

IBM WebSphere Business Services Fabric 6.2 – Lab exercises

# Assemble and deploy a legacy composite business application

What this exercise is about .....	2
Lab requirements .....	2
What you should be able to do .....	3
Exercise instructions .....	4
Part 1 Build the composite business services model.....	5
Part 2 Define and simulate business service policies .....	46
Part 3 Deploy and run the composite business service.....	87
What you should be able to do .....	122

## What this exercise is about

WebSphere Business Services Fabric greatly extends IBM's SOA based BPM platform to support a new class of applications. WebSphere Business Services Fabric includes, uses, and extends IBM's J2EE WebSphere Process Server for integration and automation of enterprise business processes. This new class of applications is called Composite Business Applications.

The components that make up WebSphere Business Services Fabric include :

- **Dynamic Assembler** - Highly scalable, service personalization and semantic services engine that runs on the WebSphere Process Server platform
- **Business Services Repository** - Maintains business service metadata optionally using WebSphere Services Registry and Repository
- **Business Services Subscriber Manager** - Controls and automates entitlements
- **Business Services Governance Manager** – Manage the Business Services Repository meta-model, provide governance for collaborative work, notification, user access configuration, and management, etc

IBM® WebSphere® Business Services Fabric's development tool is the Composition Studio environment. Composition Studio is used for visual modeling and management of business services metadata models and policies. It is an Eclipse perspective plug-in that extends WebSphere Integration Developer.

In this lab, you will build and verify the integrity of a Credit Report Business Service model using Composition Studio. This model is used by WebSphere Business Services Fabric runtime to dynamically select a credit service depending on runtime characteristics of the service call using context, contracts, and content.

Next, you will decorate endpoints with metadata and then create policies that use this information to dynamically provision appropriate services at runtime based on the service consumers needs.

In the last section you will invoke the JK Enterprises customer account open BPEL process developed and deployed using WebSphere Integration Developer. This process is running on WebSphere Process Server. The necessary services have been deployed to its underlying WebSphere Application Server container.

Your goal is to see the correct credit report service provider being selected by WebSphere Business Services Fabric's Dynamic Assembler. This behavior is enabled through the Dynamic Assembler's use of a consumer's content, context and contract information combined with declarative business policies, endpoint capabilities and other meta-data contained in the Business Services Repository.

This lab does not cover the new business spaces introduced in version 6.2. This lab is provided AS-IS, with no formal IBM support.

## Lab requirements

- You need to complete Migration V6.1.2 lab before starting this lab
- Software needed :
  - WebSphere Business Modeler Advanced version 6.2
  - WebSphere Integration Developer version 6.2 with WebSphere Process Server Unit Test Environment
  - WebSphere Tool Pack with WebSphere Business Fabric Server Unit Test Environment version 6.2

- WebSphere Monitor Toolkit version 6.2
- WebSphere Registry and Repository version 6.1.2 or later
- All PIF a and related files can be found in **WDPE-v6.2-labfile.zip**
- These labs use username: **admin** and password: **admin** to login as administrator for any of the Web consoles including WebSphere Business Services Fabric and WebSphere Process Server. Your environment might use different credential details.
- This lab requires WebSphere Process Server application port is **9081**.

## What you should be able to do

After performing this lab, participants will have:

- Reviewed the WebSphere Business Services Fabric Business Service model
- Created a business service model
- Captured the service realizations
- Verified the integrity of this business service model
- Reviewed the capabilities of the credit reporting services available
- Defined the assertions for each credit reporting service
- Run a simple simulation without policies
- Creating the Business Space variable to be used in these exercises
- Created the policies for the JK Enterprises business case
- Simulated JK Enterprises policies
- Deployed development changes into the runtime business services repository
- Ran three end-to-end test cases and evaluated the results - not simulation
- Understood how WebSphere Business Services Fabric extends and enhances WebSphere Process Server capabilities



### Prior setup

A number of tasks have been completed for you in advance in order to focus on core learning objectives.

---

## Exercise instructions

## Part 1 Build the composite business services model

### 1.1 Create a composite business service resource hierarchy

Using Composition Studio, you will create a composite business service resource hierarchy Instance.

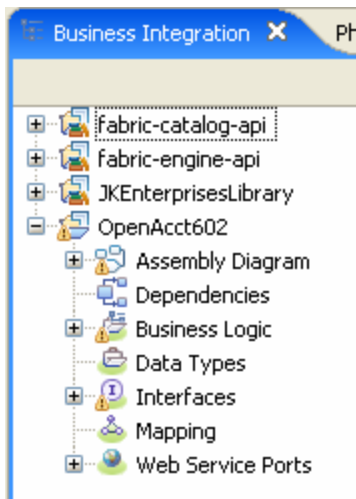
Composition Studio is used to:

- Author and describe objects and relationships developed for use by WebSphere Business Services Fabric using rich, wizard-based interfaces
- Validate the consistency and completeness of composite business services using the integrity checker
- Simulate transactions and perform “what if” analysis based on policies
- Submit changes to the governance manager for approval and publishing

\_\_1. View the OpenAccount project in WebSphere Integration Developer.

In the migration lab, you loaded a simple JK Enterprises OpenAcct602 project and have associated libraries for you. This perspective contains the typical assembly life cycle elements developed and managed using WebSphere Integration Developer. It contains the BPEL, data types, assembly diagram, endpoints, and other components.

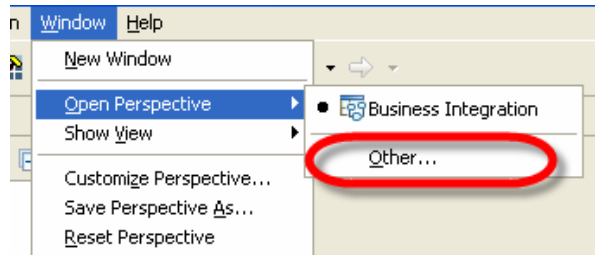
\_\_a. WebSphere Integration Developer should have opened in the default Business Integration perspective. If not, switch to Business Integration perspective from the main menu by selecting **Window > Open Perspective > Business Integration**.



\_\_2. View the WebSphere Business Services Fabric Project. A local WebSphere Business Services Fabric Project as been created for you in advance. Most of the endpoints and all of the interfaces have already been imported from WebSphere Integration Developer for you.

A WebSphere Business Services Fabric project is a container in which to develop and store an instance of the Composite Business Service model you are building. It is called “JK Customer Services Process”.

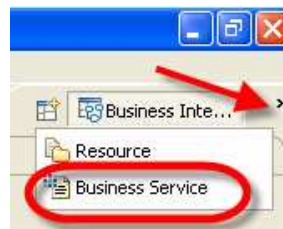
- \_\_a. Open the Business Service perspective by selecting **Window > Open Perspective > Other**



- \_\_b. Select “**Business Service**”.



- \_\_c. Click **OK**.
- \_\_d. A useful shortcut for switching perspectives in eclipse is shown in the image. Click >> in the upper right corner and select **Business Service**.



**Done during setup**



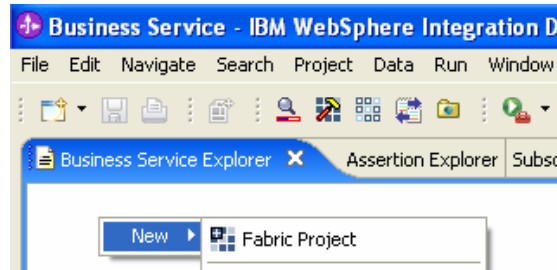
This local development project has been replicated from the “remote master” Business Service Repository.

This replication has provided you, potentially one of several developers, with the extensions defined for the JK Enterprises business needs.

- \_\_3. Import a project into Composition Studio

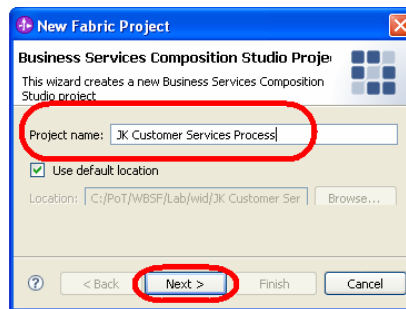
- \_\_a. Right click the blank space to the left – under Business Service Explorer

\_\_b. Click **Fabric Project**.

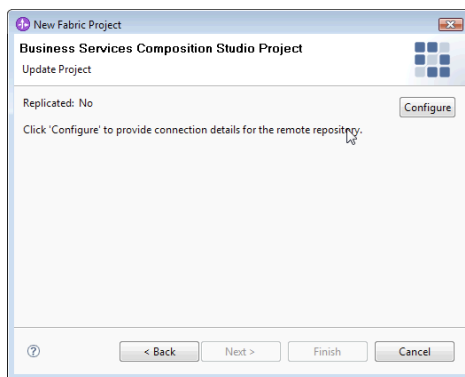


\_\_c. Enter **JK Customer Services Process** for the Project name.

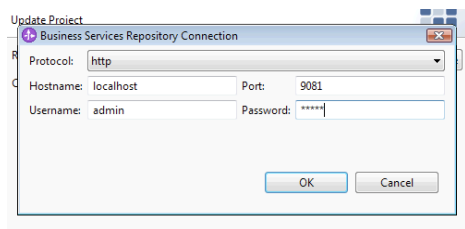
\_\_d. Click **Next**.



\_\_e. Click **Configure Project**

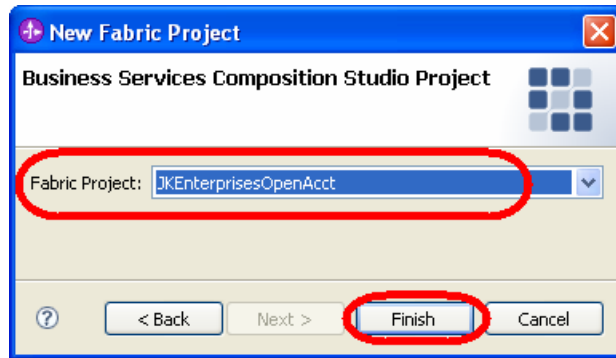


\_\_f. Enter the host name (localhost) and Port: 9081 and the username and password – admin/admin.



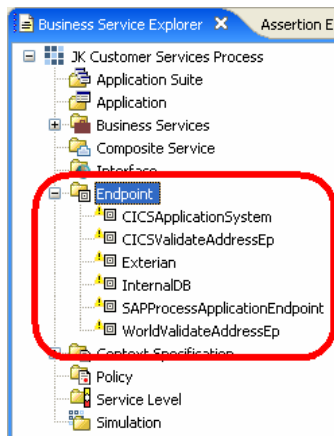
\_\_g. Click **OK** then wait for the project to replicate.

- \_\_h. Click **Next**.
- \_\_i. Select **JKEnterprisesOpenAcct** from the dropdown box and click **Finish** and wait.




As a member of a development team, you have just retrieved a project from the Business Services Repository.

- \_\_j. Click the project's plus sign to expand it.



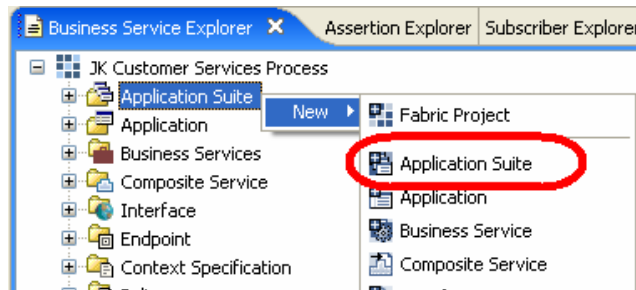
At this time, the project is empty except for six endpoints that have been provided for you in advance. You are adding two endpoints yourself during the labs. These were provided in the interest of time. The yellow marks indicate that they are not yet complete.

- \_\_4. Create your Credit Report Composite Business Service Model instance in this project.
  - \_\_a. Expand **JK Customer Services Process** project collection if necessary.
  - \_\_b. Right-click **Application Suite** and select **New > Application Suite**.



**Important!**  
Be sure to select "Application Suite" and NOT "Application"

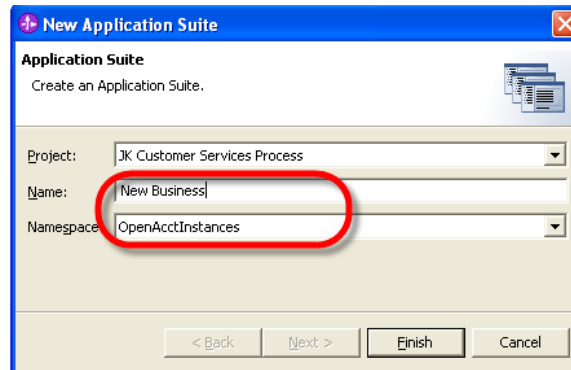




### Application Suite

Recall the definition for **Application Suite** discussed on page **Error! Bookmark not defined..**

- \_\_c. Enter **New Business** as the **Name** and verify that **OpenAcctInstances** is selected for the **Namespace** element. Click **Finish**.



- \_\_d. The editor for the newly created application suite will open. Close this editor by clicking on its **X**.

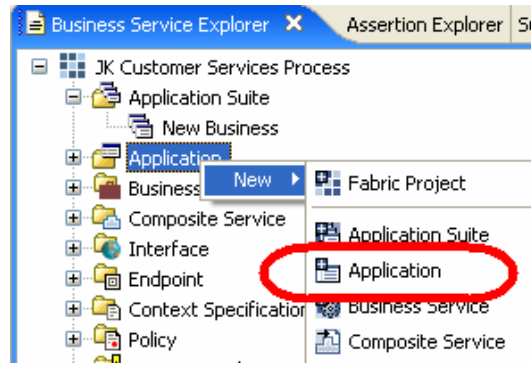


- \_\_e. Right-click **Application** and select **New > Application**.



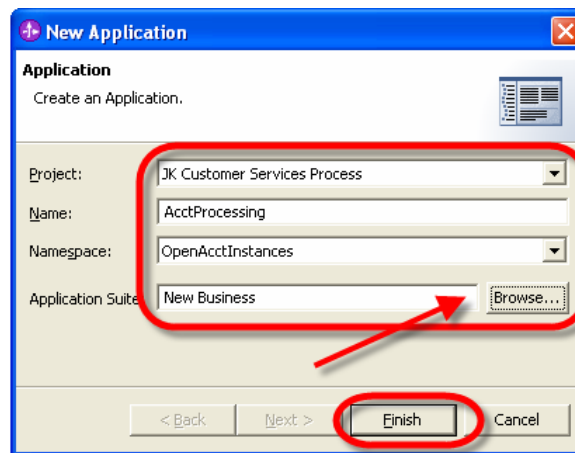
### Important!

Be sure to select "Application" and NOT "Application Suite" this time.



\_\_f. Enter or verify the data in the wizard:

Project	<b>JK Customer Services Process</b>
Name	<b>AcctProcessing</b>
Namespace	<b>OpenAcctInstances</b> (using the drop-down)
Application Suite	<b>New Business</b> (using the Browse option)



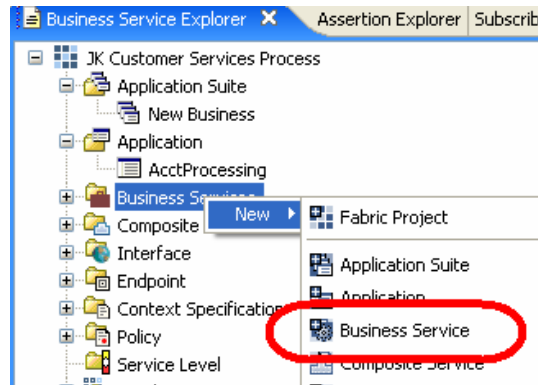
\_\_g. Click **Finish**.

\_\_h. The editor for the newly created application will open. Close this editor by clicking on its **X**.

\_\_5. Create a new Business Process Service.

This is a “process” business service because you are modeling a step in the JK Enterprises business process. Typically each process service is realized as a BPEL process.

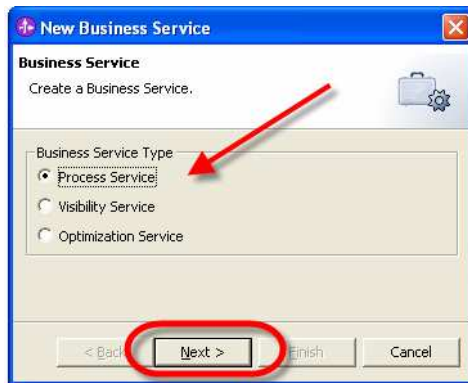
\_\_a. Right click **Business Services** and select **New > Business Service**.



**Business Services**

Recall the schematic and definition for **Business Services** discussed in the **Composite Business Service Resource Hierarchy** on page **Error! Bookmark not defined..** As you can see, you are building out the underlying resource hierarchy of the Business Services Fabric Model.

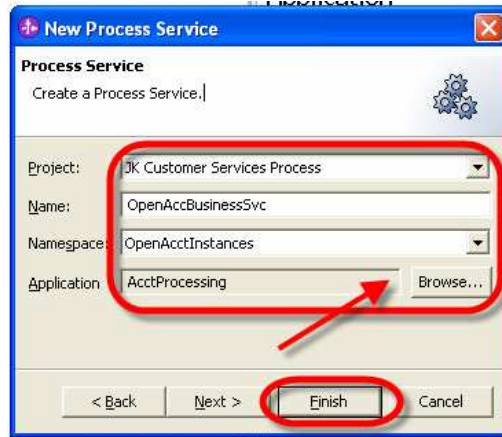
\_\_b. Select **Process Service** from the option list (it should be the default). Click **Next**.



\_\_c. Enter or verify the data in the Process Service wizard:

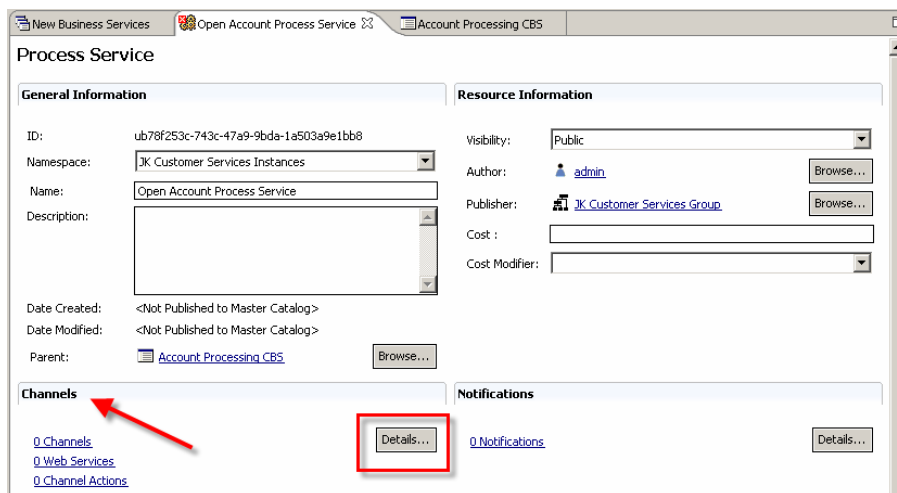
Project	<b>JK Customer Services Process</b>
Name	<b>OpenAcctBusinessSvc</b>
Namespace	<b>OpenAcctInstances</b>
Application	<b>AcctProcessing</b> (using the Browse option)

\_\_d. Click **Finish**.

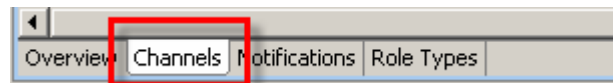


The Process Service editor will open.

- \_\_e. In the OpenAcctBusinessSvc Process Service editor, click **Details** in the **Channels** section of this window.



Alternatively you can have selected the **Channels** tab.

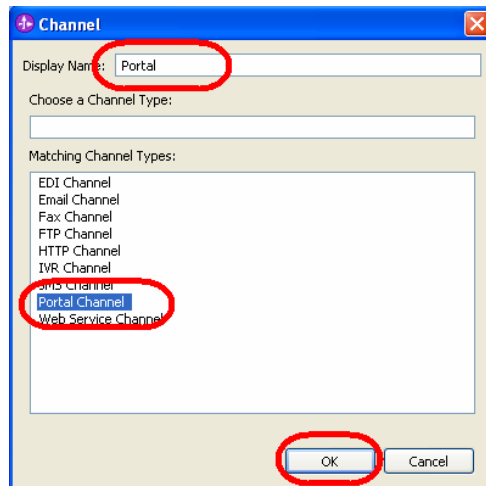


- \_\_f. When the details window opens, click **Add** to the right of the **Channels** table

- \_\_g. Enter or choose the information.

Channel Name	<b>Portal</b>
Matching Channel Types	<b>Portal Channel</b> (choose from available)

- \_\_h. Click **OK**.



### Channel

A channel is a point of consumption.

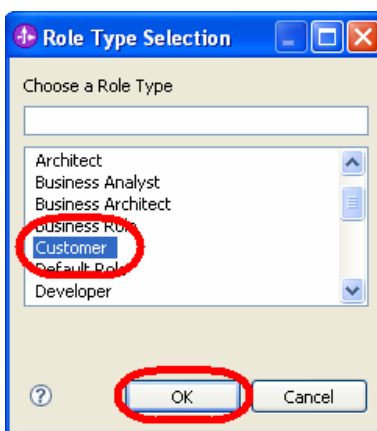
Examples include Web portal, cell phone, ATM, Enterprise Application, and so on.




This is not about who is consuming the service but rather what that “who” is using to consume the service.


Multiple channels can be defined. This is part of the “context” of a service call.

- \_\_6. Associate a role with this channel.
  - \_\_a. Click the Role Types tab at the bottom of the editor.
  - \_\_b. Click **Add Role Type**.
  - \_\_c. Select **Customer** and click **OK**.



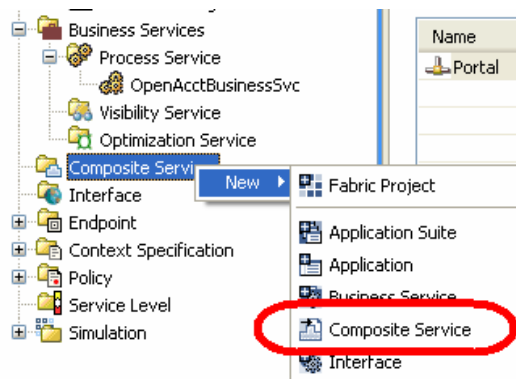
**Role type**

 A role type is the type of user that is going to consume the Business Service. Multiple role types can be selected to consume a business service. For a specific business, custom role types can be added to match company's needs. Industry Content Packs, for example, provide commonly used industry roles.

\_\_d. Click the “save icon”  or use **CTRL-S** and close this editor.

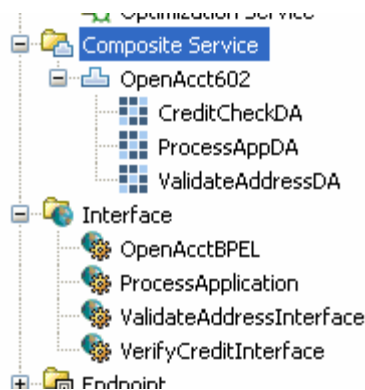
\_\_7. Create a Composite Service by importing the SCA module developed in WebSphere Integration Developer.

\_\_a. Right mouse click **Composite Service** and select **New / Composite Service**.



\_\_b. Accept the default detected from the process model contained in the Business Integration perspective and click **Finish**.

The process of importing the SCA module populates both the Composite Service and Interface collections with information.



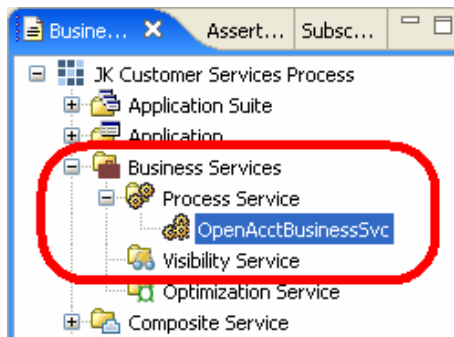
### SCA import



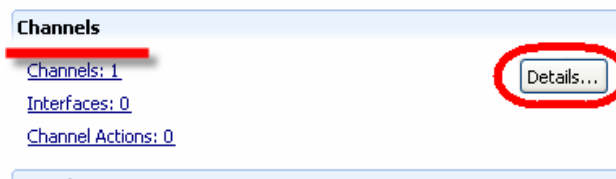
You are not storing the process module and contained BPEL in the Business Services Repository.

This import is only extracting information about the services being orchestrated and not the orchestration logic.

- \_\_c. Double-click the various **imported components** and examine the information they contain.
- \_\_d. **Close** any windows you opened. Do not save changes you might have made to these imported components.
- \_\_8. Now that you have this information, you will associate an interface with the channel you defined earlier.
  - \_\_a. Double-click the **OpenAcctBusinessSvc** business process created earlier to open its editor. (It can still be open.)



- \_\_b. Click **Details** in the Channels area.



- \_\_c. To the right of the Interfaces table, click **Add**.
- \_\_d. Select **OpenAcct602** as the Module
- \_\_e. Verify that **OpenAcctBPEL** was selected as the Interface.
- \_\_f. Click **OK**.

\_\_g. **Save** and **close** this editor.



You have finished defining the OpenAcctBusinessSvc Process Service. Next, you will tie this Business Service to the actual implementation that realizes it.

## 1.2 WebSphere Business Services Fabric enhances WebSphere Process Server

In this lab section you will become more familiar with how WebSphere Business Services Fabric is coupled with WebSphere Process Server and its Service Component Architecture (SCA) programming model.

Essentially WebSphere Business Services Fabric comes with WebSphere Process Server and is tightly coupled to and dependent on it. WebSphere Business Services Fabric's assets, including the Dynamic Assembler SCA component, run on WebSphere Process Server's underlying Java EE application server.

The Dynamic Assembler is made available for development as a widget in WebSphere Integration Developer's Business Integration perspective. During solution development, this Dynamic Assembler SCA widget is placed on the assembly diagram, wired to the BPEL solution, and then configured and instrumented.

This WebSphere Business Services Fabric Dynamic Assembler SCA component provides the declarative, dynamic runtime business policy driven service provider selection capabilities that characterize Composite Business Services Applications. These capabilities enhance and extend IBM's BPM platform.

\_\_9. Understand the invocation sequence in the general case in addition to the specific case used for this workshop. The invocation sequence is:

A "client" makes a call to a BPEL-based service / process. This invocation call provides the required context and content information such as who is requesting a service, what channel they are using,



and information in the service request data payload. [In this workshop, the context was injected using a Java™ snippet in front of the BPEL process.]

The BPEL process executes, orchestrating various service interactions as defined by the process flow.

One or more of the BPEL's "invoke" nodes requests WebSphere Business Services Fabric's Dynamic Assembler to select and provision an appropriate service provider during runtime. The invoke node references the Dynamic Assembler primitive using an import / partner link.

The Dynamic Assembler SCA component is located on the WebSphere Integration Developer Assembly Diagram. It receives the call as a proxy for the invoke node.

The Dynamic Assembler consults the Business Services Registry (cached information if available) using the context, contract, and content (message payload) information provided by the call. It then builds a merged selection policy using appropriate, available business policies. It then finds appropriate candidates and ranks them while rejecting non-compliant endpoints. Then, it selects the best available endpoint to provision on behalf of the service caller using the BPEL invoke node.

The Dynamic Assembler then invokes this selected endpoint service realization. The SCA programming model returns the response to the invoke node.

- \_\_10. Examine the JK Enterprises Open Account BPEL process to see how WebSphere Business Services Fabric extends WebSphere Process Server capabilities.

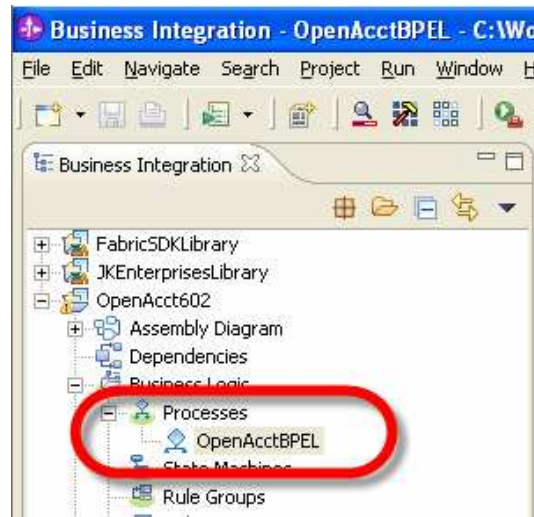
A simplified version of the BPEL process, without the data mapping nodes, is presented in the image below.



This BPEL orchestrates three services in the order: Credit Check, Validate Address, and Process Application.

- \_\_a. Switch to Business Integration perspective using the main menu by selecting **Window > Open Perspective > Other**. Select **Business Integration (default)**. Click **OK**.

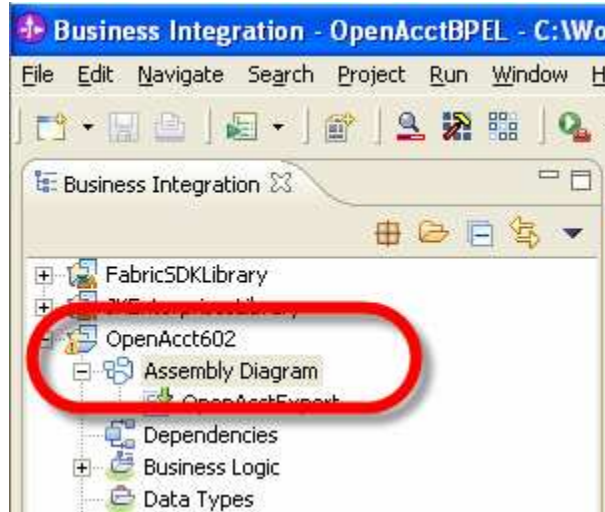
- \_\_b. Expand **OpenAcct602 -> Business Logic -> Processes** and double-click **OpenAcctBPEL** to open the BPEL editor.



- \_\_c. The invoke nodes (circled) are making the calls to Dynamic Assembler SCA components. The first and last nodes provide request and response capabilities while the other four assign nodes map appropriate data from the service caller to and from each of the three services as needed.



- \_\_d. Close this editor by clicking the **X**.
- \_\_11. Examine the Assembly Diagram that contains the Dynamic Assembler's SCA components.
  - \_\_a. Expand the **OpenAcct602** collection and double-click **Assembly Diagram**.

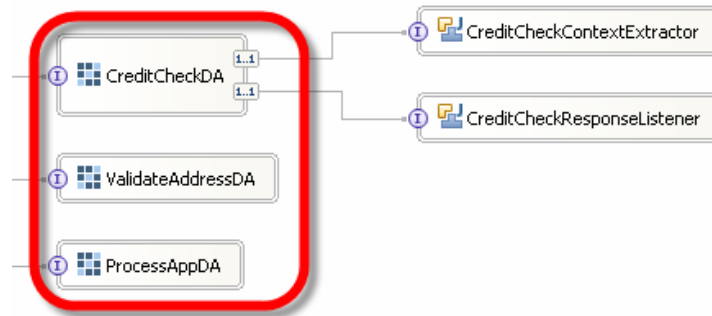


- \_\_b. Enlarge the Assembly Diagram editor as needed (double-clicking on its tab will maximize it and double-clicking that tab again will restore the previous view).
- \_\_c. Locate the component titled **OpenAcctContextInjector**.



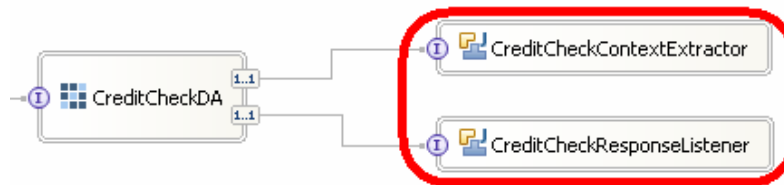
This Java snippet is used in this workshop to inject context information in to the **OpenAcctBPEL** process component wired to its right. Essentially it provides the subscription ID of the caller. This ID is used by the Dynamic Assembler in order to populate the performance database. Later you will use a Web GUI to examine performance information.

- \_\_\_d. Locate the three components to the right of the BPEL component you examined earlier.



These are instances of WebSphere Business Services Fabric's Dynamic Assembler SCA component. Generic Dynamic Assembler components were selected from the tool bar, dropped onto the Assembly Diagram canvas, configured, named, and wired to the exposed BPEL WSDL references.

- \_\_\_e. To the right of the **CreditCheckDA** component, locate the two Java components titled **CreditCheckContextExtractor** and **CreditCheckResponseListener**.

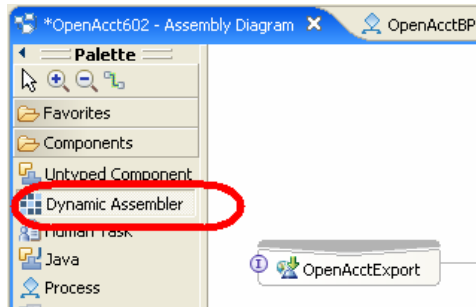


These are Java components that use WebSphere Business Services Fabric's SDK to extend the Dynamic Assembler SCA component as needed. They are referred to generically as extensions. The Fabric SDK provides a rich collection of different types of extensions.

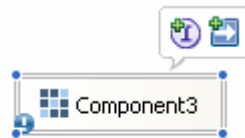
The extensions (or plug-ins) used in this lab are `extractContext` and `handleResponse` extensions. Note that the `ValidateAddressDA` and the `ProcessAppDA` do not have extensions as they are providing round-robin selection of those service endpoints.

\_\_12. Create and examine a Dynamic Assembler SCA component.

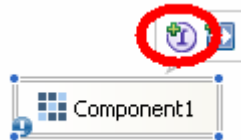
\_\_a. In the palette for the Assembly Diagram editor click the **Dynamic Assembler** component to choose it.



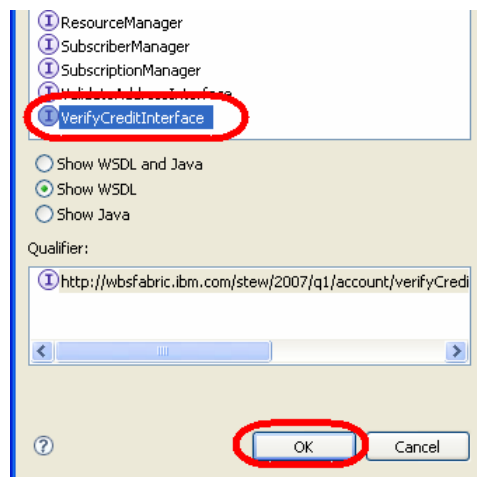
\_\_b. Then, move the mouse to a **blank area** on the canvas and click **the canvas** again to create a new instance of this primitive.



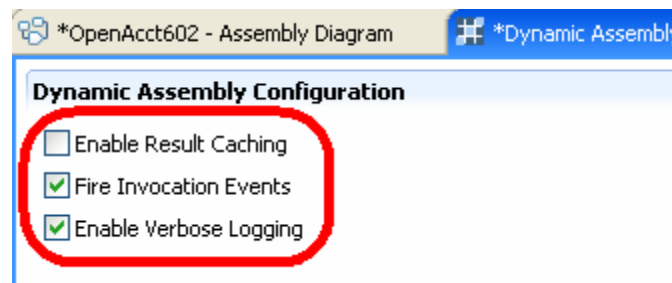
\_\_c. Click the **Add Interface icon**.



\_\_d. In the Add Interface window that opened, select **VerifyCreditInterface** and click **OK**.



- \_\_e. Double-click this new **Dynamic Assembler component** and click **Yes** when asked to implement it.
- \_\_f. Accept the folder **OpenAcct602** and click **OK**.
- \_\_g. The following configuration window will open and reveal this component's configuration. In development, you should disable caching and enable verbose logging. Select appropriate boxes as necessary.

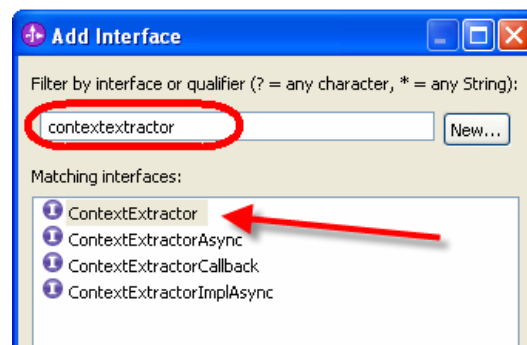


- \_\_h. **Save** and **close** this window.

