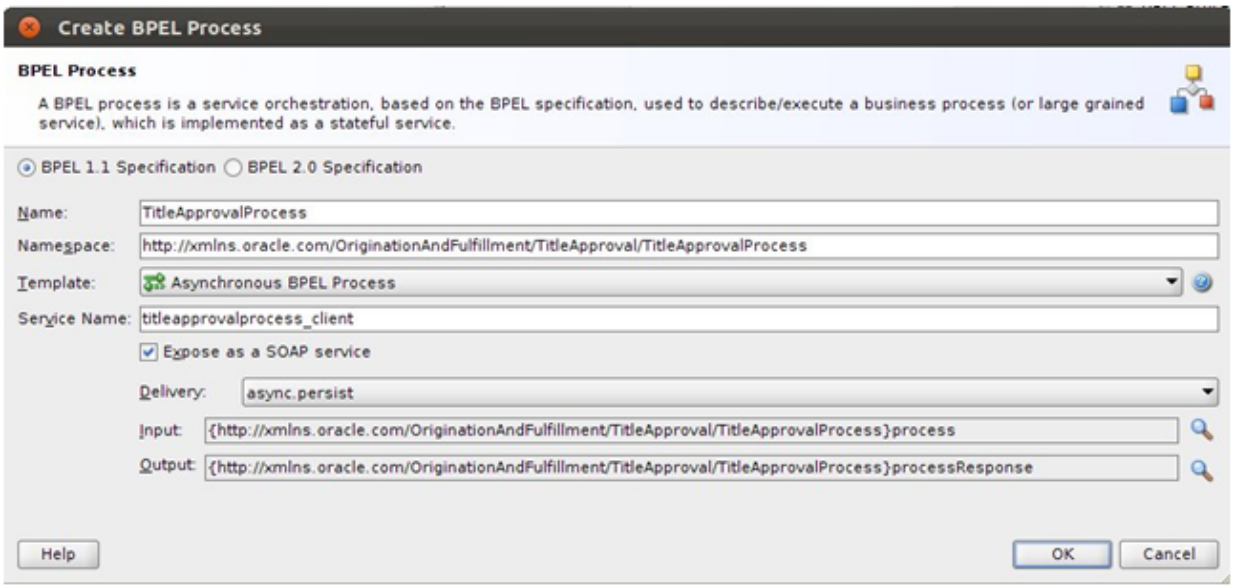
7. Click **Finish**.

Figure 6–28 Configure SOA Settings



8. The dialog *Create BPEL Process* will open. Enter a name (TitleApprovalProcess) for the process and select *Asynchronous BPEL Process* from the templates drop-down.

Figure 6–29 Configure BPEL Process Settings

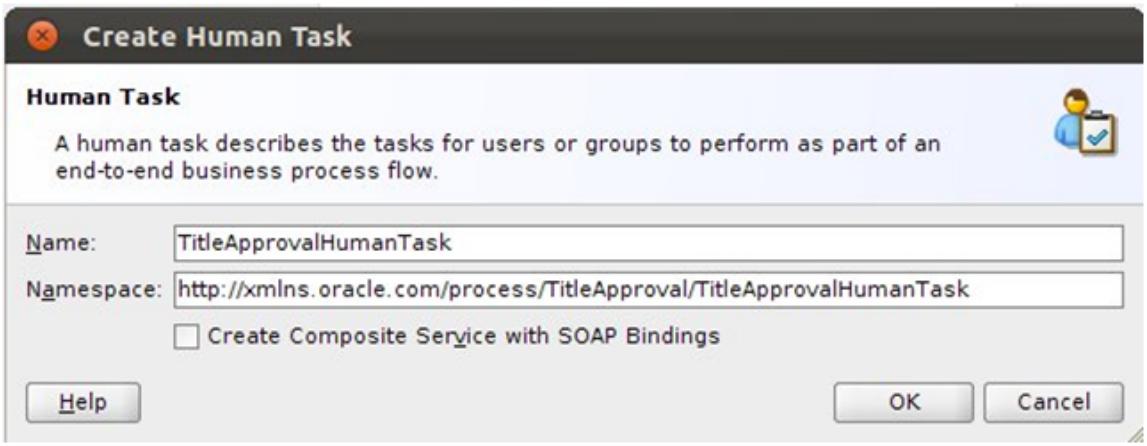


Step 2 Create Human Task

After defining the BPEL process, you will need to add the *Human Task* component to the process. To add the *Human Task*, follow these steps:

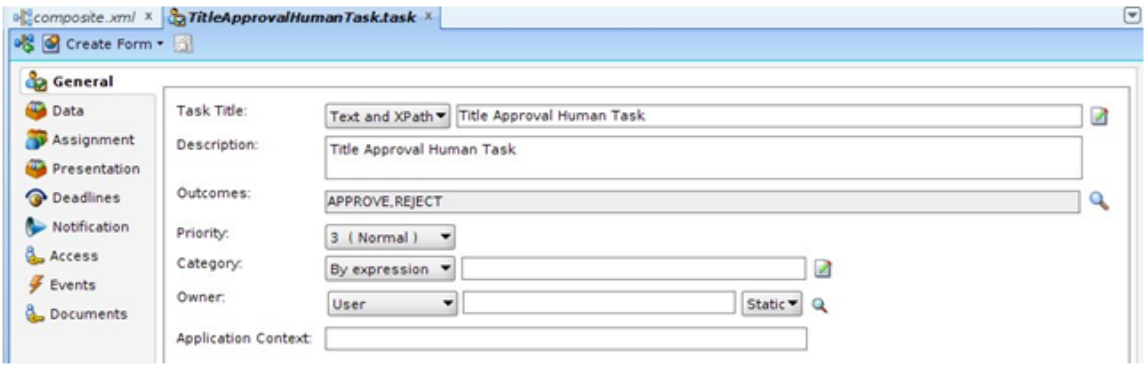
1. From the **Project Navigator** tab, select and open *composite.xml* in the *Design* mode.
2. From the **Component Palette** tab, in *SOA Components* section, select the *Human Task* component and drag and drop it onto the *components* section of the *composite.xml*.
3. In the **Create Human Task** dialog that opens, enter a name (TitleApprovalHumanTask) for the human task.
4. Click **OK**.

Figure 6–30 Enter Human Task Name



5. From *Project Navigator*, select and open *TitleApprovalHumanTask.task* file in **Design** mode. This file has the human task definition.
6. In the *General* section, specify a **Task Title** and **Description** for the human task.

Figure 6–31 Create Human Task - General Tab



7. In the *Data* section, click the icon for add task parameter.
8. In the **Add Task Parameter** dialog, specify the parameter type and name for the input to the human task. In our case, the input task parameter would be a string title.

