4 BPEL Orchestration

4.1 Introduction

When receiving large orders (greater than $1,000) we want to be more cautious and:

- Validate the customer's credit card
- Automatically accept or reject the order based on the credit card status

In this chapter, you will have someone from the sales department review and approve the order manually using the Human Task service.

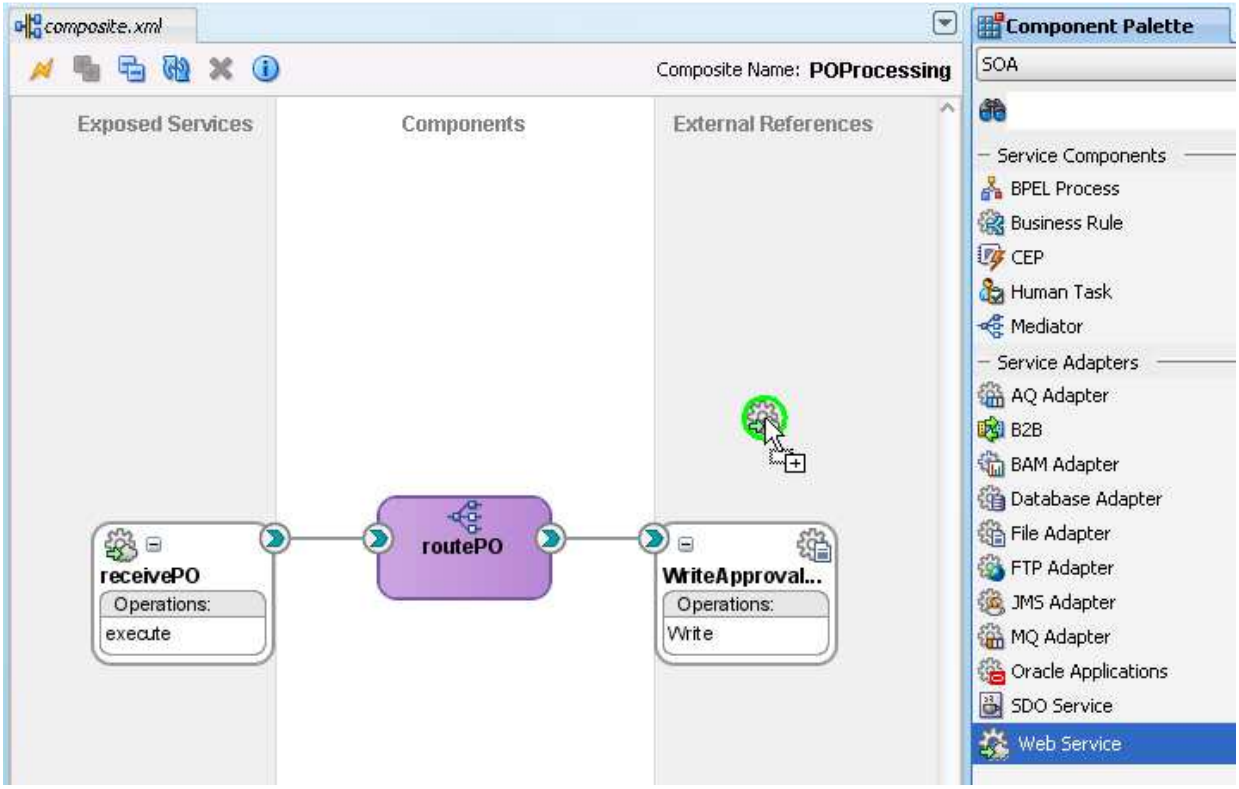
The tool for orchestrating these interactions is the BPEL process manager. The overall flow of the application uses the services created earlier as well as the Human Task service.

4.2 Modifying the composite application

You'll start by invoking the service you created in chapter 2, the credit card validation service, using SOAP.

4.2.1 Invoking the CreditCardStatus service

1. Drag-and-drop a Web Service activity into the **External References** swim lane.

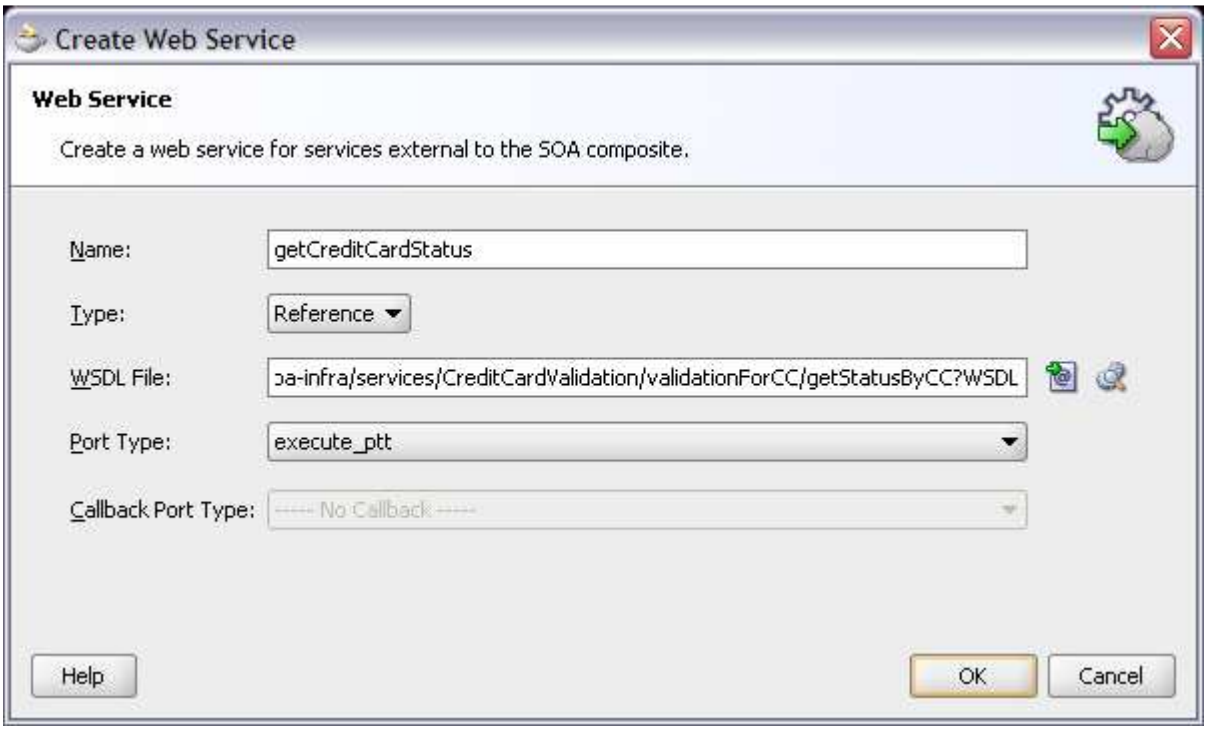


2. Set the following fields:

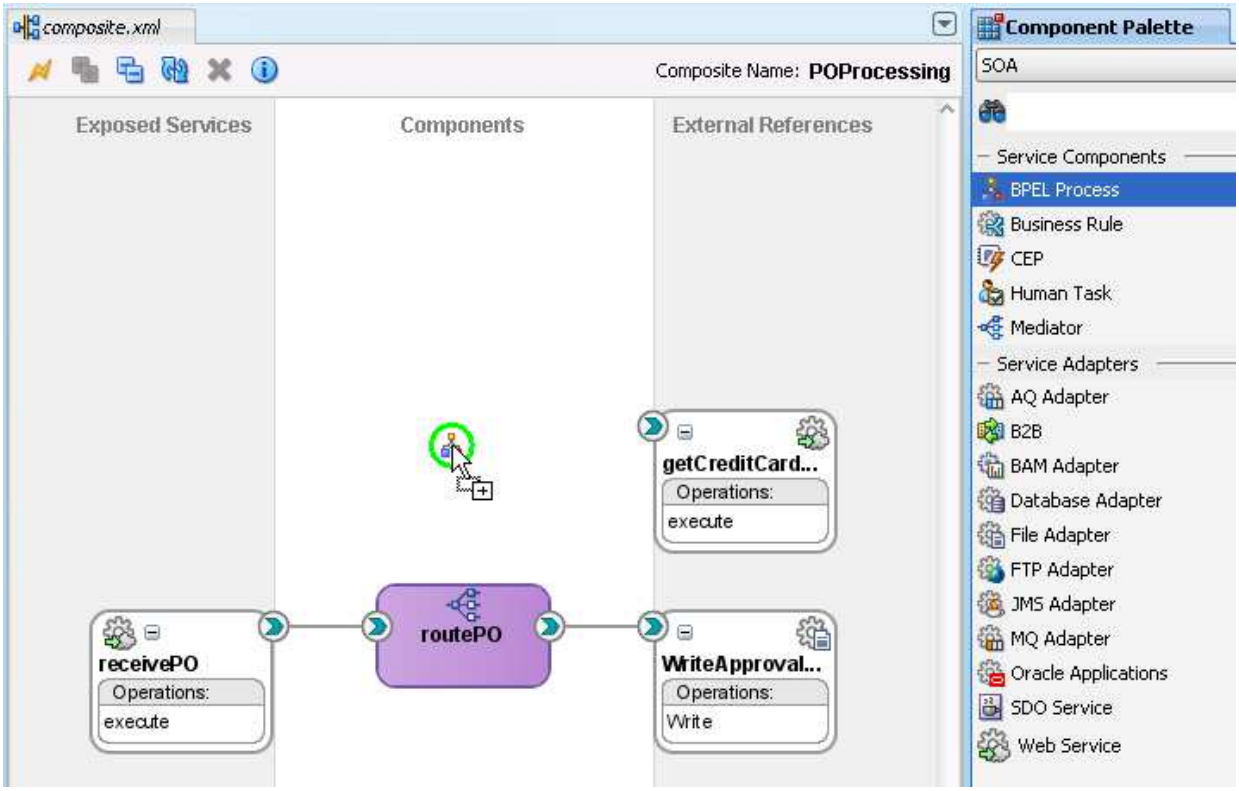
- **Name:** getCreditCardStatus
- **WSDL File:** As mentioned in chapter 2, the URL to the WSDL file can be obtained from the SOA Console, in the test screen, by clicking the **Service**

Description link. (Hint: It will be something like: http://localhost:8888/soa-infra/services/CreditCardValidation/validationForCC/getStatusByCC?WSDL)

- **Port Type:** When you copy-and-paste the WSDL URL in and press the Tab key to move to the next field, it will be updated automatically based on the contents of the WSDL.