This is the simplest implementation of a Dynamic Assembler SCA component. ValidateAddressDA and ProcessAppDA are implemented exactly the same way. All that remains is for you to wire it to an appropriate WSDL reference based on the interface you selected.

- \_\_13. Create a Dynamic Assembler extension.

- \_\_a. Select the **Java** component from the Palette and then click the assembly diagram **canvas** to the right of your new Dynamic Assembler component. This will create an SCA Java component.
- \_\_b. Select the **Add Interface** icon as you did previously. A dialog window opens for selecting the appropriate interface.
- \_\_c. Type **contextextractor** in the **Filter by reference** window to limit the number interfaces provided.
- \_\_d. Select **ContextExtractor** and click **OK**.

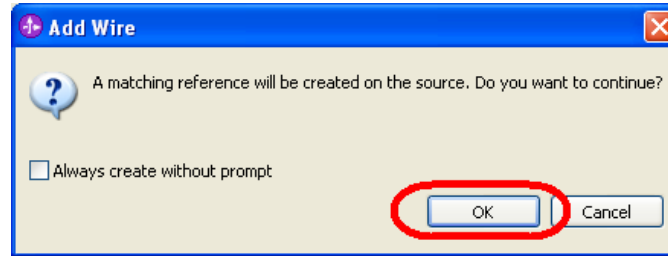


This Java component now has an interface.

- \_\_e. •Click the reference wire from the right side of the Dynamic Assembler you created and drag it to the interface of this ContextExtractor interface.



- \_\_f. In the Add Wire dialog box click **OK** to create a matching interface.



- \_\_g. Double-click the **Java component** (the one to the right) and select **Yes** to implement the component now.

- \_\_h. In the Generate Implementation widow, select **(default package)** and click **OK**.

A Java editor window opens with a stub implementation of the ContextExtractor. A developer, knowing the purpose of the associated Dynamic Assembler, determines the needed information and implement appropriately.

```

*OpenAcct602 - Assembly Diagram  Component2Impl.java
package sca.component.java.impl;

import com.ibm.websphere.fabric.da.PendingRequest;

public class Component2Impl implements ContextExtractor {
    /**
     * Default constructor.
     */
    public Component2Impl() {
        super();
    }
}
  
```

- \_\_i. **Close** this window.

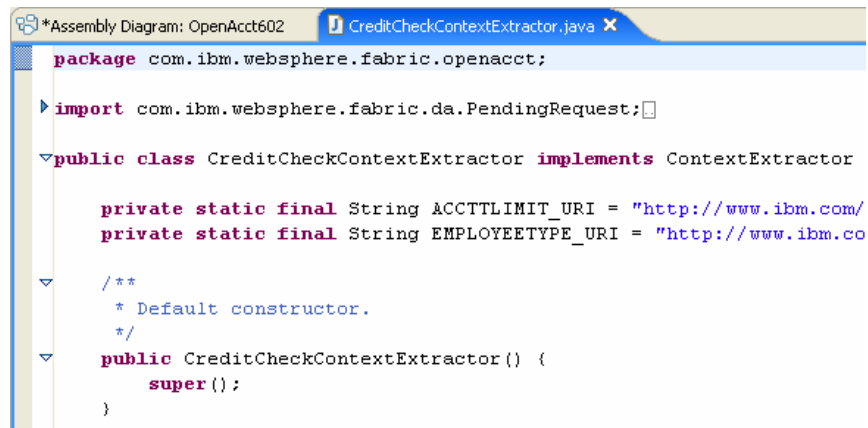
#### Out of scope



The previous steps and further extractor / constructor implementation steps are outside the scope of this workshop.

However, the previous steps are potentially useful for understanding at a finer detail how the Dynamic Assembler is implemented and works.

- \_\_14. Examine the completed CreditCheckContextExtractor used in this workshop.
- \_\_a. Double-click the Java component named **CreditCheckContextExtractor**. This will open its editor and expose the underlying completed Java implementation.



```

*Assembly Diagram: OpenAcct602  CreditCheckContextExtractor.java X
package com.ibm.websphere.fabric.openacct;

import com.ibm.websphere.fabric.da.PendingRequest;

public class CreditCheckContextExtractor implements ContextExtractor

    private static final String ACCTTLIMIT_URI = "http://www.ibm.com/
    private static final String EMPLOYEETYPE_URI = "http://www.ibm.co

    /**
     * Default constructor.
     */
    public CreditCheckContextExtractor() {
        super();
    }

```

Explore the code and compare it to the stub implementation you created earlier.

Notice that only a few additional lines of code were added. They map information from the message body for elements account limit and customer (employee) type into the message header for use by the Dynamic Assembler.

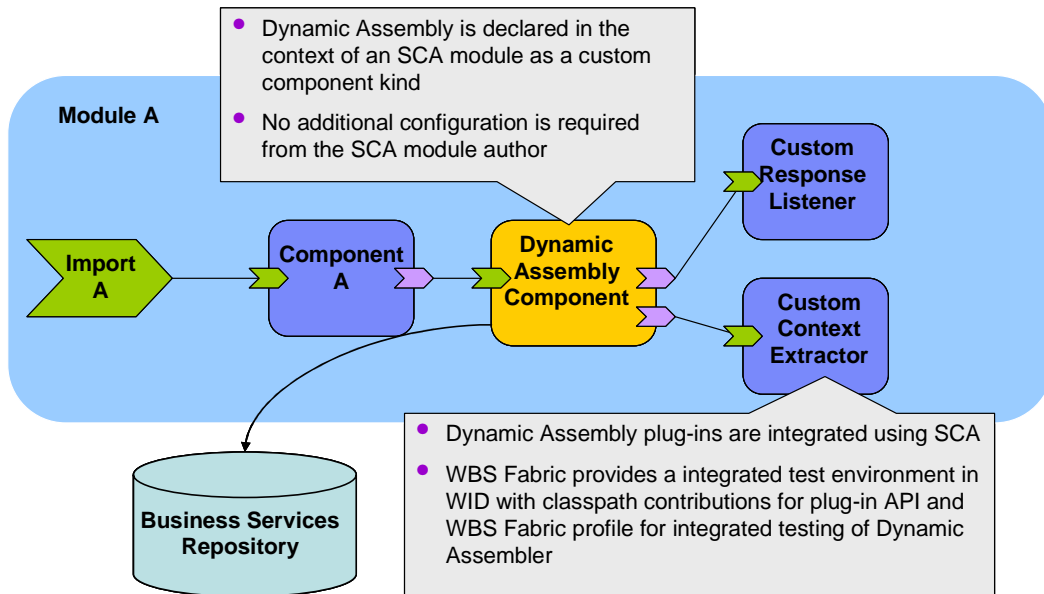
This extension augments the simple Dynamic Assembler component by locating and providing needed information for runtime selection of appropriate service endpoints as describe previously.

Most of the added code generates logging information for attendees of this workshop to see running the three test cases.

- \_\_b. Close this editor by clicking its **X**.
- \_\_c. Close the editor for **OpenAcct602 – Assembly Diagram** by clicking its **X**.
- \_\_d. Click **No** when asked to save changes.



\_\_15. Review, at a high level, the how WebSphere Business Services Fabric enhances and extends WebSphere Process Server capabilities.



WebSphere Business Services Fabric extends IBM's SOA based BPM Platform by providing capabilities for flexible, declarative, business policy based semantic mediation behavior at run time. This enables a new class of SOA based applications called Composite Business Services.

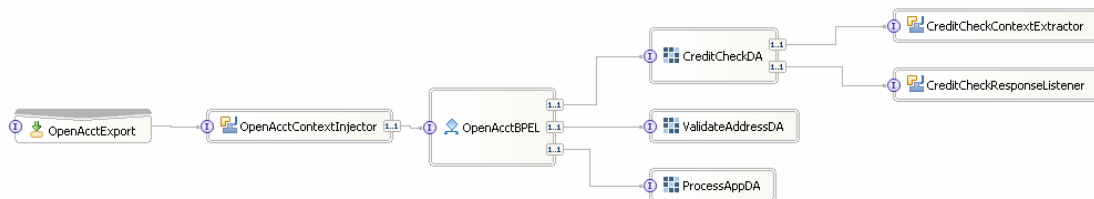
It enables dynamic business service adaptation and delivery through the use of business metadata and policies that eliminate or reduce hard coded (brittle) service provisioning (binding).



You have completed the lab.

### 1.3 Capture service realizations

A simple Open Account process you are enabling for dynamic assembly is represented in the Assembly Diagram view of WebSphere Integration Developer. It contains the BPEL process "OpenAcctBPEL". This process orchestrates three services - "ValidateAddress", "VerifyCredit", and "ProcessApplication". The appropriate realization of these abstracted services is selected by the Dynamic Assembler at runtime based on Content, Context, and Contract.



The **ValidateAddressDA** service is implemented as a round robin and can be either:

- “CICSValidateAddressEndpoint”
- “WorldMappingsValidateAddressEndpoint”

The **ProcessApplicationDA** service is implemented as a round robin and can be either:

- “SAPProcessApplicationEndpoint”
- “CICSProcessApplicationEndpoint”

The **VerifyCreditDA** service can be realized using as any of the three services and is selected appropriately by WebSphere Business Services Fabric. In this workshop, your focus is on the selection of the correct credit reporting service based on policies, capabilities, and the characteristics of the consumer of the open account process. The available credit reporting services are:

- “Exterian”
- “Equinox”
- “InternalDB”

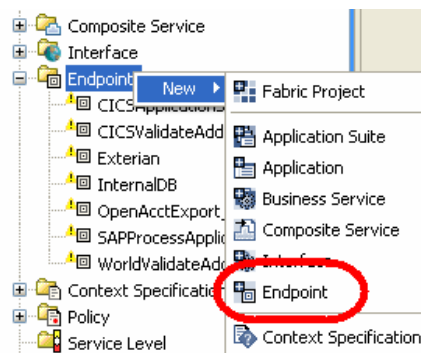
## Equinox endpoint

\_\_16. Add the Equinox endpoint to your WebSphere Business Services Fabric model.

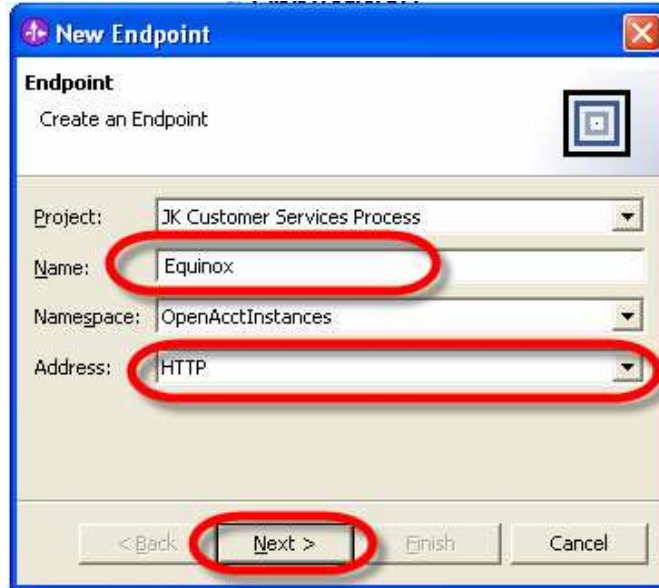
Several endpoints have already been added and configured in advance. Because they were added before you imported the SCA module, their interface references were not available. You will add these interface references shortly.

First, you will follow the entire procedure for adding an additional endpoint - the Equinox endpoint. Adding this new endpoint will quickly illustrate what “capturing service realizations” entails.

\_\_a. Right click the **Endpoint** folder in the Business Services Explorer tree view and select **New > Endpoint**.

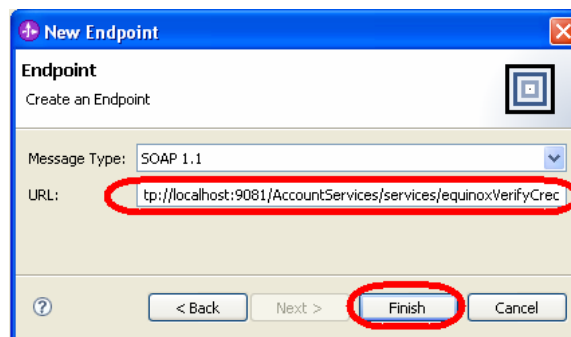


- \_\_\_b. In the Name field enter **Equinox**.
- \_\_\_c. Select **HTTP** for the Address field using the dropdown box.
- \_\_\_d. Click **Next**.



- \_\_\_e. In the window that opened, paste the **Equinox URL** from the clipboard or type it into the URL field.

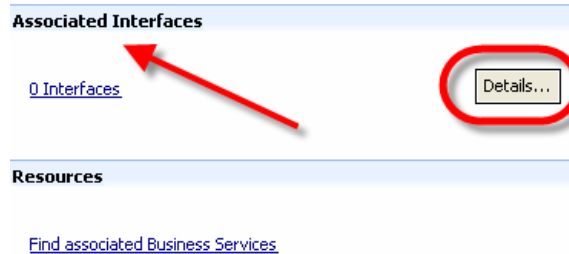
URL: **http://localhost:9081/AccountServices/services/equinoxVerifyCredit**



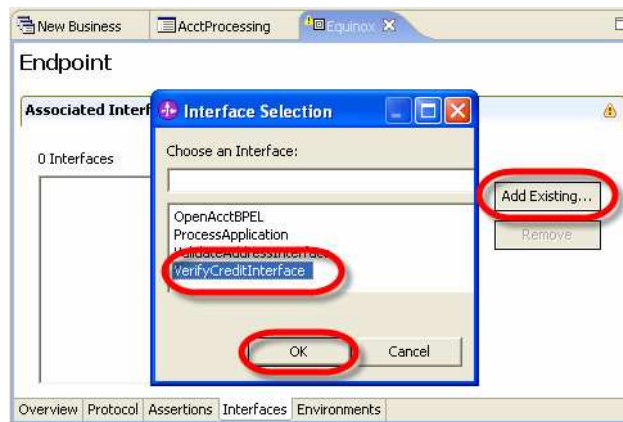
- \_\_\_f. Click **Finish**. The Endpoint editor will open.

Notice the yellow icon warning for this new endpoint in the tree view. Associated descriptions are provided under the Validations tab. This endpoint does not yet support any interfaces.

- \_\_17. Associate this new endpoint to the SCA imported interface information.
- \_\_a. In the Equinox endpoint editor, scroll down and locate the **Associated Interfaces** section and click **Details**. (or use the tab at the bottom of the endpoint editor called "Interfaces")




- \_\_b. Click **Add Existing** and select **VerifyCreditInterface**.
- \_\_c. Click **OK**.



You have associating the endpoint model in WebSphere Business Services Fabric with the service implementation that was acquired when you created the Composite Service.

- \_\_d. Click the **Environments** tab and confirm that the automatically associated environment is **Default**.

**Environments**

 In the Composition Studio Unit Test Environment (UTE), an Environments default setting of "Default" is automatically assigned.

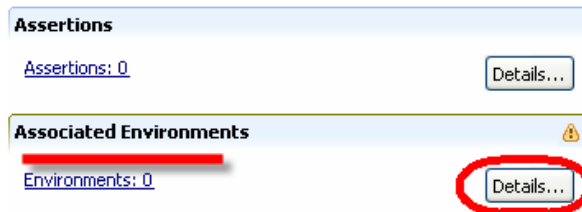
An enterprise typically has multiple environments such as development, testing, performance testing, and production. These are typically customized during setup to match your environment.

- \_\_e. Save your work by using **CTRL-S** and close this editor.

This endpoint has no yellow warnings and is configured properly.

### Complete other endpoints

- \_\_18. Associate an environment with the Web service port export for the entire process module. This endpoint was added when you imported the SCA module.
  - \_\_a. Double click **OpenAcctExport\_OpenAcctBPELHttpService** located under the Endpoint collection.
  - \_\_b. Under Associated Environments, click Details.



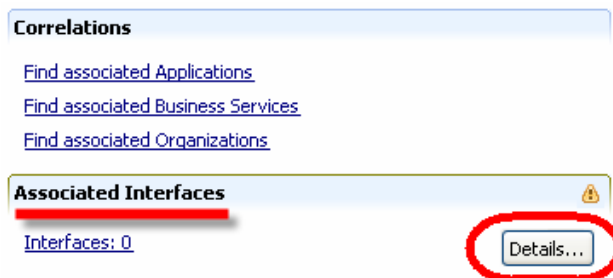
- \_\_c. Click **Add Existing** and select **Default** as the environment.
- \_\_d. Click **OK**.
- \_\_e. **Save** your work and **close** this editor.

The yellow warning marker for this endpoint has been removed.

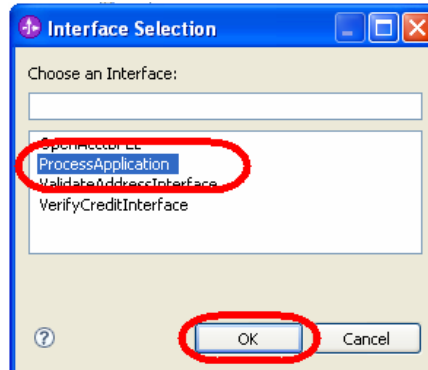
- \_\_19. Complete the configuration of the other endpoints.

Importing the SCA module populated the interface collection with information. The endpoints already provided during lab setup now need to have their interface references associated with these interfaces.

- \_\_a. Double-click **CICSApplicationSystem** under the Endpoint collection to open its editor.
- \_\_b. Click **Details** in the Associated Interfaces section.



- \_\_c. Click **Add Existing**. The Interface Selection window will open.
- \_\_d. For this CICSApplicationSystem endpoint, select **ProcessApplication**
- \_\_e. Click **OK**.



- \_\_f. **Save** and **close** this editor.

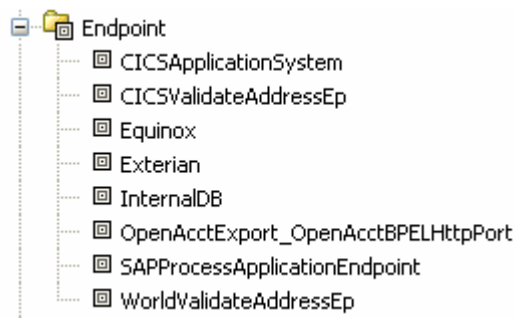
Note that the yellow warning next to this endpoint has been removed.

- \_\_20. Repeat the previous steps (a – f) to add interface references for the remaining endpoints. Use the information.

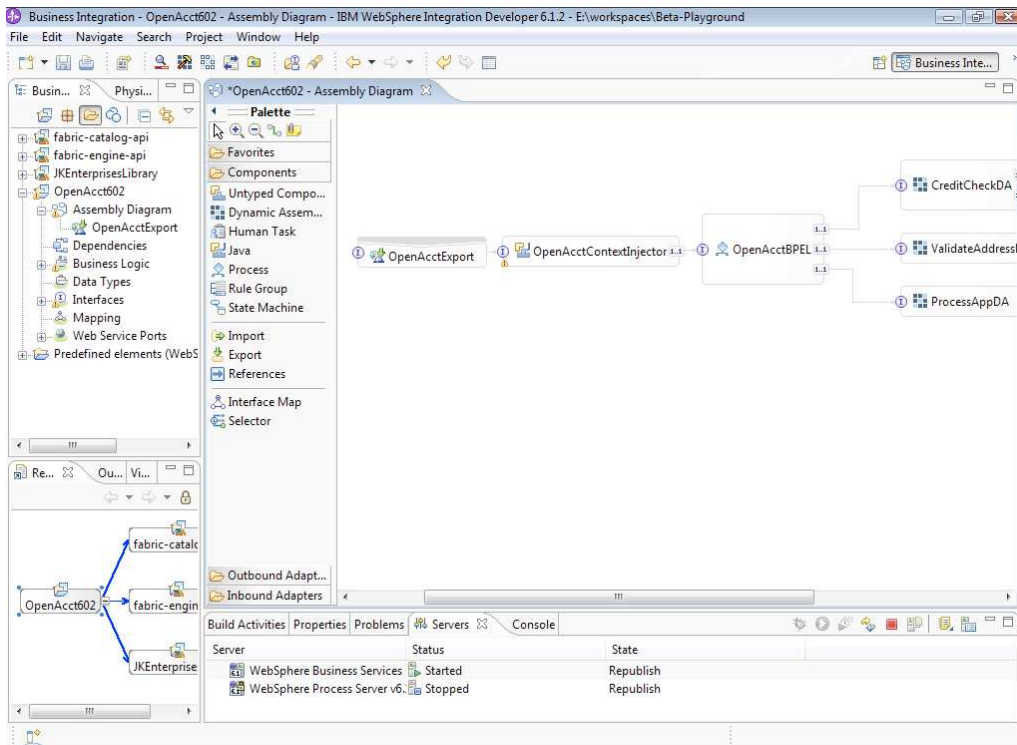
**Table 1: Endpoint Interfaces**

Endpoint Name	Interface Name
CICSValidateAddressEp	ValidateAddressInterface
Exterian	VerifyCreditInterface
InternalDB	VerifyCreditInterface
SAPProcessApplicationEndpoint	ProcessApplication
WorldValidateAddressEp	ValidateAddressInterface

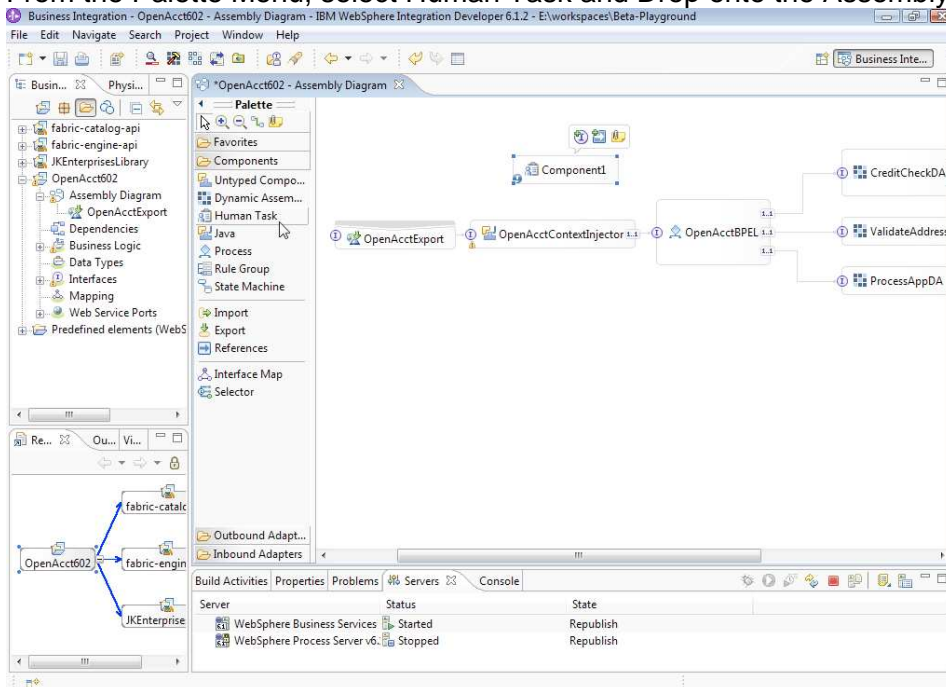
- \_\_a. **Close** any editors you have left open. If asked to **save** information, verify that the information is correct and then close the editor.
- \_\_b. Validate that there are no yellow warning icons for the endpoints.



- \_\_\_21. Now, you will create an Asynchronous Human Task Endpoint, switch to Business Integration Perspective. Expand the OpenAcct602 Module in the Project Tree and expand the Assembly Diagram, click OpenAcctExport to open the Assembly Diagram.

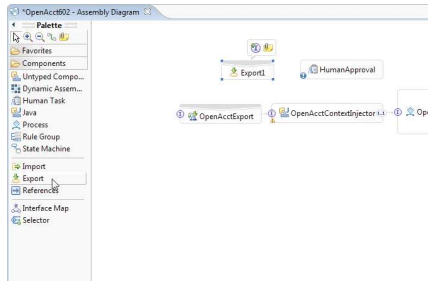


- \_\_\_22. From the Palette Menu, select Human Task and Drop onto the Assembly Diagram.



- \_\_\_23. Rename the component HumanApproval.

\_\_24. Select an Export from the Palette and drop it onto the Assembly Diagram.

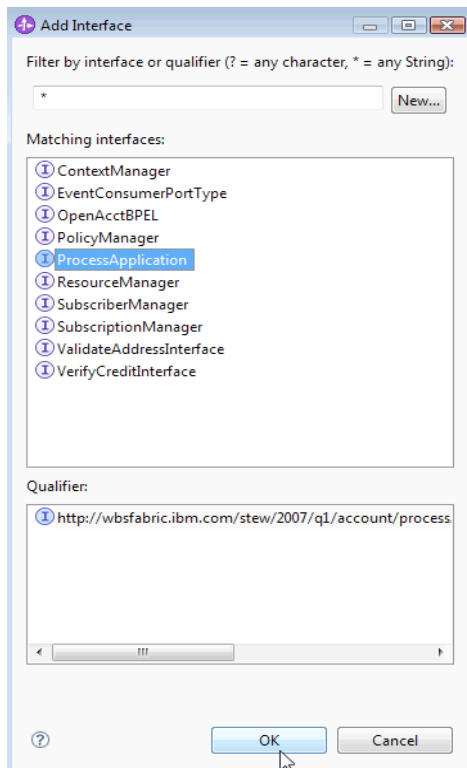


\_\_25. Rename the SCA Component as HA-Export.

\_\_26. Click the Human Approval interface, then select the "I" within the Purple circle to add an interface.



\_\_27. A floating Window should appear and select the Process Application interface and click OK.

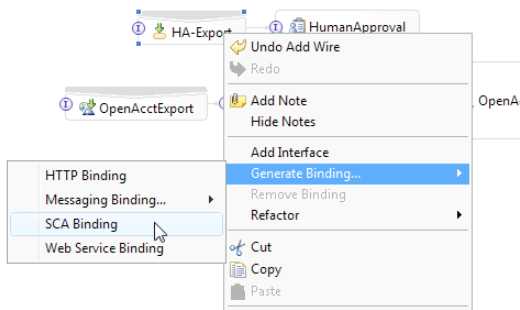


\_\_28. Repeat the previous two steps for the HA-Export component and wire the two components together.

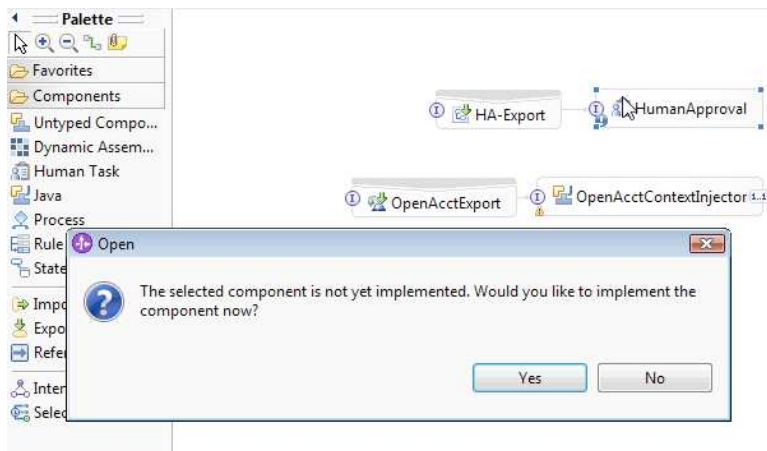




\_\_29. Right click HA-Export and select Generate Binding and choose the SCA Binding.

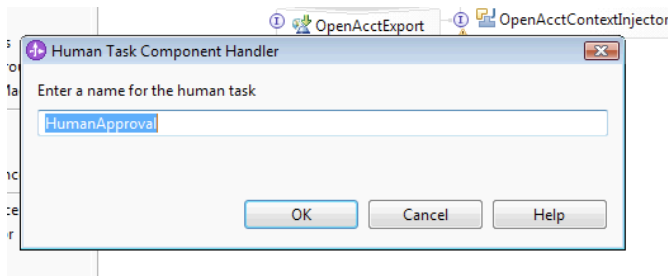


\_\_30. Double click the Human Approval Task, click Yes to generate the implementation.

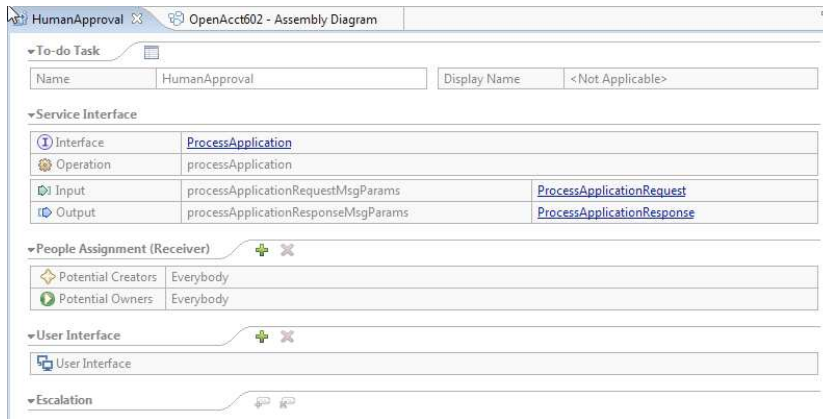


\_\_31. Accept the default and click OK.

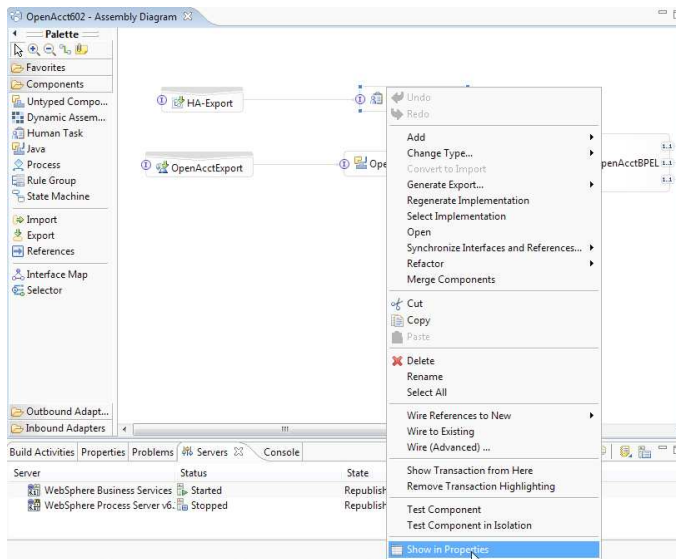
\_\_32. If not already provided, Enter HumanApproval as the the name for the Human Task and click OK.



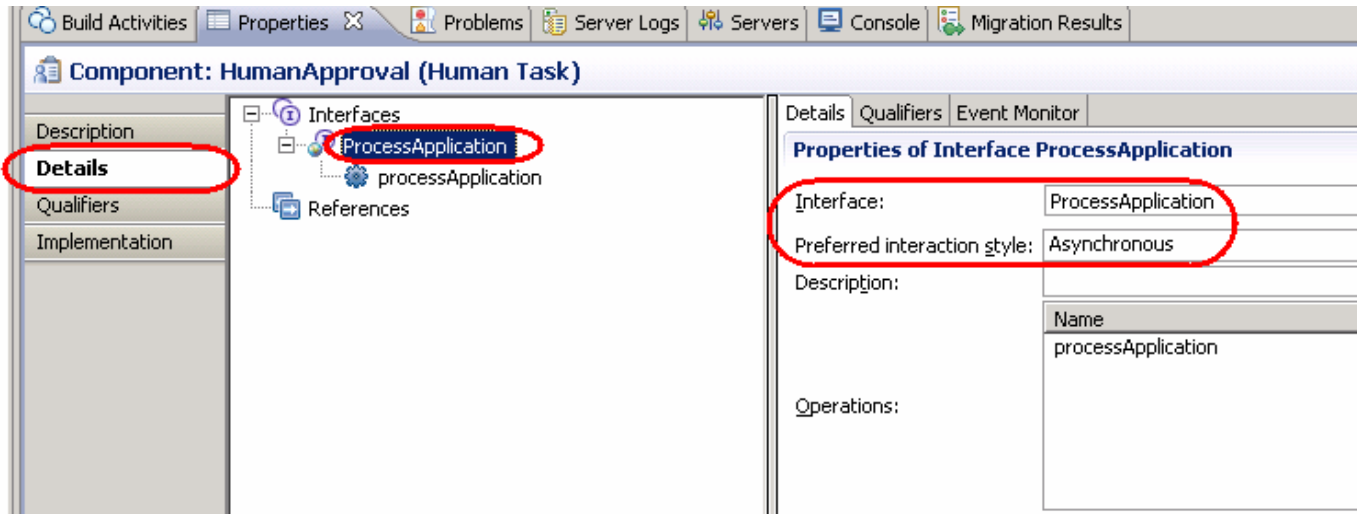
\_\_33. This window should appear. Close it once done looking at it.



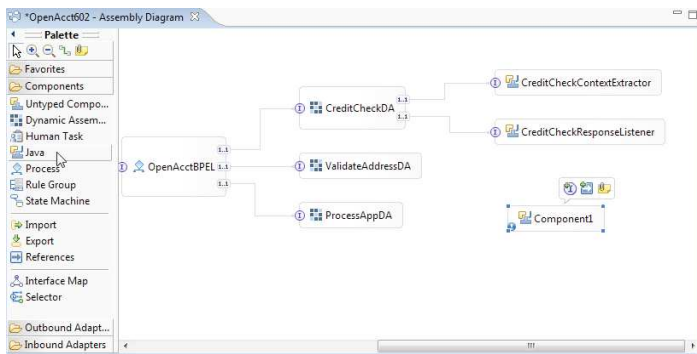
\_\_34. Return to the Assembly Diagram, right click the HumanApproval Task and select Show in Properties



\_\_35. Select the Details Tab within the Properties Tab make sure the information below is displayed.



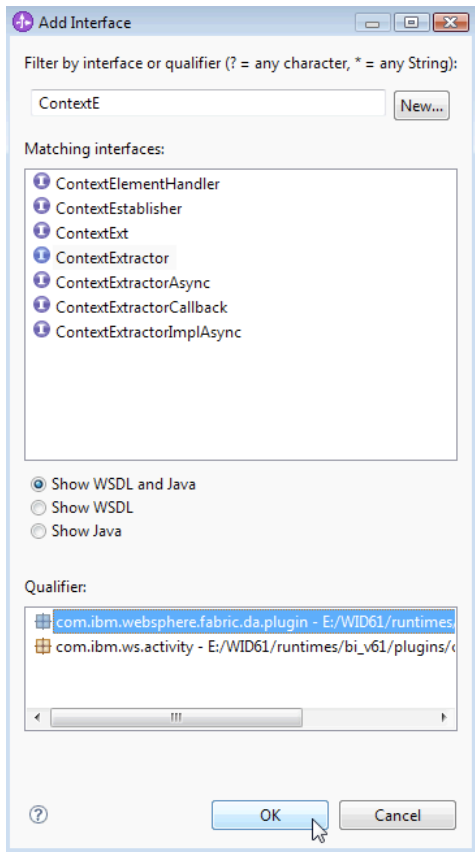
\_\_\_36. Add a Java Component next to the ProcessAppDA



\_\_\_37. Rename the component **ProcessContextExtractor**, click the component and select the add interface purple circle.



\_\_\_38. Select the **ContextExtractor** Interface and click OK.



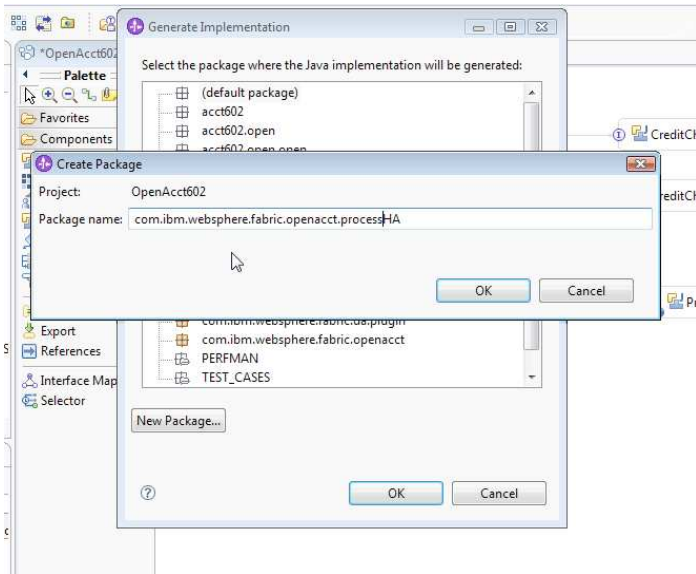
\_\_39. Wire the **ProcessAppDA** component to the **ProcessContextExtractor** Component



\_\_40. Click **OK** to generate the matching reference.