Figure 6–32 Add Human Task Parameter

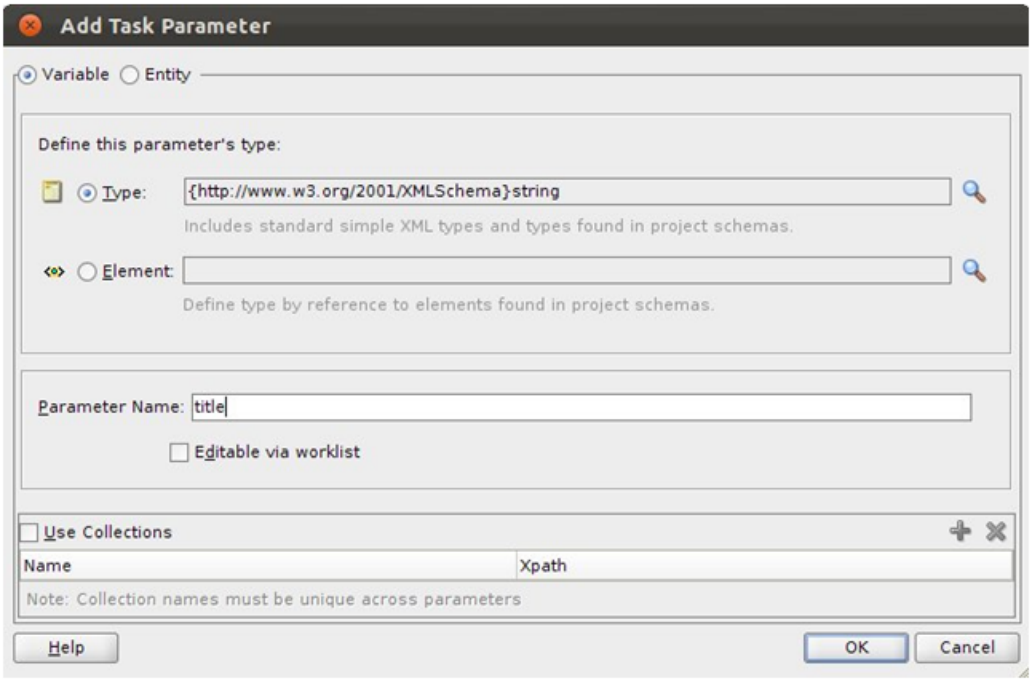
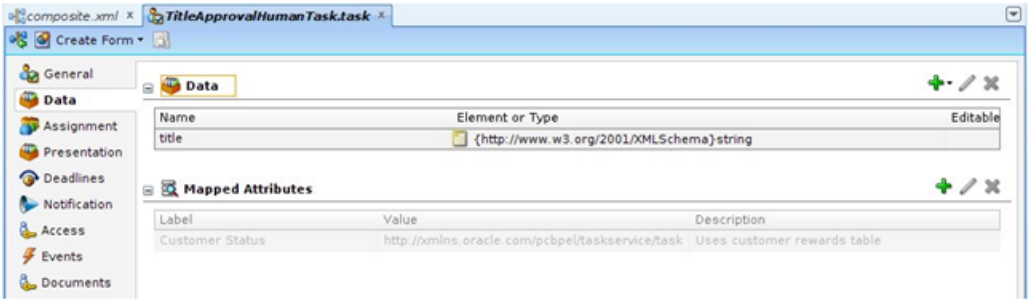


Figure 6–33 Create Human Task - Data Tab



The *Assignment* section is used to define the *Users* or *User Groups* to which the human task should be assigned.

9. Double-click **Edit Participant**.
10. In the **Add Participant Type** dialog, check the Value-based option for **Specify Attributes Using**.
11. Click the icon for adding a value.
12. Select the *User By Name* option and enter the name of your user (weblogic).

Figure 6–34 Add Participant Type Details

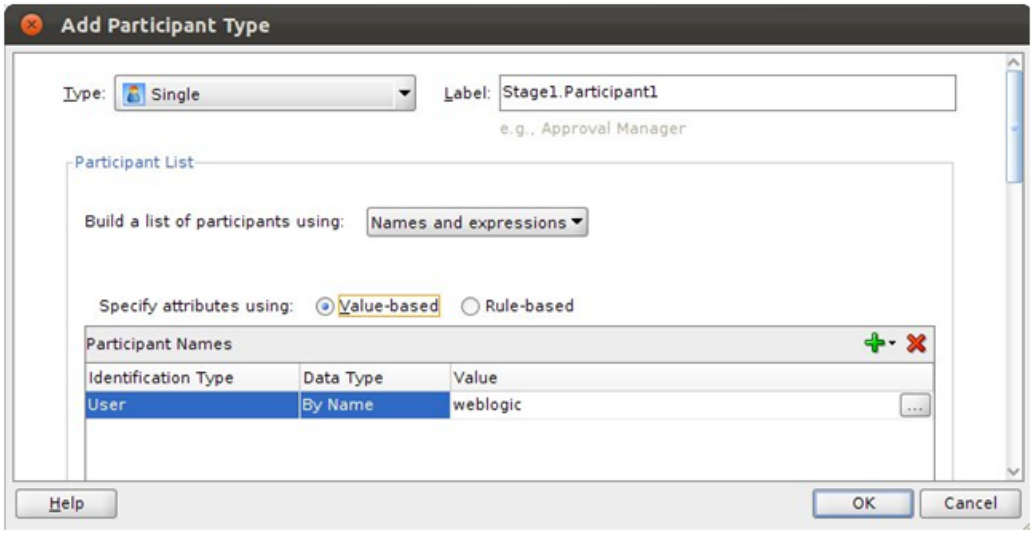
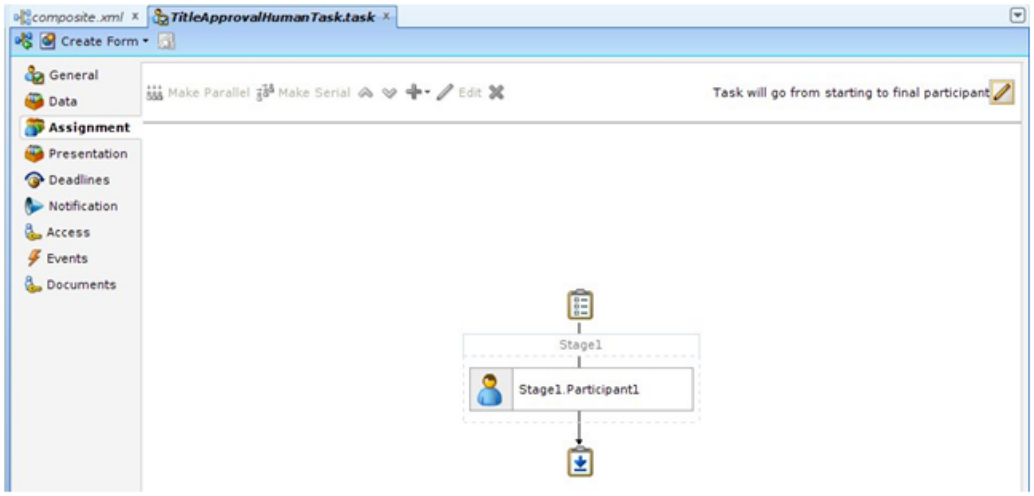
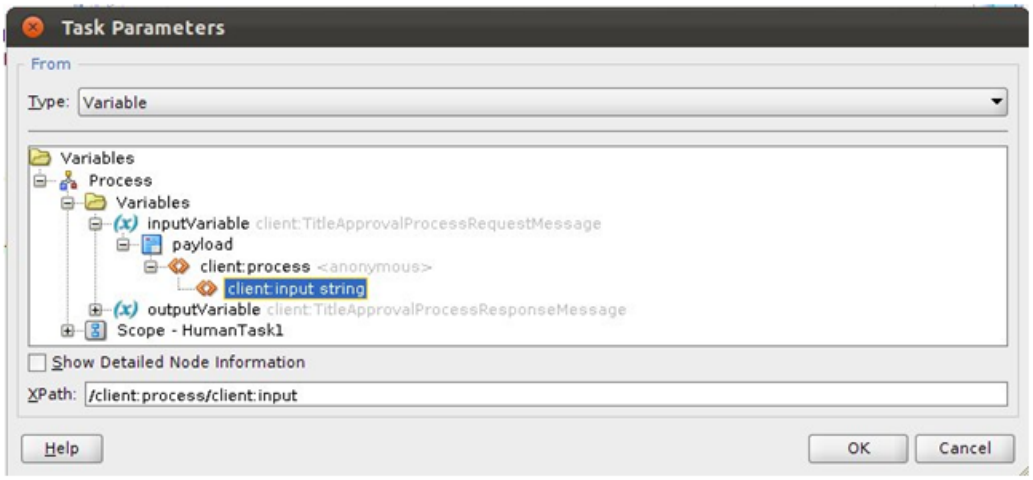