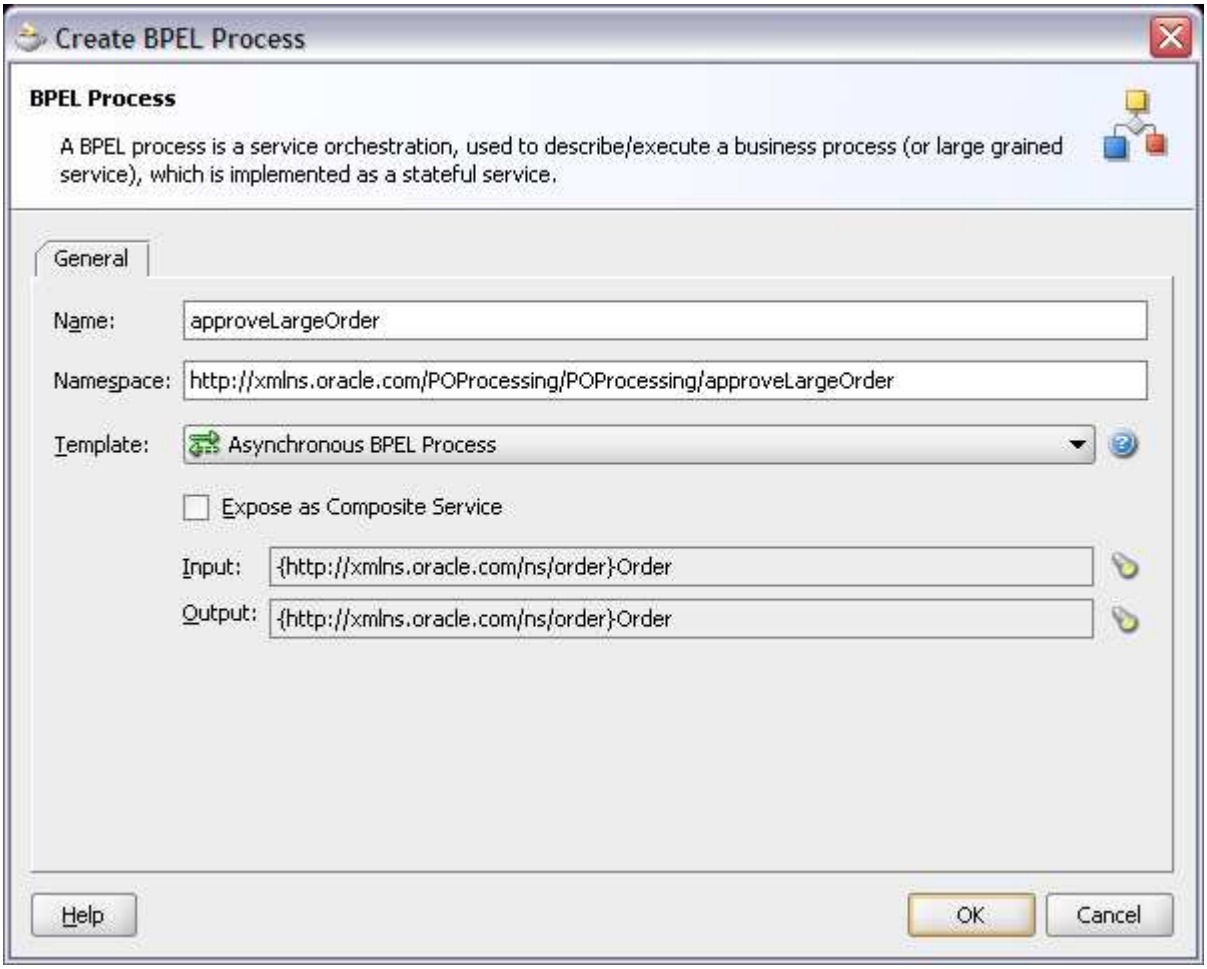
3. Click **OK**.
4. Drag-and-drop a BPEL component on to the **Components** swim lane.



5. In the **Create BPEL Process** dialog, specify the following settings:

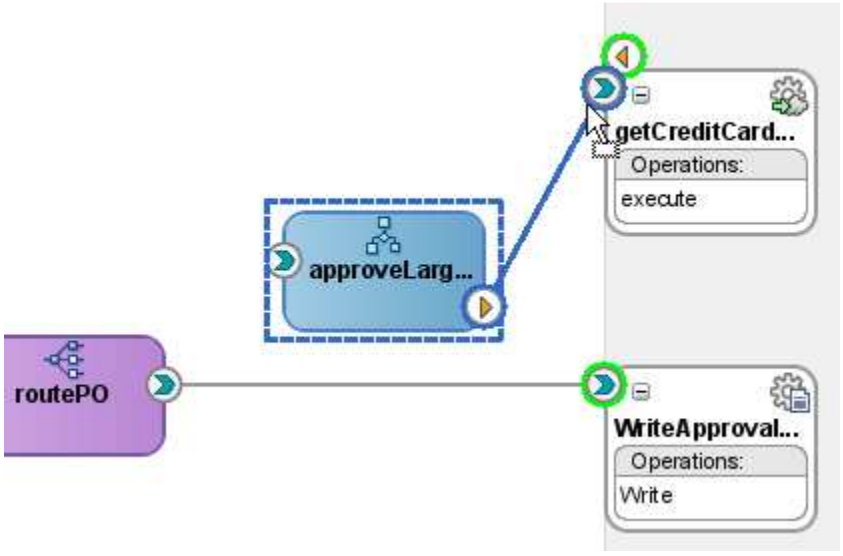
   - **Name:** approveLargeOrder
   - **Template:** Asynchronous BPEL Process
   - **Expose as Composite Service**: Unchecked
   - **Input:** Click the flashlight icon, expand **Project Schema Files > internalorder.xsd** and select **Order**
   - **Output:** Use the **Order** type, like you did for **Input**

   Note that the input and output specified here will go in to the WSDL for this service.



6. Click **OK**.

7. Wire the BPEL process and the **getCreditCardStatus** service.



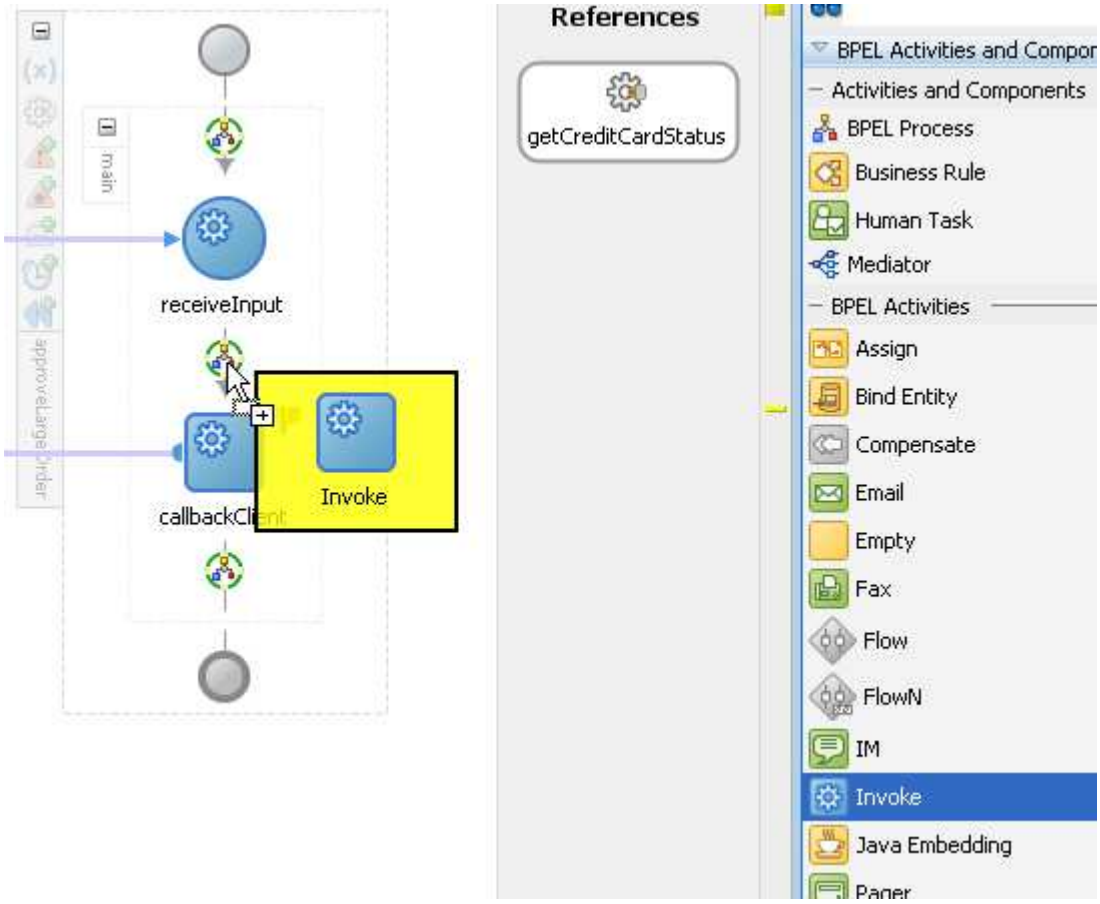
4.2.2 Designing the BPEL approval process

Next you'll build a simple BPEL process that calls the external **getCreditCardStatus** service.

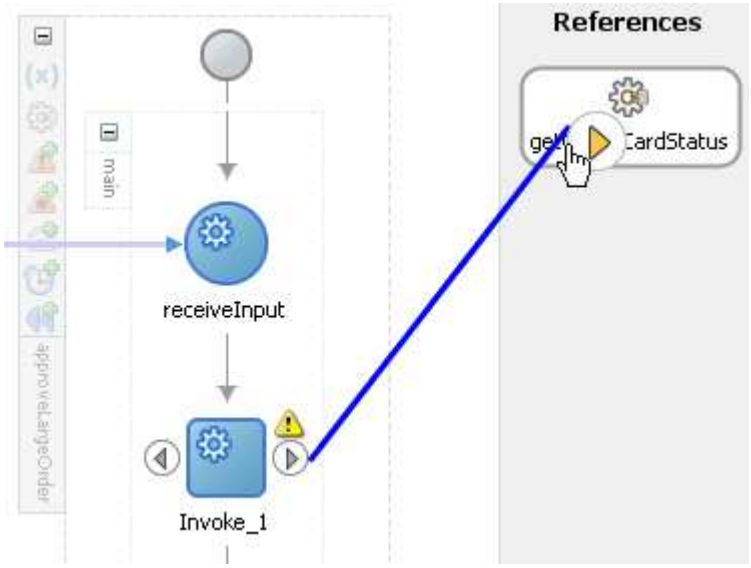
1. Double-click the BPEL component to open the BPEL editor.

   Notice the **getCreditCardStatus** partnerlink already in the **References** swim lane because you wired it in the composite.

2. Drag-and-drop an Invoke activity from the Component Palette to an insertion point under the **receiveInput** activity.



3. Drag the wire from the Invoke activity to the **getCreditCardStatus**. This tells your BPEL process to invoke that service.