\_\_41. Double click the **ProcessContextExtractor** Component and implement the component by clicking **Yes**.

\_\_42. Click the New Package button and type **com.ibm.websphere.fabric.openacct.processHA** and click **OK** then click **OK** again.



\_\_43. The following window should appear. Replace all the text with the text contained in the **ProcessContextExtractorImpl\_Java.txt**.

```

package com.ibm.websphere.fabric.openacct.processHA;

import com.ibm.websphere.fabric.da.PendingRequest;

public class ProcessContextExtractorImpl implements ContextExtractor {
    /**
     * Default constructor.
     */
    public ProcessContextExtractorImpl() {
        super();
    }

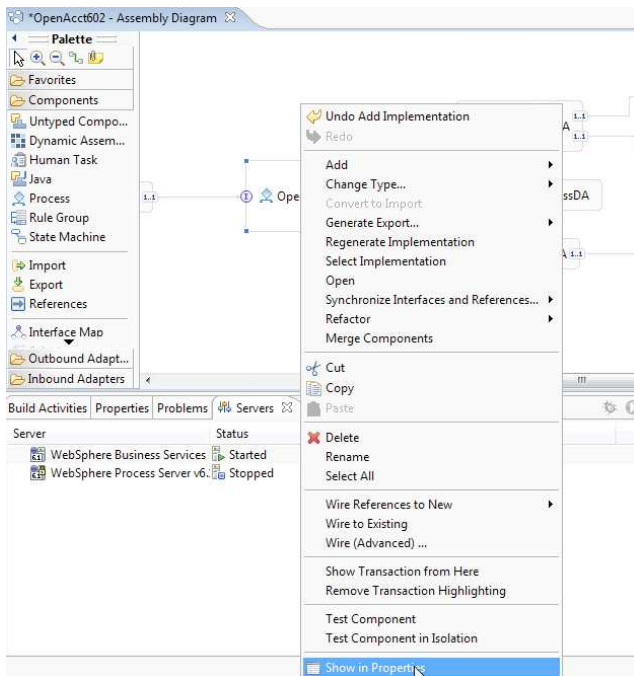
    /**
     * Return a reference to the component service instance for this implementation
     * class. This method should be used when passing this service to a partner reference
     * or if you want to invoke this component service asynchronously.
     * @generated (com.ibm.wbit.java)
     */
    @SuppressWarnings("unused")
    private ContextExtractor getMyService() {
        return (ContextExtractor) ServiceManager.INSTANCE.locateService("self");
    }

    /* (non-Javadoc)
     * @see com.ibm.websphere.fabric.da.plugin.ContextExtractor#extractContext(PendingRequest)
     */
    public Context extractContext(PendingRequest arg0)
        throws UnexpectedContentException {
        //TODO Needs to be implemented.
        return null;
    }
}

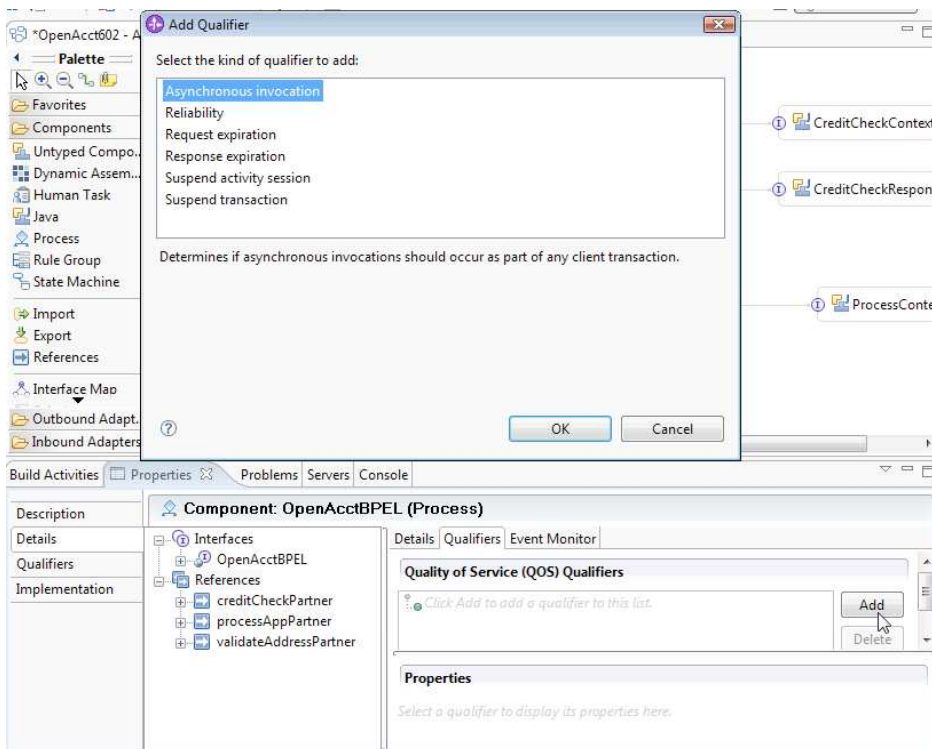
```

\_\_44. Save and close the window.

\_\_45. Right click the **OpenAcctBPEL** and select **Show in Properties**



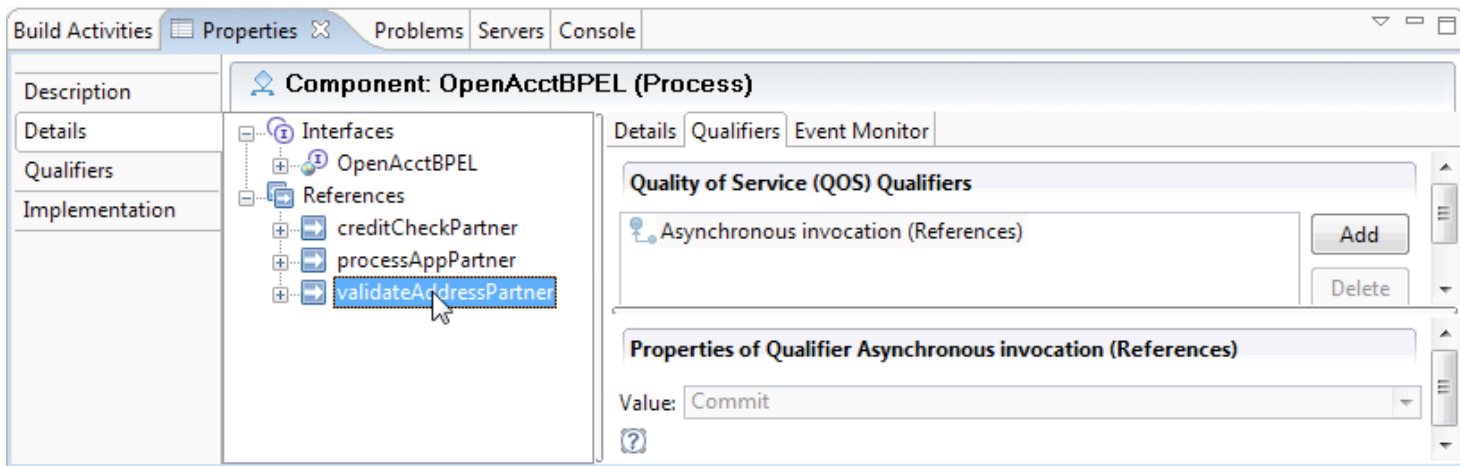
\_\_46. Select the Details Tab within the **Properties** Tab and expand the References Tree, select the References and then select the Qualifier Tab. Click the **Add** button and add the Asynchronous invocation. Click OK.



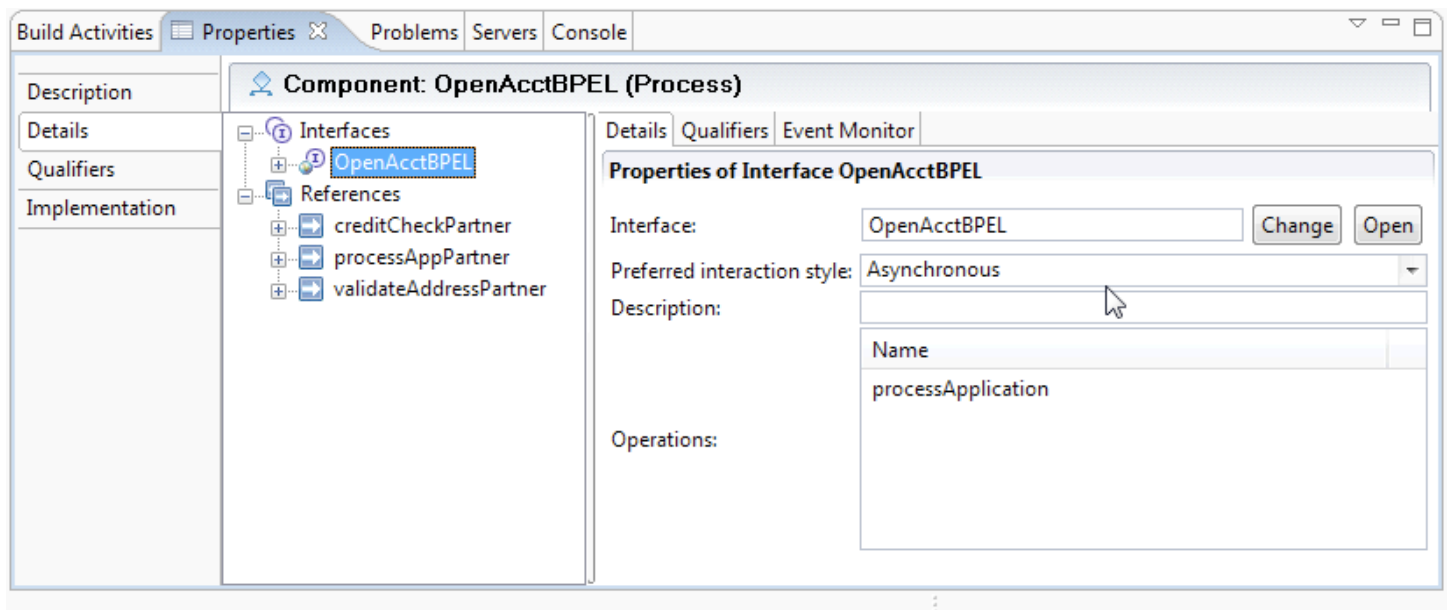
\_\_47. Make sure you choose the Commit Value for the Asynchronous invocation Properties.



\_\_48. Verify these properties have been assigned to the three references: creditCheckPartner, processAppPartner, and validate AddressPartner.

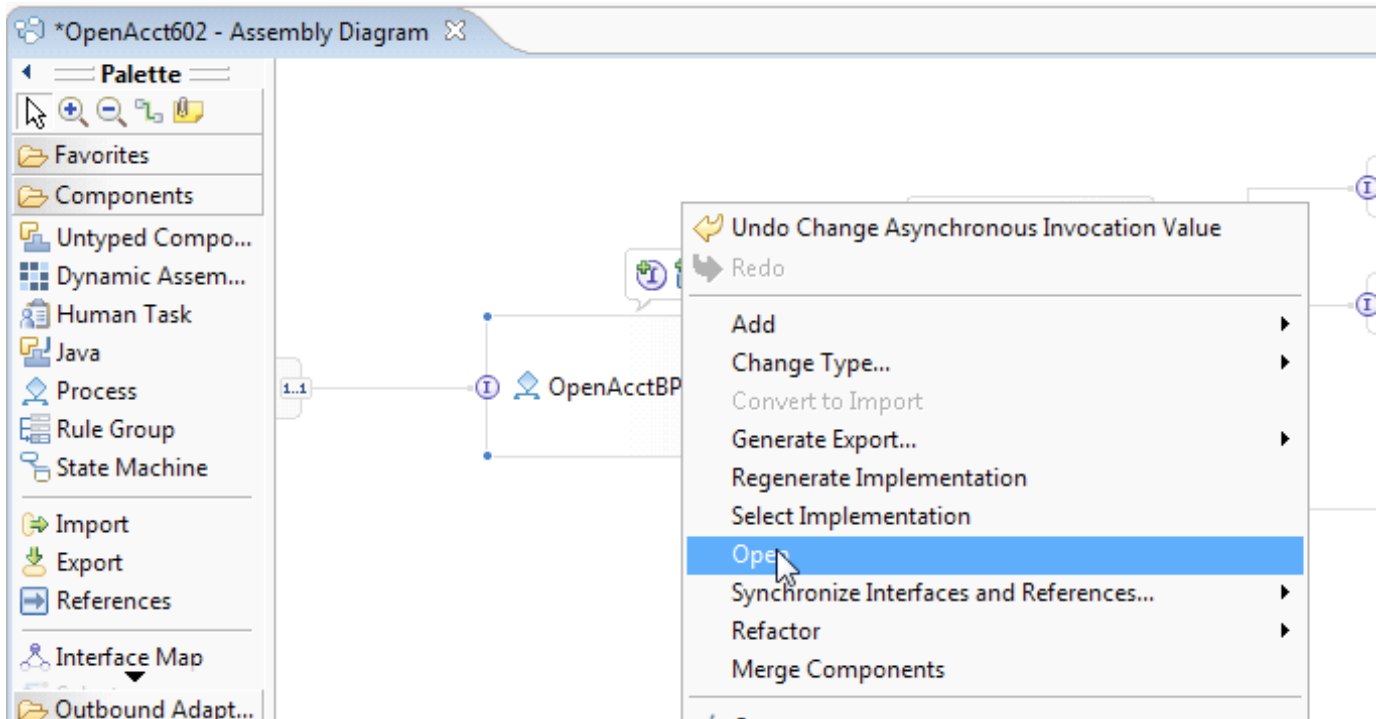


\_\_49. Select the OpenAcctBPEL and the details tab, change the preferred interaction style to Asynchronous.

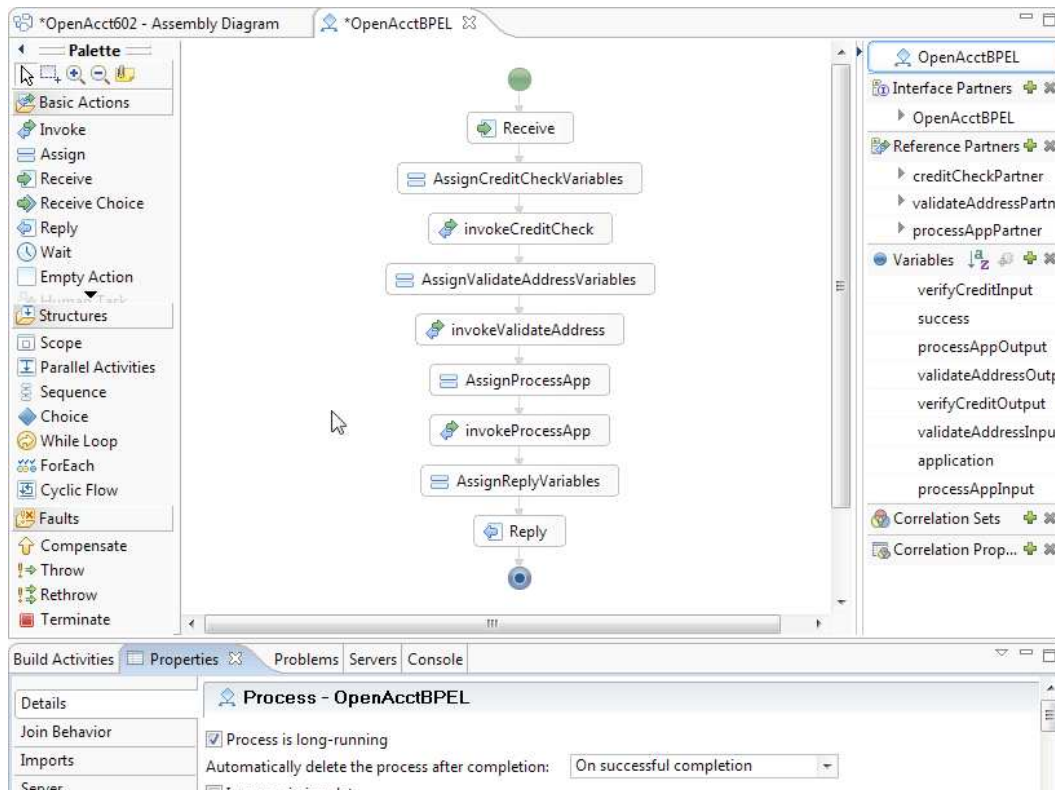


\_\_50. Save all changes by selecting File > Save All

\_\_51. Return to the Assembly Diagram and Right-click the OpenAcctBPEL, select Open.

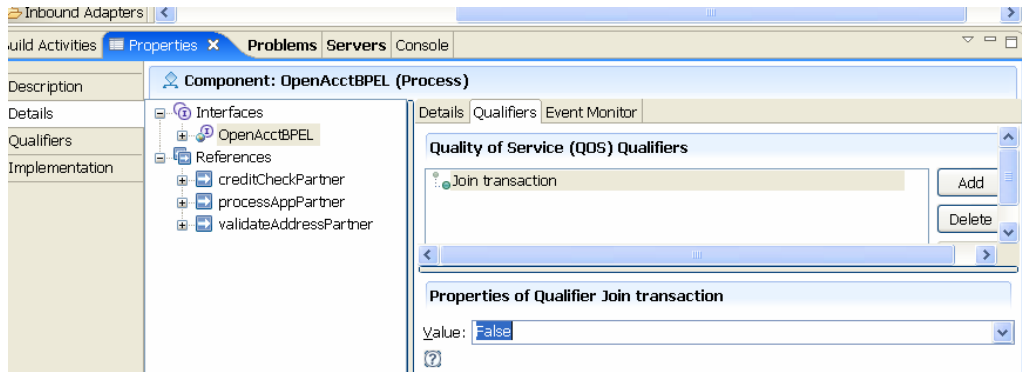


\_\_52. This will open the BPEL window. Select show in properties, then the details tab and select Process is long running.

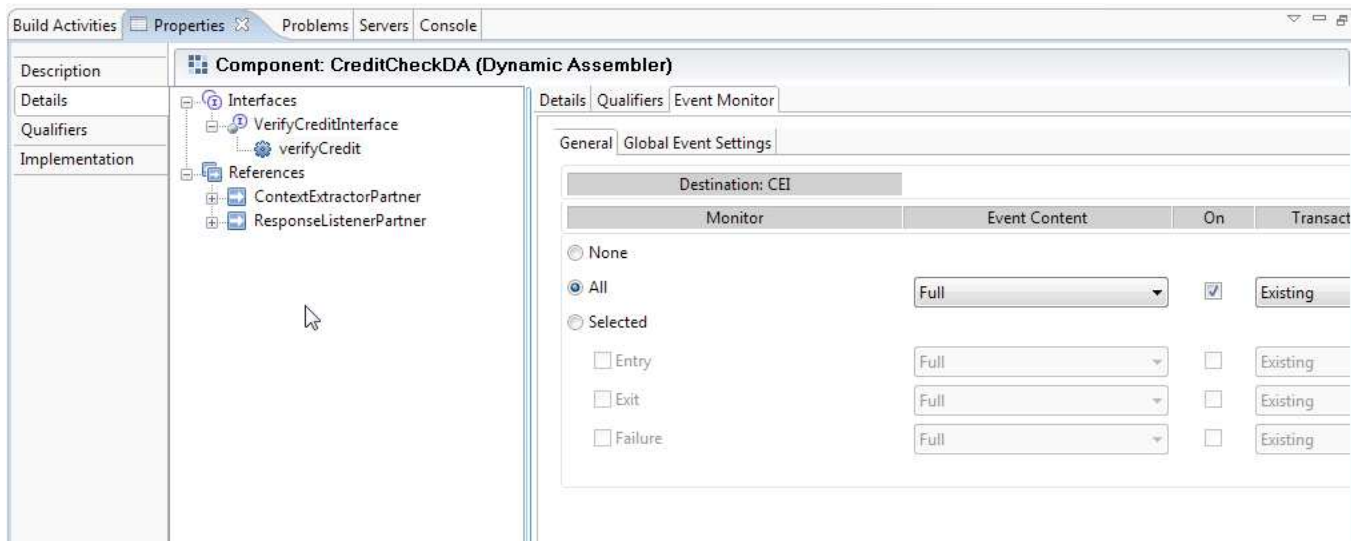




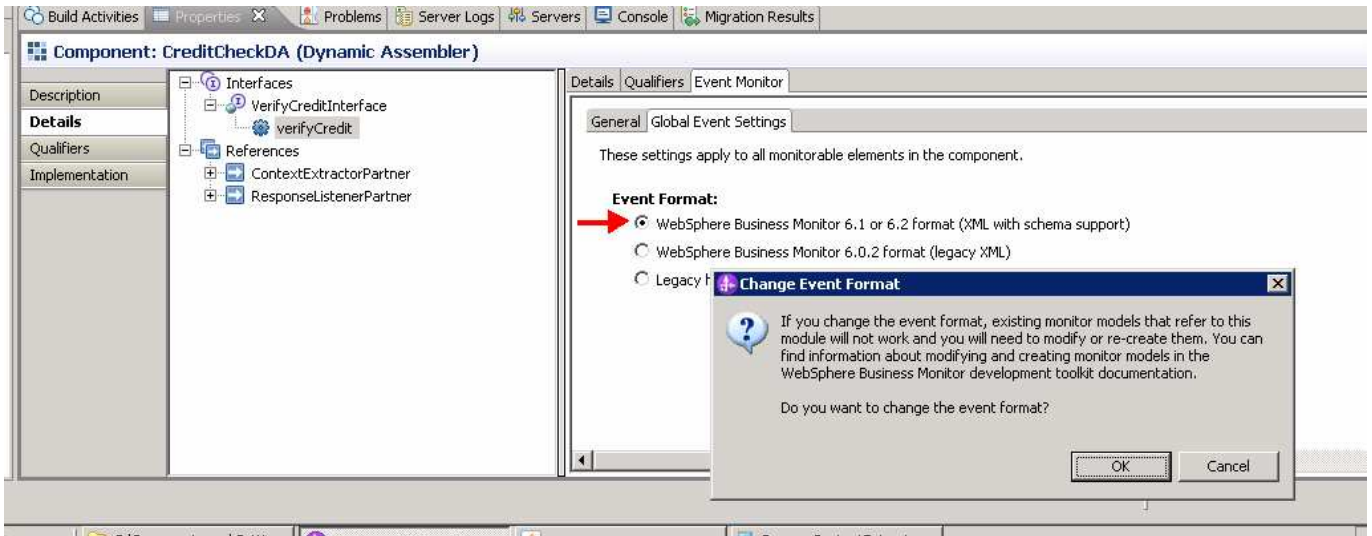
- \_\_53. Save changes by clicking Ctrl +Shift + S
- \_\_54. Return to the Assembly Diagram and right click OpenAcctBPEL. Select Show in Properties
- \_\_55. Select the OpenAcctBPEL and the Qualifiers tab
- \_\_56. Set Join Transaction value to False



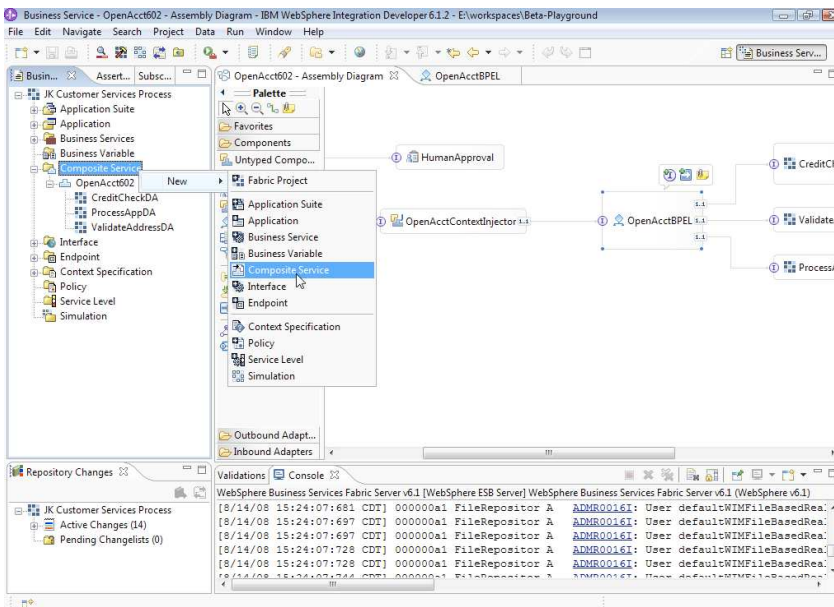
- \_\_57. Save changes by clicking Ctrl +Shift + S
- \_\_58. Right click the CreditCheckDA and select Show in Properties. Select the Details tab and expand the Interfaces tree and select the verifyCredit interface. Click the Event Monitor tab and select "All" and leave the default values of Full and Existing. This will enable Fabric Events to be monitored later in the lab.



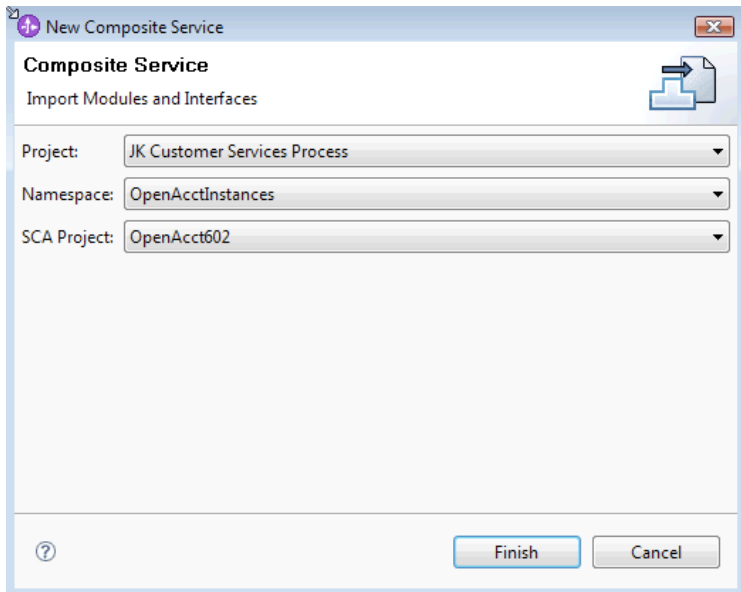
- \_\_59. Now, switch to Global Event Settings Tab, select the WebSphere Business Monitor 6.2 format. A window should appear and click OK.



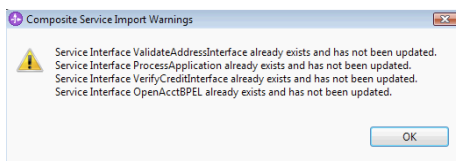
- \_\_60. Save Project by clicking the Floppy Disk in the upper left corner. Wait for the Workspace to finish rebuilding.
- \_\_61. Select Project -> Clean from the menu bar and clean all projects.
- \_\_62. Switch to Business Service Perspective, Select New Composite Service.



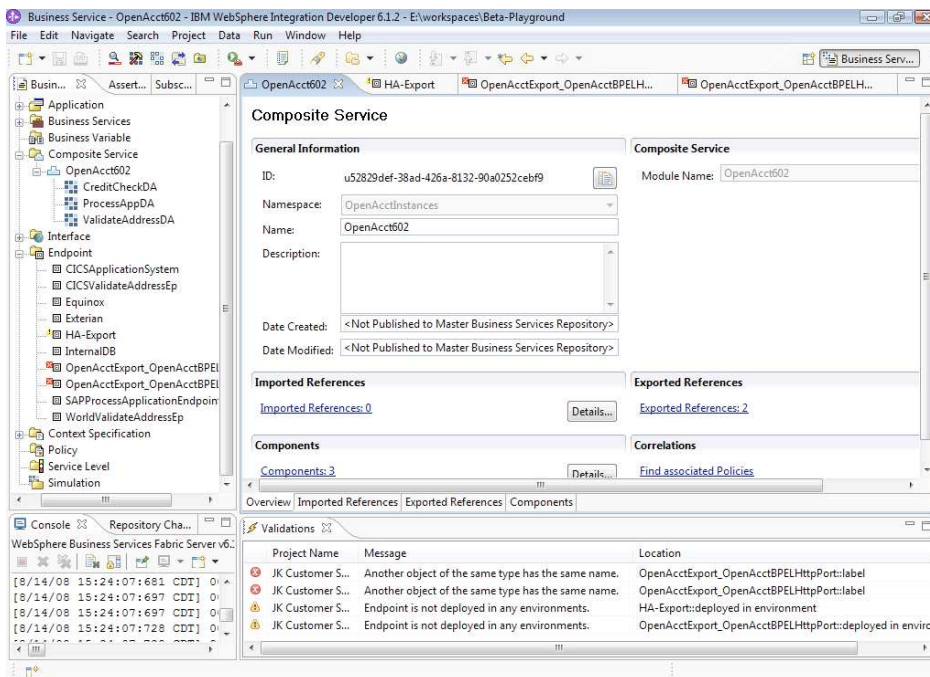
- \_\_63. Accept the Defaults and click Finish.



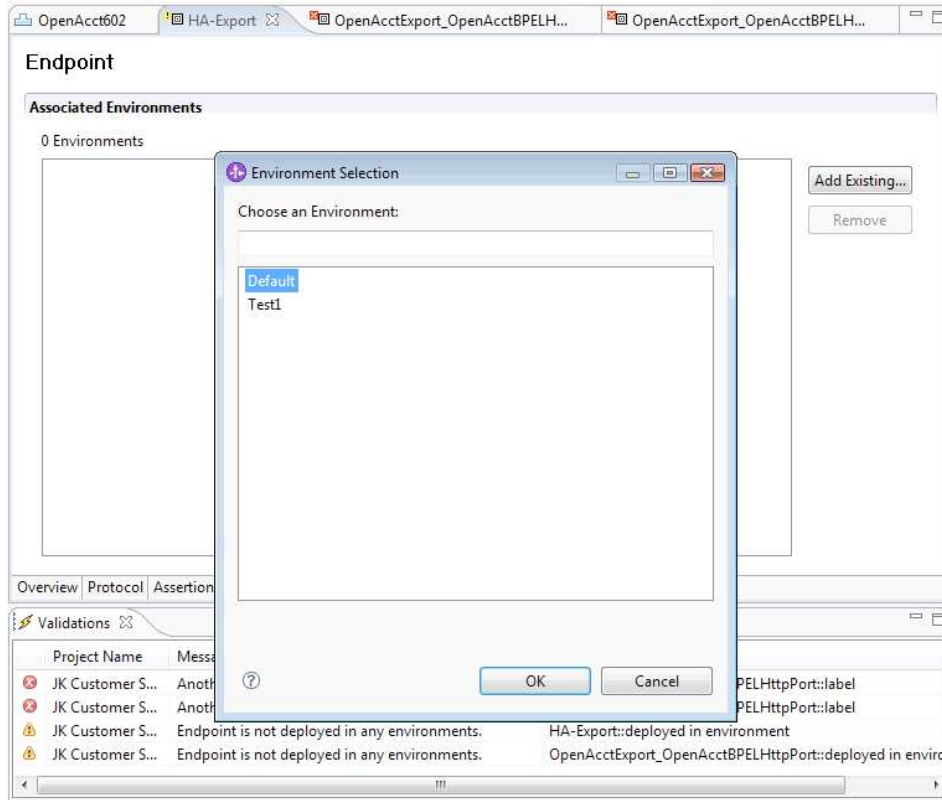
\_\_64. A window with import warnings and click OK.



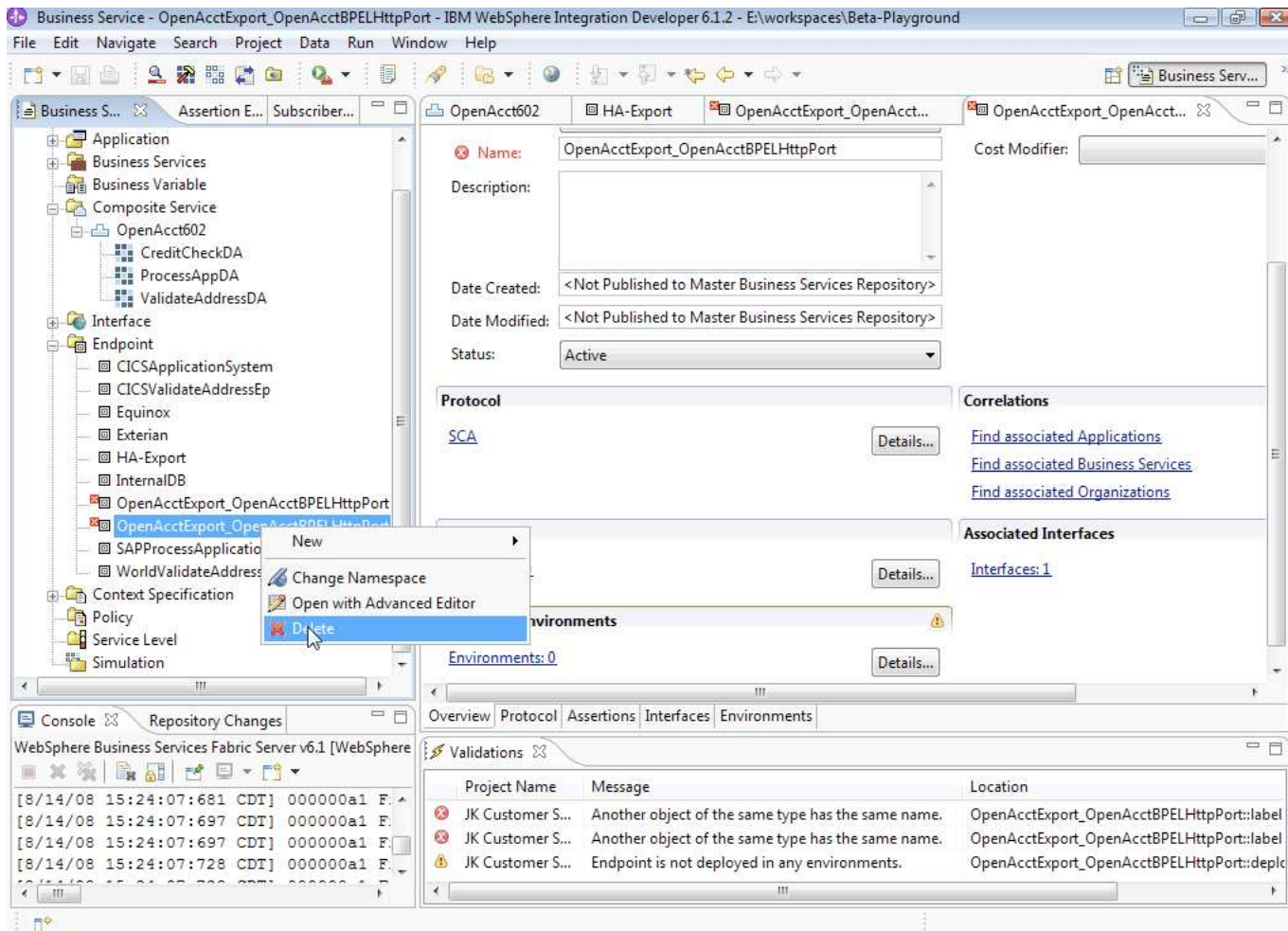
\_\_65. Your Workspace should appear similar to the one displayed below:



\_\_66. Open the HA-Export Endpoint and select the Environment sub-tab, then click Add Existing and Choose Default. Click OK.



\_\_67. If you have two OpenAcctExport\_OpenAcctBPELHttpPort endpoints, delete the extra endpoint. The extra endpoint does not have an environment defined.



\_\_68. Save the project by Ctrl+Shift+S



You have completed this lab. In the next part you will define business policies and then use them to simulate dynamic selection of an available credit reporting services based on declarative business policies.