Figure 6–35 Create Human Task - Assignment Tab



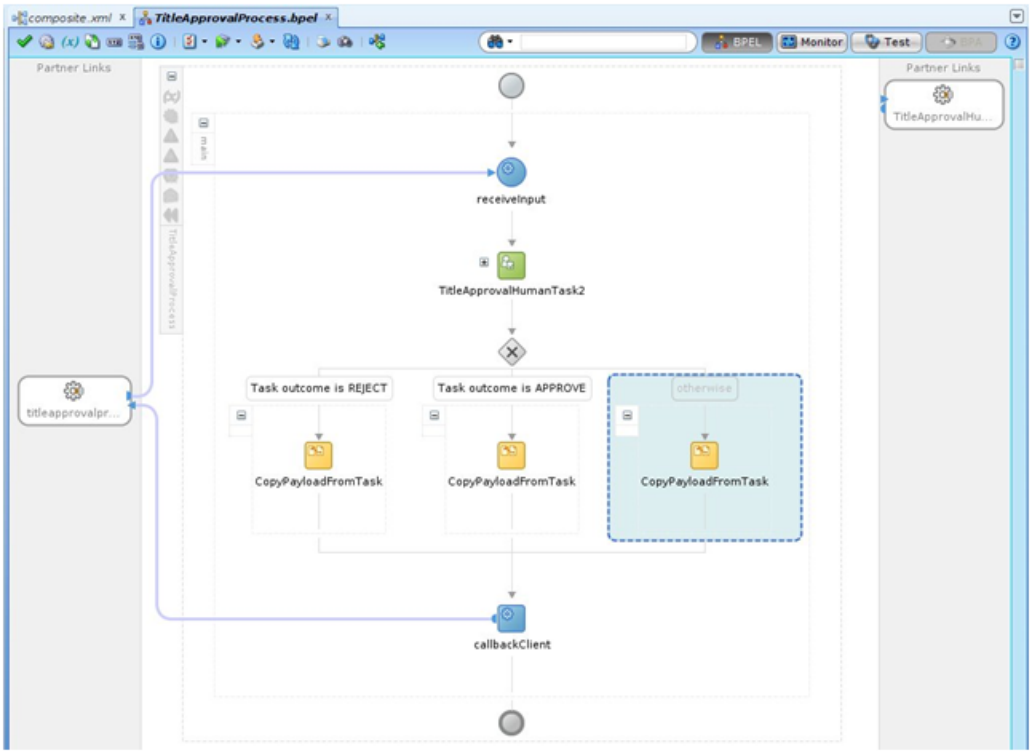
13. From the Project Navigator, open the *TitleApprovalProcess.bpel* file in **Design** mode.
14. From the **Component Palette**, select the component *Human Task* and drag-drop it on to the BPEL process.
15. Click the icon to add task parameter.
16. In the **Task Parameters** dialog, select the *string* input to the BPEL process.

Figure 6–36 Select Human Task Parameters



17. In the task outcomes *Switch* in the BPEL process, delete the condition for *otherwise*.

Figure 6–37 Create Human Task - Delete Condition



18. From the **Component Palette**, select the *Assign* component and drag-drop it to the *REJECT* outcome of the switch.
19. Enter a *name* (rejectTitle) for the component.

20. In the *Copy Rules* section of the assign, use the *Expression Builder* to set the output *string variable* of the BPEL process to '<title> - Rejected'.

Figure 6–38 Create Human Task - Expression Builder

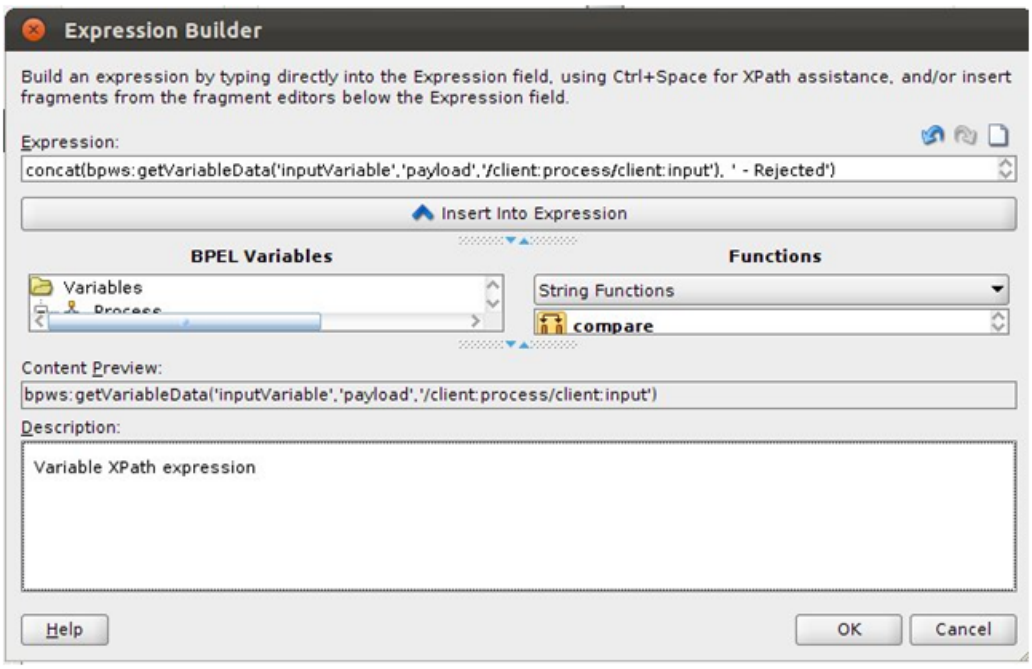
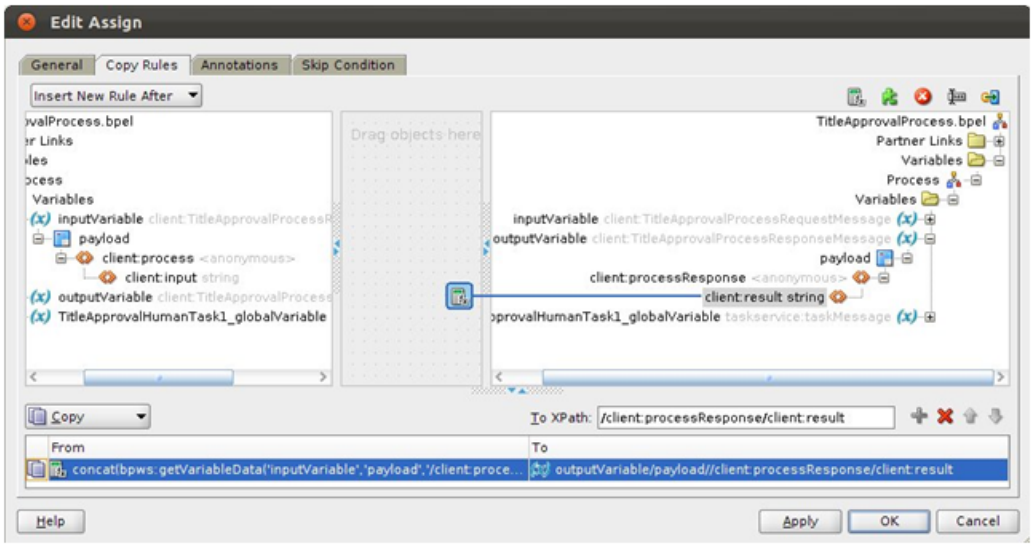
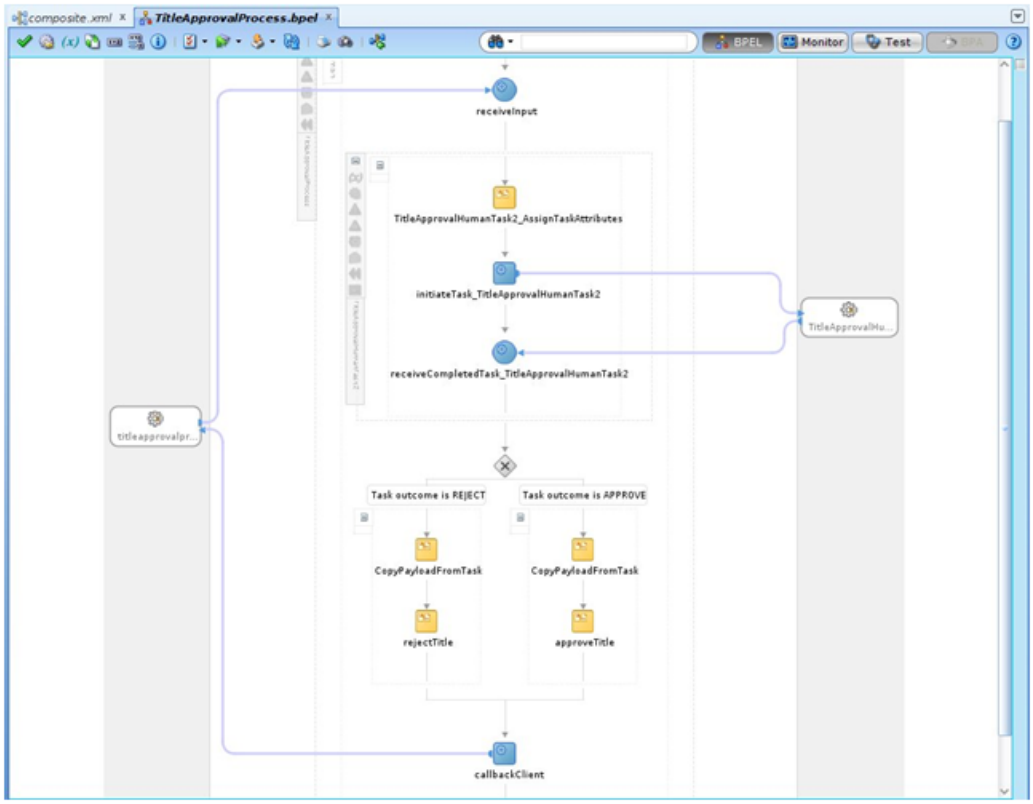