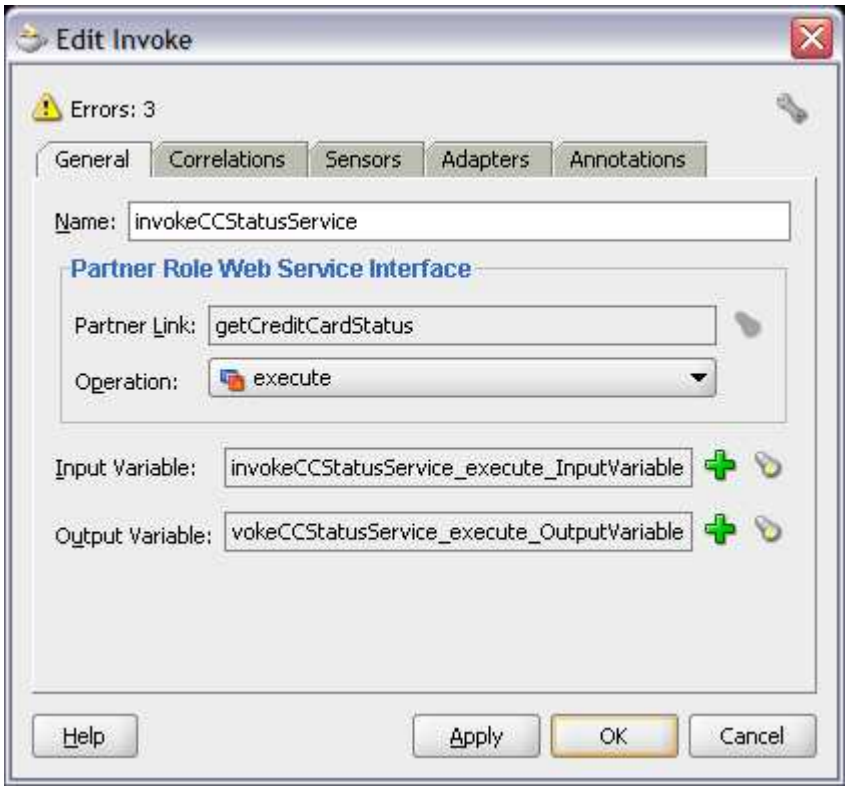


4. In the **Edit Invoke** dialog, specify the following:

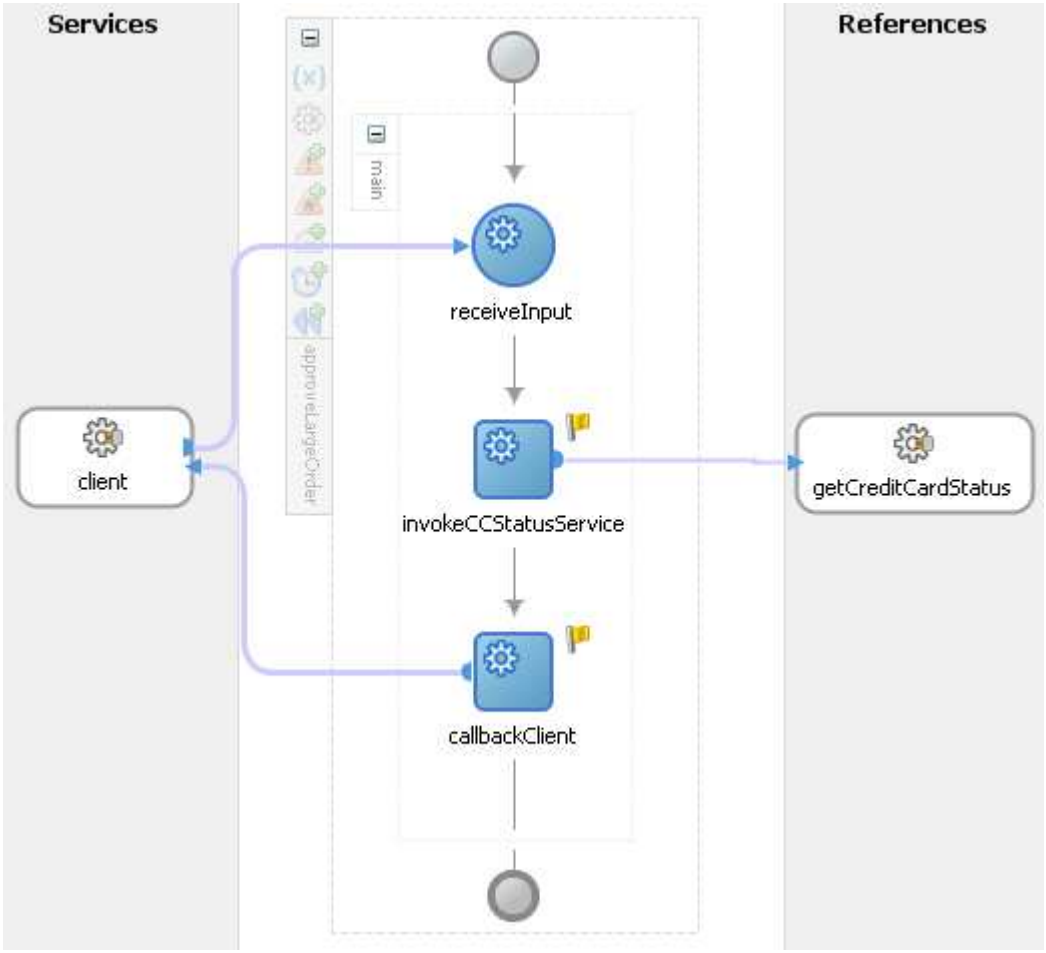
   - **Name:** invokeCCStatusService
   - **Input Variable:** Click the green plus icon, then press **OK** to create a new global variable, accepting the default name and type.

     This variable contains the data that will be sent to the service, or the input to the service.
   - **Output Variable:** Click the green plus icon, then press **OK** to create a new global variable, accepting the default name and type.

     This variable contains the data that will be returned by the service, or the output of the service.



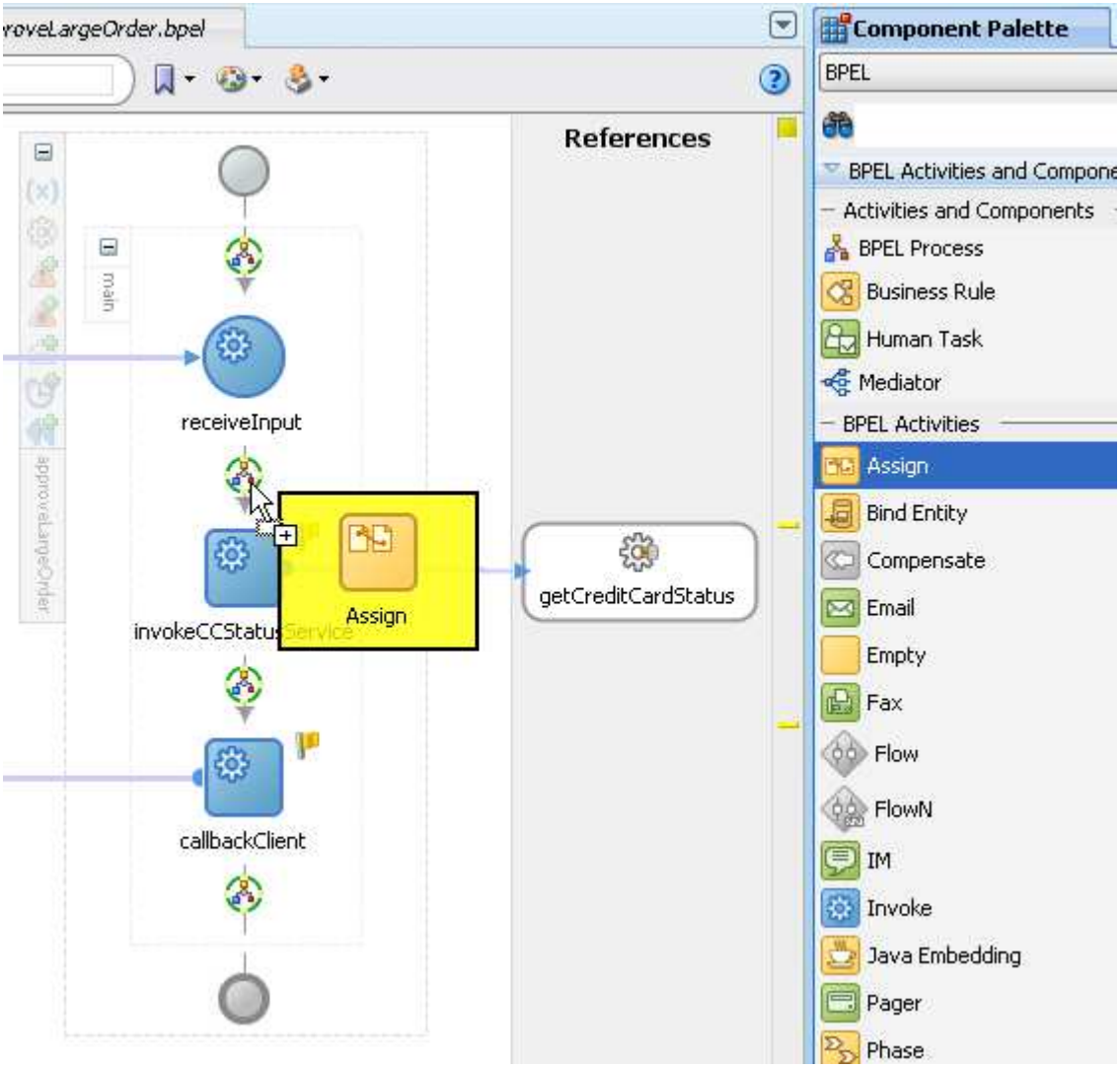
5. Click **OK**.
6. Your BPEL process looks like this so far:



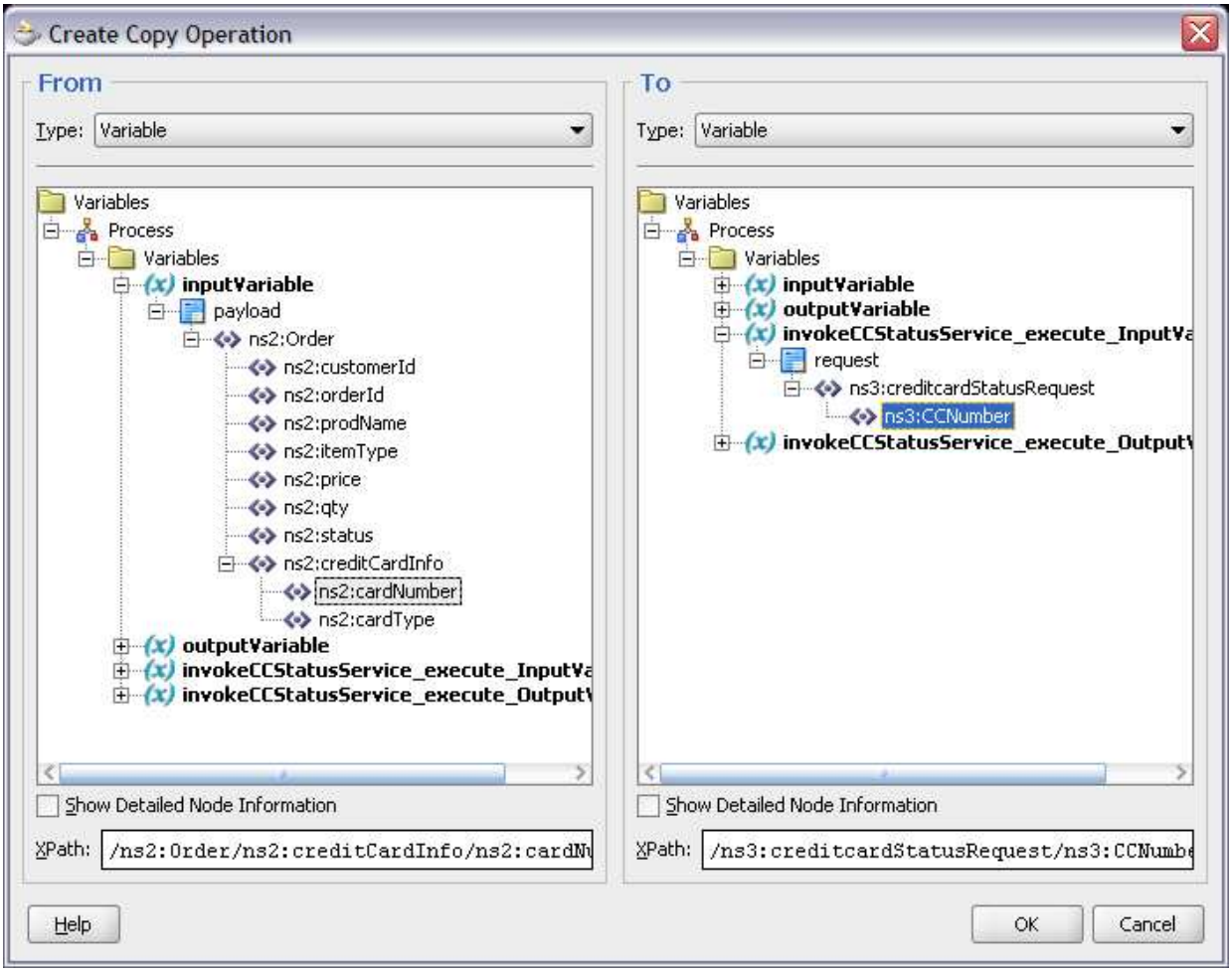
7. We have created the variables that will be used when interacting with the **getCreditCardStatus** service, but they haven't been populated. The output variable will automatically be populated when the service returns a result, but you need to populate the input variable yourself that's going to be passed to the service.

   In BPEL, you use an Assign activity to assign data to a variable. In this case you want to assign the credit card number that passed into the **POProcessing** service to the **getCreditCardStatus** service.

   Drag-and-drop an Assign activity above your Invoke activity.