## Part 2 Define and simulate business service policies

### 2.1 Review credit reporting services

WebSphere Business Services Fabric uses customer context, message content, and generates a merged business policy contract to provision the correct service. For reference a review of the available credit reporting services and their capabilities follows:

- **InternalDB** – An internal Web service that uses a database containing stored credit reports for customers with “EXISTING” accounts. The cost of retrieving a credit report from this service is \$0.00 per transaction. It only supports credit requests that are considered to be “SMALL”. Also, this service is not available on Sundays due to maintenance.
- **Equinox** – A vendor provider of credit assessment reports. They charge \$1.50 per transaction. This service can support both “NEW” and “EXISTING” customers. It only supports credit requests that JK Enterprises considers “SMALL”. The accuracy of their data is considered to be of medium grade quality because they update their records on a quarterly basis. They support credit requests 24 x 7 x 365 – always available. *This is the endpoint you built out in the previous lab section and whose policies you will focus on in this lab section.*
- **Exterian** – A vendor provider of credit assessment reports. They charge \$5.00 per transaction. This service can support both “NEW” and “EXISTING” customers. JK Enterprises only uses this service for credit requests that are considered “LARGE”. The accuracy of this credit reporting agency’s data is very high. They update their records on a monthly basis - and charge a premium for this. They support credit requests 24 x 7 x 365 – always available.
- **HA-Export** – this endpoint was added to demonstrate the asynchronous capability with a Human Task a feature now available as of Fabric version 6.1.2. It is triggered when an account credit request is greater than \$20,000.

### 2.2 Declare endpoint assertions

By executing the steps in this section, you are informing WebSphere Business Services Fabric of the capabilities associated with the credit service providers available to JK Enterprises.

These capabilities are called “assertions”. They are stored in WebSphere Business Services Fabric’s meta-model inside the Business Services Repository. At run time the Dynamic Assembler uses these endpoint assertions, the service consumer’s information, and declarative business policies to select the “best” service provider endpoint that meets the requirements.


There are five dimensions along which assertions can be applied:

- ◆ Performance
- ◆ Reliability
- ◆ Interoperability
- ◆ Security
- ◆ Manageability

You will only apply a few assertions in this lab.

## Equinox assertions

- \_\_69. Refocus the endpoint editor for the **Equinox** endpoint.
  - \_\_a. Double-click **Equinox** in the Endpoint folder to open its editor.
- \_\_70. Add the account size assertion to the Equinox credit service.



**Assertions are used to describe capabilities or requirements.**

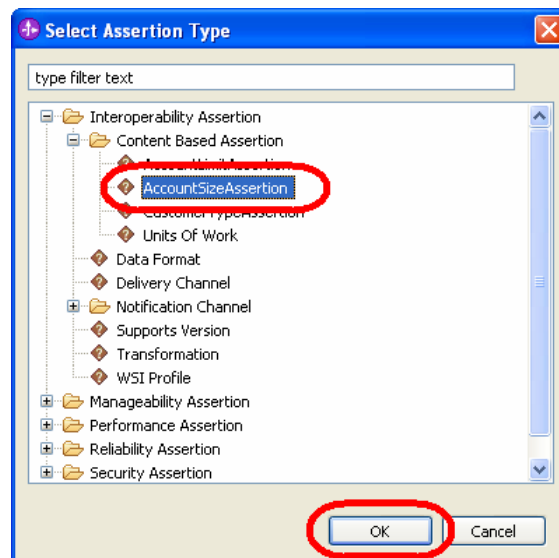
Endpoint assertions describe the capabilities of endpoints.

Policy assertions describe requirements for different contexts used by the Dynamic Assembly for endpoint selection.

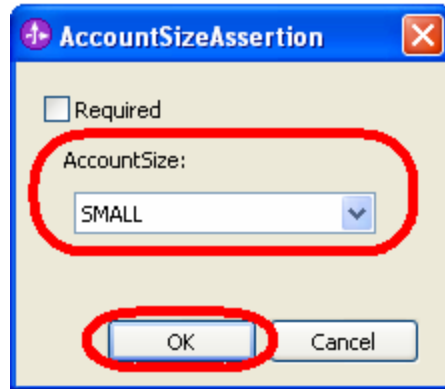
- \_\_a. Click the **Assertions** tab for this editor.



- \_\_b. Click **Add** located in the upper right corner of the Endpoint Capabilities table.  
Recall that JK Enterprises only wants to use this credit reporting service for small loan requests.
- \_\_c. Expand the **Interoperability** folder and, under it, expand the **Content Based Assertion** folder.
- \_\_d. Select **AccountSizeAssertion** and click **OK**.



- \_\_e. In the window that opens, select **SMALL** from the drop down box then click **OK**.



You have created an instance of this assertion type and associated it with this endpoint. There is now a row for this assertion in the table.

- \_\_71. Add the CustomerType assertion to the Equinox credit service.

Recall that JK Enterprises wants both new and existing customers to be supported by this service.

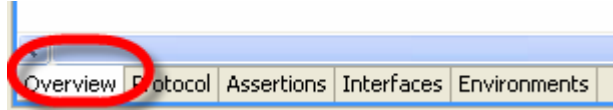
- \_\_a. With focus still on the Assertions tab, click **Add**.
- \_\_b. Expand the **Interoperability** and then the **Content Based Assertion** folders.
- \_\_c. Select the **CustomerTypeAssertion** and click **OK**.
- \_\_d. In the window that opens, select **EXISTING** as customer type and click **OK**.  
A second row will have been added to the assertion table indicating that this endpoint supports existing customers.
- \_\_e. Add a CustomerTypeAssertion of value **NEW** using the same procedure.  
A third row is now visible in the assertion table.
- \_\_f. Verify that you have three rows in the Endpoint Capabilities assertion table as shown below.

Type	Required	Value
AccountSizeAssertion	<input type="checkbox"/>	SMALL
CustomerTypeAssertion	<input type="checkbox"/>	EXISTING
CustomerTypeAssertion	<input type="checkbox"/>	NEW



\_\_72. Add the cost that Equinox charges JK Enterprises for their credit report service.  
Recall that they charge \$1.50 per credit report transaction.

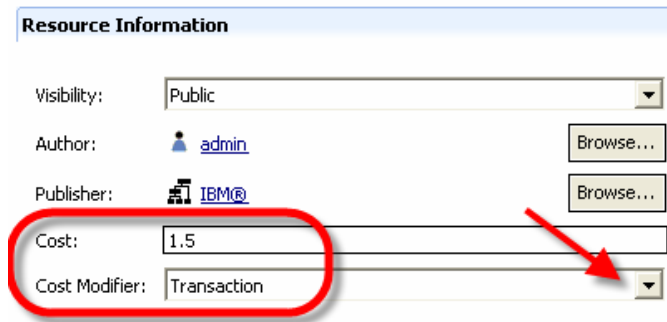
\_\_a. Click the **Overview** tab at the bottom of the editor.



\_\_b. Locate the **Resource Information** area of the editor. (upper right corner)

\_\_c. Enter the information for the **Cost** and **Cost Modifier** fields.

<b>Cost:</b>	<b>1.5</b>
<b>Cost Modifier:</b>	<b>Transaction</b> (using dropdown box)



\_\_d. **Save** your work.

\_\_e. **Close** the Equinox editor.

**Other endpoint assertions**

\_\_73. Validate other endpoint assertions.

Assertions have already been added for InternalDB and Exterian credit services.

\_\_a. Open each of these endpoints and examine their values to verify correctness.

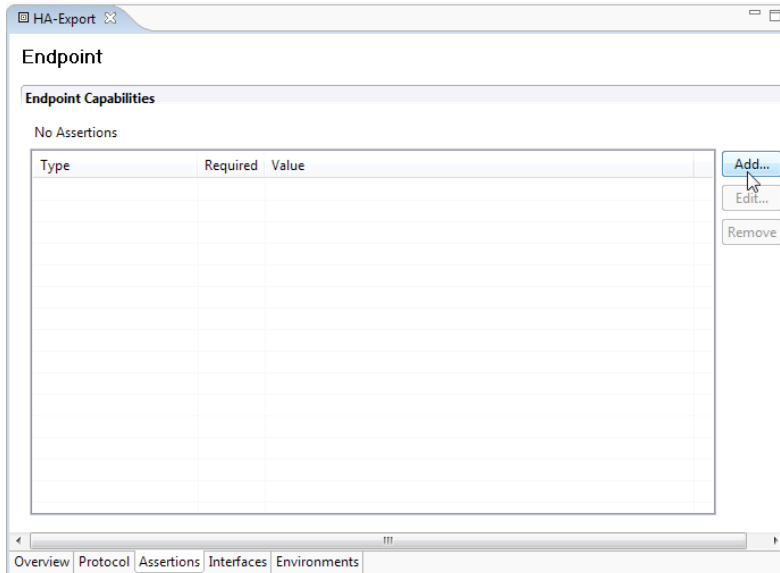
**Table 2: Endpoint Assertions**

Assertions	InternalDB	Exterian
Customer Type	EXISTING	NEW & EXISTING
Account Size	SMALL	LARGE
Cost	0.0	5.00

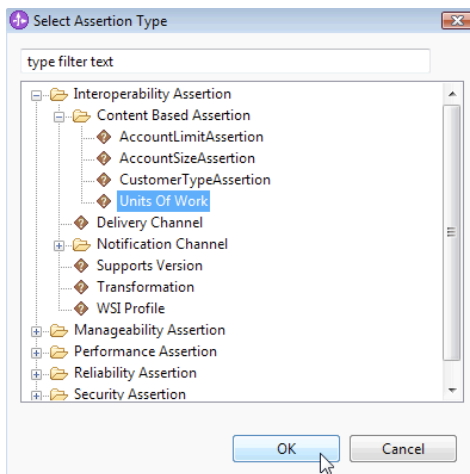
\_\_b. **Close** their editors.

You have defined or validated the endpoint assertions (capabilities) for the three credit reporting services.

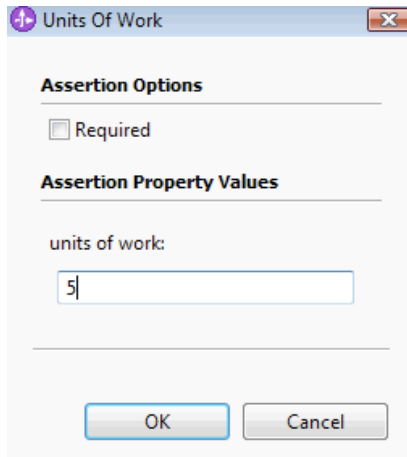
- \_\_74. Double click the HA-Export Endpoint editor and select the assertions tab, click Add.



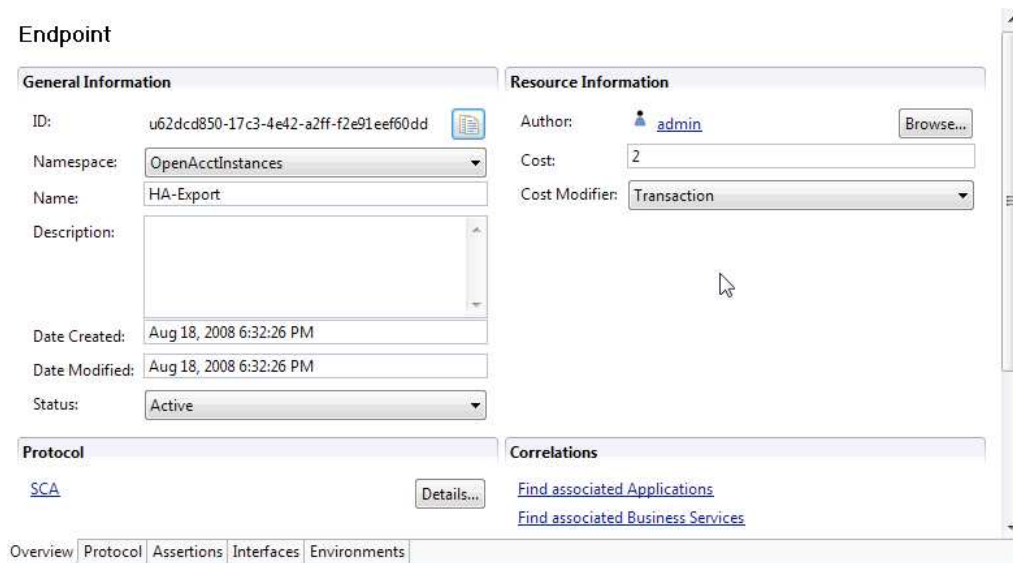
- \_\_75. Select the Units of Work Assertion and click OK



- \_\_76. Enter a value of 5 and click OK.



\_\_77. Add 2 for Cost and Select Transaction for Cost Modifier



\_\_78. Save the HA-Export Endpoint.

\_\_79. Save and Close all the Endpoint Editor pages by pressing Ctrl+Shift+ W.

## 2.3 Simulate a simple case

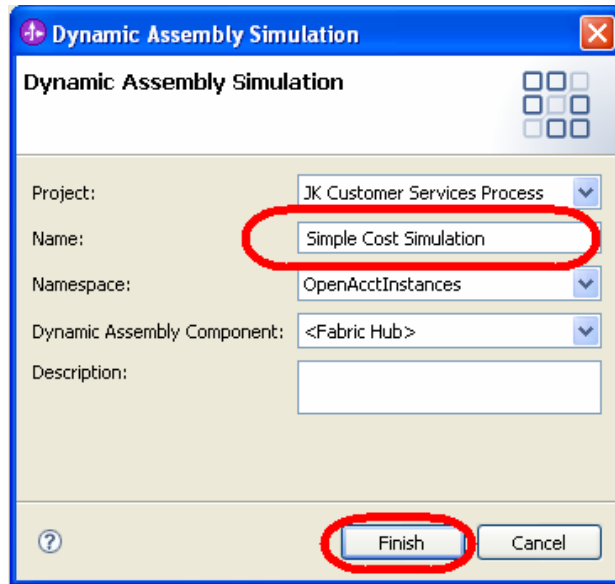
You will run a simulation to observe endpoint selection behavior before defining JK Enterprises business policies for endpoint selection.

This will show one aspect of how WebSphere Business Services Fabric selects endpoints.

\_\_80. Open Policy Simulator to run a simple simulation.

\_\_a. Right click the **Simulation** collection and select **New > Simulation**.

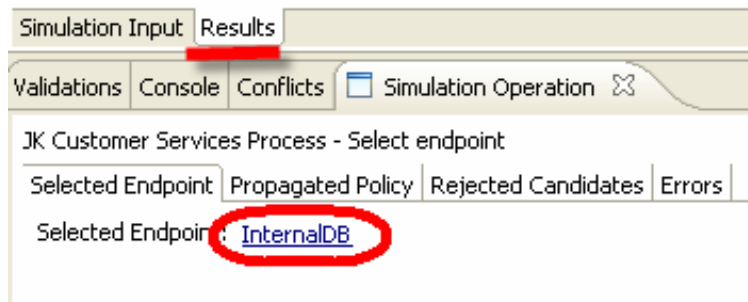
- \_\_b. In the Name field enter **Simple Cost Simulation**, accept the other default values
- \_\_c. Add a useful description, and click **Finish**.



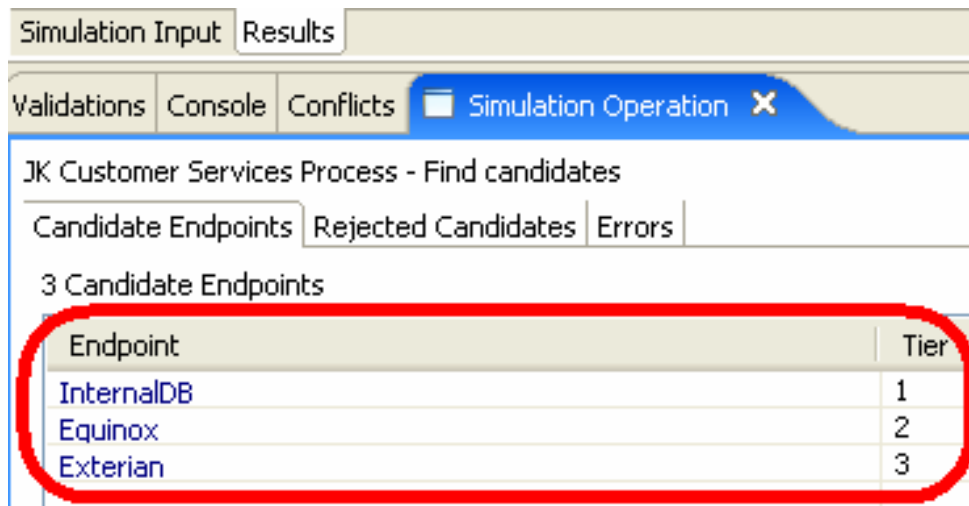
The Business Policy Simulation configuration window will appear.

- \_\_d. In the Required Context area, click **Browse** and select **VerifyCreditInterface**, and click **OK**.
- \_\_e. Click **Run** located in the bottom right corner.
- \_\_81. Analyze the results of this simulation.  
Recall that you have not yet specified any policies.

- \_\_a. The Results tab will become active, indicating the simulation steps taken.
- \_\_b. Below the Results tab, the Simulation Operation tab contains the results of the simulation. The InternalDB credit service was selected.  
The Console tab periodically becomes active and can move in front of the Simulations Operation tab.




- \_\_c. Click **Details** associated with the **Find candidates** operations area of the results tab.  
In the Simulation Operation tab under Candidate Endpoints, Three services were evaluated and ranked.



The Dynamic Assembler chose the InternalDB endpoint. This is because the default criterion used to select a service is cost. InternalDB is the cheapest endpoint available.

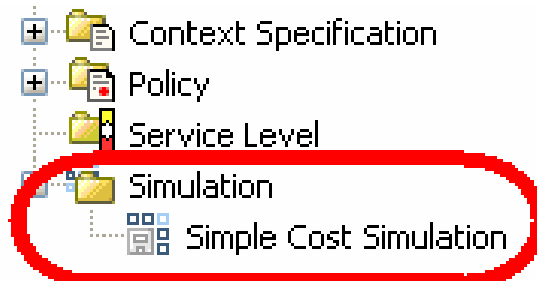
If no business policies are available and available endpoints are essentially equal, lowest cost wins. (You will create some business policies shortly.)

**Optional further investigation**

 As an exercise, vary the transaction costs of the endpoints (overview tab) and re-run this simulation. Return costs to their original values and save them before proceeding.

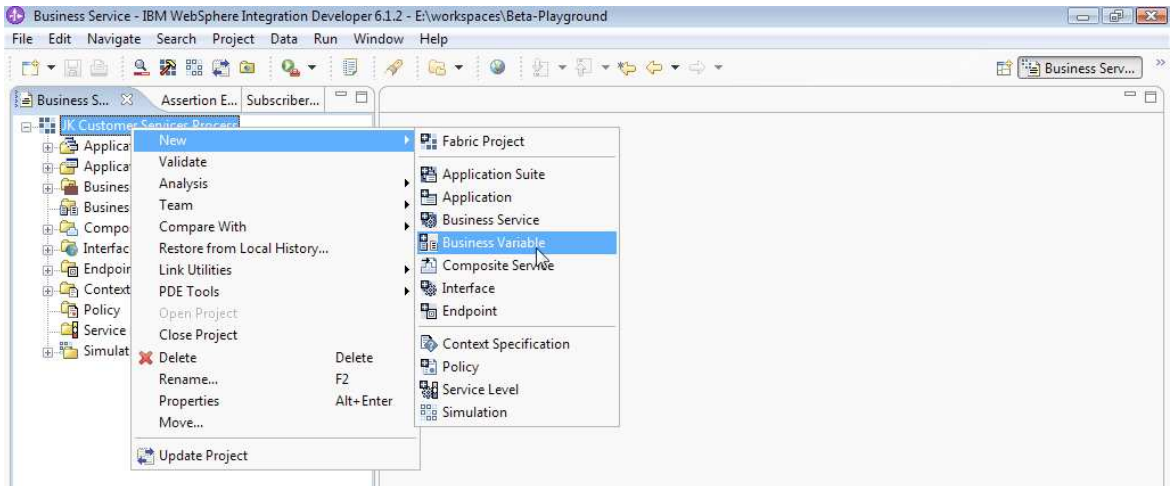
- \_\_d. **Save and close** the **Simple Cost Simulation** window.

Notice that this test case has been saved to the Simulation folder. This collection provides a collection of reusable simulation.

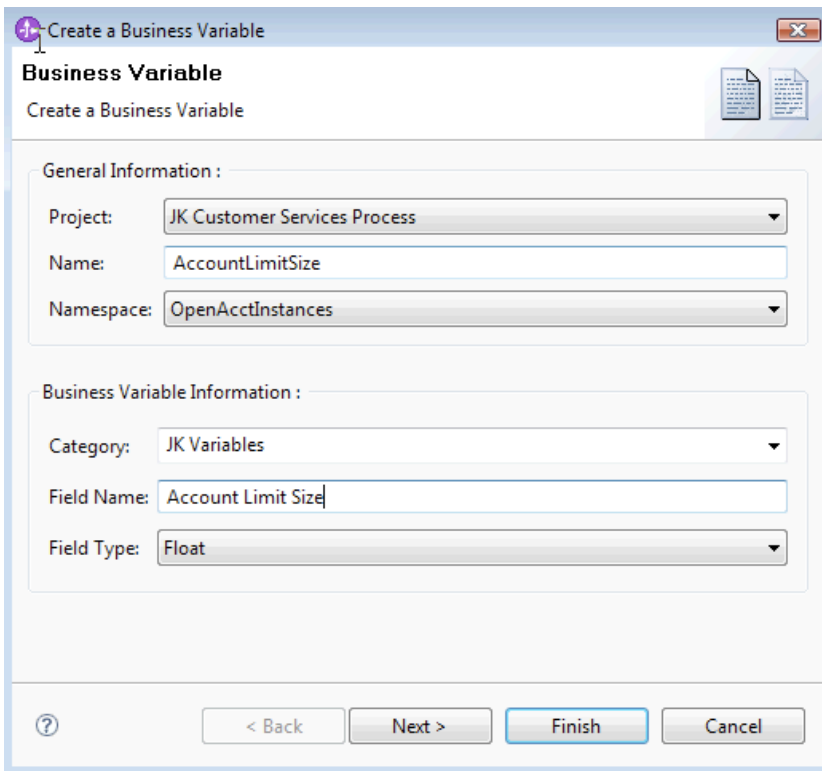


## 2.4 Create Business Space Variable

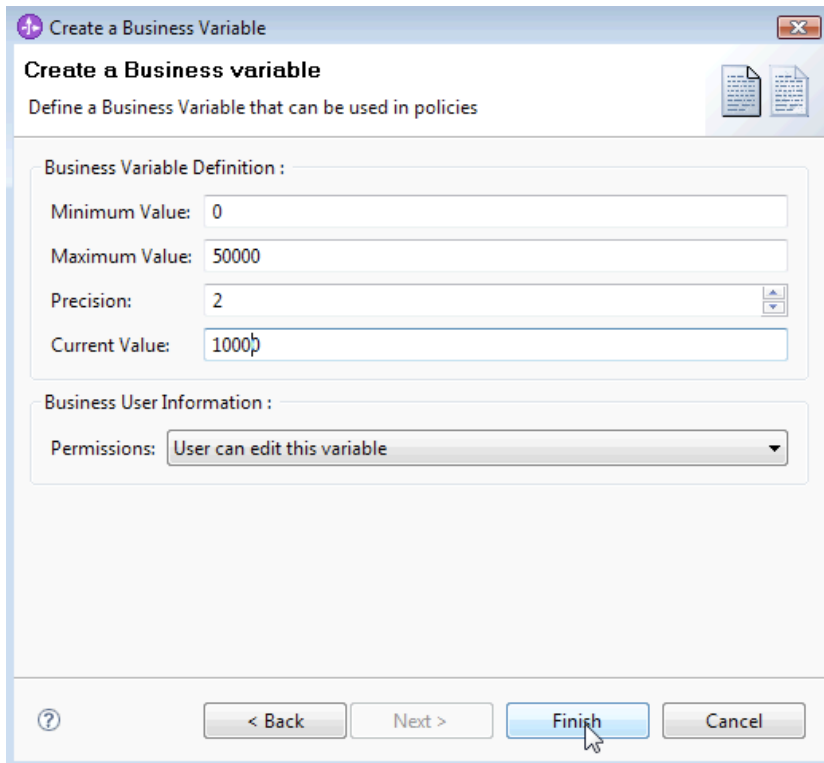
- \_\_82. In the Business Service Perspective, select New Business Variable



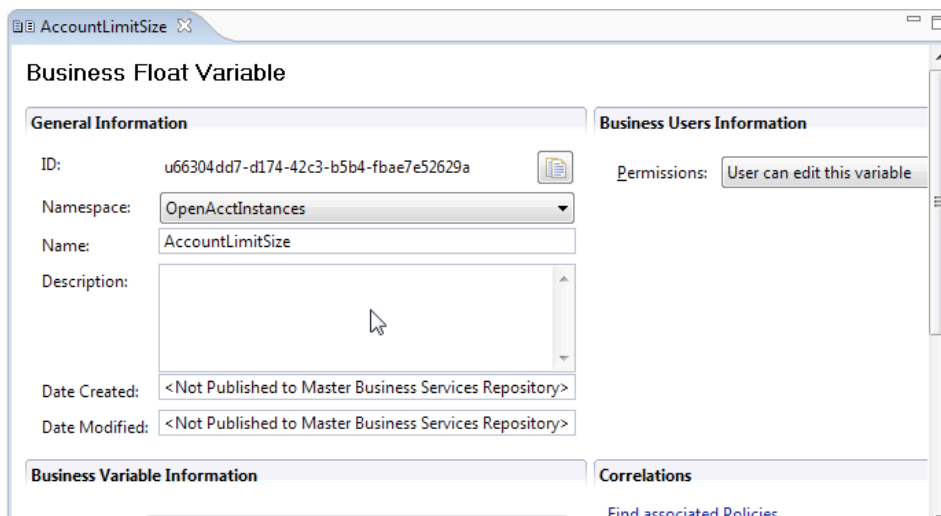
\_\_83. Enter text in each of the fields below and select Next.



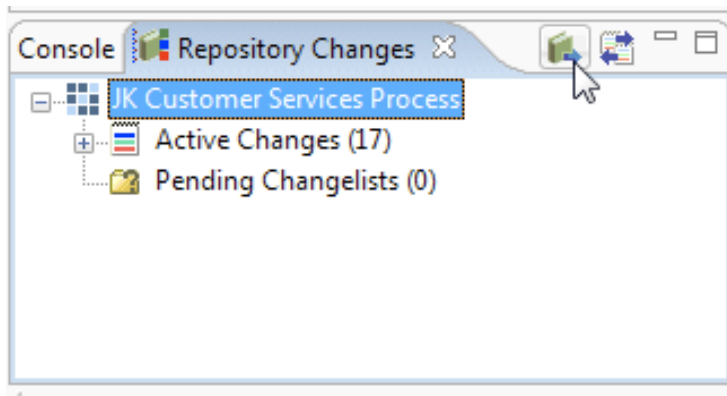
\_\_84. In the next window, finish defining the Business Variable and click Finish



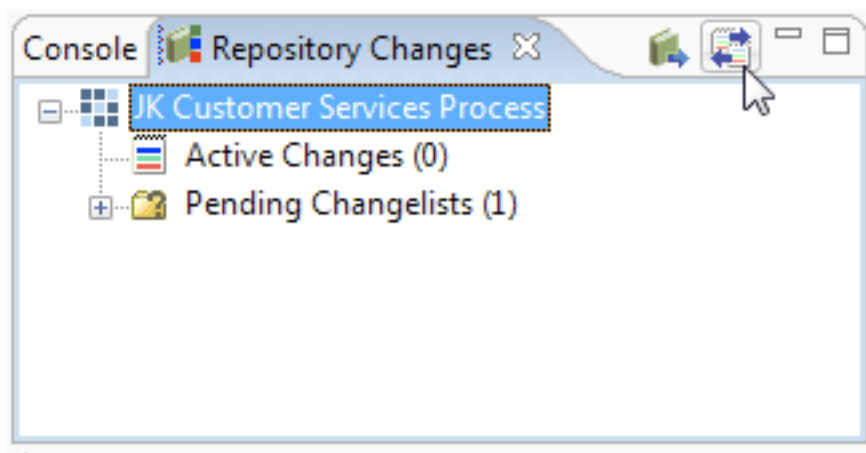
\_\_85. The Business Variable editor window should now appear. Save the project.



\_\_86. In the Repository Changes Tab, click the submit changeset button.

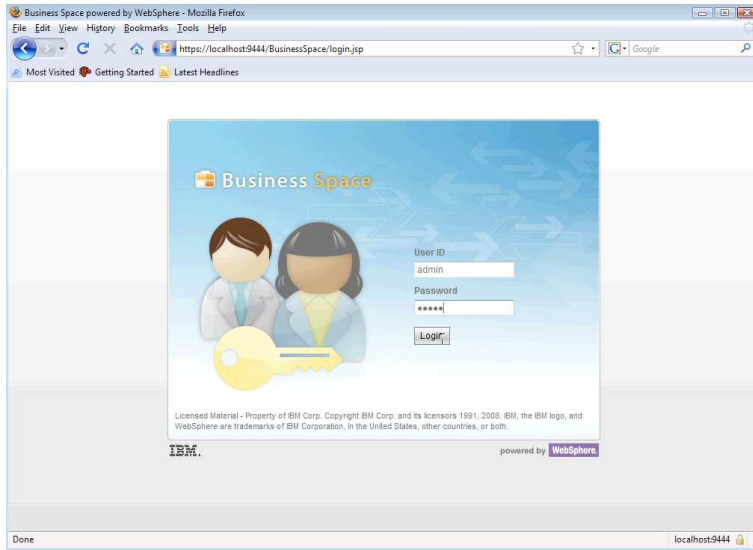


- \_\_87. Select Next, Add All, then Finish. After the changeset has successfully been completed. Click OK.
- \_\_88. Click the Update Project button from the repository Changes Tab.

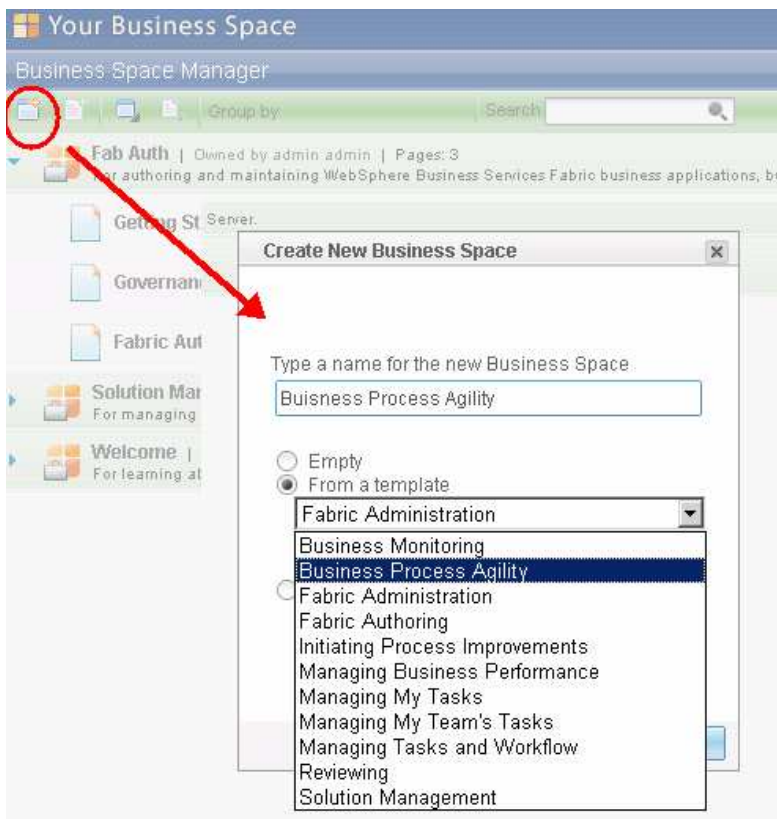


- \_\_89. Click Finish and then OK.
- \_\_90. Open a New Web Browser to <https://localhost:9444/BusinessSpace/banner.jsp> The following webpage should appear. Login as the administrator using admin/admin.





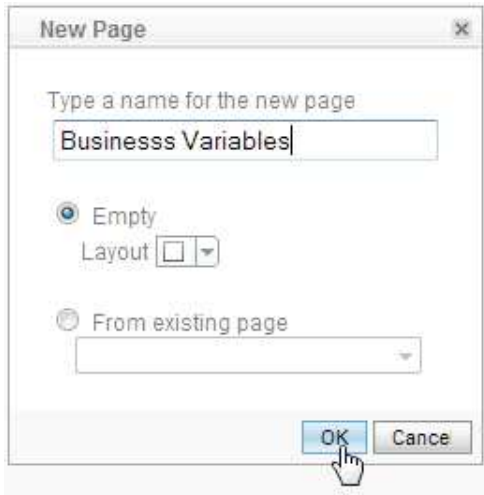
- \_\_91. In the browser select the New Business Space Icon, select **Business Process Agility** template, provide a name and click the **OK** button.



- \_\_92. In the Business Space Browser select **New Page** icon.



\_\_93. A floating window should appear, provide a name and click OK.

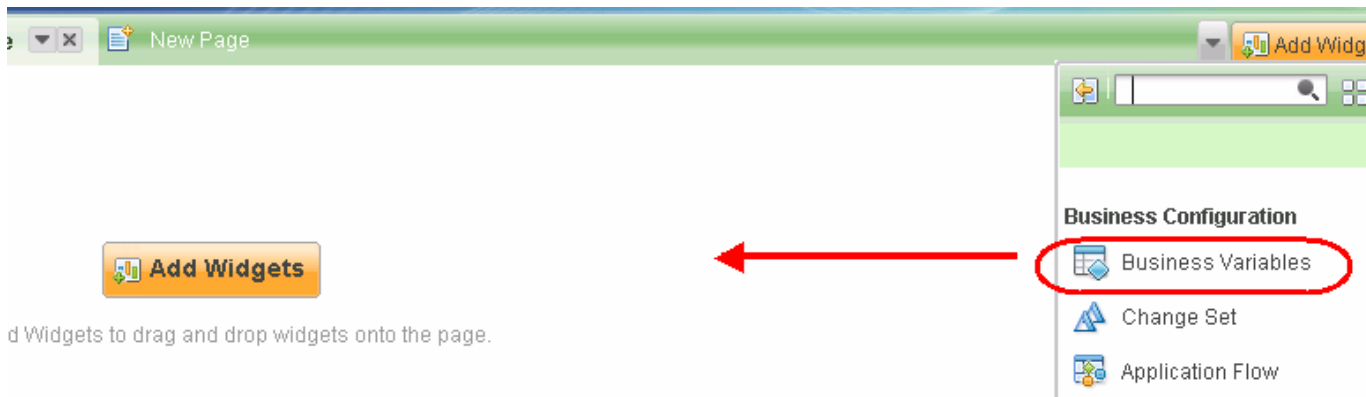


\_\_94. Click the **Add a widget** link



Click Add Widgets to drag and drop widgets onto the page.

\_\_95. Add a **Business Variables Widget** and click OK.



\_\_96. A successful implementation of the Business Space Variable is seen below:

The screenshot displays the 'Business Variable Configuration' page. At the top, there are tabs for 'Business Policy Configuration', 'Business Variable Configuration', 'Governance', and 'Business Variable'. Below the tabs, the 'Business Variables' section is visible. A 'Show: All' dropdown menu is present. The 'JK Variables' section lists 'AccountLimitSize' with an edit icon. To the right, the 'AccountLimitSize' variable is shown with its value 'Account Limit Size: 10000'. At the bottom of the list, there are two input fields, both containing the number '1'.

## 2.5 Create the JK Enterprises policies

JK Enterprises has established business policies for when the different credit services are to be used. These easily modifiable “declarative” policies represent how JK Enterprises wants to do business at this time. The Dynamic Assembly component consults these business policies at run time before selecting a service that meets the service caller’s needs.

In this section you will create and capture this policy information in WebSphere Business Services Fabric.

- \_\_97. Switch back to **WebSphere Integration Developer > Business Services** perspective.
- \_\_98. Review the JK Enterprises Endpoint Assertions and Use Cases.






Recall that:

Customer types requesting credit are EXISTING or NEW.  
Account size (Credit request amounts) are SMALL or LARGE:

Below is a tabular representation of JK Enterprises service provider endpoint capabilities (assertions) and use cases for service consumers.