Figure 6–39 Create Human Task - Copy Rules



21. Save all changes to the human task. The BPEL process should look as shown in the figure below.

Figure 6–40 Create Human Task - BPEL Process



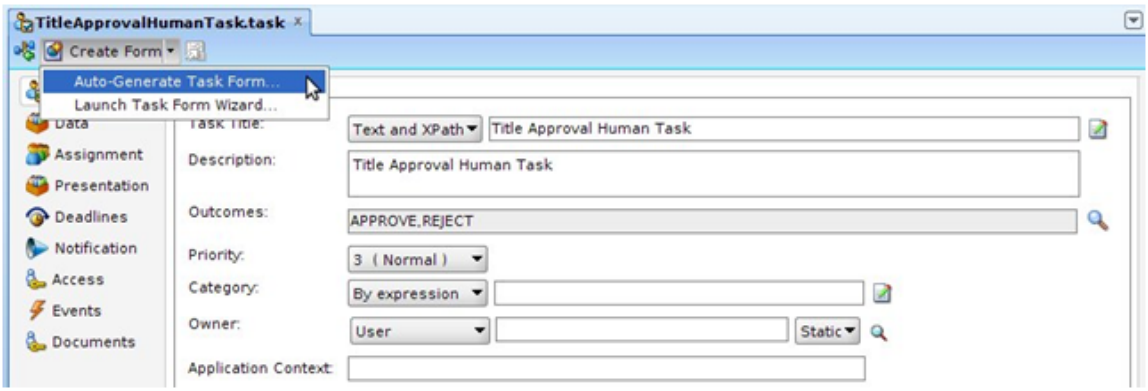
22. Deploy the SOA process to the server as mentioned in the previous example.

Step 3 Create Human Task Form

The *Human Task* is visible to assigned users in the *BPM Worklist* application. To display the task parameters and payload for the task, you will need to create a *task-flow* with the *Human Task* Form. This task form can be auto generated through the process. Follow these steps:

1. From the Project Navigator, open the *TitleApprovalHumanTask.task* file in **Design** mode.
2. Click the button for *Create Task Form*.
3. Select the *Auto-generate Task Form* option from the context menu.

Figure 6–41 Select Human Task Form



4. Enter a name (TitleApprovalHumanTask) and *location* for the human task form project.
5. Click **Finish**.

The generated human task form project will have default file names. Re-Factor file names for form and page definition using appropriate naming conventions.

Step 4 Deploy Human Task Form Project

You will need to deploy the human task form project on the UI server for the previously deployed SOA process. To deploy, follow these steps:

1. Clean and build the project.
2. Right-click the project and select *Deploy* from the context menu.
3. In the **Deploy** dialog that opens, select the *Deploy to Application* Server option from the list and click *Next*.
4. Select the appropriate UI server for the SOA server.
5. Click **Finish**.