8. Double-click the Assign activity to edit it.
9. Click the **General** tab and change the name to **assignCCNumber**.
10. Click the **Copy Operation** tab.
11. Click the green plus icon and select **Copy Operation** to open the **Create Copy Operation** dialog, and specify the following.
    - In the From side, select **Variables > Process > Variables > inputVariable > payload > Order > creditCardInfo > cardNumber**
    - In the **To** side, select **Variables > Process > Variables > invokeCCStatusService_execute_InputVariable > request > creditcardStatusRequest > CCNumber**

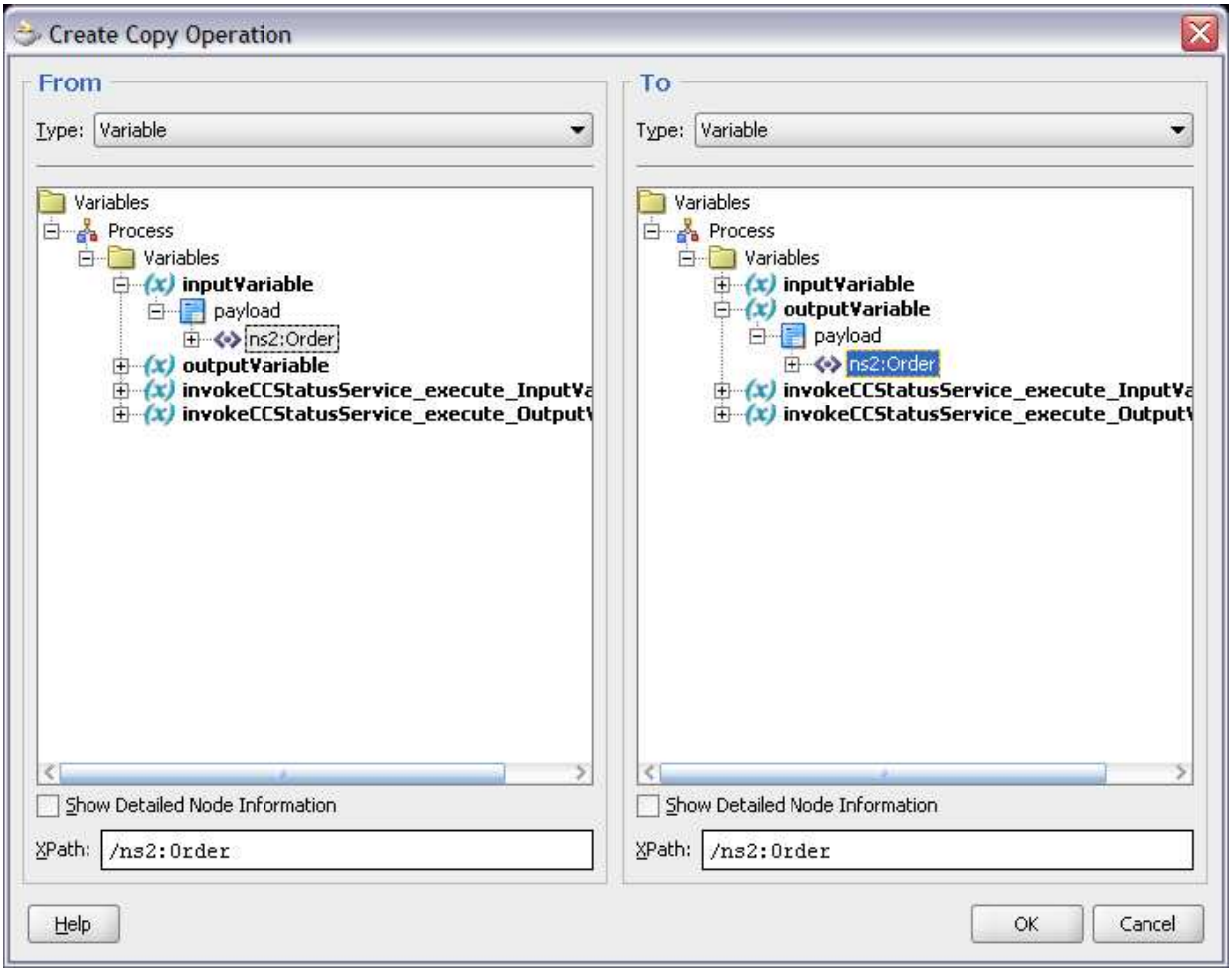


12. Click **OK**.

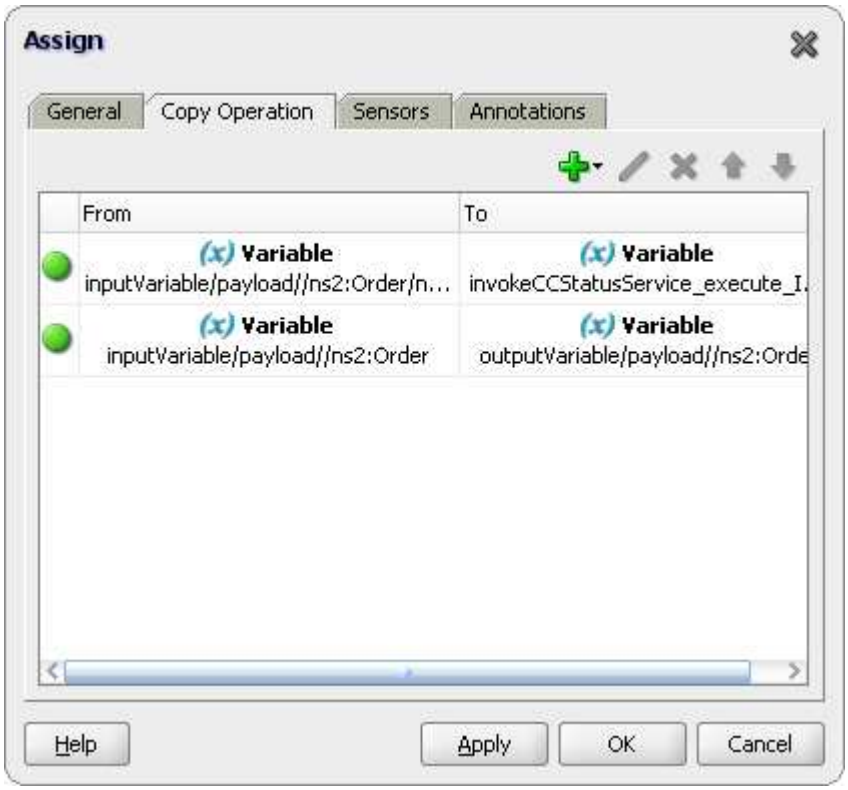
13. Back in the Assign dialog, add a second copy operation by click the green plus icon and selecting **Copy Operation**, and specify the following.

    - In the From side, select **Variables > Process > Variables > inputVariable > payload > Order**
    - In the **To** side, select **Variables > Process > Variables > outputVariable > payload > Order**

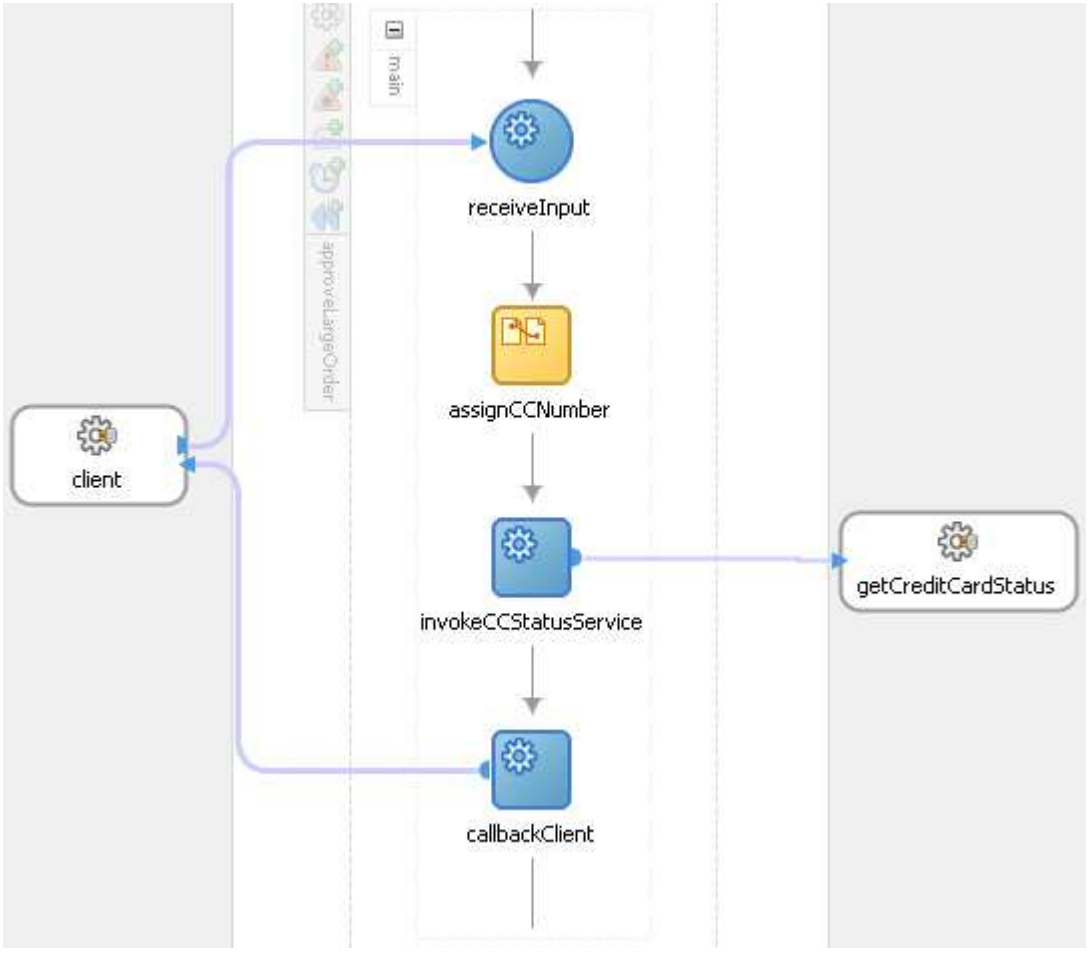
    You are doing this because BPEL process will return the input data, but with some updates which will be made later in the BPEL process.



14. The **Assign** dialog now looks like this:



15. Click **OK** to return to the BPEL process which now looks like this:



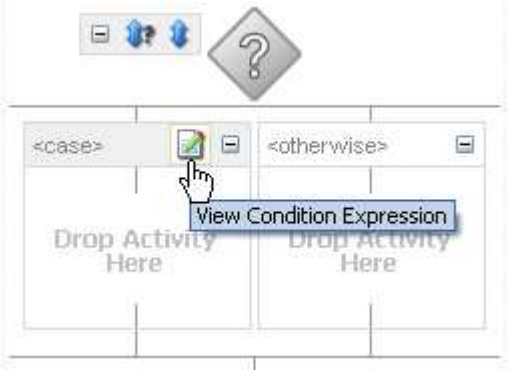
16. The input variable to **getCreditCardStatus** is now populated. The Invoke activity will pass that data to the service. The next step is to process the return data from the service, the output.

    Drag-and-drop a Switch activity underneath the **invokeCCStatusService Invoke** activity.

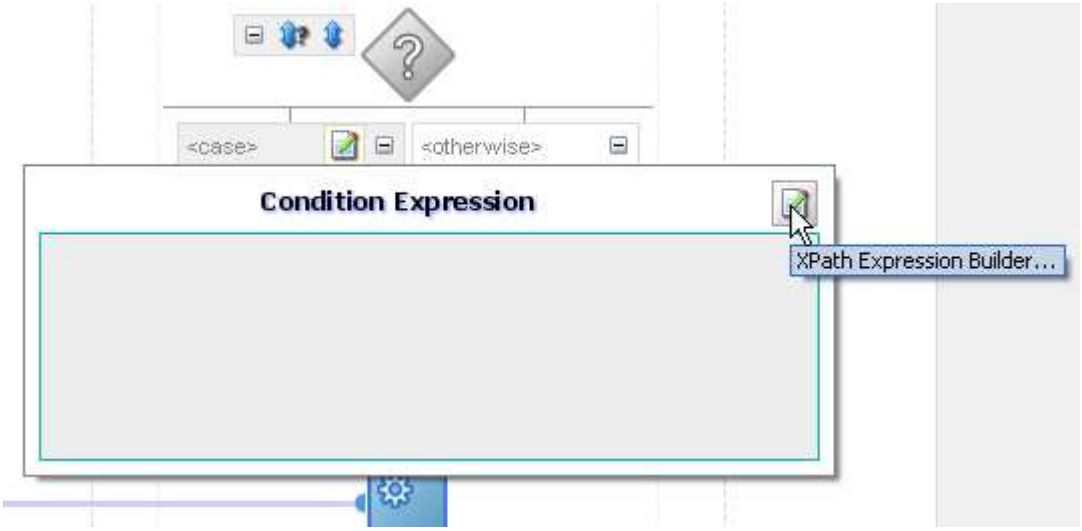
17. Double-click the name of the Switch underneath the icon (which is probably something like **Switch_1**) and rename it to **EvalulateCCStatus**. Note: You can also double-click the Switch icon and change the name in the subsequent dialog, but if you double-click the text itself you can change the name in-place.