The top row depicts the service providers available while the left column represents service consumer use cases.

The check marks represent which service provider endpoint is to be selected based on the context, content and contract of the service consumer’s request.

<b>SERVICE PROVIDER</b> <b>SERVICE CONSUMER</b>	<b>InternalDB</b> Cost: \$0.00 Customer Type Supported: EXISTING Account Size Supported: SMALL Availability: Monday through Friday	<b>Equinox</b> Cost: \$1.50 Customer Type Supported: NEW and EXISTING Account Size Supported: SMALL Availability: 24x7	<b>Exterian</b> Cost: \$5.00 Customer Type Supported: NEW and EXISTING Account Size Supported: LARGE Availability: 24x7
Customer = NEW Account Size = SMALL (\$5000) Day of the Week = Monday			
Customer = NEW Account Size = LARGE (\$15000) Day of the Week = Tuesday			
Customer = EXISTING Account Size = SMALL (\$5000) Day of the Week = Tuesday			
Customer = EXISTING Account Size = LARGE (\$25000) Day of the Week = Tuesday			
Customer = EXISTING Account Size = SMALL (\$5000) Day of the Week = Sunday			

### Small account size policy


\_\_99. Create the first of two policies for account size.

The JK Enterprises policy that you are going to create specifies account size based on customer's AccountLimit information contained in the service request. (This information is provided in the request message data set as "content").

This is the policy you will establish.

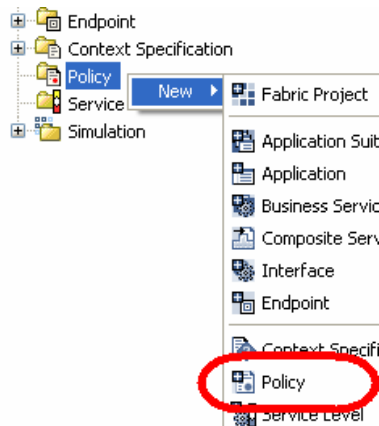
If AccountLimit < \$10,000 (or as redefined by the Business Space Variable) then AccountSize = SMALL

If AccountLimit > or = \$10,000 (or as redefined by the Business Space Variable) then AccountSize = LARGE



**Policy formula**  
 A policy has the formula:  
 If (condition) then (contract)

\_\_a. Right click the **Policy** folder and select **New > Policy**.

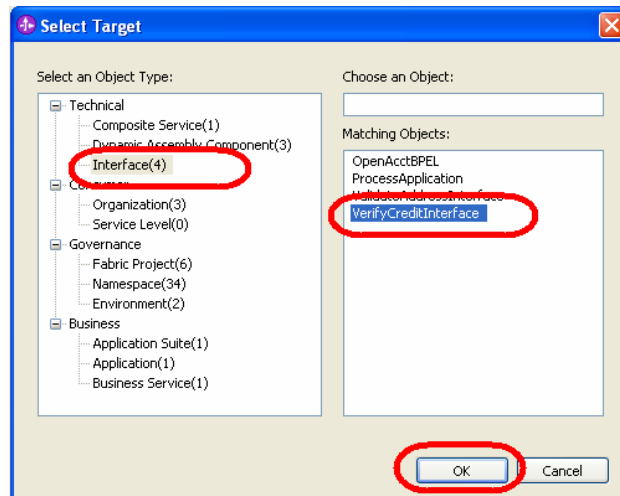


\_\_b. In the Create a Policy wizard, enter **SmallAcctSize** for Name.

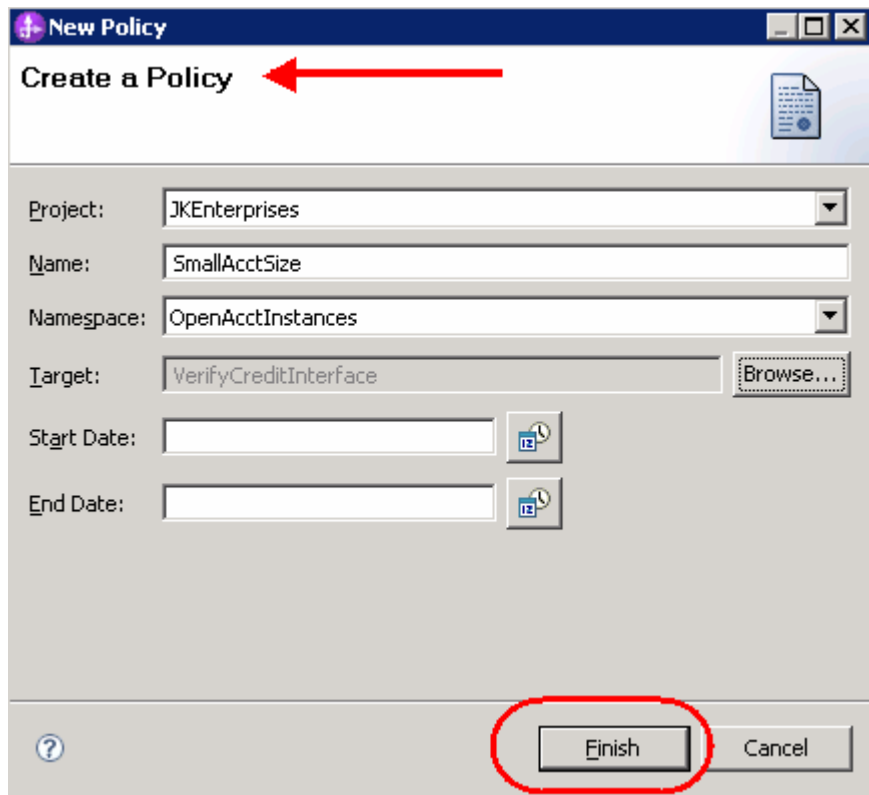
\_\_c. Click **Browse** for the Target field

\_\_d. Click **Interface** under the Technical object collection on the left.


\_\_e. Under Matching Objects, select **VerifyCreditInterface** and click **OK**.



\_\_f. When focus returns to the “**Create a Policy**” window click **Finish**.

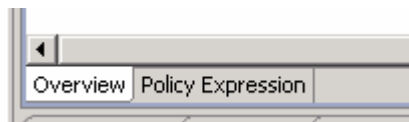


**Policy availability**

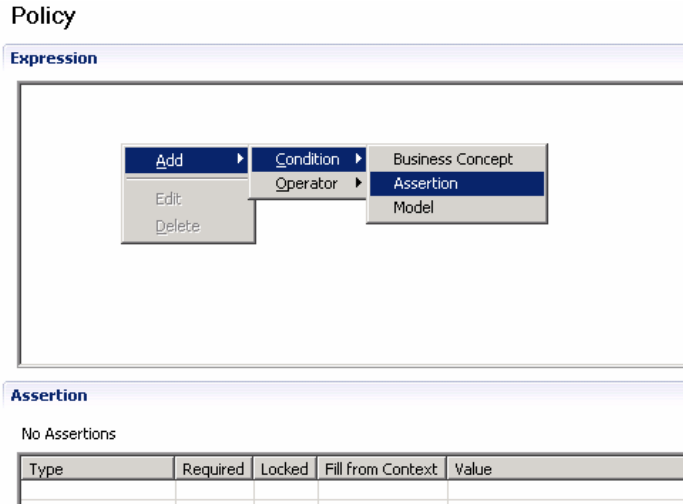


Leaving the Start Date and End Date empty means that this policy is available and in effect as soon as it is applied using governance to the production runtime Business Services Repository and will not expire.

\_\_100. Click **Policy Expression** tab



\_\_101. Right click in the **Expression** box and select the information:



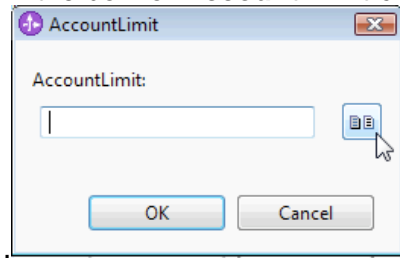
\_\_102. Add the expression :

**Content Type**    **AccountLimit** (select from drop down list)  
**Comparison**    **is less than** (select from drop down list)

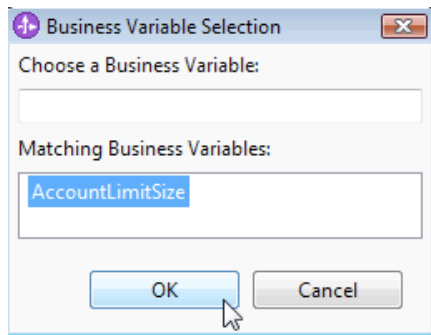
\_\_a. Click **Add Content**.



\_\_b. In the box for **AccountLimit** click the icon to the left of the text field.



\_\_c. Choose the Business Variable that was created earlier and click OK.



This represents the condition or left side of the policy equation.

(condition)                      then                      (contract)

If AccountLimit < set Business Variable      then      AccountSize = **SMALL**

\_\_d. Be sure that you selected AccountLimit and not AccountSize. Click **OK**.

Account limit is the information about customer account that is used to help determine what type of credit report is appropriate for them.

\_\_e. Click **OK** three times.

\_\_f. Click **Finish** to close the Policy Content window.

\_\_g. In Assertion table click **Add**.



## Policy

**Expression**

```
.... [AccountLimit] < {AccountLimitSize}
```

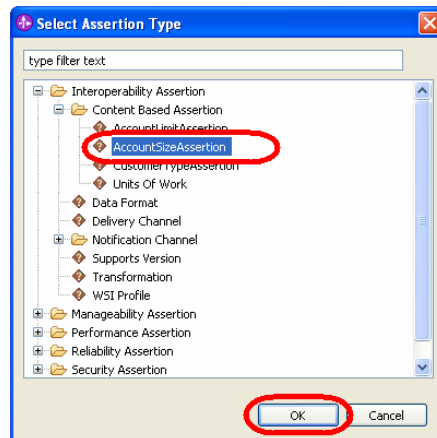
**Assertion** ←

No Assertions

Type	Required	Locked	Fill from Context	Value	
					<b>Add...</b>
					Edit...
					Remove

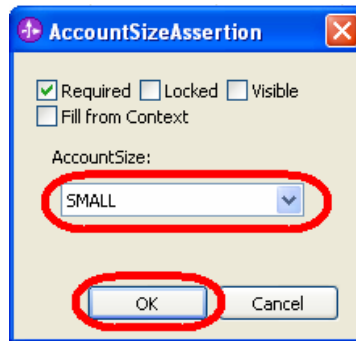
\_\_h. Expand the **Interoperability Assertion** folder and under it, expand the **Content Based Assertion** folder.

\_\_i. Select **AccountSizeAssertion** and click **OK**.



A window for specifying the AccountSizeAssertion contract has opened.

\_\_j. Select **SMALL** and click **OK**.



SMALL and LARGE are extensions available for this (and possibly other) business service models. Recall the discussion earlier when assigning assertions (capabilities) to your endpoints. You are now assigning assertions to this policy. (Policy Assertions)

\_\_k. Verify that a row has been added to the Policy Assertions table.

Assertion						
1 Assertion						
Type	Required	Locked	Fill from Context	Value		
AccountS...sassertion	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SMALL		

\_\_l. **Save** the policy you have developed then **close** this editor.

A policy has been added to the collection under Policy in the tree view.

### Large account size policy

\_\_103. Create a policy for customers requesting LARGE loans.

The policy you are developing next is as follows:

If AccountLimit > or = to \$10,000 then AccountSize = **LARGE**

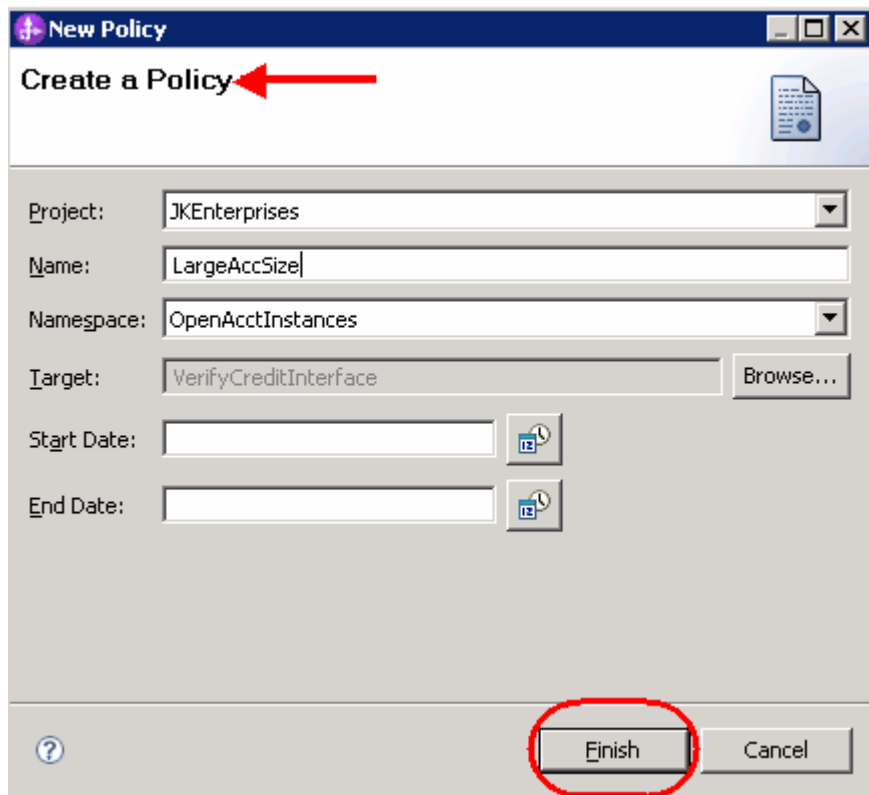
\_\_a. Right click the **Policy** folder and select **New > Policy**.

\_\_b. In the Create a Policy window, enter **LargeAcctSize** as the policy name and click **Browse**.

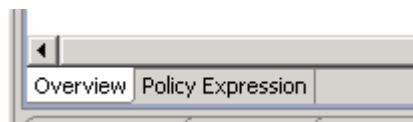
\_\_c. Under Technical object, click **Interface**.

\_\_d. Select **VerifyCreditInterface** and click **OK**.

\_\_e. Click **Finish** in the Create a Policy window.

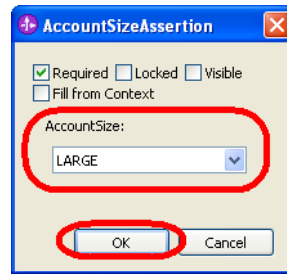


\_\_f. Click Policy Expression tab





- \_\_o. The AccountSizeAssertion window will open. Select **LARGE** for AccountSize and then click **OK**.

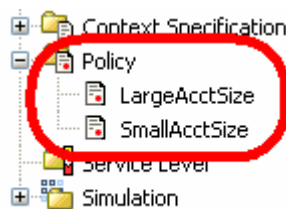


- \_\_p. The final result will look like:

Assertion				
1 Assertion				
Type	Required	Locked	Fill from Context	Value
AccountSizeAssertion	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	LARGE

- \_\_q. **Save** the policy you have developed and **close** its editor.

There are now two policies, LargeAcctSize and SmallAcctSize, in the Policy folder.



## Customer type policy

- \_\_104. The policy that you will create next will specify that a customer type is required by the Dynamic Assembler - NEW or EXISTING.

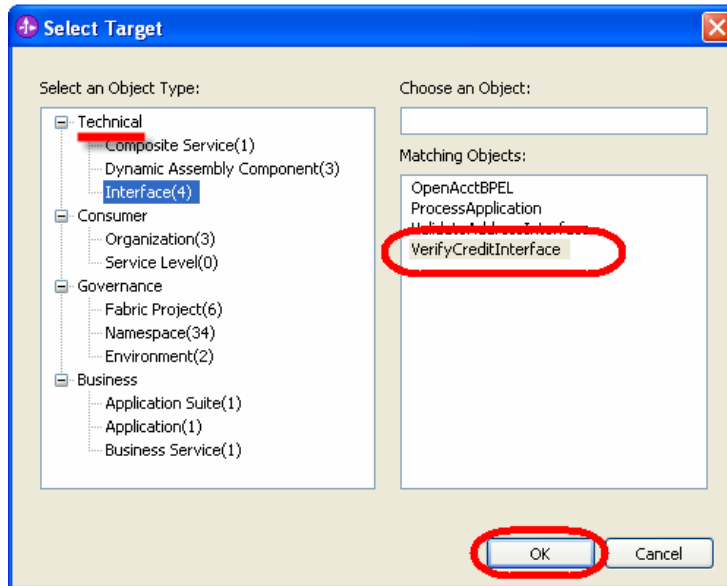
The service caller provides a set of data when making a service call. One of the data fields contains customer type information. Only the values NEW or EXISTING are allowed. There is no need to define a specific policy assertion contract. It is provided by the caller's message data.

The endpoints are decorated with information about the types of customers that they support. InternalDB only supports EXISTING customers. The other credit services support both NEW and EXISTING customers.

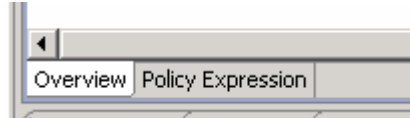
- \_\_a. In the tree view to the left, right-click the **Policy** folder and select **New > Policy**.

- \_\_b. In the Create a Policy window, enter **CustomerType** as the policy name and click **Browse**.

- \_\_c. Expand the **Technical** collection and click **Interface**. Then, select **VerifyCreditInterface** and click **OK**.




- \_\_d. When focus returns to the “Create a Policy” click **Finish**.
- \_\_e. Click Policy Expression tab



- \_\_f. Right click in the **Expression** box and add **OR** condition

**Prefix Notation**

 The expression box in the policy editor adds policies in prefix notation. There fore you first define the operator – **OR** and then add the left and right side of the operator – **Customer Type = NEW OR Customer Type = EXISTING**

## Policy

### Expression

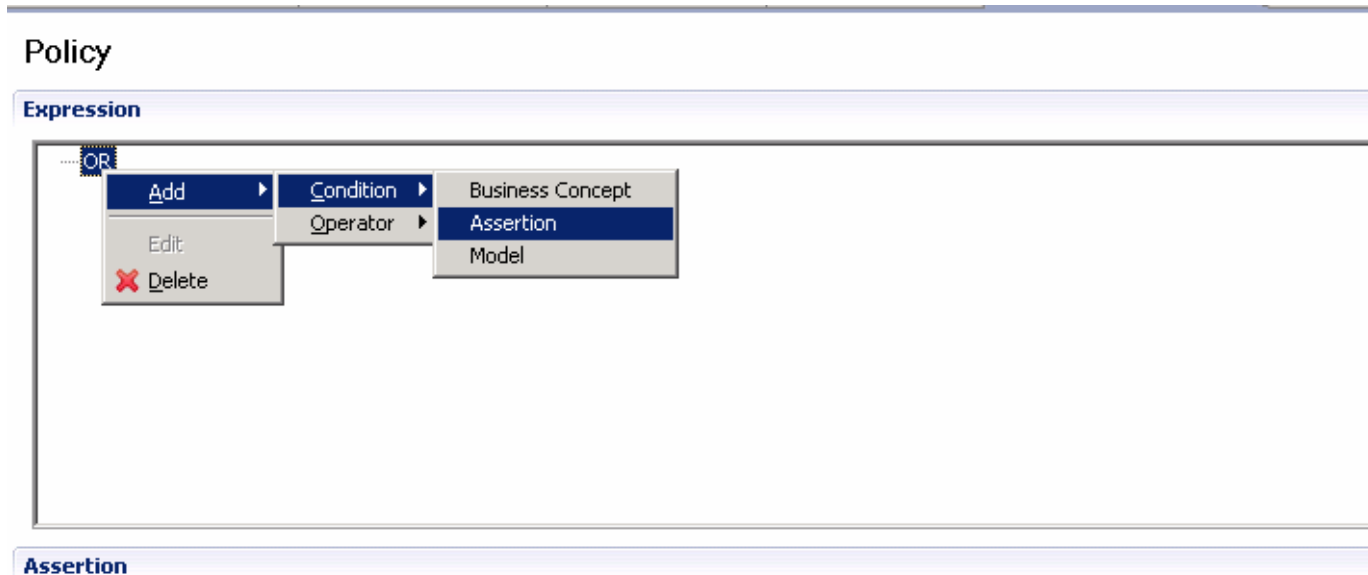
The image shows a graphical user interface for building an expression. It features a context menu with three main options: 'Add', 'Edit', and 'Delete'. The 'Add' option is expanded to show two sub-options: 'Condition' and 'Operator'. The 'Operator' option is further expanded to show three logical operators: 'NOT', 'OR', and 'AND'. The 'OR' operator is currently selected.

### Assertion

No Assertions

Type	Required	Locked	Fill from Context	Value

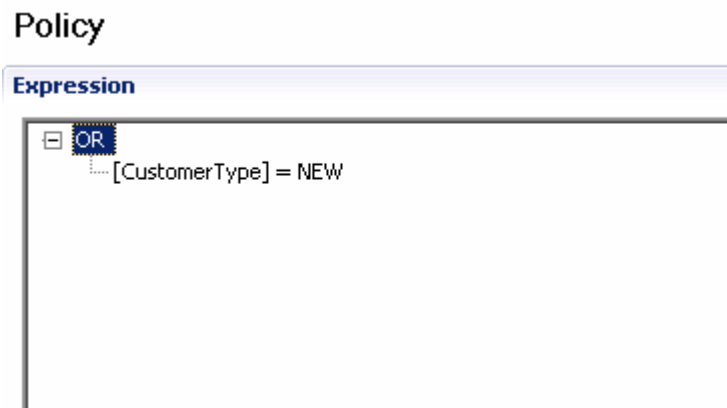
- \_\_g. Next you will assertions by right clicking the **OR** operator in the **Expression** box and selecting **Assertion**.



- \_\_h. Specify the assertion information.

<b>Content Type</b>	<b>CustomerType</b> (select from drop down list)
<b>Comparison</b>	<b>is equal to</b> (select from drop down list)
<b>Add Content</b>	<b>NEW</b> (click Add Content and select )

- \_\_i. Click **OK** and click **OK** again.  
 \_\_j. Verify that you have generated:



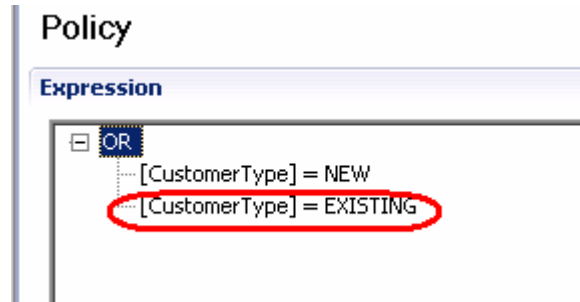
- \_\_k. Similarly add the second assertion condition to the OR operator. Specify the information stated below



<b>Content Type</b>	<b>CustomerType</b> (select from drop down list)
<b>Comparison</b>	<b>is equal to</b> (select from drop down list)
<b>Add Content</b>	<b>EXISTING</b> (select from drop down list)

\_\_l. Click **OK** and click **OK** again.

\_\_m. Verify that you have generated the information below :



\_\_n. In the Assertion table below click **Add**.

**Policy**

**Expression**

OR

- [CustomerType] = NEW
- [CustomerType] = EXISTING

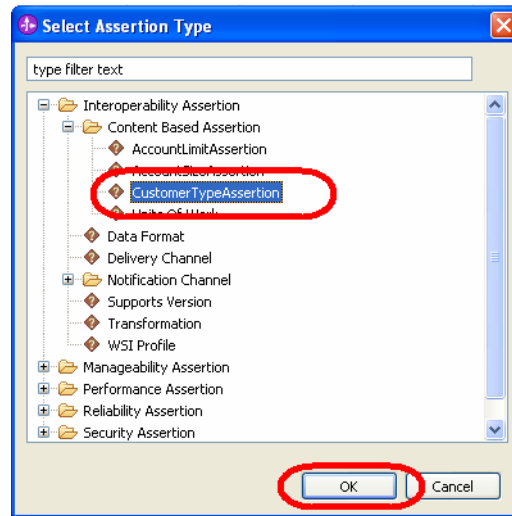
**Assertion**

No Assertions

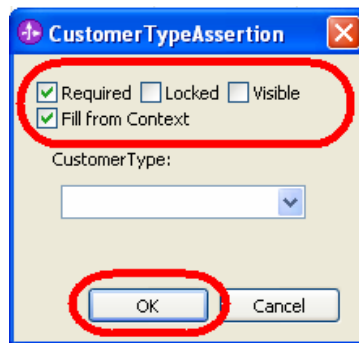
Type	Required	Locked	Fill from Context	Value	Add...
					Edit...
					Remove

\_\_o. Expand **Interoperability Assertion** and **Content Based Assertion** folders.

\_\_p. Select **CustomerTypeAssertion** and then click **OK**.



\_\_q. In the CustomerTypeAssertion window that opened, check the selection box for **Fill from Context** and verify that **Required** is checked as well. Click **OK**.



\_\_r. The final result will look like :

Type	Required	Locked	Fill from Context	Value
CustomerTypeAssertion	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

\_\_s. **Save** and **close** this policy editor to add this third policy to Policy collection.

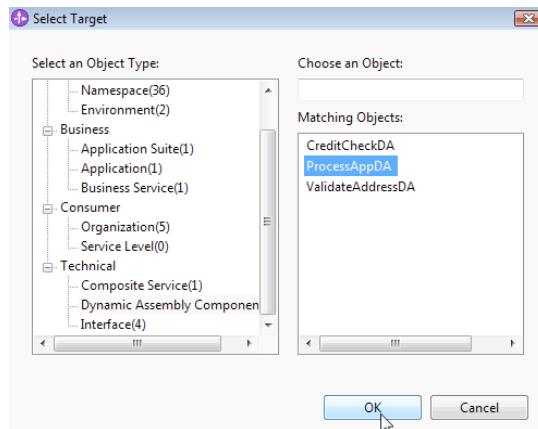
\_\_t. **Save all** open windows with **Ctrl+Shift+S**

\_\_u. **Close all** open windows **Ctrl+Shift+W**

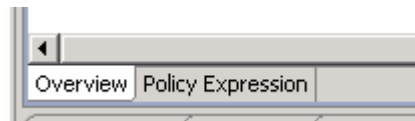
## Policy to trigger the Human Task (Asynchronous Capability)

\_\_105. The policy that you will create next will trigger the Human Task that you added to the assembly diagram in a previous lab. Basically, any loan request that is greater than \$20,000 will require human approval before selecting the appropriate endpoint.

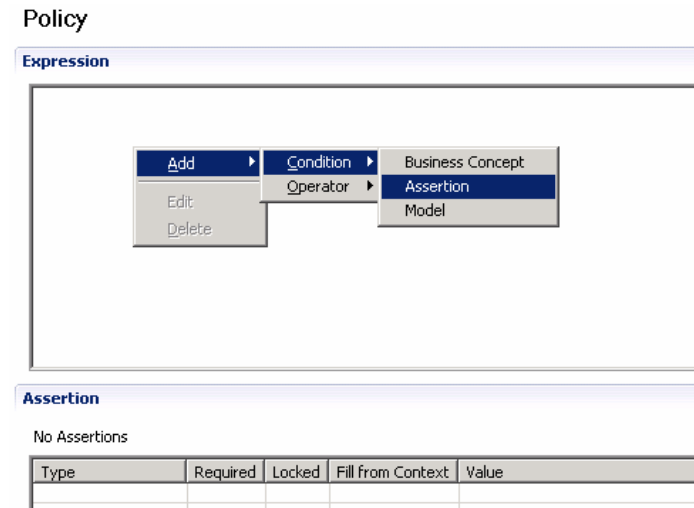
- \_\_a. In the tree view to the left, right-click the **Policy** folder and select **New > Policy**.
- \_\_b. In the Create a Policy window, enter **AccountLimit** as the policy name and click **Browse**.
- \_\_c. A window should appear and select the ProcessAppDA and click OK.



- \_\_d. When your focus is back on Create Policy window click **Finish**
- \_\_e. Click Policy Expression tab



\_\_f. Right click in the **Expression** box, select this:



\_\_g. Specify this information using the steps described earlier.

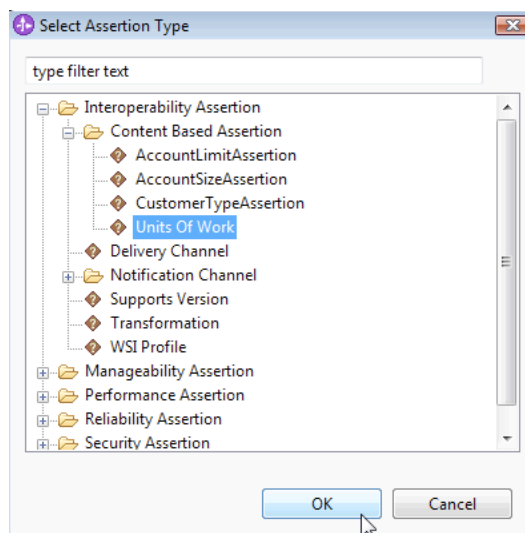
<b>Content Type</b>	<b>AccountLimit</b> (select from drop down list)
<b>Comparison</b>	<b>is greater than</b> (select from drop down list)
<b>Add Content</b>	<b>20000</b>

\_\_h. Be sure that you selected AccountLimit and not AccountSize. Click **OK**.

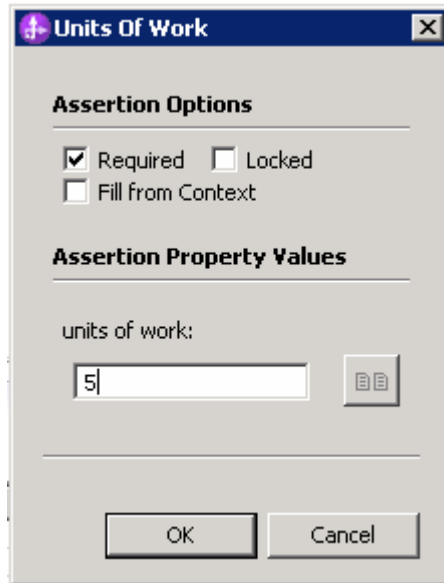
\_\_i. Click **OK** again (three times).

\_\_j. In Assertion table below click **Add**

\_\_k. Select the **Units of Work** assertion and click **OK**.



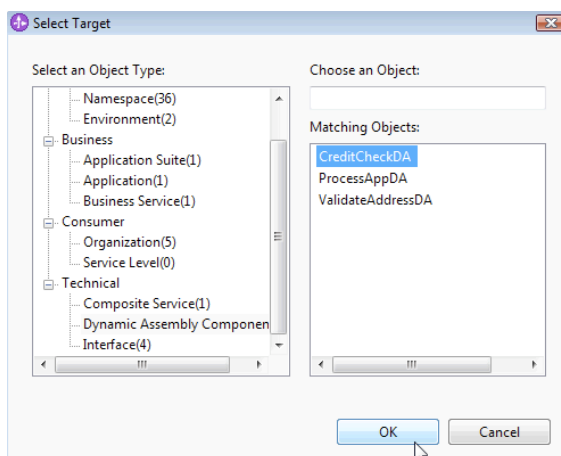
- \_\_l. In the window, leave the **Required** checked and type 5 in the text field and click **OK**.



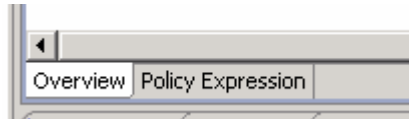
- \_\_m. Save and close the editor.

## Monitoring policy

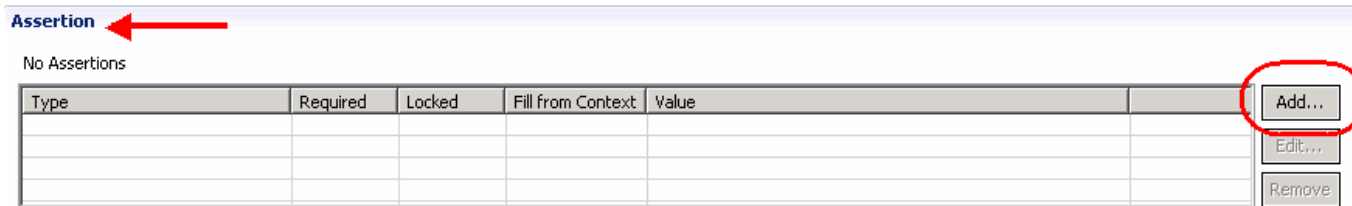
- \_\_106. The policy that you will create next is needed for monitoring Fabric events
  - \_\_a. In the tree view to the left, right-click the **Policy** folder and select **New > Policy**.
  - \_\_b. In the Create a Policy window, enter **FabricEventsPolicy** as the policy name and click **Browse**.
  - \_\_c. Select the **CreditCheckDA** and click **OK**.



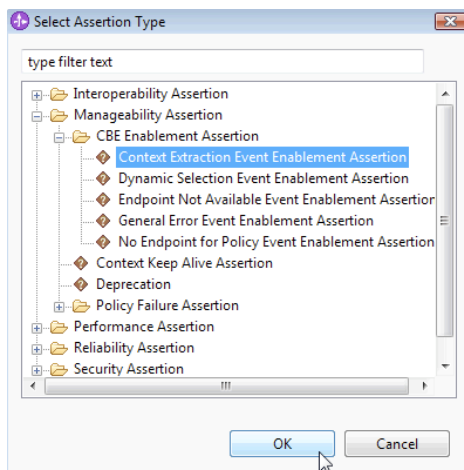
- \_\_d. Then click **Finish**.
- \_\_e. Click Policy Expression tab



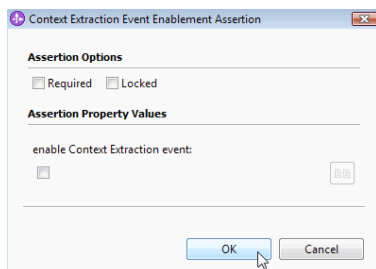
\_\_f. In the Assertion table click **Add**



\_\_g. Expand **Manageability Assertion** -> **CBE Enablement Assertion**. Select the **Context Extraction Assertion** and click **OK**.



\_\_h. Remove the required checkmark, **click the enable Context Extraction Event**, and click **OK**.



\_\_i. Click the Add button and repeat the previous three steps for the remaining four assertions:

1	Dynamic Selection Event
2	Endpoint Not Available Event Enablement
3	No Endpoint for Policy Event Enablement
4	Technical Error Event Enablement

\_\_j. The final assertion table should be similar to :

Assertion				
5 Assertions				
Type	Required	Locked	Fill from Context	Value
Dynamic Selecti...ement Assertion	<input type="checkbox"/>	<input type="checkbox"/>		True
Endpoint Not A...ment Assertion	<input type="checkbox"/>	<input type="checkbox"/>		True
No Endpoint for...ement Assertion	<input type="checkbox"/>	<input type="checkbox"/>		True
Technical Error ...lement Assertion	<input type="checkbox"/>	<input type="checkbox"/>		True

\_\_k. Save and close the policy editor window.

## 2.6 Simulate JK Enterprises policy usage

The simulation capabilities of the Composition Studio perspective of WebSphere Integration Developer provide a facility for testing endpoint selection scenarios. Detailed results are available for each step in Dynamic Assembler's process of determining which service is appropriate. This simulation allows developers to test, troubleshoot, and modify or author policies to generate the necessary results for the business.

Endpoint Selection consists of following three major steps or operations:

**Build Selection Policy:** Find all policies applicable for the current context and composes them into a merged selection policy

**Find Candidates:** Filter endpoint candidates that do no match the requirements for the selection (rejected candidates) and rank candidates that meet the requirements

**Select Endpoint:** Pick the best endpoint from the candidates base their ranking

Previously, you ran a simple simulation without policies. That simulation always resulted in the selection of the least expensive endpoint. Because there were no business policies, lowest cost was provisioned.

In this section you will simulate and test the behavior of the Dynamic Assembler as it selects the appropriate endpoint service using policies you previously defined.

## Context Specification

It is a best practice to create a custom Context Specification for implemented Dynamic Assembly components. This context specification component formalizes the contract between the service consumer and the Dynamic Assembler.

There are two default Context Specifications provided. They are:

- ◆ **Fabric Hub Context:** Supports earlier versions with the deprecated hub deployment model. The only required dimension is the service interface.
- ◆ **SCA Default Context:** Default context specification used with SCA-style Dynamic Assembly invocations. The required dimensions are Composite Service, Dynamic Assembler Component, and Interface – all of which are provided automatically when using the SCA component for Dynamic Assembly.