Figure 6–42 Select Human Task Form Deployment Action

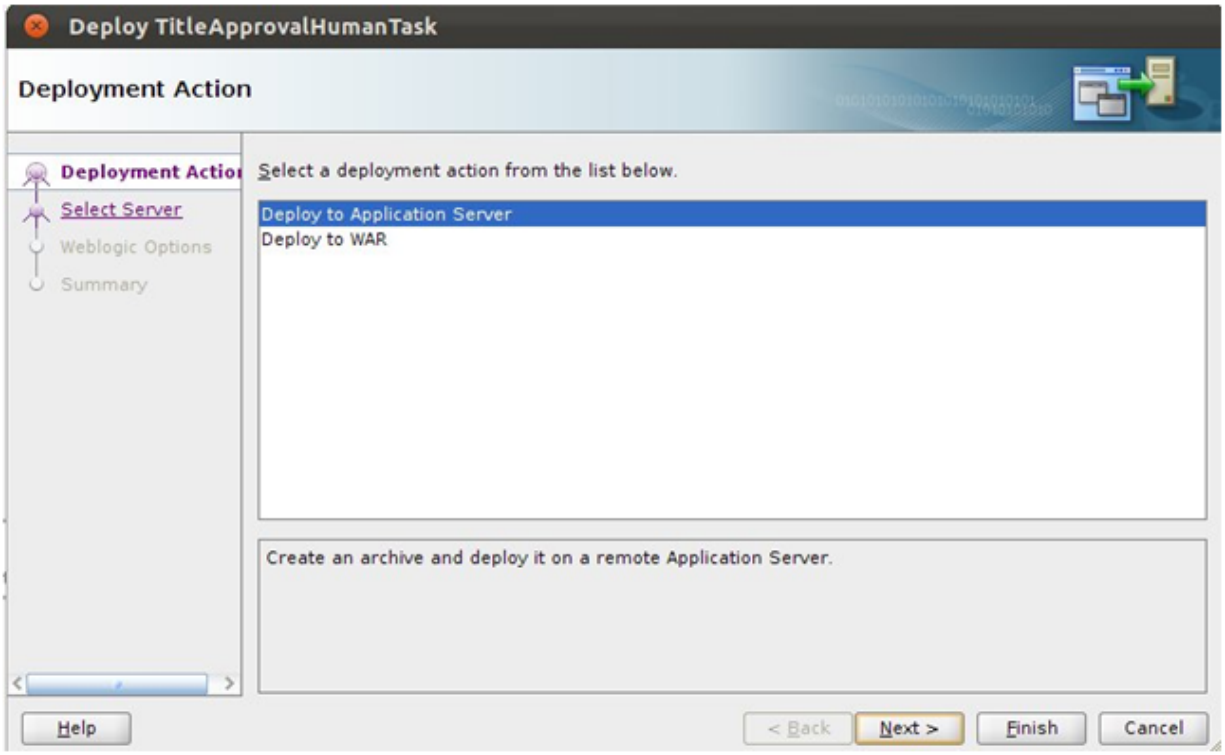
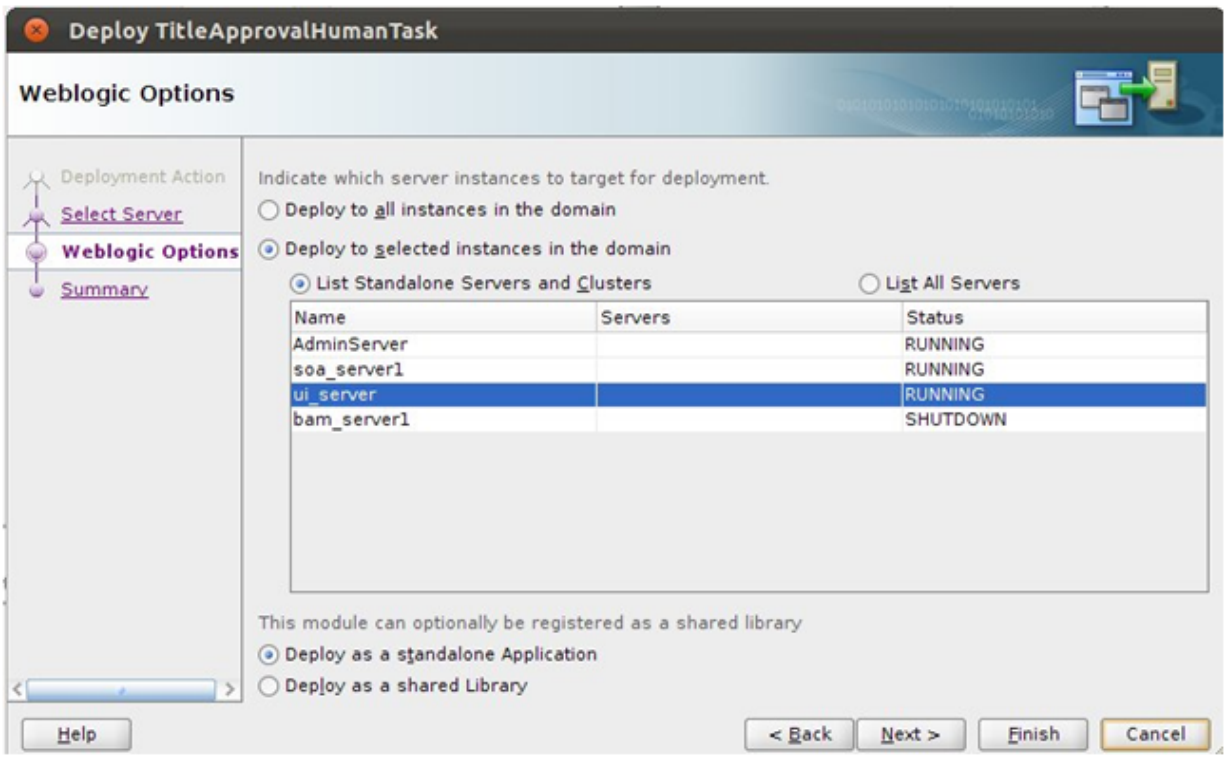


Figure 6–43 Select Human Task Form - Weblogic Options



Step 5 Add Customizable Scope to SOA Application

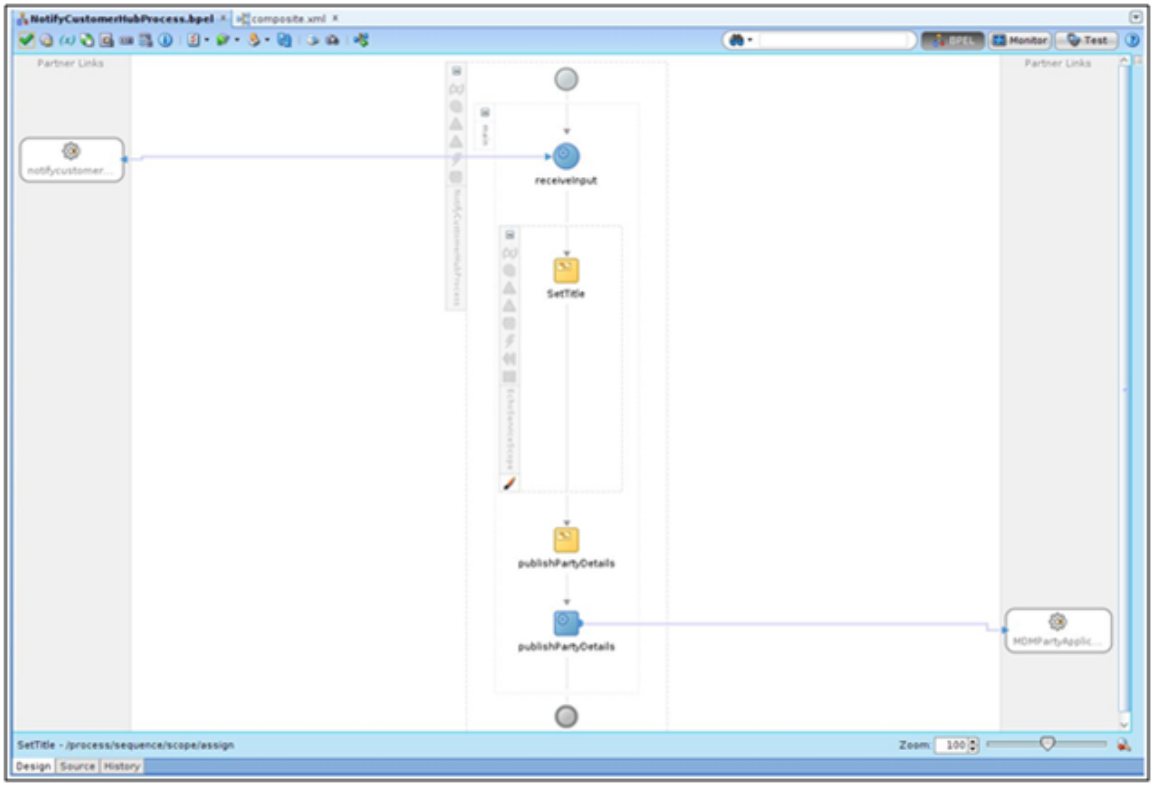
To demonstrate customizations of a SOA process, we will be using the BPEL process *NotifyCustomerHubProcess* present in the composite *com.ofss.fc.workflow.process.NotifyCustomerHub*.