18. Expand the Switch by clicking the small plus icon next to it.

19. Click the **View Condition Expression** button.

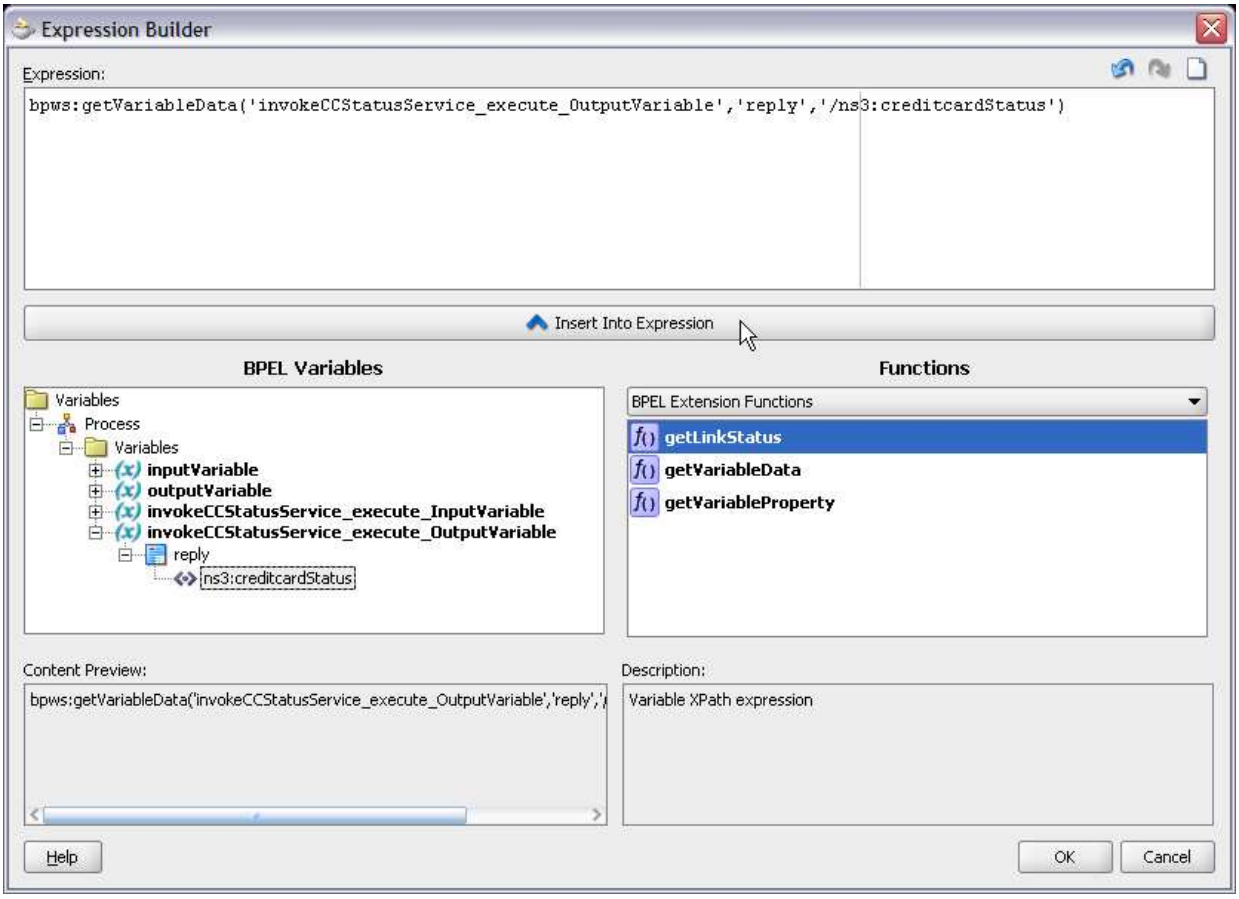


20. Click the **XPath Expression Builder** button.

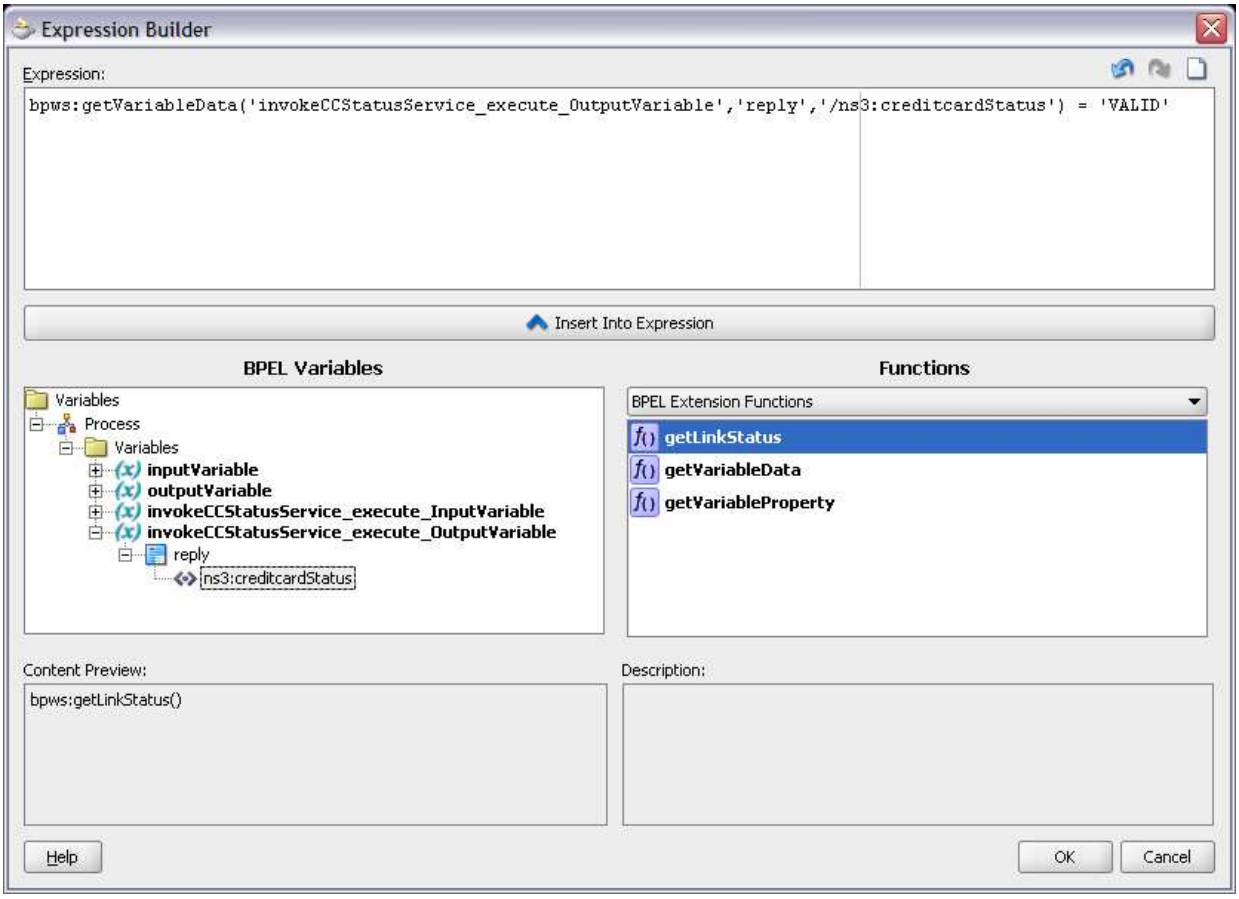


21. In the **BPEL Variables** field, expand **Variables > Process > Variables > invokeCCStatusService_execute_OutputVariable > reply** and select **creditCardStatus**.

22. Click the **Insert Into Expression** button (it's the wide button under the **Expression** field).

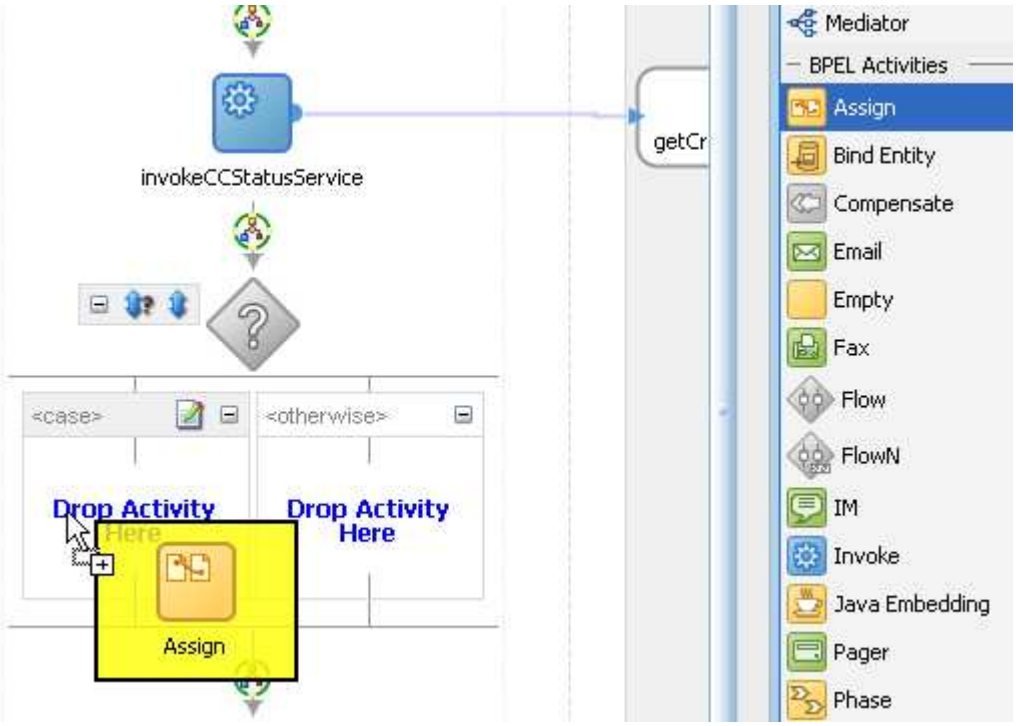


23. Put your cursor in the **Expression** field and at the endand add: **= 'VALID'**

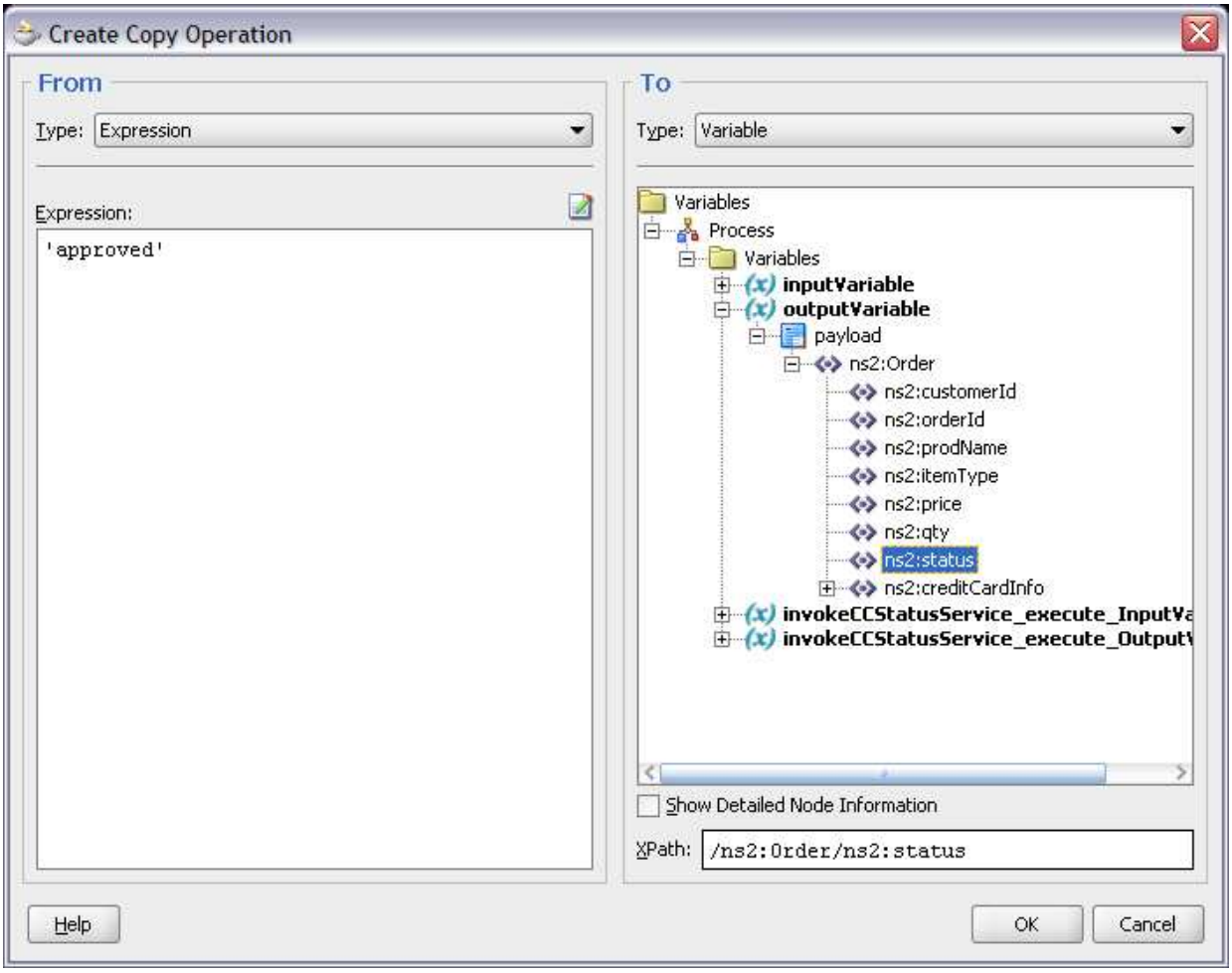


24. Press **OK**.
25. Click outside the **Condition Expression** popup to close it.
26. If that condition is true, then BPEL will execute any activities in the **<case>** part of the switch. If not, any activities in the **<otherwise>** section will be executed.