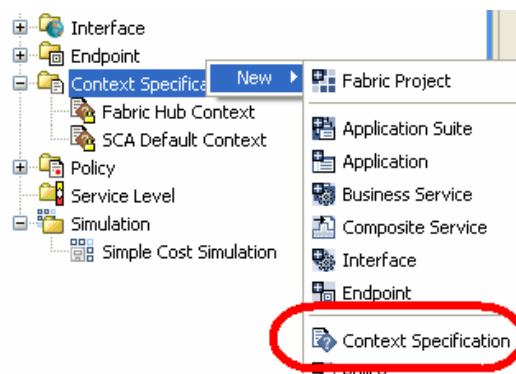
When you built and ran your Simple Cost Simulation, only the interface was provided for you. You did not select and specify values in the Optional Context portion of the Simulation Input parameter view.

Besides strongly formalizing the contract between service consumer and the Dynamic Assembler, another benefit of utilizing context specifications is to provide fine-grained control during simulation and runtime.

Creating a simulation for that Dynamic Assembler component will automatically add your required context dimensions to the simulation configuration. One only needs to specify the values before running simulations.

\_\_107. Create a Context Specification to be used for the simulations you will run shortly.

\_\_a. Right click the **Context Specification** folder and select **New > Context Specification**.



\_\_b. Enter **VerifyCreditContextSpec** for the name and click **Finish**.

\_\_c. In the configuration window that opens, click its **Dimensions** tab at the bottom.

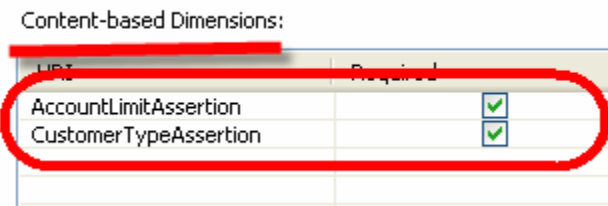
\_\_d. Click **Add** associated with the **Content-based Dimensions** table.

\_\_e. Select **AccoutLimitAssertion** and click **OK**.

\_\_f. Click **Add** again, select **CustomerTypeAssertion**, and click **OK**.



- \_\_\_g. Verify that your **Content-based Dimensions** are now configuration as shown in the image.

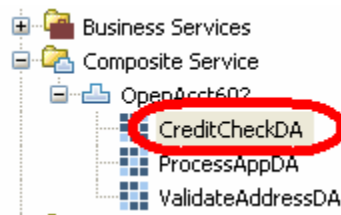


- \_\_\_h. **Save** and **close** this editor.

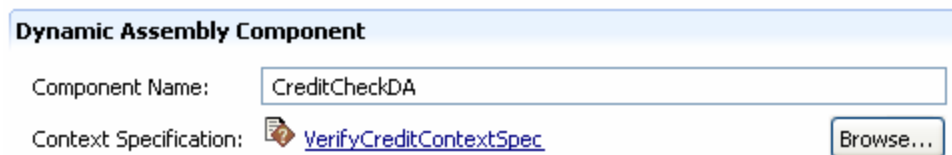
The Context Specification folder now contains your custom specification.

- \_\_\_108. Apply this context specification to the credit report Dynamic Assembler component.

- \_\_\_a. Expand the **Composite Service** folder and then expand the **OpenAcct602** process module's collection of Dynamic Assembler components.
- \_\_\_b. Double click **CreditCheckDA** to open its configuration editor.



- \_\_\_c. Select context specification associated with this Dynamic Assembler Component by clicking on **Browse**
- \_\_\_d. Select **VerifyCreditContextSpec** and click **OK**.
- \_\_\_e. Verify that you have the configuration shown below :



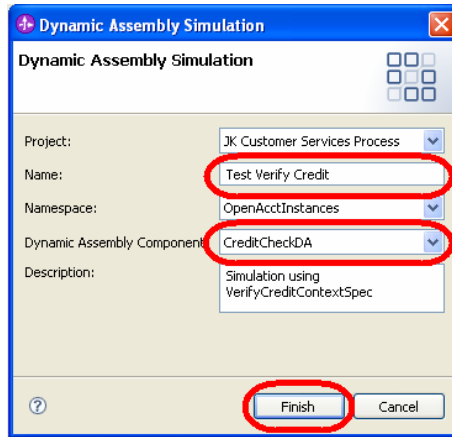
- \_\_\_f. **Save** and **close** this editor.

### Simulate InternalDB

- \_\_\_109. Simulate a JK Enterprises use case that should select InternalDB.

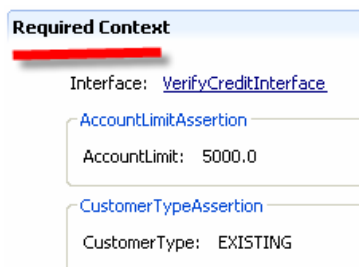
- \_\_\_a. Right click the **Simulation** collection and select **New > Simulation**.

- \_\_b. In the Name field enter **Test Verify Credit**.
- \_\_c. Select **CreditCheckDA** for the Dynamic Assembly Component field.
- \_\_d. Accept the other default values and click **Finish**.



The Simulation configuration window will open.

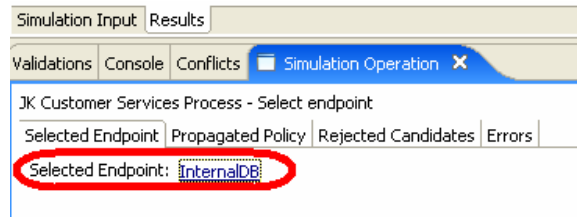
- \_\_e. Select AccountLimitType and CustomerType by clicking **Add Content Based Dimensions**
  - \_\_f. Click **Edit** for AccountLimit, enter **5000**, and then click **OK**.
  - \_\_g. Click **Edit** for CustomerType; select **EXISTING** from the dropdown box, and click **OK**.
- The results should look like :



- \_\_h. Click **Run** located in the bottom right corner.

\_\_110. Analyze the simulation results

- \_\_a. View the results using the **Simulation Operation** tab located below the simulation window.



InternalDB was selected as expected.

The Dynamic Assembler selected this endpoint because the account size for the credit request was SMALL (<10000) and customer was EXISTING.

This behavior adheres to how JK Enterprises' business policies "declare" how it wants to use various credit reporting services today.

**Endpoint tier ranking**

- \_\_111. Review the steps that the Dynamic Assembler took as it determined that InternalDB was the correct credit service for this service consumer request simulation.

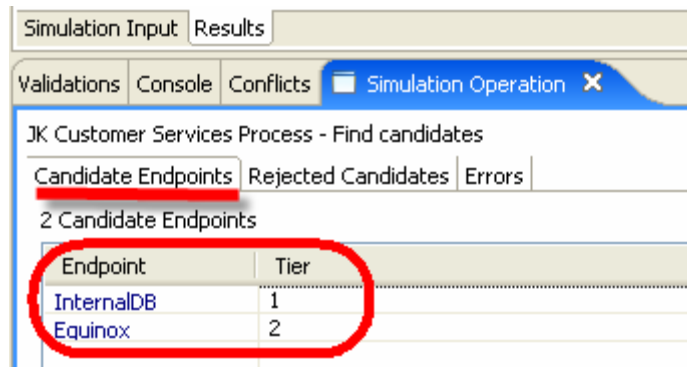
The green checks represent these steps.

- \_\_a. In the Results tab for this simulation, click the **Details** hyperlinks for:

- Build selection policy
- Find candidates
- Select endpoints

- \_\_b. Click the **Details** link for **Find candidates**.

Notice that two endpoints meet the specified requirements. They are shown under the Simulation Operation tab under the Candidate Endpoints tab.




When endpoint candidates are selected during the “Find candidates” step, they are placed into tiers. Each of the usingble candidates identified must meet all of the requirements specified in the merged selection policy. They also have to match any required endpoint specified assertions.

These candidates are then assigned a tier number or ranking based the normalized (per transaction) cost of the endpoint and the number of matched optional assertions.

The tier number is an integer number. The lowest number is the best ranking. Cost takes precedence only when all required and optional policy and endpoint assertions are equally met. Essentially, the candidate endpoints are considered equal.

A lower cost endpoint with fewer matched optional assertions is ranked more highly (less favorably) than a higher cost endpoint that matches more optional assertions.

All usingble candidates that have the same cost AND the same number of optional assertions matched are placed into the same tier. If more than one suitable endpoint is ranked in the best tier (lowest number), the Dynamic Assembler will alternately pick among these endpoints in a round-robin fashion. This distributes the load across essentially equivalent endpoints.



**No round-robin during simulation**

A round-robin between suitable endpoints is only applied to the server version of the Dynamic Assembler.

When using the Dynamic Assembler Simulator, a single endpoint is repeatedly selected from among equivalently ranked endpoints based on its creation timestamp.

### Simulate Exterian

\_\_112. Simulate a JK Enterprises use case that should select Exterian as the result.

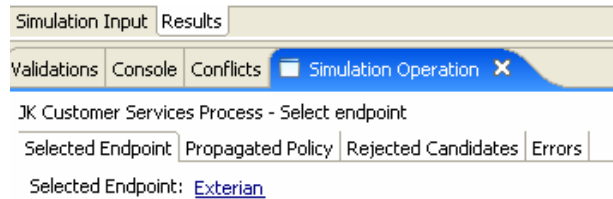
You will use the values:

<b>Customer</b>	<b>NEW</b>
<b>Account Size</b>	<b>15000 (LARGE)</b>

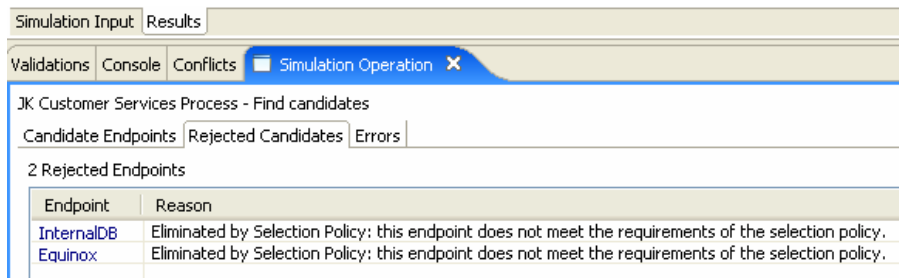
- \_\_a. With the Test Verify Credit simulation window still open, click the **Simulation Input** tab.
- \_\_b. Click **Edit** for AccountLimit, enter **15000**, and then click **OK**.
- \_\_c. Click **Edit** for CustomerType, select **NEW** from the dropdown box, and click **OK**.
- \_\_d. Click **Run**.

\_\_113. Analyze the simulation results

As expected, the selected endpoint is now Exterian.



- \_\_a. In the **Results** tab, click the **Details** hyperlink for **Find candidates**.
- \_\_b. Click the **Rejected Candidates** tab under the **Simulation Operation** tab.
- \_\_c. Two credit services, InternalDB and Equinox were rejected because they did not meet the selection policy's requirements.



**Simulate Equinox**

\_\_114. Simulate a JK Enterprises use case that should select Equinox.

Use the values:

<b>Customer</b>	<b>NEW</b>
<b>Account Size</b>	<b>5000 (SMALL)</b>

- \_\_a. Click the **Simulation Input** tab.
  - \_\_b. Click **Edit** for AccountLimit, enter **5000**, and then click **OK**.
  - \_\_c. Verify that CustomerType is **NEW**.
  - \_\_d. Click **Run**.
- \_\_115. Analyze the simulation results
- As expected, the selected endpoint is now Equinox.
- \_\_a. **Save** and **Close** the simulation window.

You created a custom context specification and policies. Then, you simulated the behavior of the business services model you defined. .



## Part 3 Deploy and run the composite business service

### 3.1 Local development change set submission

\_\_116. Submit your local change set to update the Business Services Repository.

A change set is a collection of related meta-data changes that an individual developer has made in their localized project.



#### Local and remote repositories

WebSphere Integration Developer with its Business Integration and Composition Studio perspectives contains a Unit Test Environment. It includes a fully functional WebSphere Process Server for stand-alone development and testing.

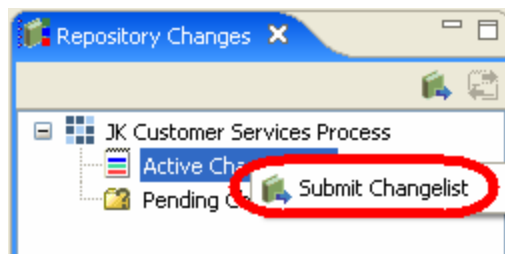
During development and simulation of Composite Business Services, a local eclipse repository is consulted.

When running an actual process rather than simulating it, WebSphere Business Services Fabric's Dynamic Assembler consults an actual Business Services Repository.

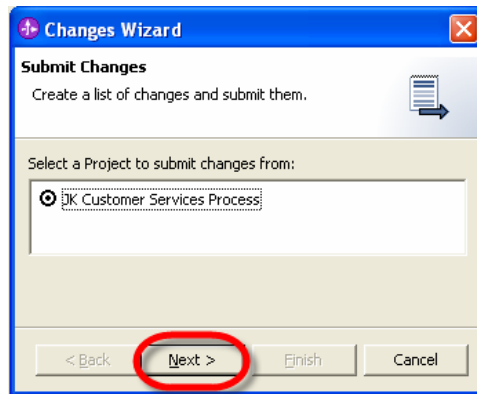
Before running actual process executions, you first need to update that runtime repository.

\_\_a. Locate the Repository Changes view in the bottom left of this eclipse environment.

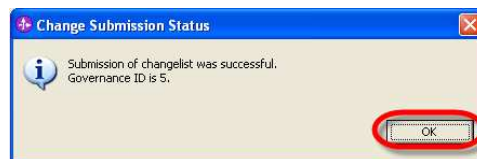
\_\_b. Right click **Active Changes** and click **Submit Changelist**.



\_\_c. Click **Next** in the Changes Wizard.

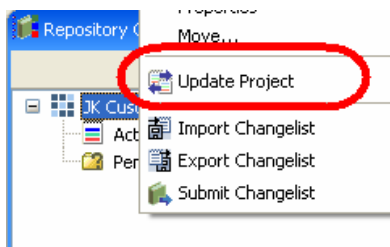


- \_\_\_d. Click **Add All >>** to move all of your development changes to the Selected Changes window and then click **Finish**.
- \_\_\_e. In the Confirm Submit to Business Services Repository window, click **Yes**.
- \_\_\_f. In the Change Submission Status window click **OK**.



Notice that there is now an entry in the "Pending Changesets" folder.

- \_\_\_g. At the root of the tree, right-click **JK Customer Services Process** and select **Update Project**.



- \_\_\_h. In the Update Project dialog box, select **Finish**.
- \_\_\_i. In the Project Update Results dialog box, select **OK**.

This process has moved your changes to the runtime Business Services Repository. WebSphere Process Server will now have access to this metadata during runtime. Additionally, you also resynchronized your local Composition Studio environment.

In this Unit Test Environment, governance approval of change sets is automatic.

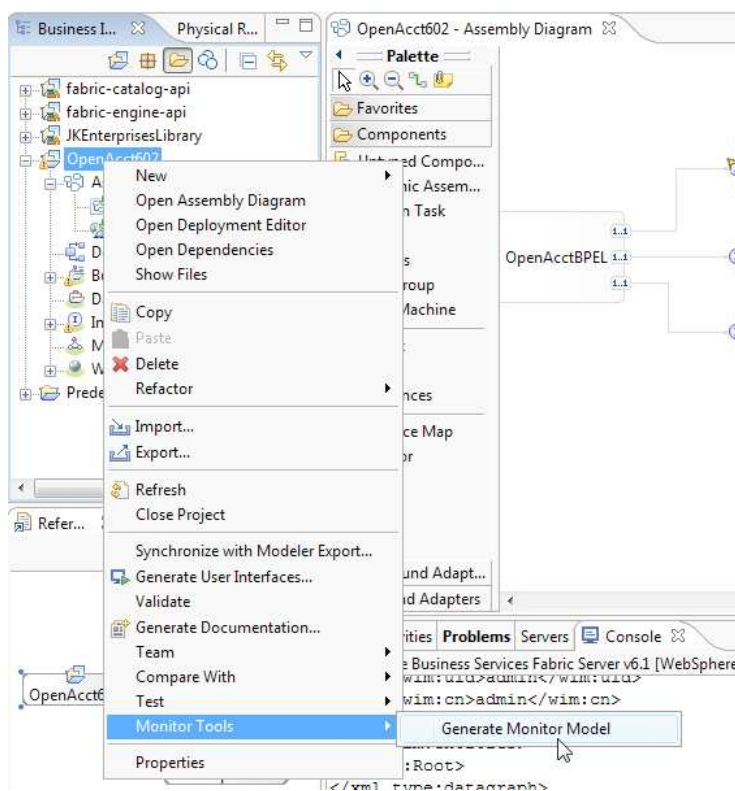


### 3.2 Create the Monitor model

You are ready to create the Monitor model and then run the end-to-end JK Enterprises Open Account BPEL process. This process orchestrates three services (ValidateAddress, VerifyCredit, and ProcessApplication) in order to open a new loan account.

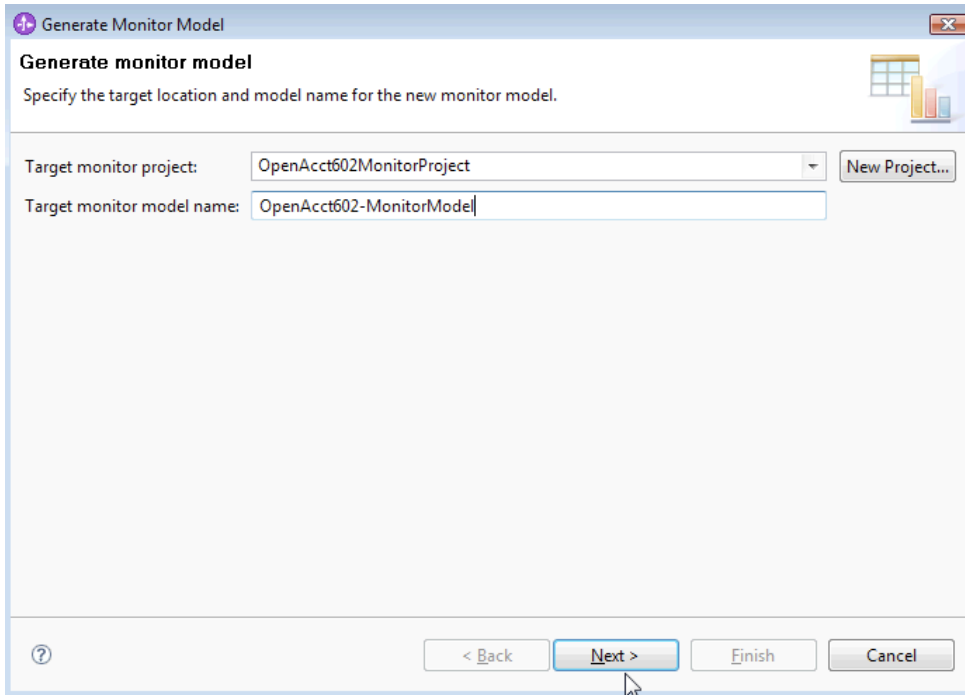
Your goal is to validate that the correct credit report service provider is being selected and provisioned by the Dynamic Assembler for the four use cases.

- \_\_\_117. Switch to the Business Integration perspective and click OK.
- \_\_\_118. Right click the OpenAcct602 module, select Monitor Tools and Generate Monitor Model to create a new Monitor Project.



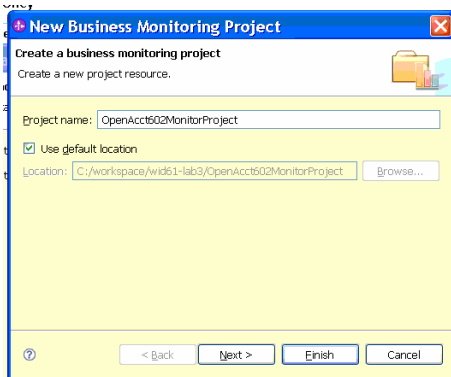
- \_\_\_119. In the Generate Monitor Model window enter:
  - \_\_\_a. Target monitor project : OpenAcct602MonitorProject
  - \_\_\_b. Target monitor model name : OpenAcct602MonitorModel

\_\_c. Click Next.

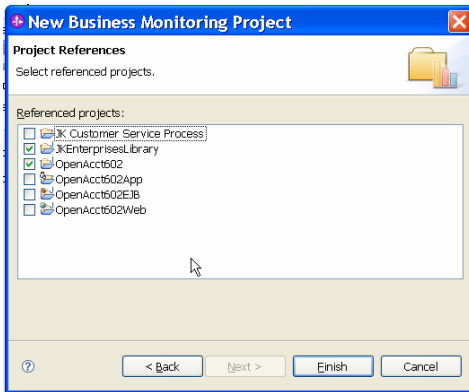


\_\_120. Click Yes to create the Project.

\_\_121. In the New Business Monitoring Project click Next.

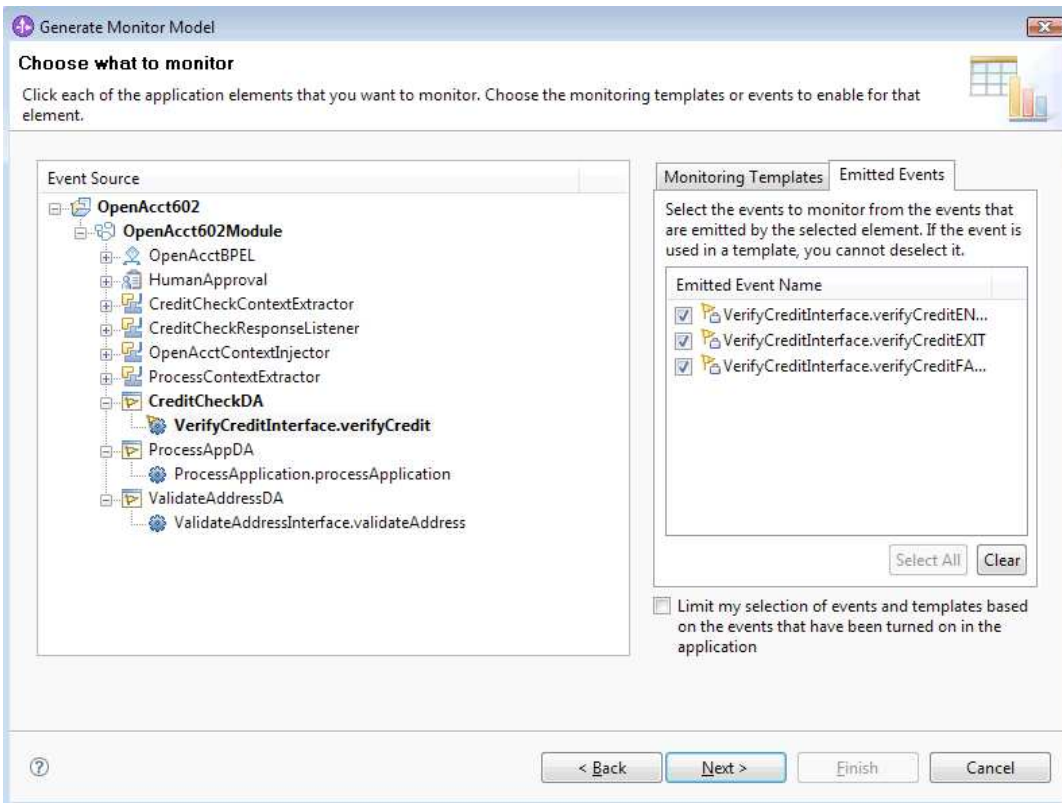


\_\_122. Select the JKEnterpriseLibrary, OpenAcct602. Click Finish

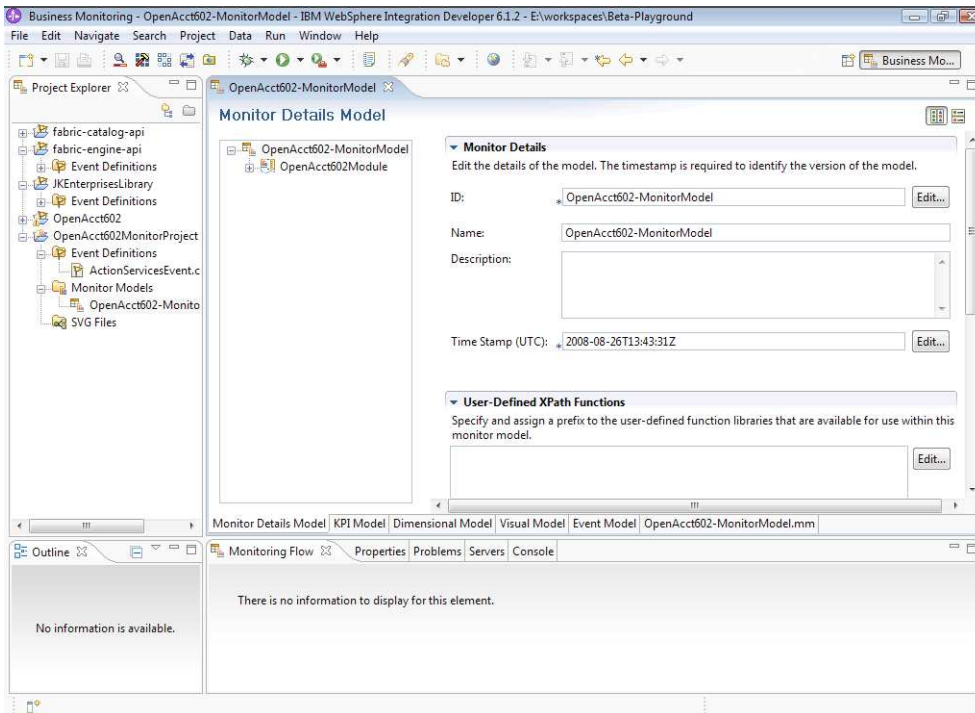


\_\_123. A window should appear. Expand the CreditCheckDA and select the VerifyCreditInterface.verifyCredit.

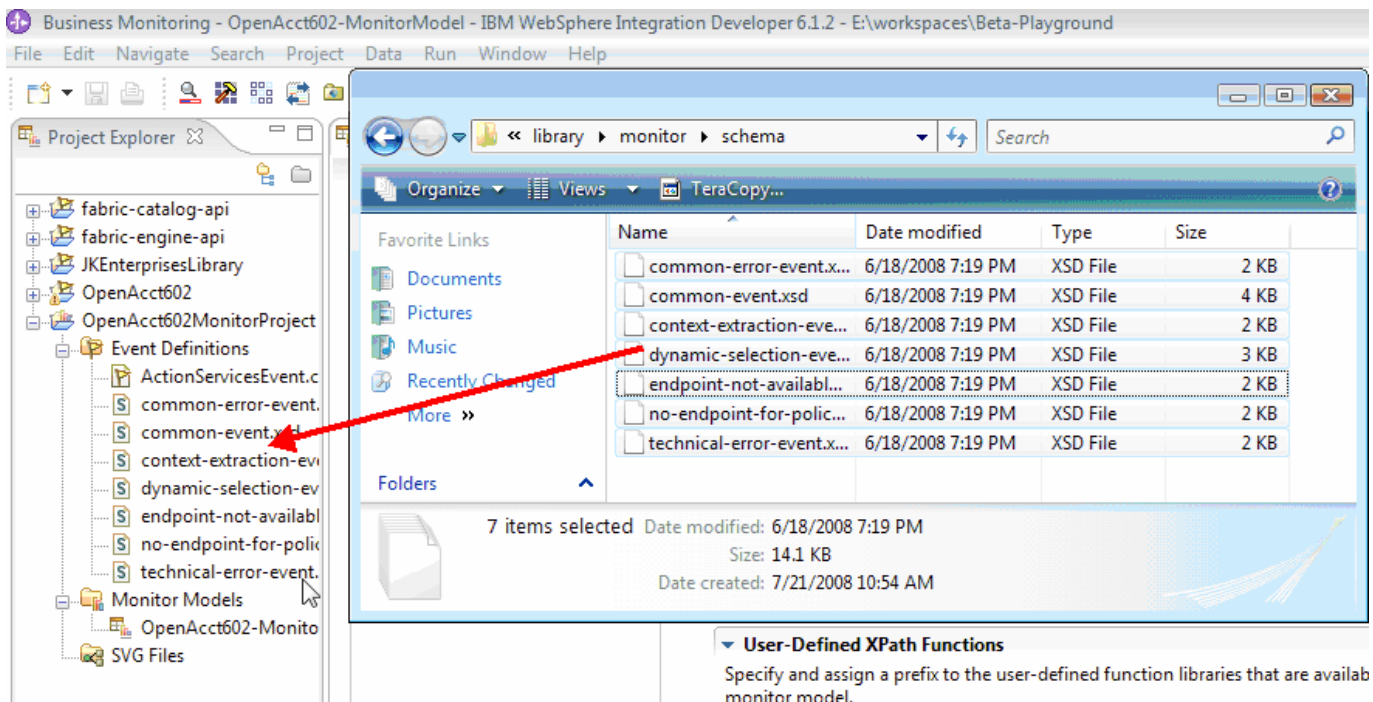
\_\_a. Select all the Emitted Events and then click Next.



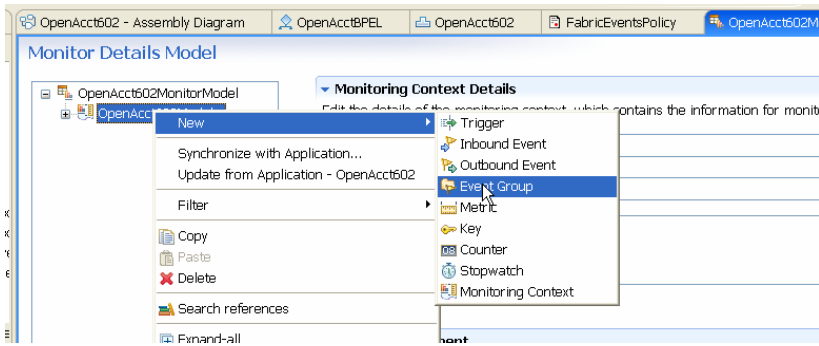
\_\_124. Click Finish and then click Yes to switch to the Business Monitoring Perspective. If you are asked to open Getting Started guide click No.



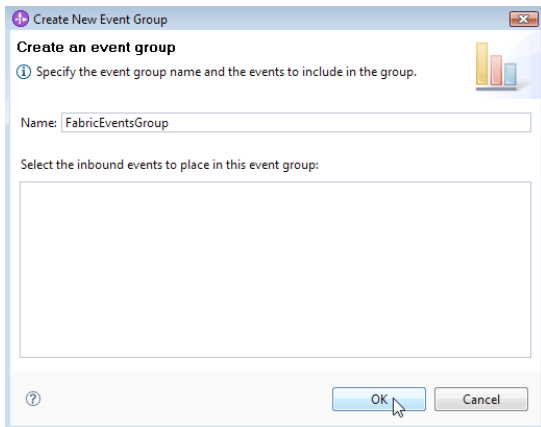
\_\_125. Open in Windows Explorer, <ToolPack\_Home>\library\monitor\schema folder and drag the files into the OpenAcct602MonitorProject > Events Definitions.



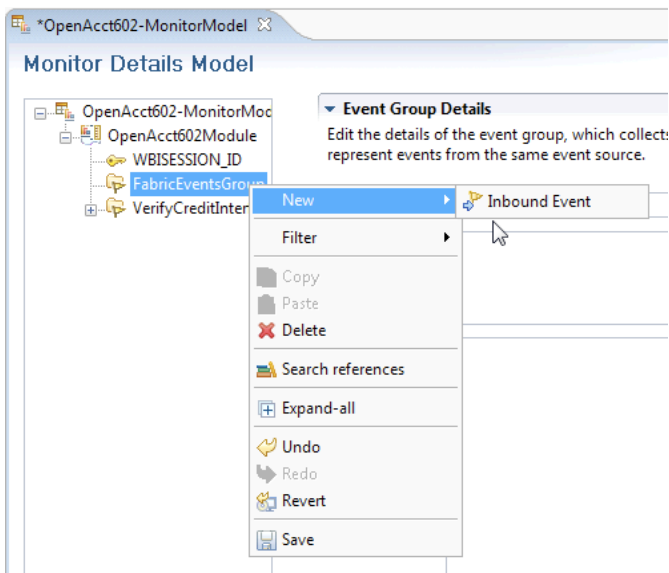
\_\_126. In the Monitor Details Model right click **OpenAcct602Module** and select **New > Events Group**



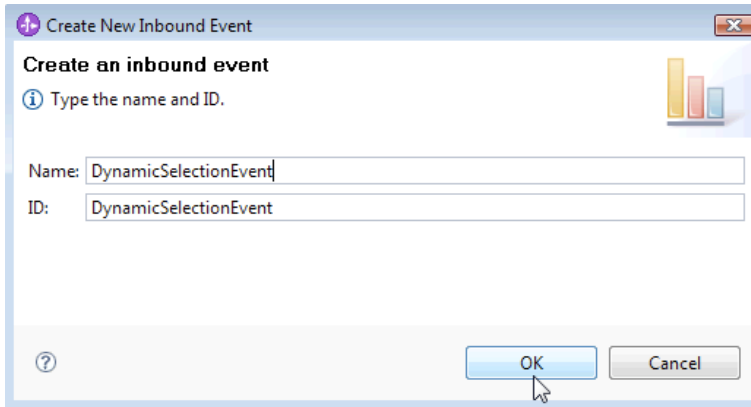
\_\_127. Give the Events Group name **FabricEventsGroup** and click **OK**.



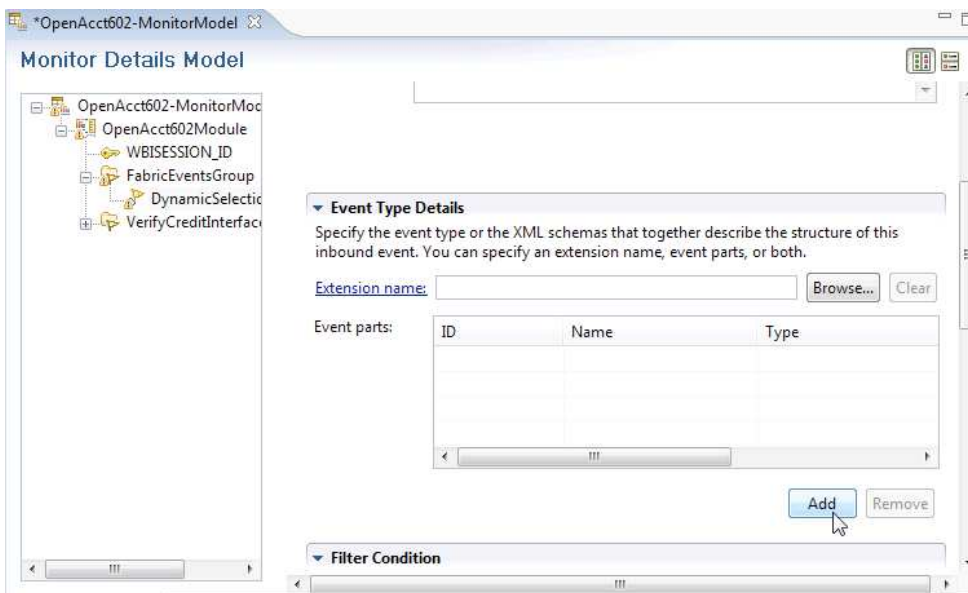
\_\_128. **FabricEventsGroup** is created in the tree view. Right click it and select **New > Inbound Event**



\_\_129. Enter name **DynamicSelectionEvent** and click **OK**.