Open the SOA application which contains the base composite which will be customizing. The aforementioned process is present in the *OriginationAndFulfillment* application inside the *com.ofss.fc.workflow.NotifyCustomerHub project*.

To add a customizable scope to the BPEL process, follow these steps:

1. Open the *NotifyCustomerHubProcess.bpel* file in **Design** mode.
2. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the *Scope* component and drop it on to the BPEL process as shown in the figure.
3. Double click the component and enter appropriate name for the component.
4. Drag and drop the existing *Assign* component labeled *setTitle* on to the newly added *Scope* component.

Figure 6–44 Add Customization Scope to SOA Application



5. Right-click the *Scope* component and select *Customizable* from the context menu.
6. Save all the changes and restart JDeveloper in *Customization Developer Role*.

Step 6 Customize the SOA Composite

After adding a *Customizable Scope* to the base composite, you can start performing customizations in JDeveloper's *Customization Developer Role*.

When you open the *NotifyCustomerHubProcess.bpelf* ile in Design mode, you will notice that all other components in the process, except the customizable Scope component, are disabled. This means that your customizations are limited to that scope.

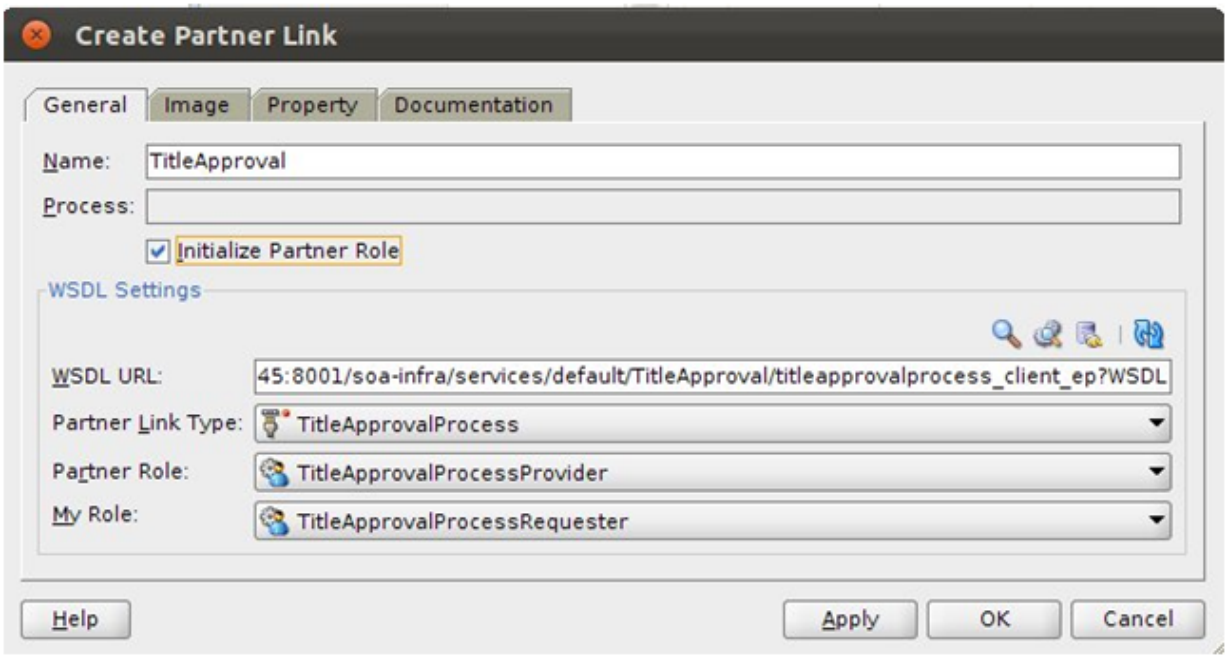
In the following sections, we will be adding a *Partner Link* call to the previously created *TitleApproval BPEL* process and other required components in the customization mode.

Step 7 Add Partner Link Component

To add a *Partner Link* to the BPEL process, follow these steps:

1. From the Project Navigator, open the NotifyCustomerHubProcess.bpel file in Design mode.
2. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the Partner Link component and drop it on to the *Partner Links* section of the BPEL process.
3. In the *Create Partner Link* dialogue that opens, enter appropriate name (TitleApproval) for the partner link.
4. Select the following options:
    a. **TitleApprovalProcess** as the Partner Link Type.
    b. **TitleApprovalProcessProvider** as the Partner Role.
    c. **TitleApprovalProcessRequester** as the My Role.

Figure 6–45 Add Partner Link Component

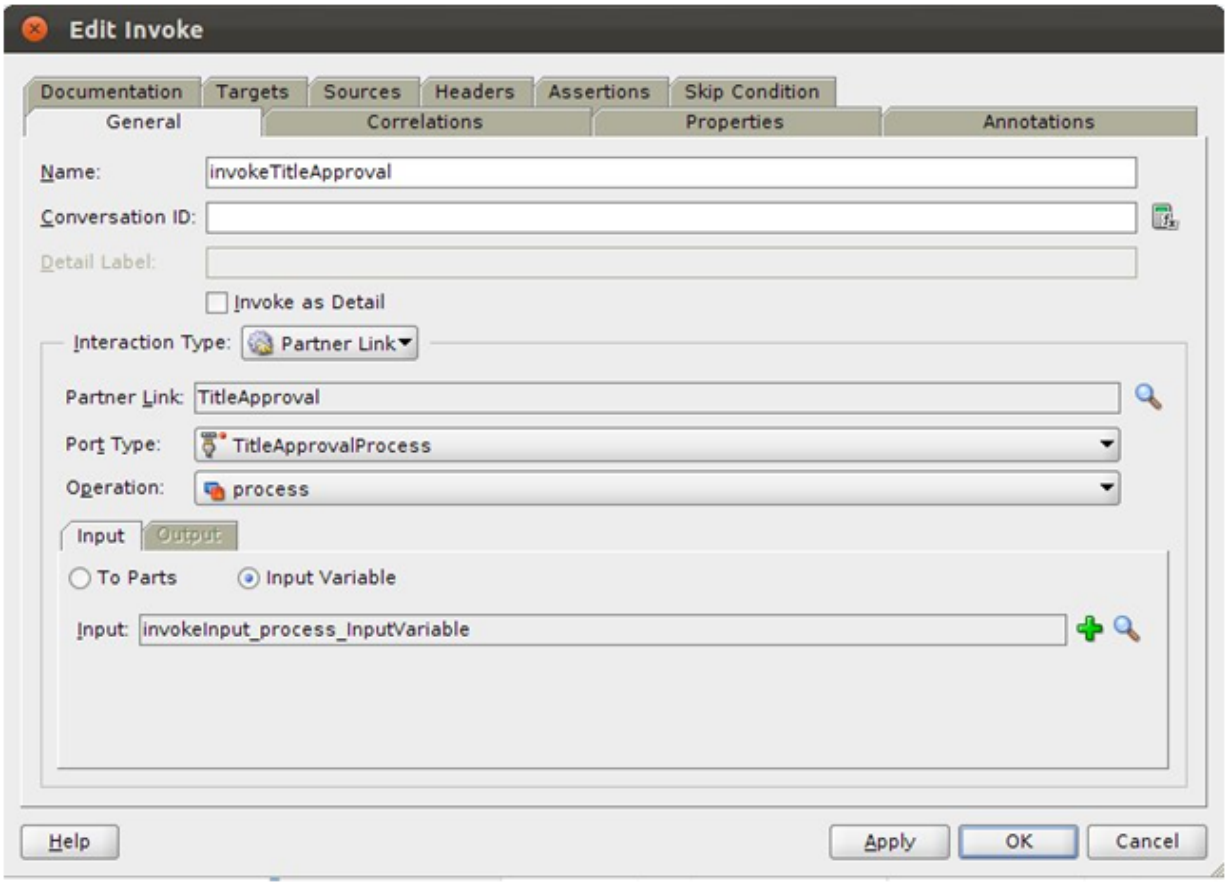


Step 8 Add Invoke Component

You will need to add an *Invoke* component to invoke the previously added partner link call to *TitleApproval*. To add *Invoke component*, follow these steps:

1. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the *Invoke* component and drop it on the BPEL process inside the *Scope* component.
2. Click the *Invoke* component and drag it to the previously added *TitleApproval* partner link.
3. Double-click the *Invoke* component.
4. In the *Edit Invoke* dialogue that opens, enter an appropriate name (invokeTitleApproval) for the component.
5. Click the icon for adding a new variable in the *Input Variable* section.
6. Click **OK** to save the changes.