    Drag-and-drop an Assign activity into the <case> section of the Switch.

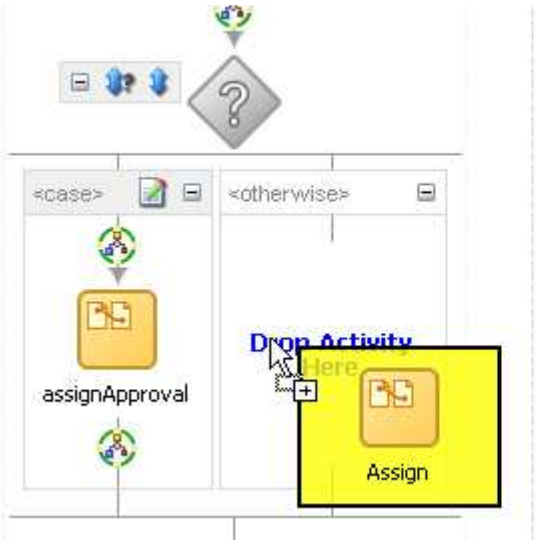


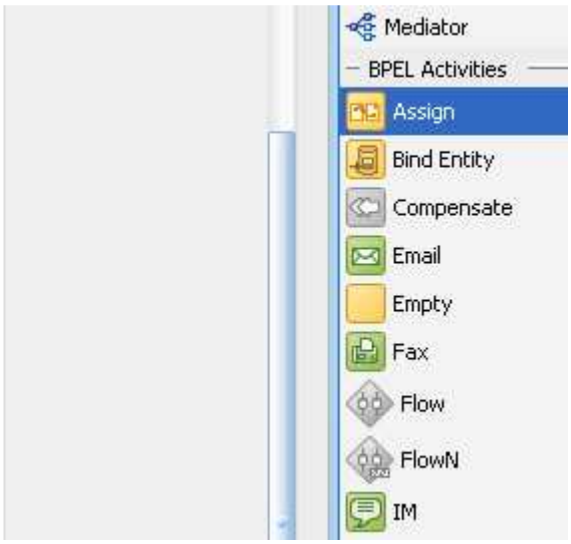
27. Double-click the name of the Assign (which will be something like **Assign_2**) and rename it to **assignApproval**.
28. Double-click the Assign icon to open the **Assign** dialog.
29. Click the green plus icon and add a new copy operation.
30. In the **From** section, change the **Type** poplist to **Expression**.
31. In the **Expression** field, type: 'approved'.
32. In the **To** section, select **Variables > Process > Variables > outputVariable > payload > Order > status**.



33. Click **OK**.

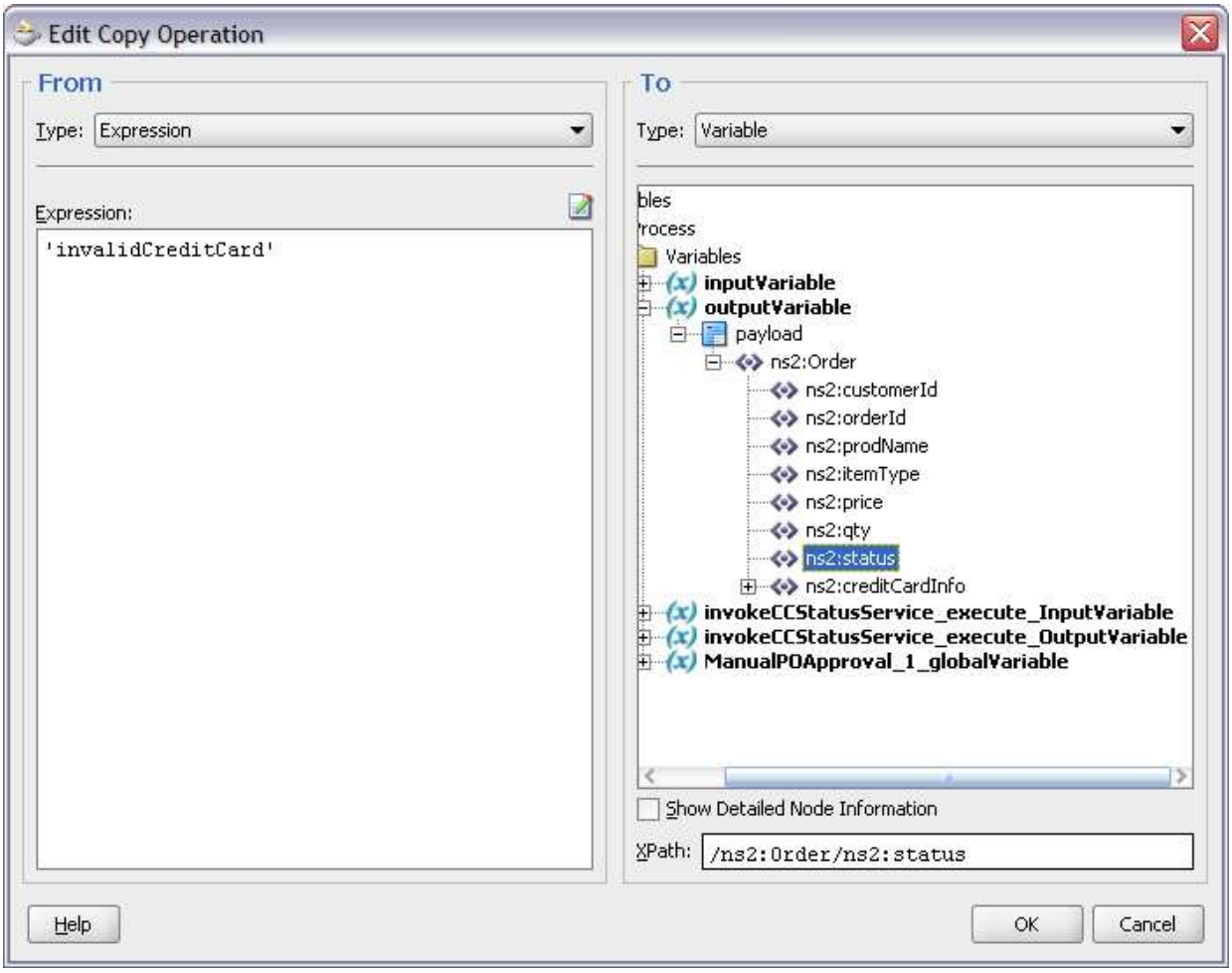
34. Drag-and-drop an Assign activity into the <otherwise> section of the Switch.



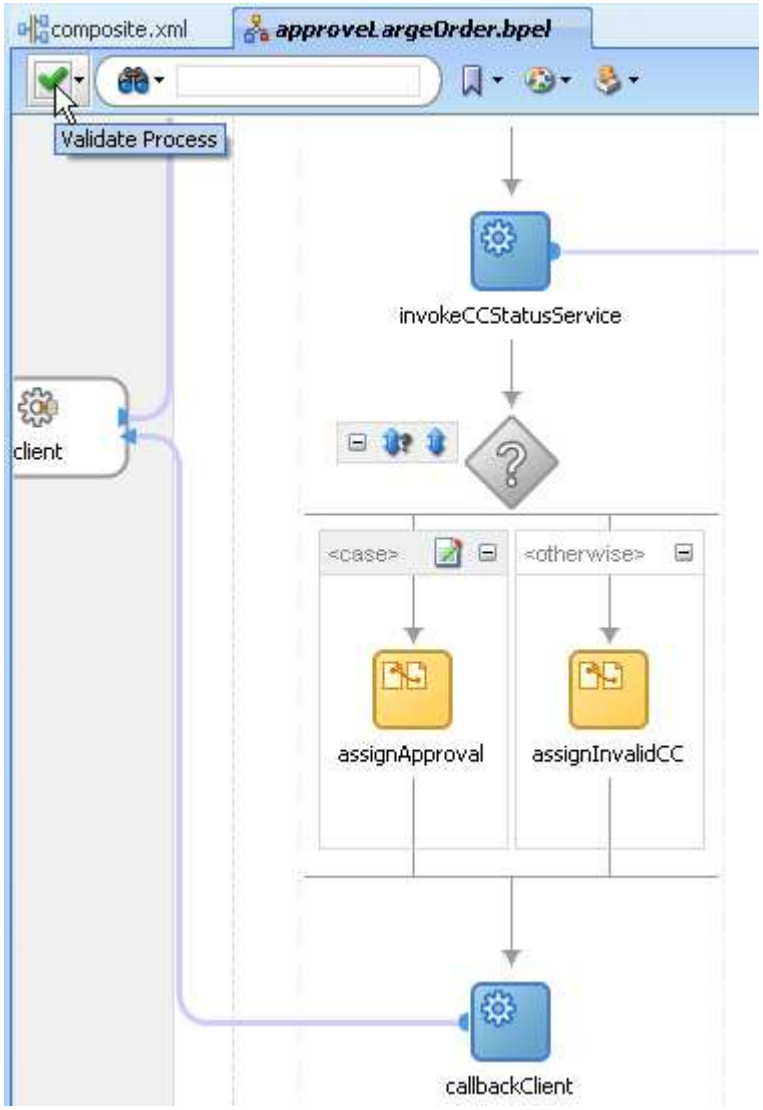


35. Rename it to **assignInvalidCC**.

36. In the same way you just did, assign the value '**invalidCreditCard**' to the **status** field of the **outputVariable** variable.



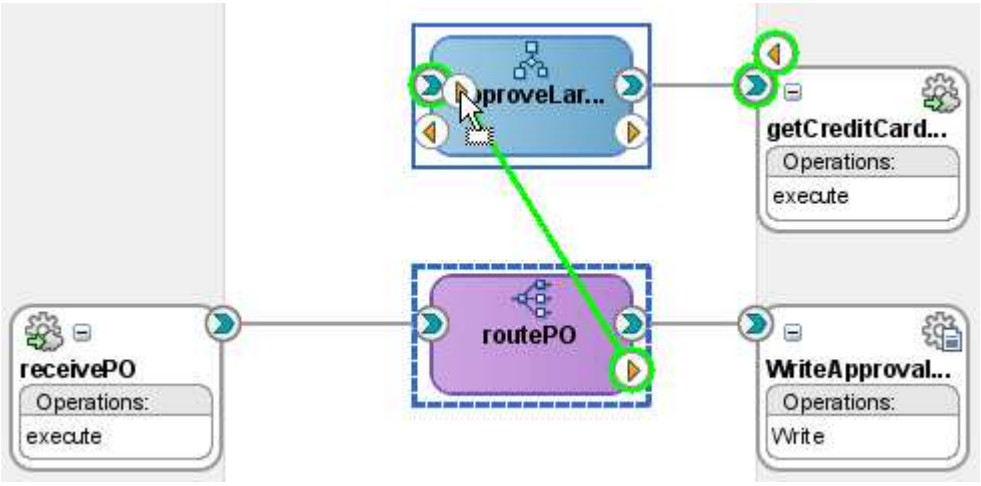
37. At the top of BPEL designer, click the green check mark to validate your process. Any yellow flags you had should disappear and you should not have any warning messages.