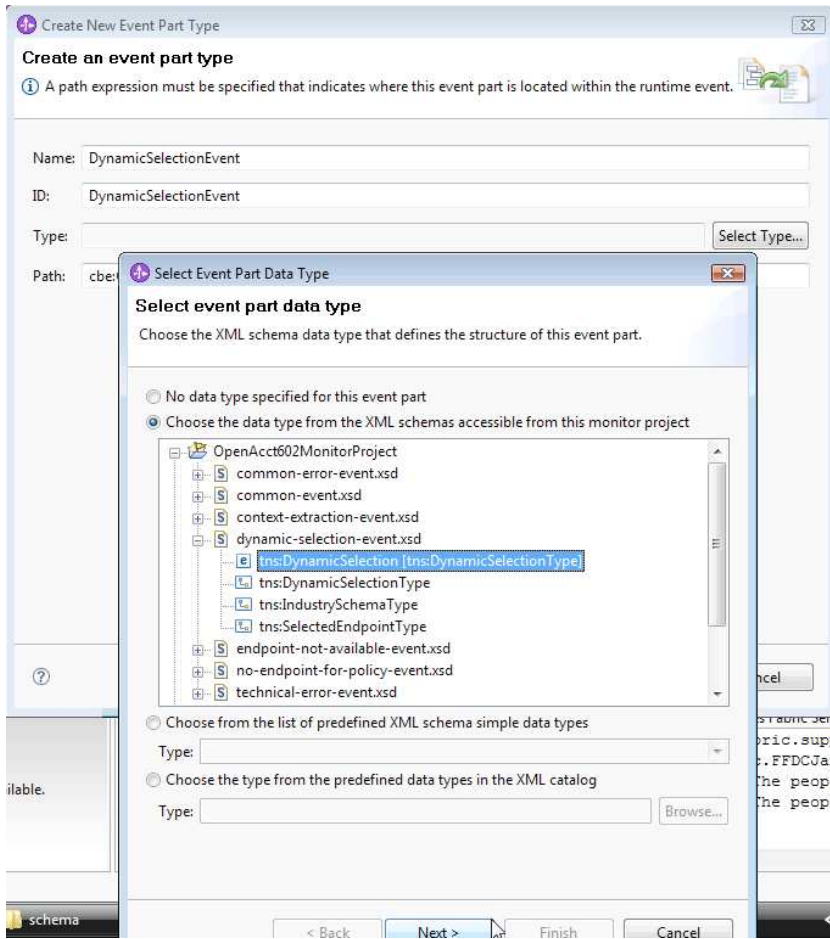


\_\_130. Scroll Down and click **Add** under the Event Type Details

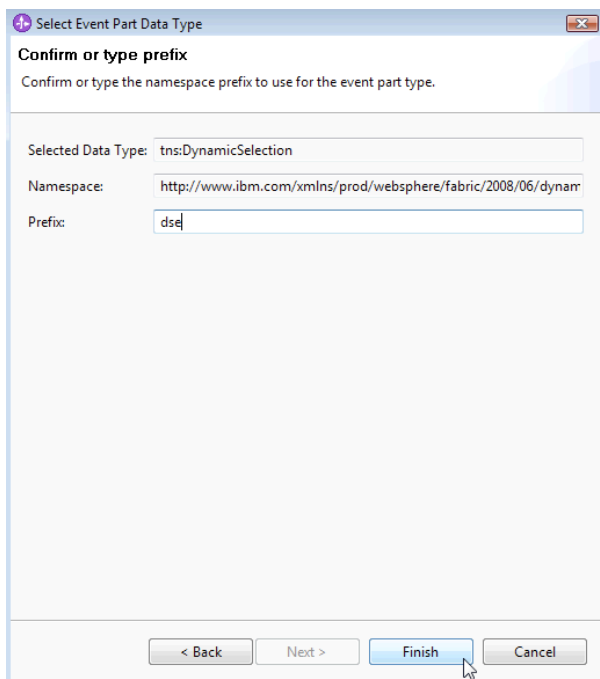


\_\_131. Enter Name: **DynamicSelectionEvent**

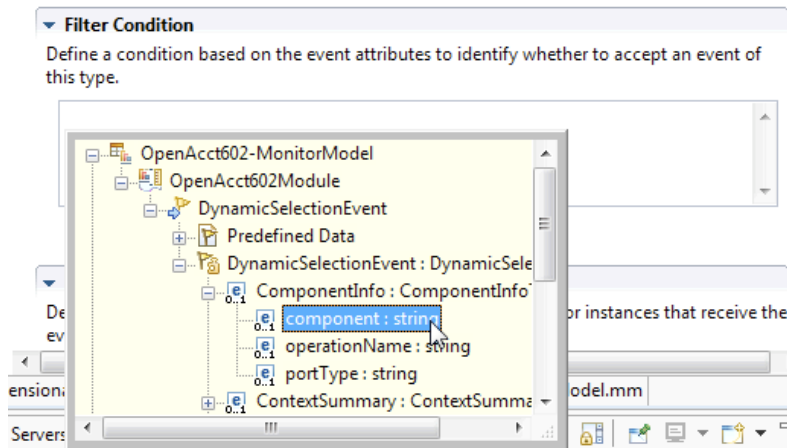
- \_\_a. Click **Select Type..**
- \_\_b. Expand the tree and select the **Dynamic Selection Type**
- \_\_c. Click **Next.**



\_\_132. Type **dse** as a prefix and click **Finish**

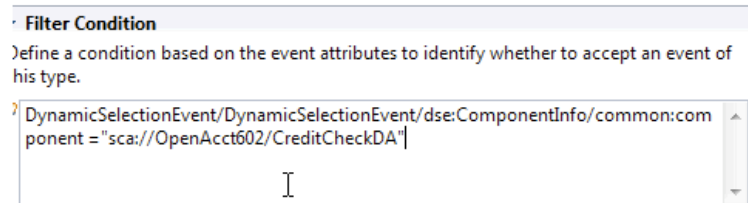


- \_\_133. Click **Finish** in the Create New Event Part type window.
- \_\_134. In the Filter Condition, type CTRL-<space> to open the window, expand the tree, and select **component string** from under the **Dynamic Selection Event**.



- \_\_135. Complete the string by adding **"sca://OpenAcct602/CreditCheckDA"** to the end. The final string should be :

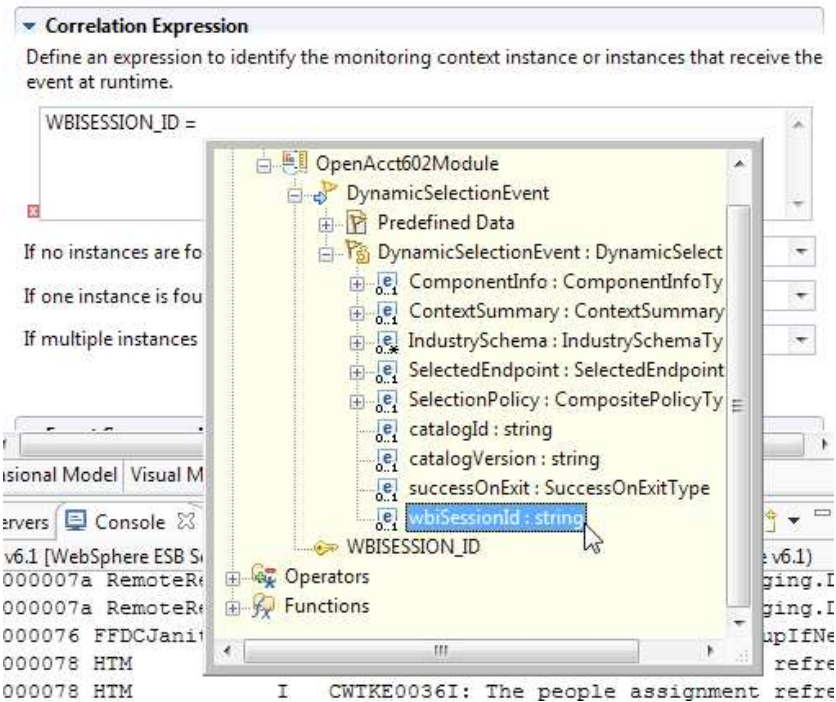
*DynamicSelectionEvent/DynamicSelectionEvent/dse:ComponentInfo/common:component="sca://OpenAcct602/CreditCheckDA"*



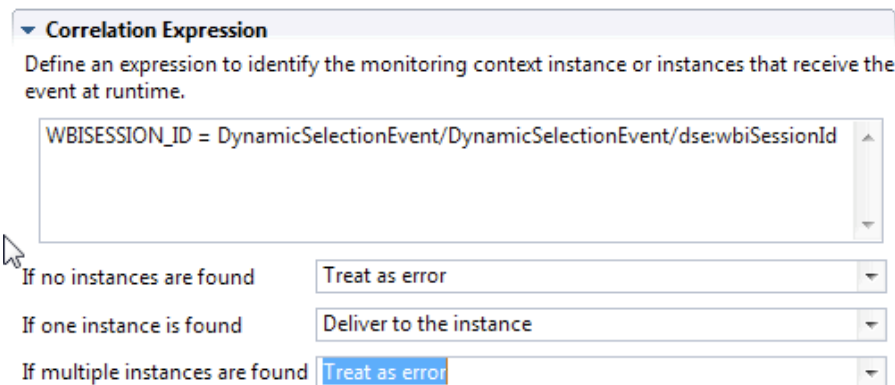
- \_\_136. Scroll down to the **Correlation Expression**, use ctrl-<space> to select the **wbiSessionId** from the **DynamicSelectionEvent**. The final string is

*WBISESSION\_ID=DynamicSelectionEvent/DynamicSelectionEvent/dse:wbiSessionId*



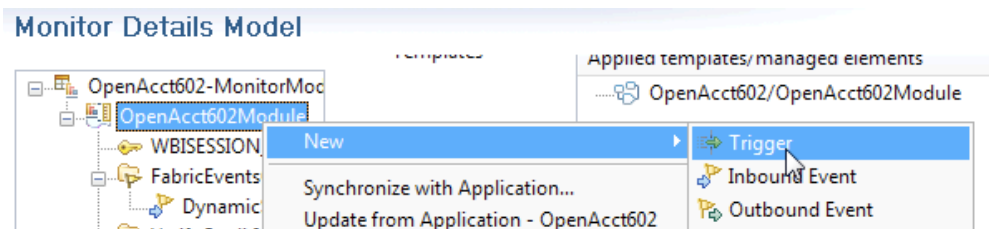


\_\_137. Change the values for drop downs as described.

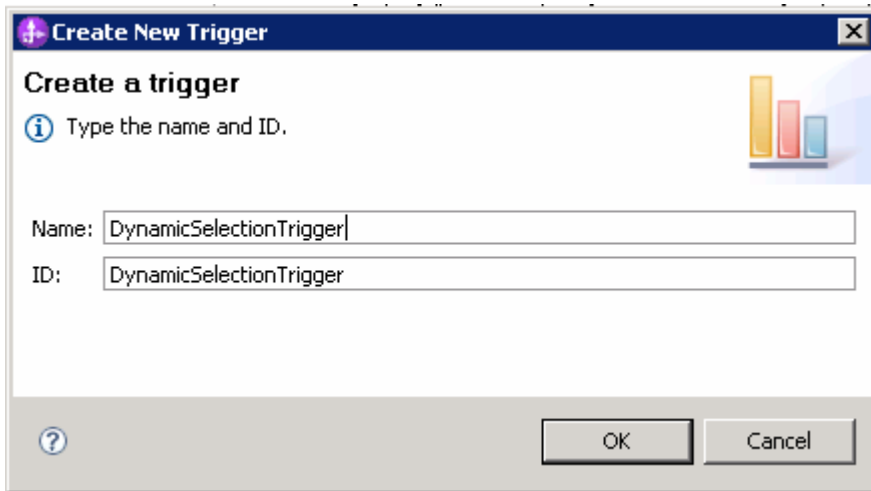


\_\_138. Save changes by clicking Ctrl+Shift+s

\_\_139. Create a new trigger by selecting the module

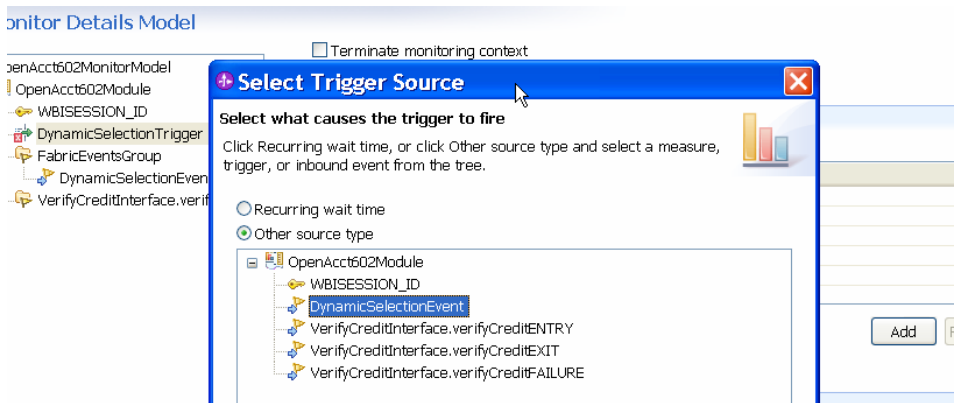


\_\_140. Enter **Trigger** name **DynamicSelectionTrigger**

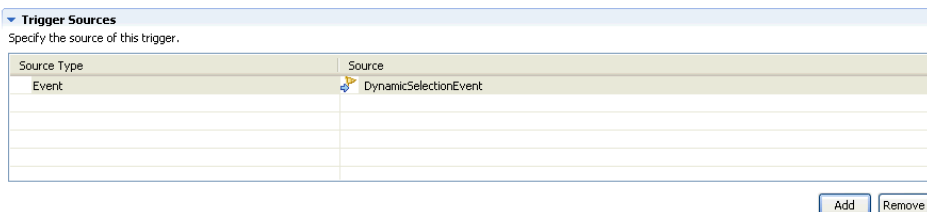


\_\_141. Enter **Trigger Source** by clicking **Add**

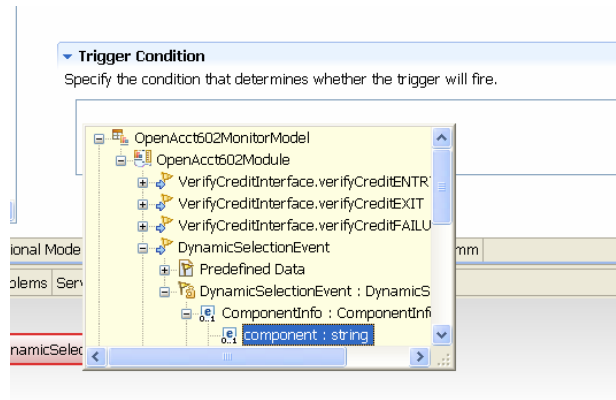
\_\_142. Select **DynamicSelectionEvent**



\_\_143. Click **OK** and verify the trigger source has been added.

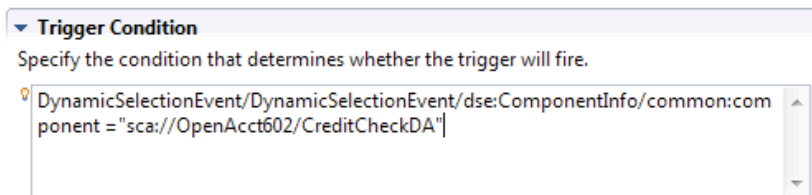


\_\_144. In the Trigger Condition, type CTRL-<space> to open the window, expand the tree, and select **component: string** from under the **DynamicSelectionEvent**.

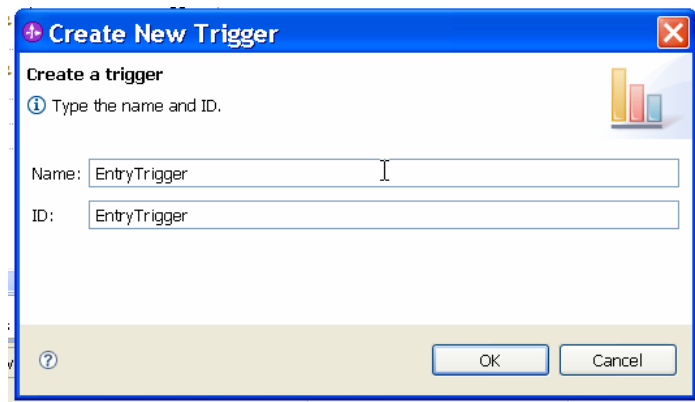


\_\_145. Make sure your final TriggerCondition is the same as shown below. The final string is :

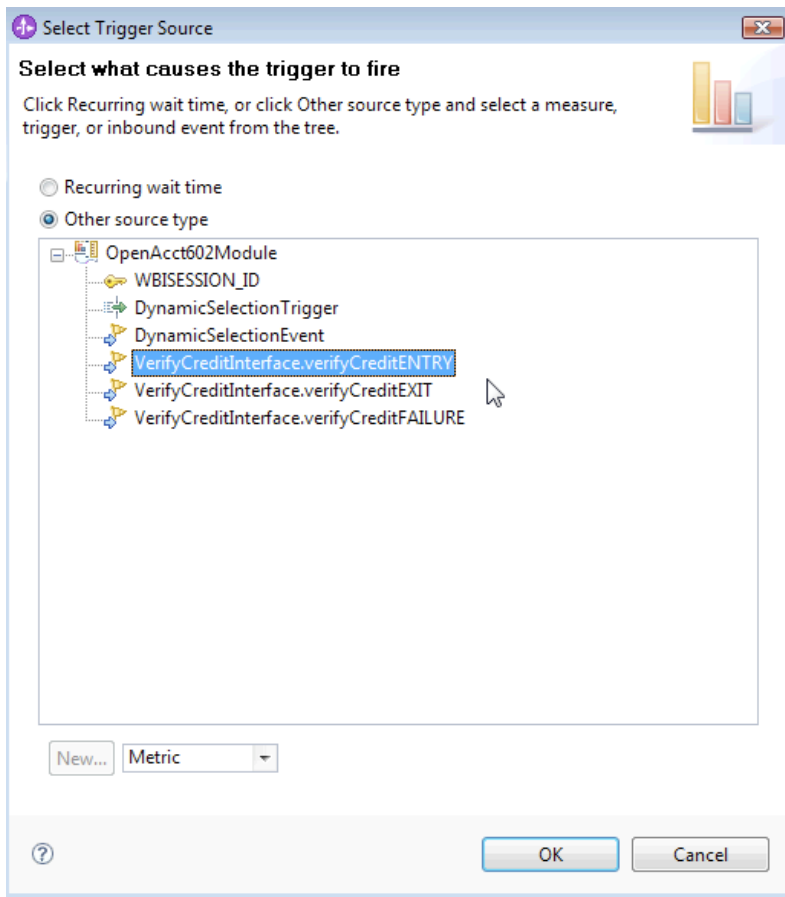
*DynamicSelectionEvent/DynamicSelectionEvent/dse:ComponentInfo/common:component="sca://OpenAcct602/CreditCheckDA"*



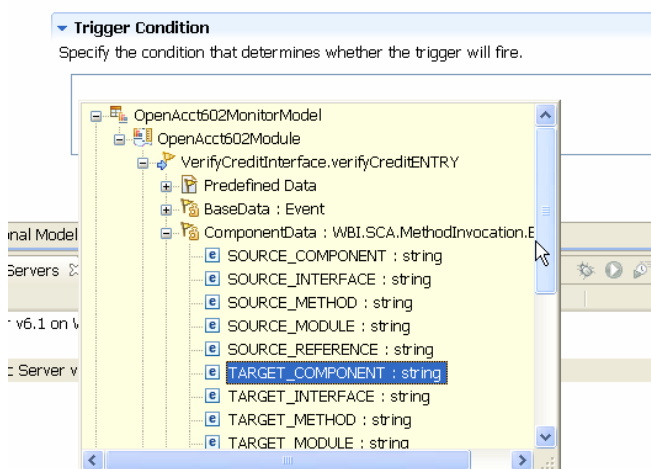
\_\_146. Create another Trigger and name it **EntryTrigger**



\_\_147. Add Trigger source **VerifyCreditEntry** and click **OK**.



\_\_\_148. Add a Trigger condition :  
 VerifyCreditInterface.verifyCreditENTRY/ComponentData/sca:TARGET\_COMPONENT =  
 "CreditCheckDA"



\_\_\_149. The final EntryTrigger Source and Condition should be exactly as follows:

**Trigger Sources**  
Specify the source of this trigger.

Source Type	Source
Event	VerifyCreditInterface.verifyCreditENTRY

**Trigger Condition**  
Specify the condition that determines whether the trigger will fire.

VerifyCreditInterface.verifyCreditENTRY/ComponentData/sca:TARGET\_COMPONENT = "CreditCheckDA"

\_\_150. Create a new Trigger and name it **ExitTrigger**

\_\_151. Add Trigger Source **VerifyCreditInterface.verify.CreditExit** and press **OK**.

**Select Trigger Source**

Select what causes the trigger to fire  
Click Recurring wait time, or click Other source type and select an item from the tree.

Recurring wait time  
 Other source type

- OpenAcct602Module
  - WBISESSION\_ID
  - DynamicSelectionTrigger
  - EntryTrigger
  - DynamicSelectionEvent
  - VerifyCreditInterface.verifyCreditENTRY
  - VerifyCreditInterface.verifyCreditEXIT**
  - VerifyCreditInterface.verifyCreditFAILURE

\_\_152. Fill out the **ExitTrigger** Source and Condition such that it looks like:

*VerifyCreditInterface.verifyCreditEXIT/ComponentData/sca:TARGET\_COMPONENT = "CreditCheckDA"*

**▼ Trigger Sources**  
Specify the source of this trigger.

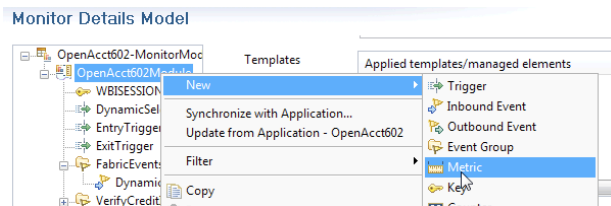
Source Type	Source
Event	VerifyCreditInterface.verifyCreditEXIT

**▼ Trigger Condition**  
Specify the condition that determines whether the trigger will fire.

VerifyCreditInterface.verifyCreditEXIT/ComponentData/sca:TARGET\_COMPONENT = "CreditCheckDA"

\_\_153. Save All changes

\_\_154. Create a New Metric by right clicking on the module, select **New -> Metric**.

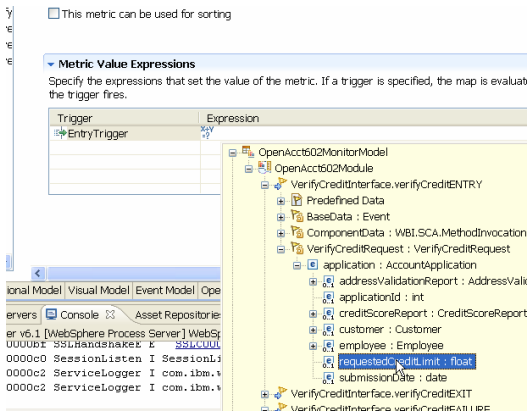


\_\_155. Name the metric **CreditRequested** and click OK

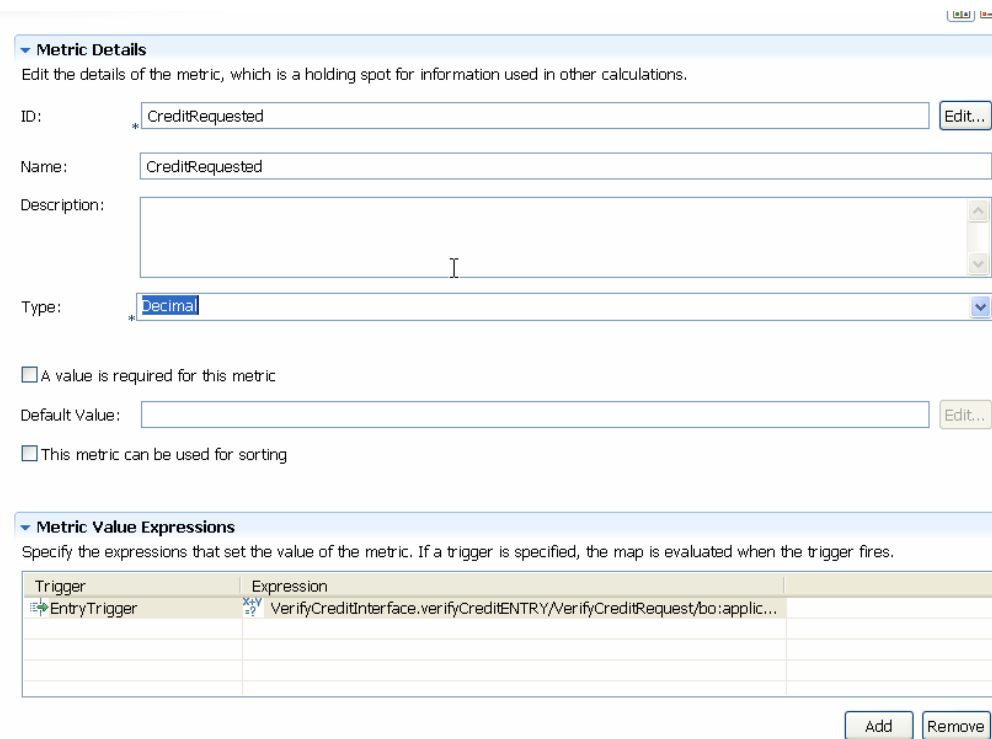
\_\_156. Select Type **Decimal**

\_\_157. Select **EntryTrigger** under Trigger

\_\_158. Select Expressions:  
**VerifyCreditInterface.verifyCreditENTRY/VerifyCreditRequest/bo:application/requestedCreditLimit**



\_\_159. The final screen should look like :



\_\_160. Create a new metric and this time name it **EndPointSelected** and click **OK**. Fill out the fields such that they match the ones shown.

**Metric Details**  
 Edit the details of the metric, which is a holding spot for information used in other calculations.

ID: \*EndpointSelected

Name: EndpointSelected

Description:

Type: \*String

Maximum String Length: 256

Allocate additional space in database to accommodate Unicode string for globalization

A value is required for this metric

Default Value:

This metric can be used for sorting

---

**Metric Value Expressions**  
 Specify the expressions that set the value of the metric. If a trigger is specified, the map is evaluated when the trigger fires.

Trigger	Expression
DynamicSelectionTrigger	DynamicSelectionEvent/DynamicSelectionEvent/dse:SelectedEndpoint/dse:address

Trigger: *DynamicSelectionTrigger*

Expression: *DynamicSelectionEvent/DynamicSelectionEvent/dse:SelectedEndpoint/dse:address*

\_\_161. Create a new metric and this time name it **TargetComponent**, click **OK**. Fill out the fields such that they match the ones shown.

Monitor Details Model

OpenAcct602-MonitorModel

- OpenAcct602Module
  - WBISESSION\_ID
  - CreditRequested
  - EndpointSelected
  - TargetComponent
  - DynamicSelectionTrigger
  - EntryTrigger
  - ExitTrigger
  - FabricEventsGroup
    - DynamicSelectionEvent
  - VerifyCreditInterface.verifyCredit

**Metric Details**  
 Edit the details of the metric, which is a holding spot for information used in other calculations.

ID: \*TargetComponent

Name: TargetComponent

Description:

Type: \*String

Maximum String Length: 256

Allocate additional space in database to accommodate Unicode string for globalization

A value is required for this metric

Default Value:

This metric can be used for sorting

---

**Metric Value Expressions**  
 Specify the expressions that set the value of the metric. If a trigger is specified, the map is evaluated when the trigger fires.

Trigger	Expression
ExitTrigger	VerifyCreditInterface.verifyCreditEXIT/ComponentData/sca:TARGET_COMPONENT



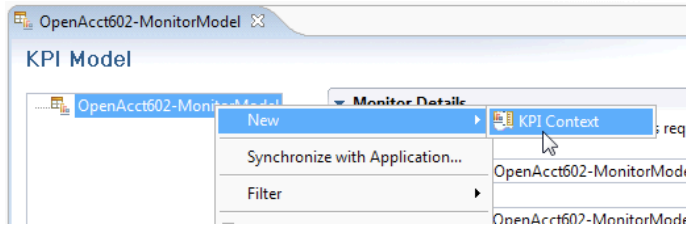
Trigger: *ExitTrigger*

Expression: *VerifyCreditInterface.verifyCreditEXIT/ComponentData/sca:TARGET\_COMPONENT*

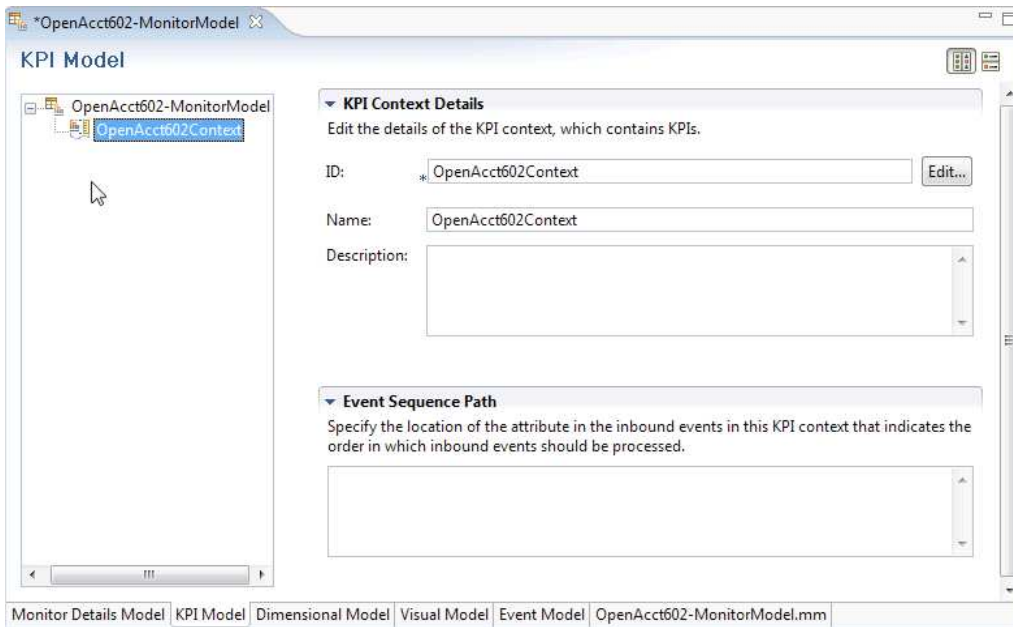
\_\_162. Switch to the KPI Model Tab



\_\_163. Right click the module and select new **KPI Context**

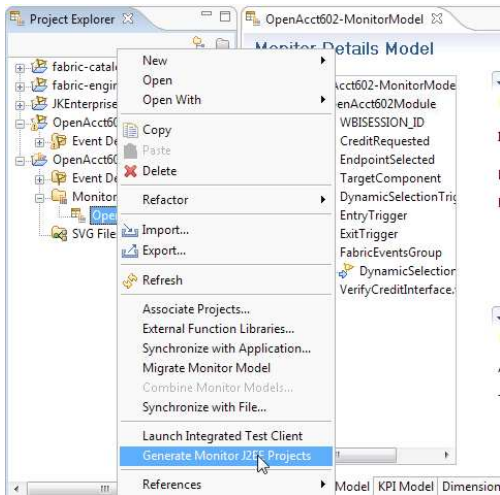


\_\_164. Provide the name **OpenAcct602Context** and click **Finish**. The screen should appear as follows:

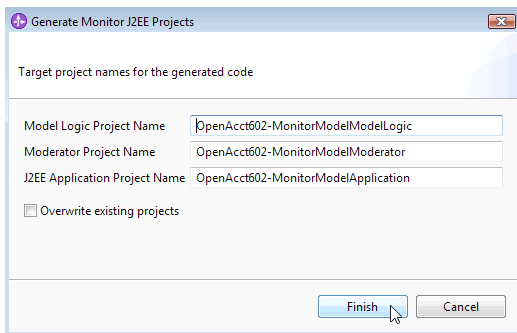


\_\_165. Save All changes by Ctrl+Shift+S

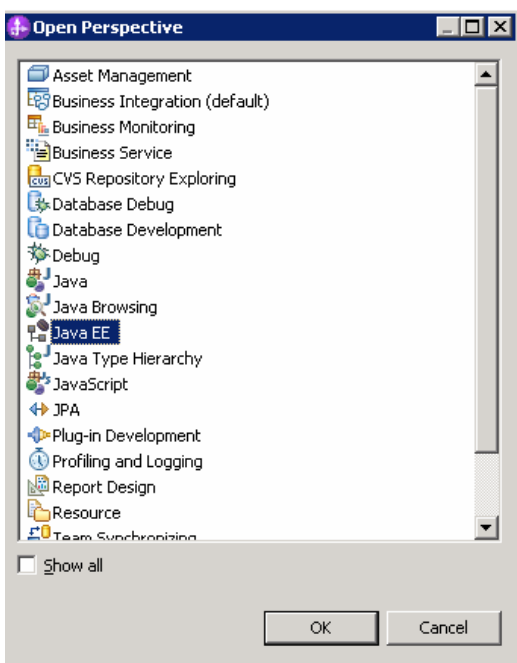
\_\_166. Next Generate the Monitor Java EE project, from the project explorer window expand the OpenAcct602MonitorProject, expand the Monitor Models folder and right click the Monitor Model and click Generate Monitor Java EE Project



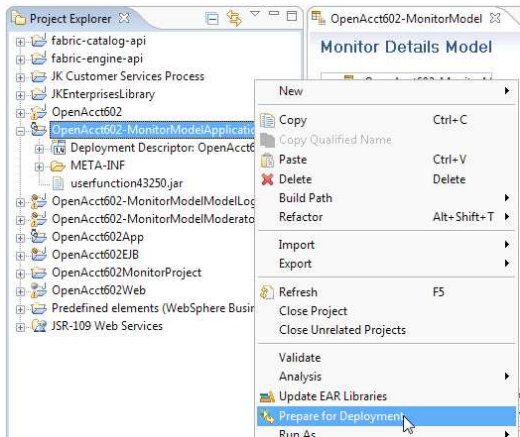
\_\_167. In the window that opens provide Target Project Names and click Finish.