Figure 6–46 Add Invoke Component



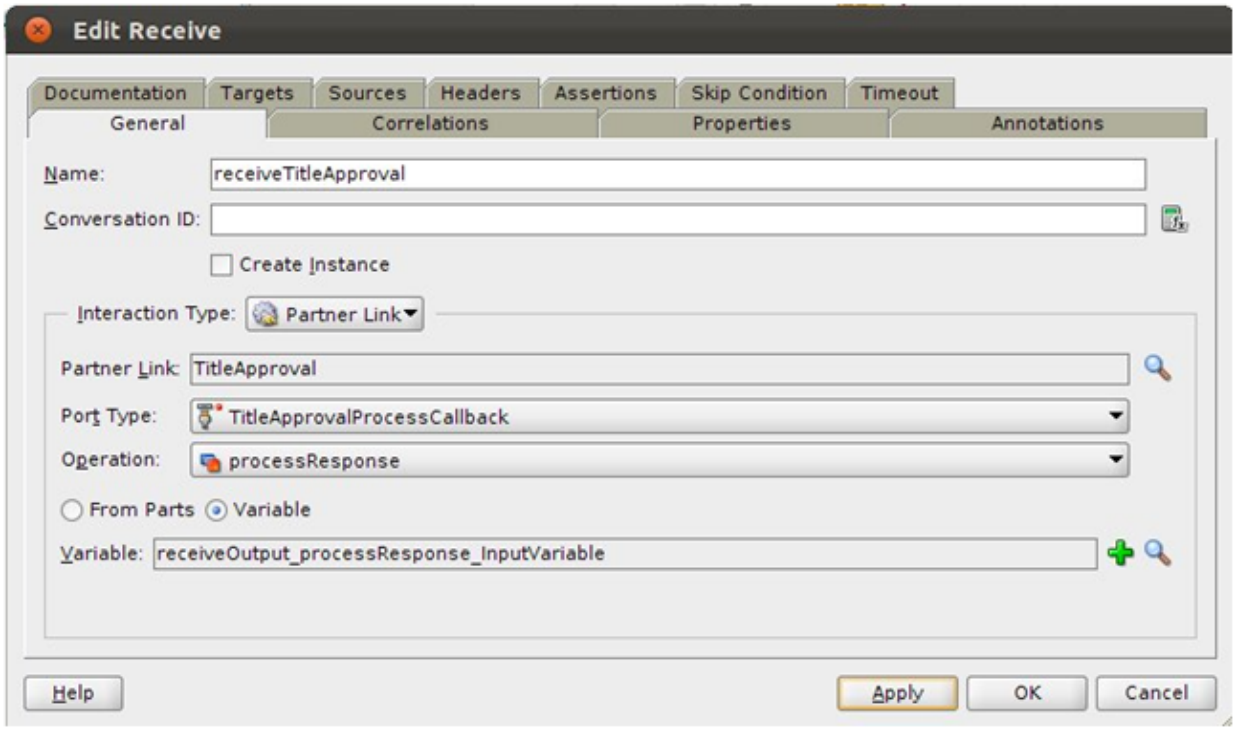
Step 9 Add Receive Component

You will need to add a *Receive* component to receive output from the previously added partner link call to *TitleApproval*. To add *Receive* component, follow these steps:

1. From the *Component Palette* panel on the right side, in the *BPEL Constructs* section, drag the *Invoke* component and drop it on the BPEL process inside the *Scope* component.
2. Click the *Receive* component and drag it to the previously added *TitleApproval* partner link.
3. Double-click the *Receive* component.

4. In the *Edit Receive* dialogue that opens, enter an appropriate name (receiveTitleApproval) for the component.
5. Click the icon for adding a new variable in the *Output Variable* section.
6. Click **OK** to save the changes.

Figure 6–47 Add Receive Component using BPEL functions



Step 10 Add Assign Components

An *Assign* component is used to assign values to a variable. These values can be directly assigned from one variable to another or modified using BPEL functions available.

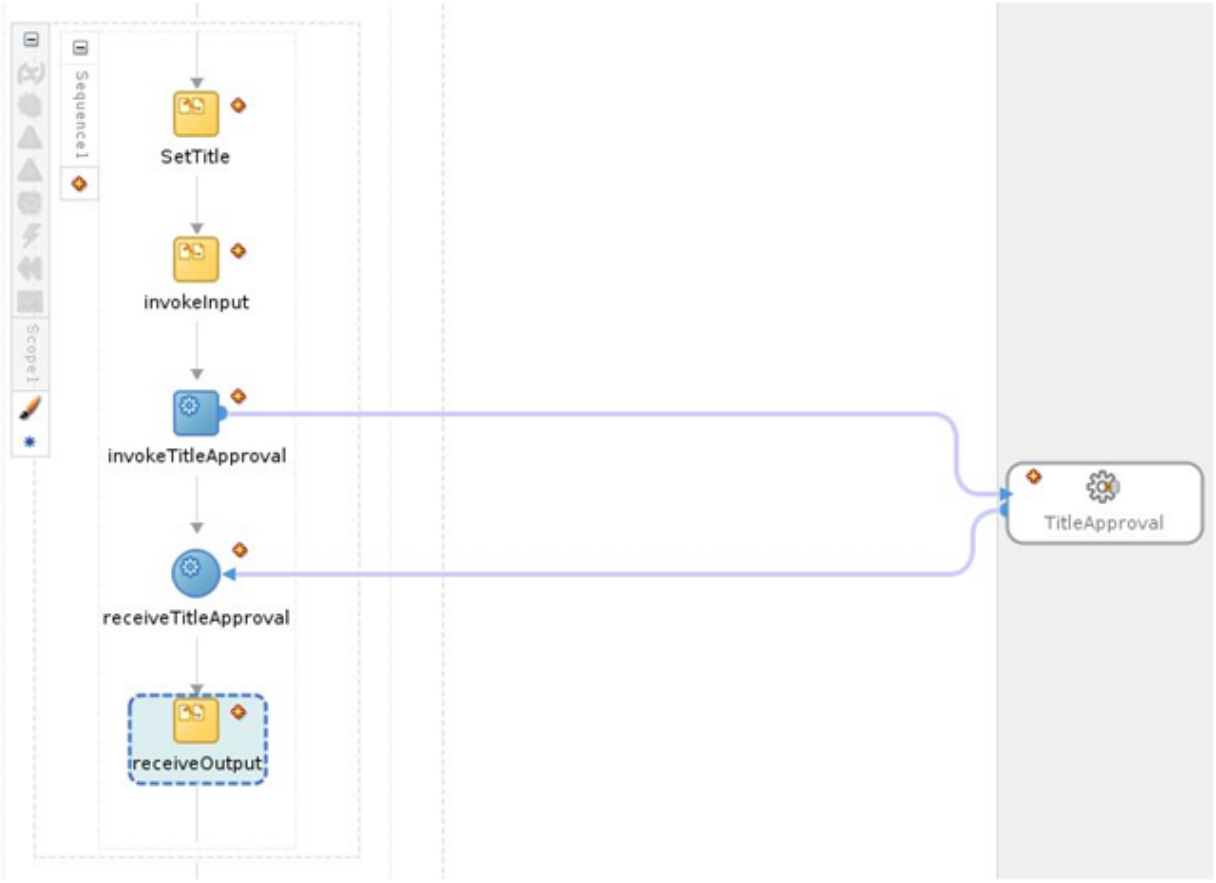
The *TitleApproval* accepts a single *string* as an input and gives a single *string* as an output. The *Input Variable* and *Output Variable* defined in the previously created *invokeTitleApproval* and *receiveTitleApproval* components will be used to hold the input value for the *TitleApproval* and the output returned respectively.