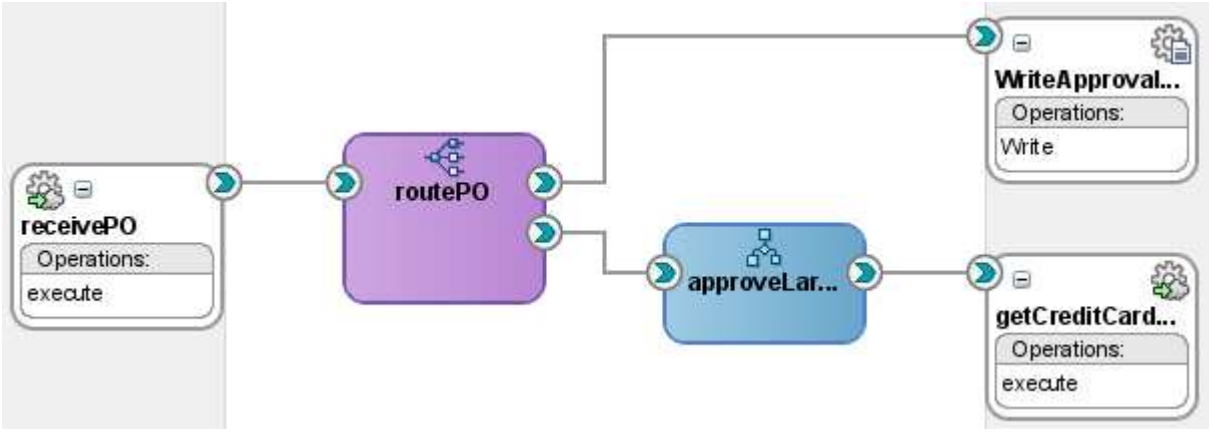
38. Save the BPEL process and close the window to return to the composite.

4.2.3 Modifying the Mediator component

1. Wire the Mediator to the BPEL process.



2. Your composite now looks like this:

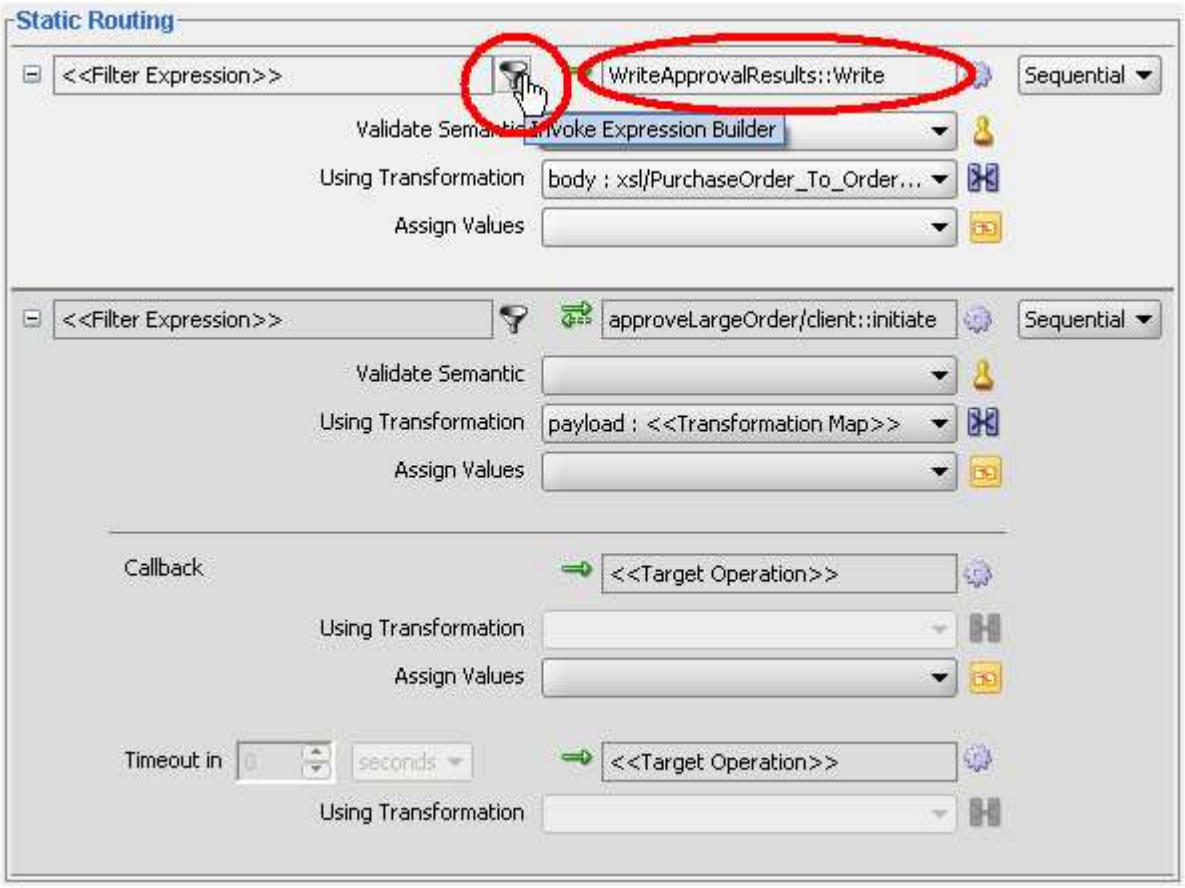


3. Now the Mediator is routing requests to both the **WriteApprovalResults** service and the **approveLargeOrder** BPEL process. Sometimes this is what you want a Mediator service to do, but in this case a new order should either be automatically approved or have to be approved by going through the **approveLargeOrder** process.

   If you recall, orders under $1,000 should be automatically approved while orders greater than or equal to $1,000 need to go through an approval process. The Mediator is capable of creating a content-based routing service to enable this kind of processing.

   Double-click the **routePO** Mediator component to open the Mediator editor.

4. Click on the filter icon, called **Invoke Expression Builder** which looks like a funnel, for the **WriteApprovalResults::Write** request operation.

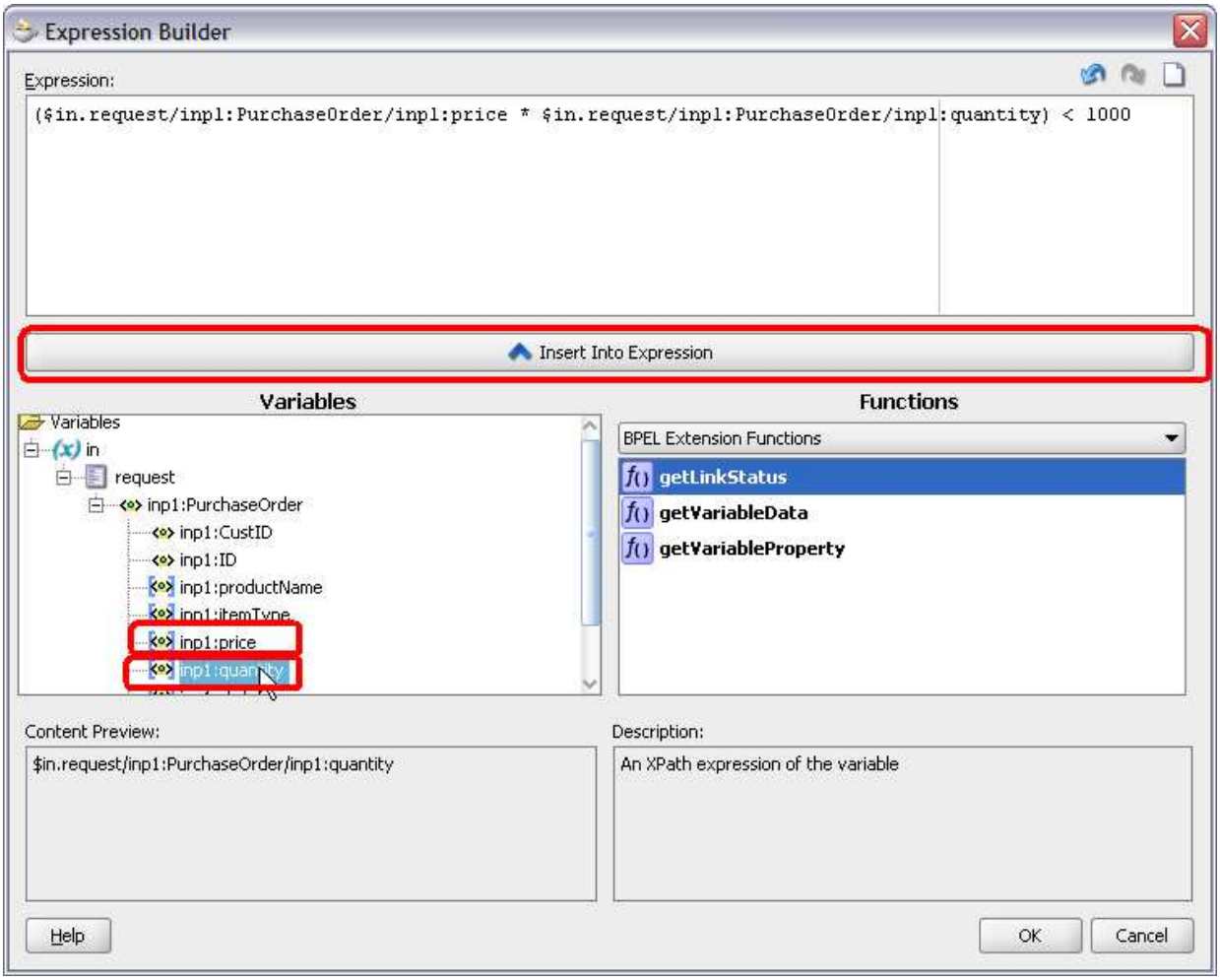


5. In the **Expression Builder** dialog, build up the following expression:

```
($in.request/inp1:PurchaseOrder/inp1:price *
$in.request/inp1:PurchaseOrder/inp1:quantity) < 1000
```

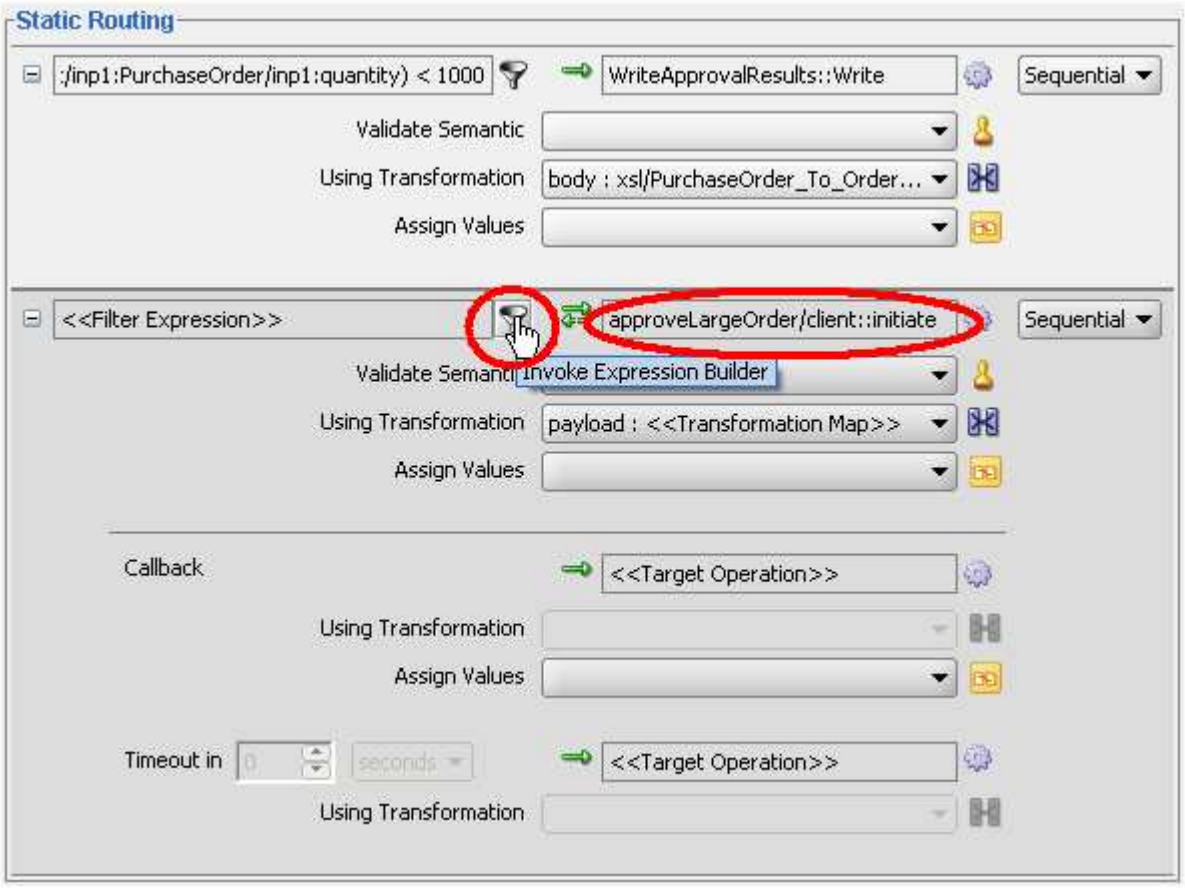
Note: The namespaces (e.g., **inp1**) may be different for you, but you can ignore that.