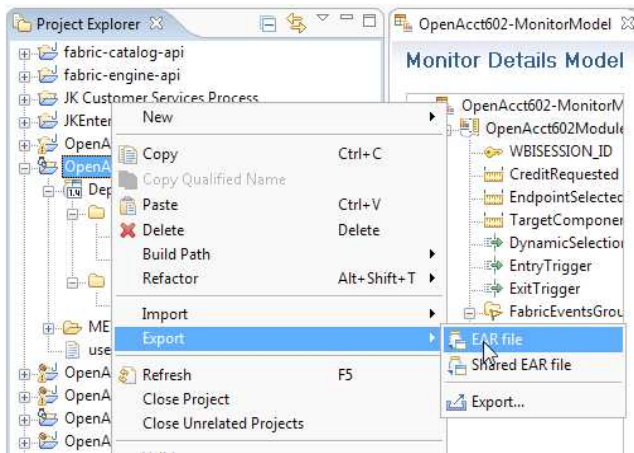
\_\_168. Switch to the Java EE perspective



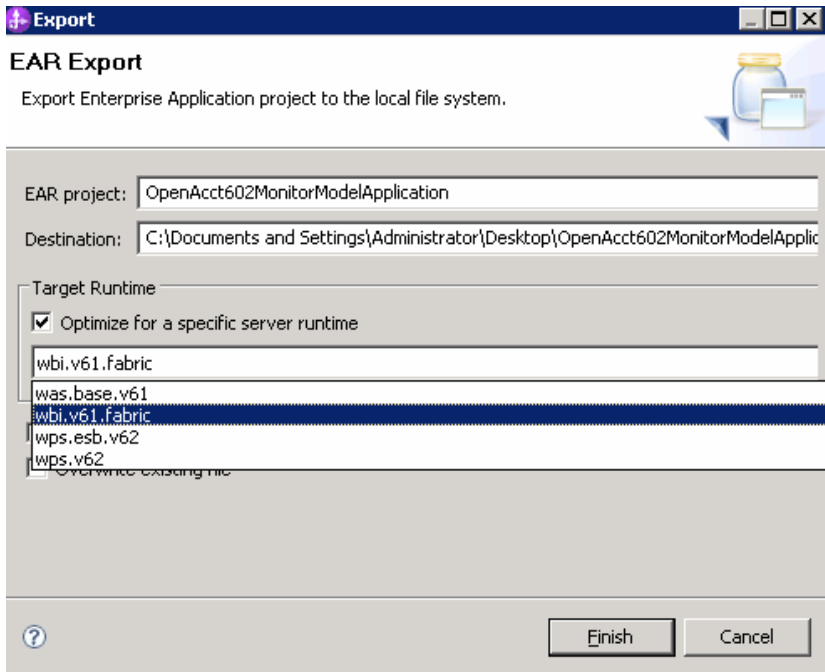
\_\_169. Right click the created Monitor model application, and click prepare for deployment



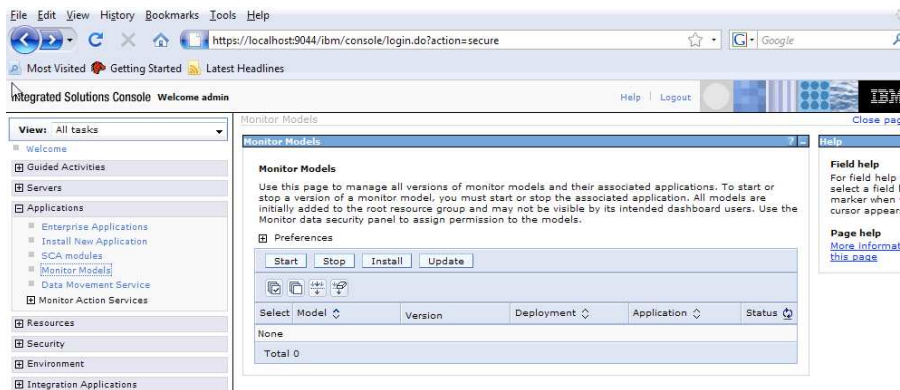
\_\_170. Export the EAR file



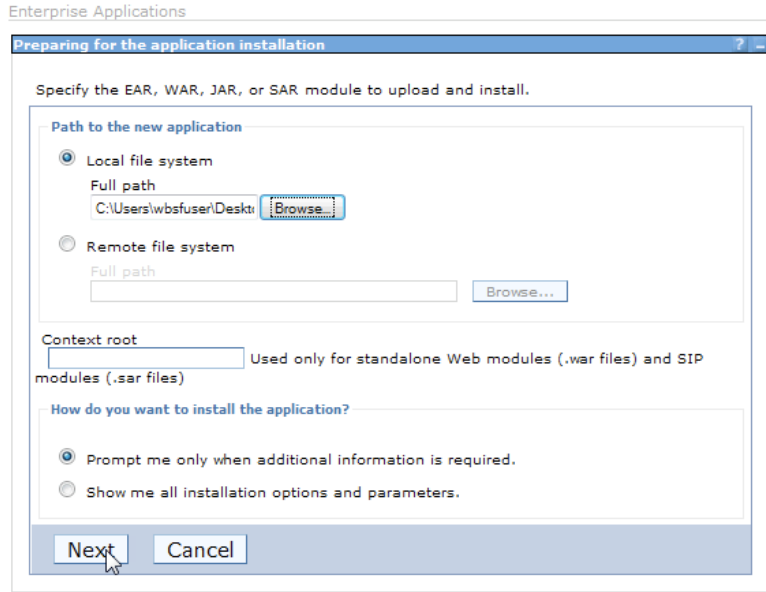
\_\_171. Name the EAR file and save to the desktop. Uncheck Optimize for a specific server runtime. Click **Finish**



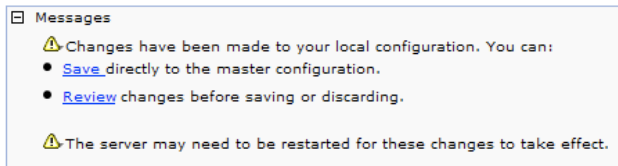
\_\_172. Return to the WebSphere Process Server Administrator console, expand the Applications and select Monitor Models.



\_\_173. After clicking the installation button, browse and select the EAR file you just created. Click Next through the panels and accept the defaults and then click Finish.



\_\_174. Save the changes to the Master Configuration



\_\_175. Start the EAR file

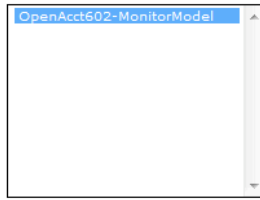


\_\_176. Expand the Security menu, select the Monitor Data Security, click root.



\_\_177. Select the Monitor Model, and select the Personal-KPI-Administrator and click Users.

**Models**

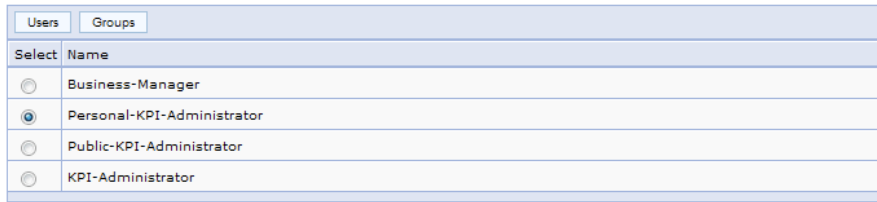


**Roles**

Select a role for this resource group, and click either Users or Groups.

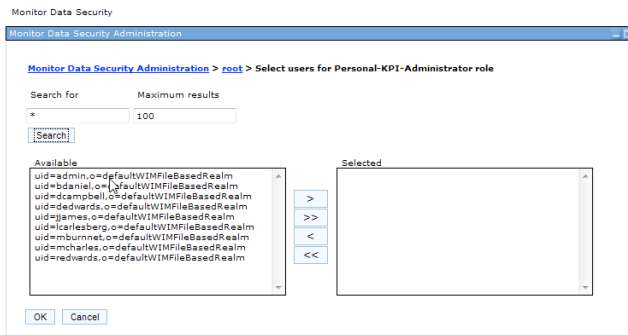
Please note that the following requirements must be met in order to assign users or groups to a role

- Administrative Security must be enabled
- Application Security must be enabled
- Federated Repositories must be the selected User Account Repository



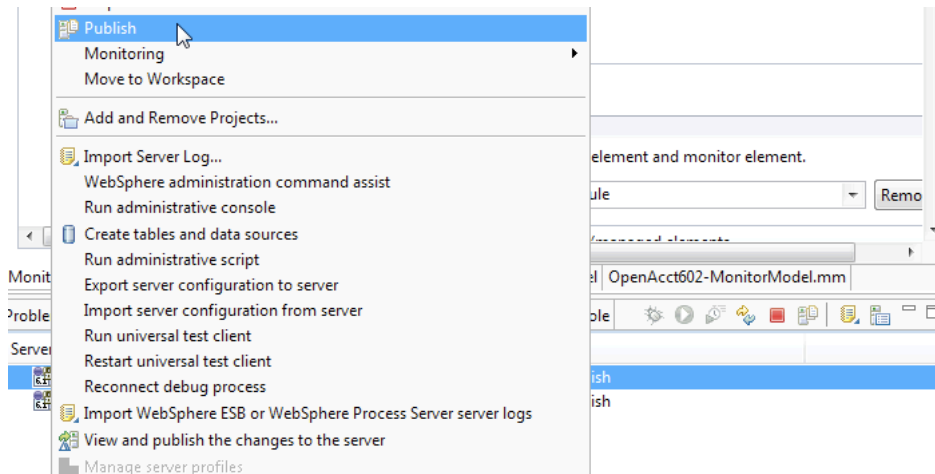
OK Cancel

\_\_178. Click Search and add the Administrator User and click OK.

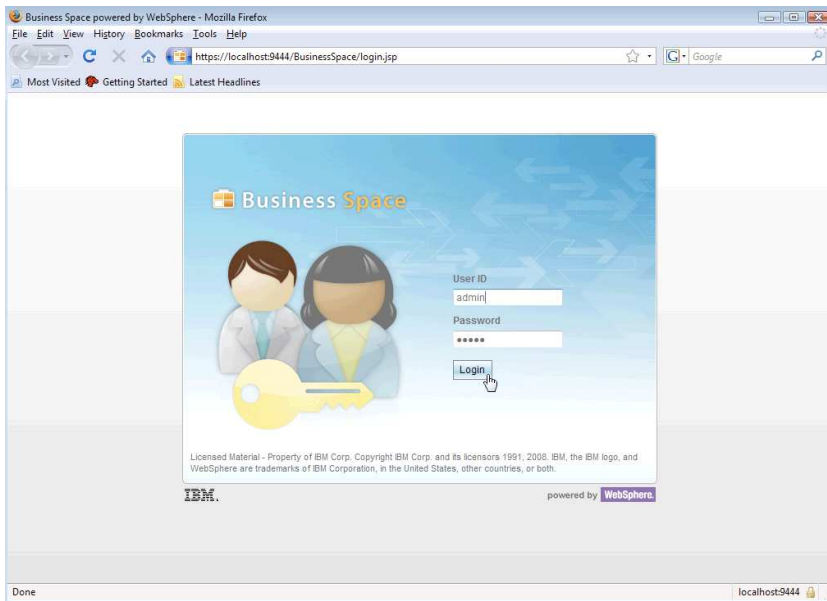


\_\_179. Repeat the previous 2 steps for the Business Manager user, Public-KPI-Administrator, and KPI-Administrator. Click OK.

\_\_180. Switch back to WebSphere Integration Developer > Business Integration perspective. In the servers tab, right click the Fabric server and select Publish.



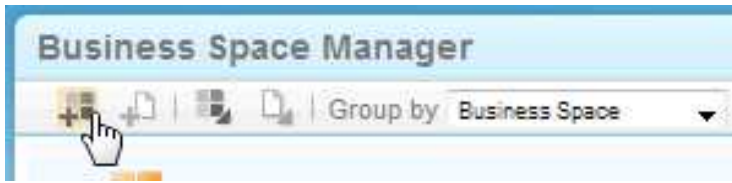
\_\_181. Login into Business Space



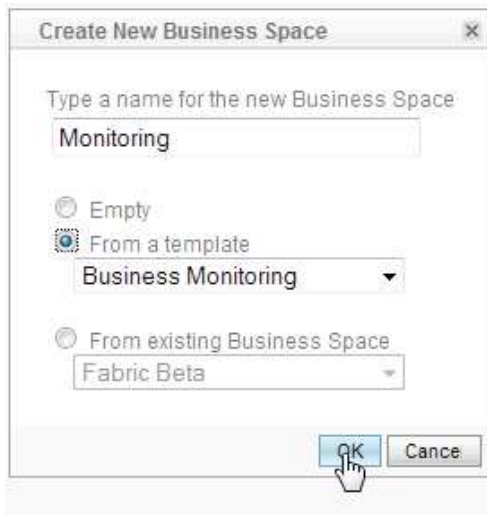
\_\_182. Switch Perspective within Business Space



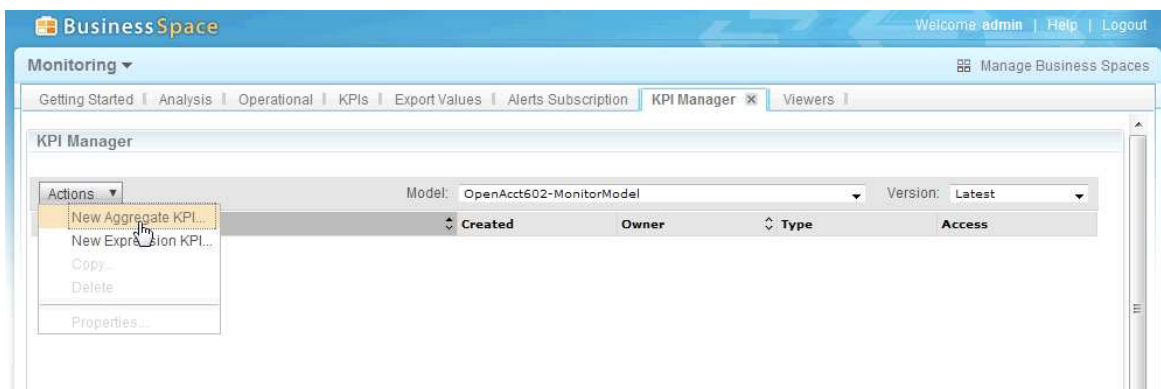
\_\_183. Create a New Area within Business Space



\_\_184. Name the Business Space area and select from Template and press OK.



\_\_185. The Business Space should provide a template and Select the KPI Manager Tab. Select the Monitor Model – OpenAcct602MonitorModel and select New Aggregate KPI.



\_\_186. Name the KPI, Credit Requested for Equinox:



**New Aggregate KPI Properties**

Name Definition Range Other Preview

\* KPI name:

Description:

Model associated with KPI:

Access:  
 Personal  
 Shared





\_\_187. Switch to the Range Tab and edit the Ranges by Adding Rows

**New Aggregate KPI Properties**

Name Definition **Range** Other Preview

Target:

Range definition:  
 Numerical  
 Percentage

Range Name	Start Value	End Value	Color	Icon	Delete
Normal	= <input type="text" value="0"/>	< <input type="text" value="10000"/>		-	
High	= <input type="text" value="10000"/>	< <input type="text" value="20000"/>		-	

\_\_188. Switch to the Definition Tab and select :

**KPI Properties**

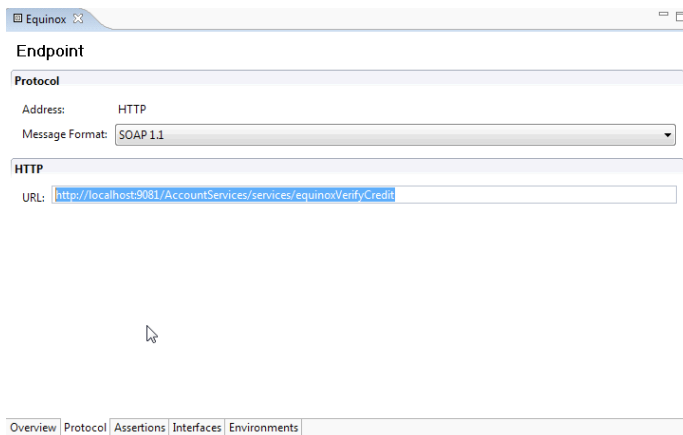
Name **Definition** Range Other Preview

\* Operator:  ▼

\* Metric:  ...

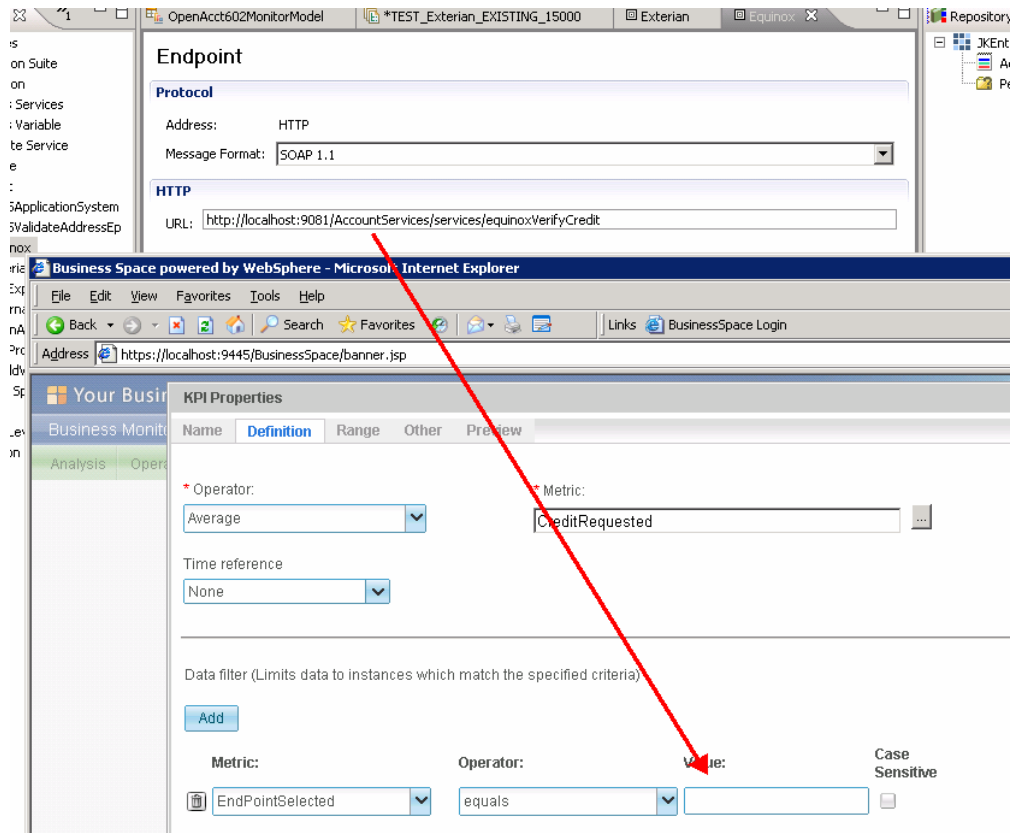
Time reference:  ▼

- \_\_189. Click OK and you should see a new entry 'Credit Request for Equinox'
- \_\_190. Switch to WebSphere Integration Developer > Business Services perspective for this step only.
  - \_\_a. Double click Equinox under Endpoints
  - \_\_b. From the Endpoint Editor switch to the Protocol tab and copy the URL.

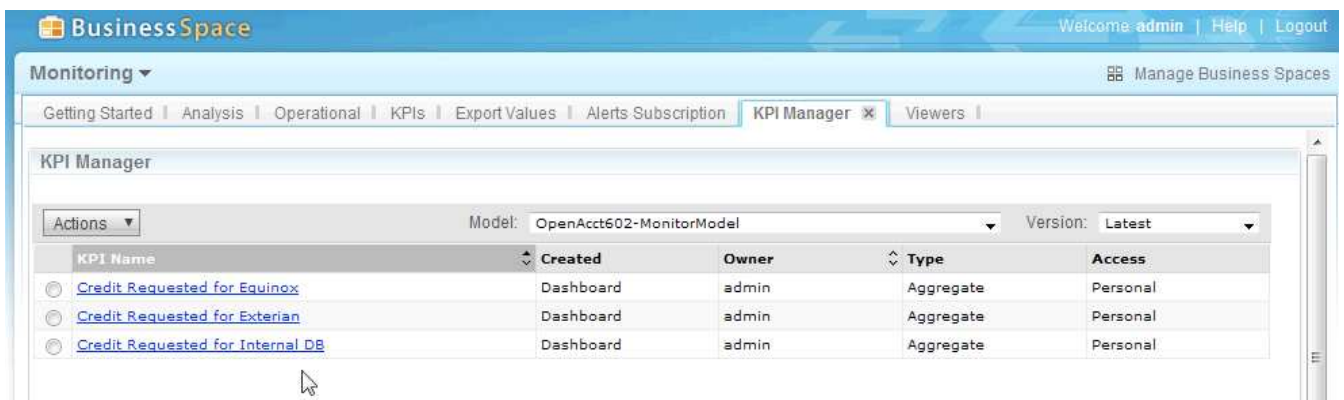


- \_\_191. Switch back to Business Space and click 'Credit Requested for Equinox'.
  - \_\_a. Switch to Definition tab and click Add for Data filter.
  - \_\_b. Place the copied URL as the value for the Endpoint Metric within the Definition tab.

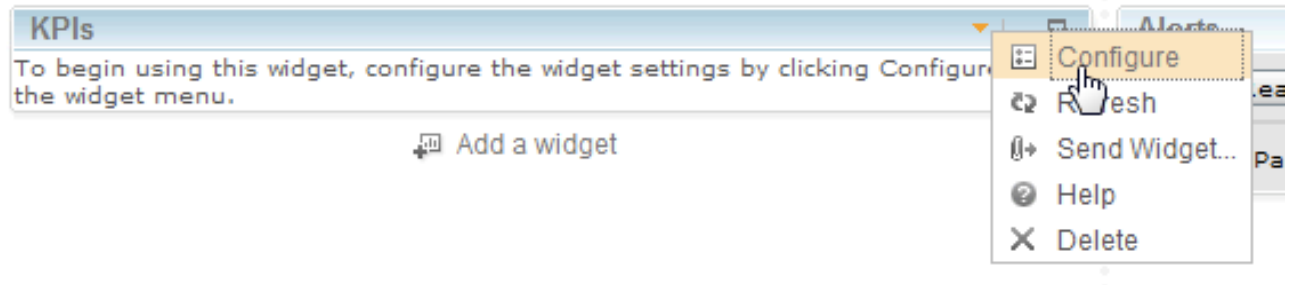
**NOTE:** Business Space does not provide the option for this metric if you do not save (click OK) before adding Data filter.



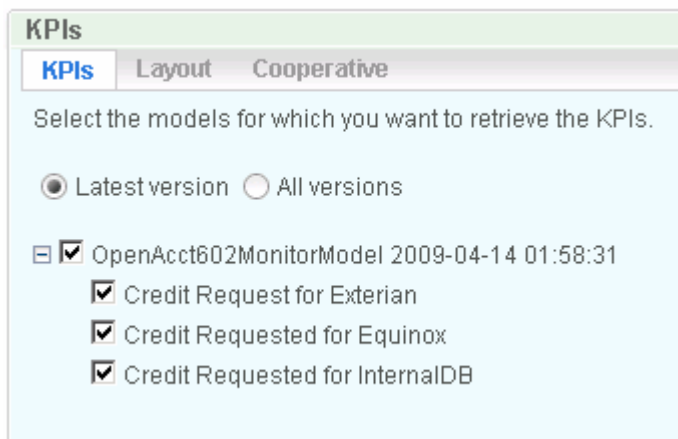
\_\_192. Repeat the process for the Exterian and InternalDB endpoints.



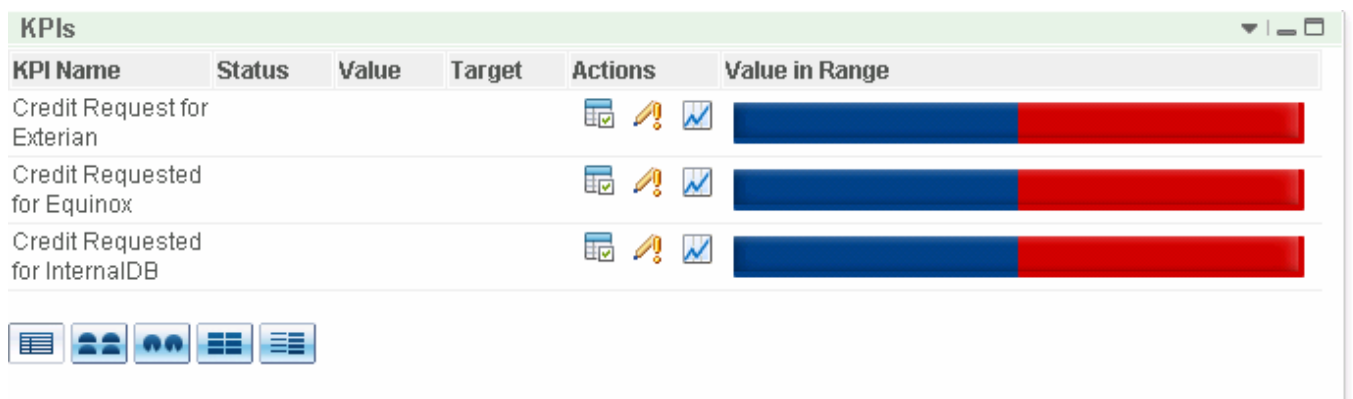
\_\_193. Switch to the KPIs tab, click the down triangle to configure the widget.



\_\_194. Select the KPIs that were just created and click OK



\_\_195. This should open the gauges as a dashboard widget. If the do not appear as such, click the ½ gauge icon at the lower left of the widget.



### 3.3 Run solution on WebSphere Process Server

You are ready to run the end-to-end JK Enterprises Open Account BPEL process. This process orchestrates three services that ValidateAddress, VerifyCredit, and ProcessApplication in order to open a new loan account.

Your goal is to validate that the correct credit report service provider is being selected and provisioned by the Dynamic Assembler for three use cases.

**Keeping things simple**

In this learning lab you are keeping things simple by focusing only on one service realization – the credit report service request.

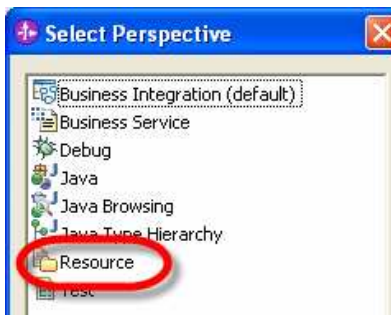


Both the address validation and application processing service calls, though driven by the Dynamic Assembler round-robin between equivalently ranked services.

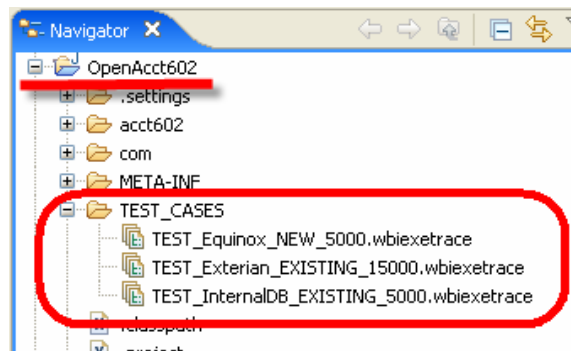
In real world enterprise class Component Business Services Applications, many service realizations and / or process invocations are driven using dynamic declarative policy based selection.

\_\_196. Switch to WebSphere Integration Developer's test environment in order to run three test cases.

\_\_a. Switch to Business Integration Resource perspective using the menu bar by selecting **Window > Open Perspective > Other**. Select **Resource** and click **OK**.



\_\_b. In the Navigator tab on the left expand the **OpenAcct602** resource folder and then expand the **TEST\_CASES** folder.



There are three previously configured test cases provided:

TEST\_Equinox\_NEW\_5000.wbiextrace

TEST\_Exterian\_EXISTING\_15000.wbiextrace

TEST\_InternalDB\_EXISTING\_5000.wbiextrace

### InternalDB test execution

The InternalDB Web service is JK Enterprises' internally developed and hosted credit report service. Recall that it stores reports for EXISTING customers but only supports credit requests for customers with SMALL account types – that is less than \$10,000.

\_\_197. Run the test case for InternalDB service.

- \_\_a. Double click **TEST\_InternalDB\_EXISTING\_5000.wbiextrace** to open the test environment.
- \_\_b. Under the Detailed Properties section on the right, scroll down to locate and validate that the data elements and values are set.

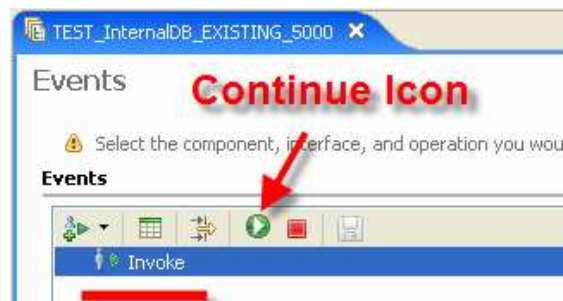
<b>application.employee.type</b>	<b>EXISTING</b>
<b>application.requestedCreditLimit</b>	<b>5000</b>

Minimizing customer and employee address will help.

Initial request parameters

Name	Type	Value
application	AccountApplication	✓
applicationId	int	✓ 0
submissionDate	date	✓ 2008-01-29 -0700
customer	Customer	✓
employee	Employee	✓
employeeId	int	✓ 0
firstName	string	✓ Albert
lastName	string	✓ Cszaszar
address	Address	✓
socialSecurityNumber	int	✓ 0
driversLicenseNumber	int	✓ 0
driversLicenseDate	string	✓
type	string	✓ EXISTING
requestedCreditLimit	float	✓ 5000
addressValidationReport	AddressValidationReport	✓
creditScoreReport	CreditScoreReport	✓

- \_\_c. To run this test case, in the upper left portion of the Events test area, click **Invoke** to highlight it and then click the **continue icon**.

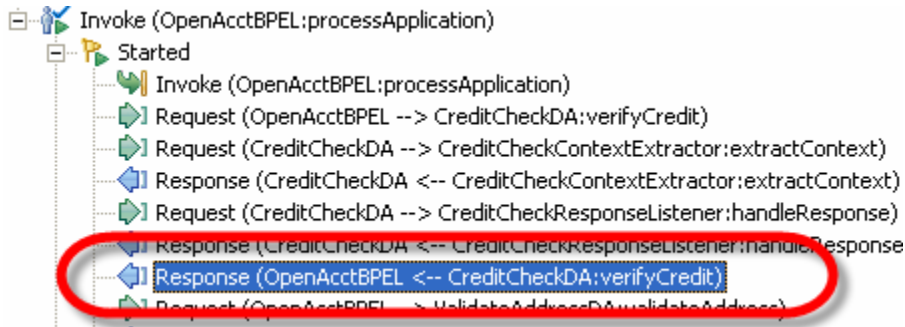


**i First run is slow**  
 This first test run takes quite a while because the test infrastructure is being loaded and the application is being deployed to the process server

\_\_d. Each step in the invocation process will generate a request and response element. Locate the newly generated event response for the credit service response called:

**Response (OpenAcctBPEL <-- CreditCheckDA:verifyCredit)**

\_\_e. Click this element to expose the response information in the Response parameters window to the right.



\_\_f. In the Response parameters table, examine the value for **reportingEntity.name** is **Internal**.

Response parameters

Name	Type	Value
[-] VerifyCreditResponse	VerifyCreditResponse	
[-] creditScoreReport	CreditScoreReport	
creditScore	int	550
[-] reportingEntity	ReportingEntity	
name	string	Internal
[-] address	Address	
line1	string	1055 Madison Avenue
line2	string	<unset>
state	string	NY
city	string	New York
zipCode	string	10299

As expected, the InternalDB credit reporting endpoint was selected. It even returned a credit score of 550.

\_\_198. Examine the logging provided by the Dynamic Assembler.

\_\_a. Double-click the **Console** tab at the bottom of the test window to maximize it.

\_\_b. Scroll up and locate the output information shown below. Then, examine the trace below it.

```

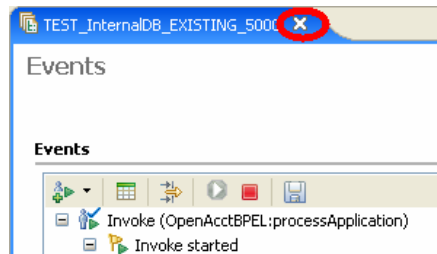
O ++++++INSIDE VERIFY CREDIT CONTENT EXTRACTOR++
O CreditLimit is 5000.0
O Employee Type EXISTING
O ++++++EXITING VERIFY CREDIT CONTENT EXTRACTOR+
I com.ibm.ws.fabric.support.g11n.logging.DelegatingLog in
I com.ibm.ws.fabric.support.g11n.logging.DelegatingLog in
O ++++++INSIDE Verify credit response listener++++
O <dataobject>
    
```

\_\_c. Scroll down and read the logging information generated.

With verbose logging turned on in this test environment, it is possible to see detailed steps taken by the Dynamic Assembler.

\_\_d. Double click the **Console** tab to return it to its original location.

\_\_e. Click the **X** to close the test window and click **No** when asked to save changes.



### Equinox test execution

The Equinox credit reporting service is a vendor credit report service provider. JK Enterprises uses this service for both NEW and EXISTING customers but only when their account limit is considered SMALL – that is less than \$10,000.

\_\_199. Load and run the second test case for validating **Equinox** credit service selection.

\_\_a. Double-click **TEST\_Equinox\_NEW\_5000.wbiexetrace** to open the test environment.

\_\_b. Locate and validate that the data elements and values are set.

<b>application.employee.type</b>	<b>NEW</b>
<b>application.requestedCreditLimit</b>	<b>5000</b>

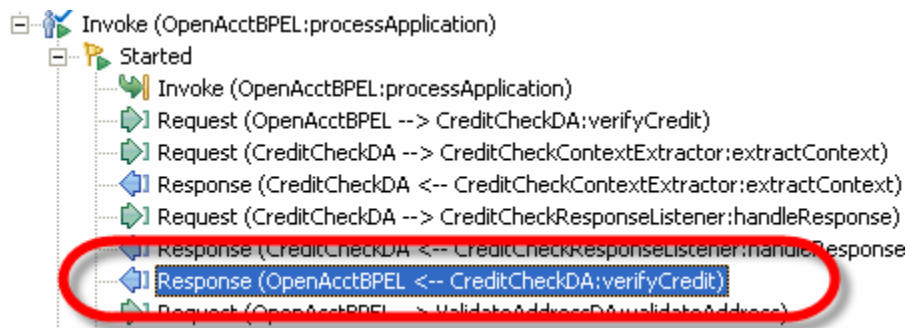


employee	Employee	✓
employeeId	int	✓ 0
firstName	string	✓
lastName	string	✓
address	Address	✓
socialSecurityNum	int	✓ 0
driversLicenseNum	int	✓ 0
driversLicenseState	string	✓
type	string	✓ NEW
requestedCreditLimit	float	✓ 5000
addressValidationReport	AddressValidationReport	✓

\_\_c. Click **Invoke** to highlight it, and then click the **continue icon** to start this test run.

\_\_d. Click the new event trace response for the credit service:

**Response (OpenAcctBPEL <-- CreditCheckDA:verifyCredit)**



\_\_e. In the **Response parameters** table on the right verify that the **reportingEntity** selected is **Equinox**.

Name	Type	Value
VerifyCreditResponse	VerifyCreditResponse	
creditScoreReport	CreditScoreReport	
creditScore	int	650
reportingEntity	ReportingEntity	
name	string	Equinox
address	Address	

\_\_f. Click the **X** to close the test window and click **No** when asked to save changes.

**Exterian test execution**

The Exterian credit reporting service is the most expensive and best credit reporting service used by JK Enterprises at this time. They use it only for customer accounts that are considered LARGE regardless of whether customer is NEW or EXISTING. Today, JK Enterprises has policies in place that declare LARGE to be equal to or greater than \$10,000.

\_\_200. Load and run the third test case to test for the selection of the Exterian credit service.

\_\_a. Double click **TEST\_Exterian\_EXISTING\_15000.wbiextrace**.

\_\_b. Locate and validate that the test data is correct.

<b>application.employee.type</b>	<b>EXISTING</b>
<b>application.requestedCreditLimit</b>	<b>15000</b>

\_\_c. As before, highlight the Invoke element on the left and click the continue icon.

\_\_d. Click the new event trace response for the credit service.

### Response (OpenAcctBPEL < -- CreditCheckDA:verifyCredit)

\_\_e. Validate that the **reportingEntity** is **Exterian**.

creditScore	int	720
<input type="checkbox"/> reportingEntity	ReportingEntity	<b>Exterian</b>
name	string	
<input type="checkbox"/> address	Address	

\_\_f. Click the **X** to close this test window and click **No** when asked to save changes.



You have deployed the model and metadata that you developed in Composition Studio to the runtime Business Services Repository. Then, you performed three Open Account BPEL process runs on a live WebSphere Process Server. You verified that WebSphere Business Services Fabric's Dynamic Assembler selected appropriate service realizations for each test case.

## What you should be able to do

In the first part you have build out the Composite Business Services Model using the Composition Studio perspective. You have understood how and why WebSphere Business Services Fabric captures endpoint service realizations. You also verified the integrity of your model.

These are the steps you completed:

- Review Business Services Fabric Model
- Create a Composite Business Service Model
- Capture the Equinox service realization (two other credit reporting service were done for you)
- Verify the Business Service Model Integrity

In the next part, simulations allowed a developer of a Business Services Application to test, validate, and understand how the WebSphere Business Services Fabric's Dynamic Assembler will perform at run time.

In that part you completed:

- Reviewed the capabilities of each credit reporting service endpoint
- Defined these capabilities as assertions for each endpoint
- Reviewed the JK Enterprises business policy for retrieving credit reports from different providers
- Created and assigned a context specification for a specific Dynamic Assembler component in the process model

- Defined these policies in WebSphere Business Services Fabric using Composition Studio
- Used Policy Simulator to validate the expected behavior of the Business Services Application you are developing

In the last part, you setup the test environment and then ran saved test cases on WebSphere Process Server. You observed WebSphere Business Services Fabric's Dynamic Assembler select the appropriate credit report service. This best endpoint selection was driven by declarative policies you implemented for JK Enterprises and the content, context, and contract of the service requests. You also explored how WebSphere Business Services Fabric extends IBM's SOA Foundation using WebSphere Process Server.

In that part you did:

- Deployed, ran, and evaluated the results of three end-to-end test cases run on WebSphere Process Server
- Understood how WebSphere Business Services Fabric enhances and extends WebSphere Process Server



You have successfully completed all the labs.