In our case, we will need to add two *Assign* components for following purposes:

- To populate the *Input Variable* of the *invokeTitleApproval* component with the value returned by the existing *setTitle* component.
- To populate the *setTitle* component with the value returned in the *Output Variable* of the *receiveTitleApproval* component.
- Add the two required *Assign* components and save all changes.

The customized process should look as shown in the figure below.

Figure 6–48 Add Assign Component



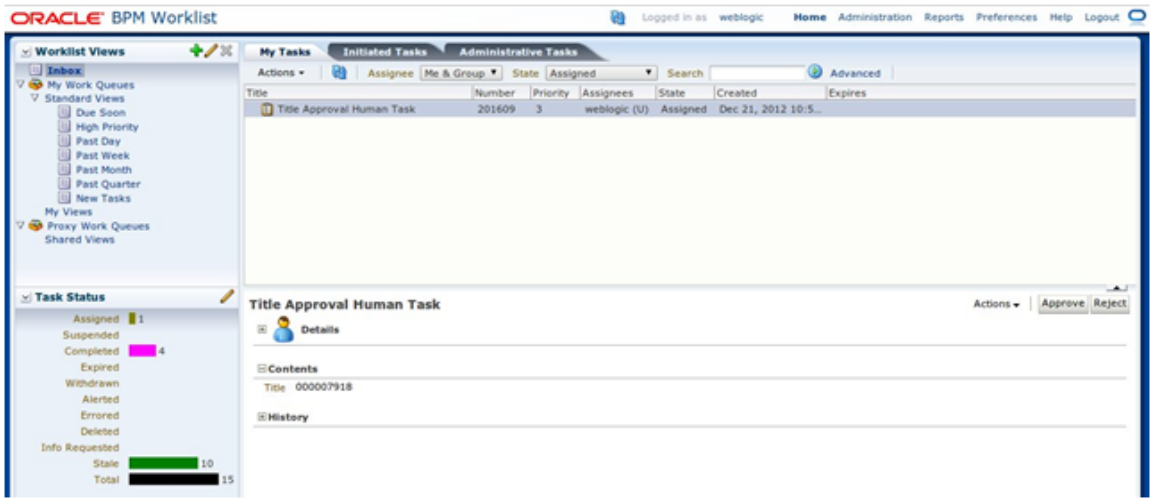
Step 11 Deploy and Test Customized SOA Composite

After performing the customizations, build the project and deploy it to a SOA Server. You will need to include the *Customization Class JAR* in the runtime classpath of the deployed application.

To test the customized composite, follow these steps:

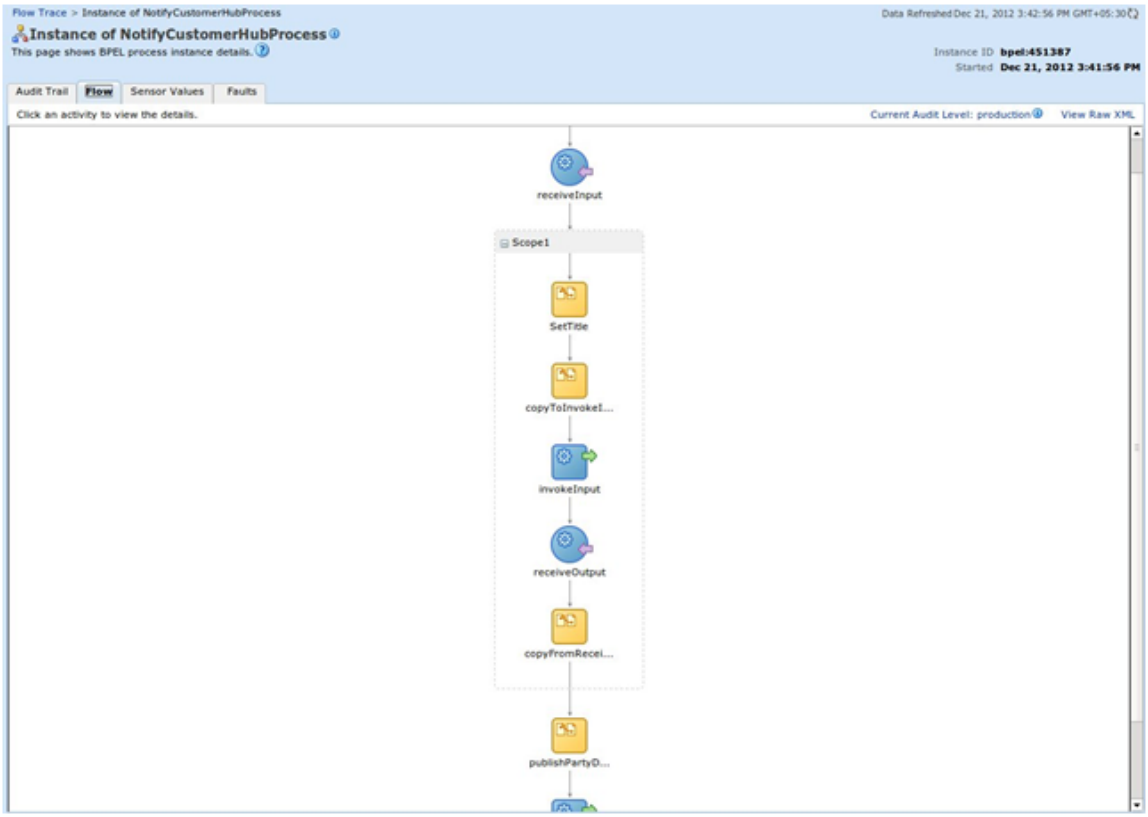
1. Log in to the *em* console and select the composite from the *SOA Domain*.
2. Click **Test** and enter appropriate input. A *Human Task* will be created and assigned to the user as specified in the human task definition.
3. Log in to the *BPM Worklist* application with the appropriate user. You will be able to see the previously created task on the dashboard.
4. Select the task from the list and click *Approve* or *Reject* button to perform approve or reject action on the task.

Figure 6–49 Deploy and Test Customized SOA Composite - My Tasks Tab



5. On the *Dashboard* panel of the *em*, click the composite *Instance*.
6. In the *Flow* panel of the screen, you will be able to see the flow of the customized composite.

Figure 6–50 Deploy and Test Customized SOA Composite - Flow



7. Click the *invokeTitleApproval* component to see the request xml for the partner link call to *TitleApproval* process.

Figure 6–51 Deploy and Test Customized SOA Composite - Invoke Input

8. Click the *receiveTitleApproval* component to see the response xml for the partner link call to *TitleApproval* process.

Figure 6–52 Deploy and Test Customized SOA Composite - Receive Output