Hint: Expand the nodes in the **Variables** section to find the field you want and press the **Insert Into Expression** button to add them.



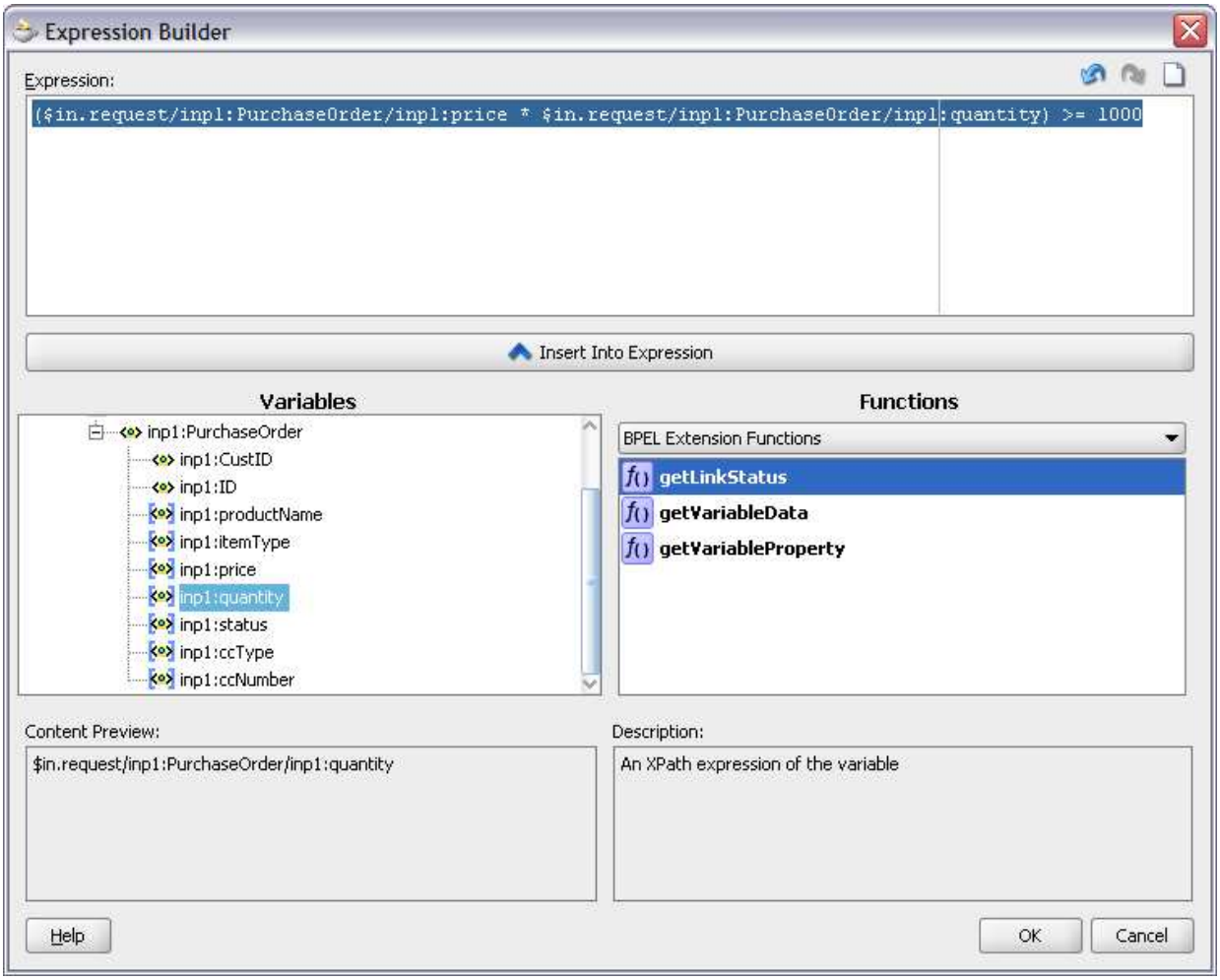
6. Click **OK**.

7. Similarly, click the filter icon for the request invocation of **approveLargeOrder/client::initiate**.



8. Add the following expression:

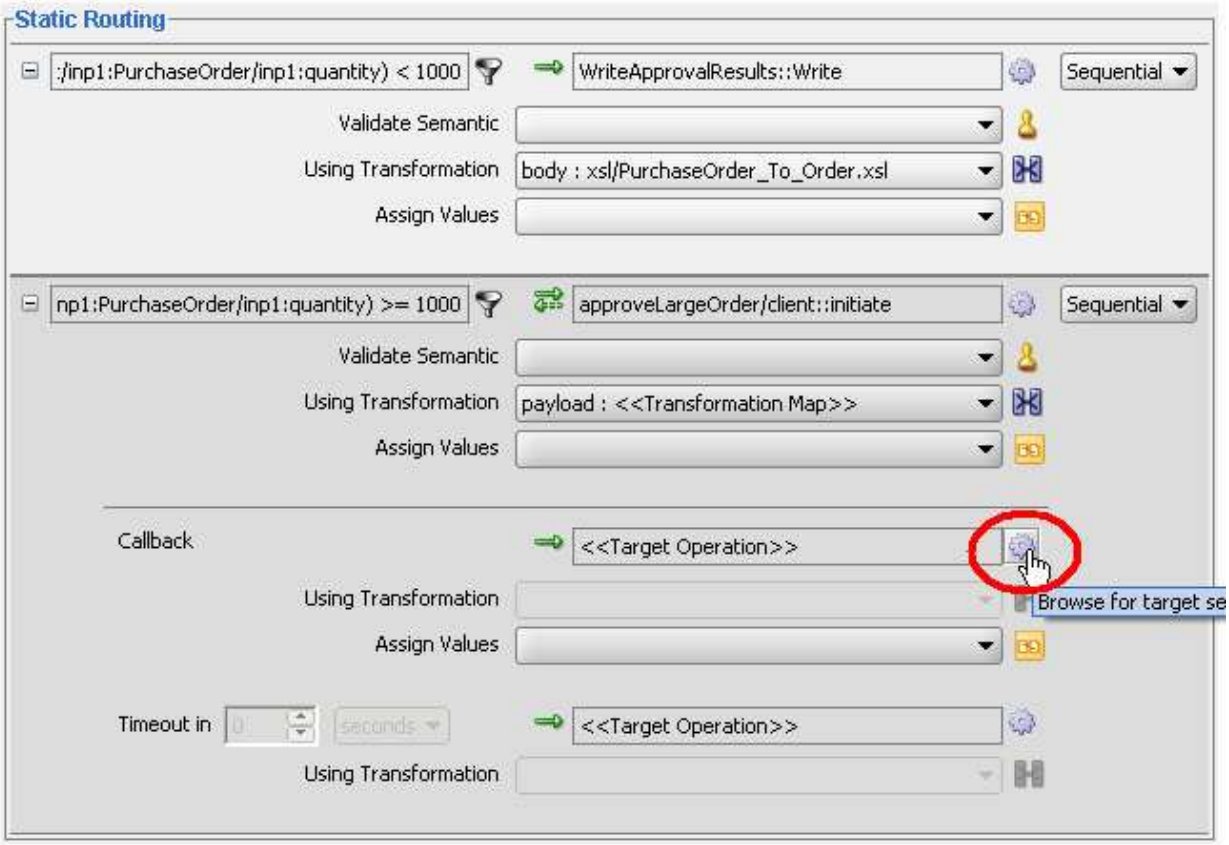
```
($in.request/inp1:PurchaseOrder/inp1:price *
$in.request/inp1:PurchaseOrder/inp1:quantity) >= 1000
```



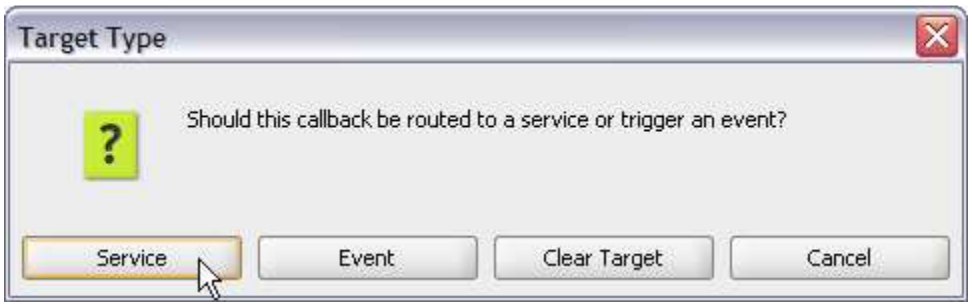
9. Click **OK**.

10. You also need to set the callback of the sysnchronous BPEL process to call the file adapter service.

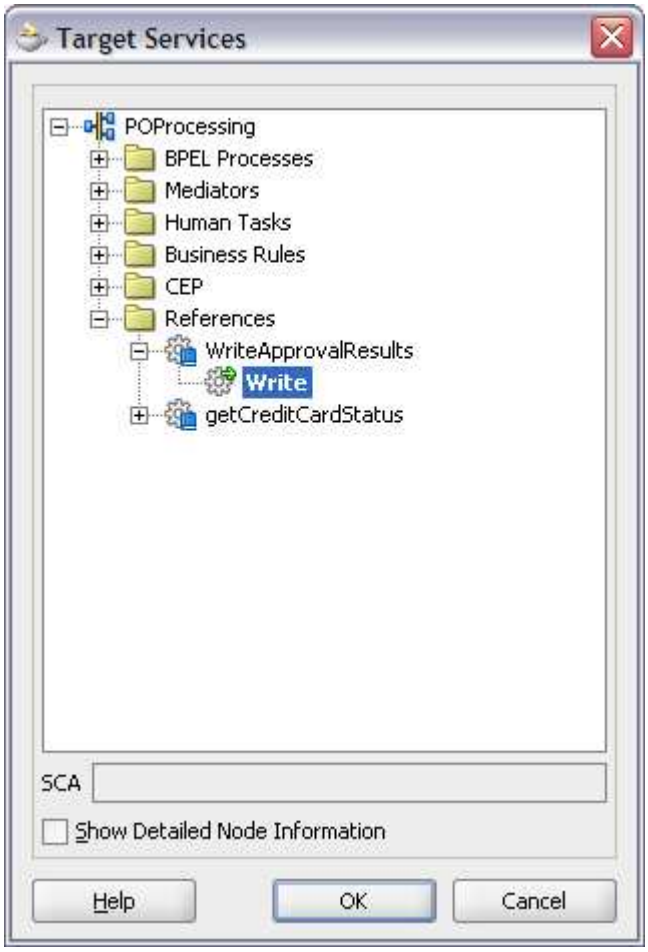
    Click the cog icon next to the **Target Operation** field in the callback section.



11. In the **Target Type** dialog, click the **Service** button.



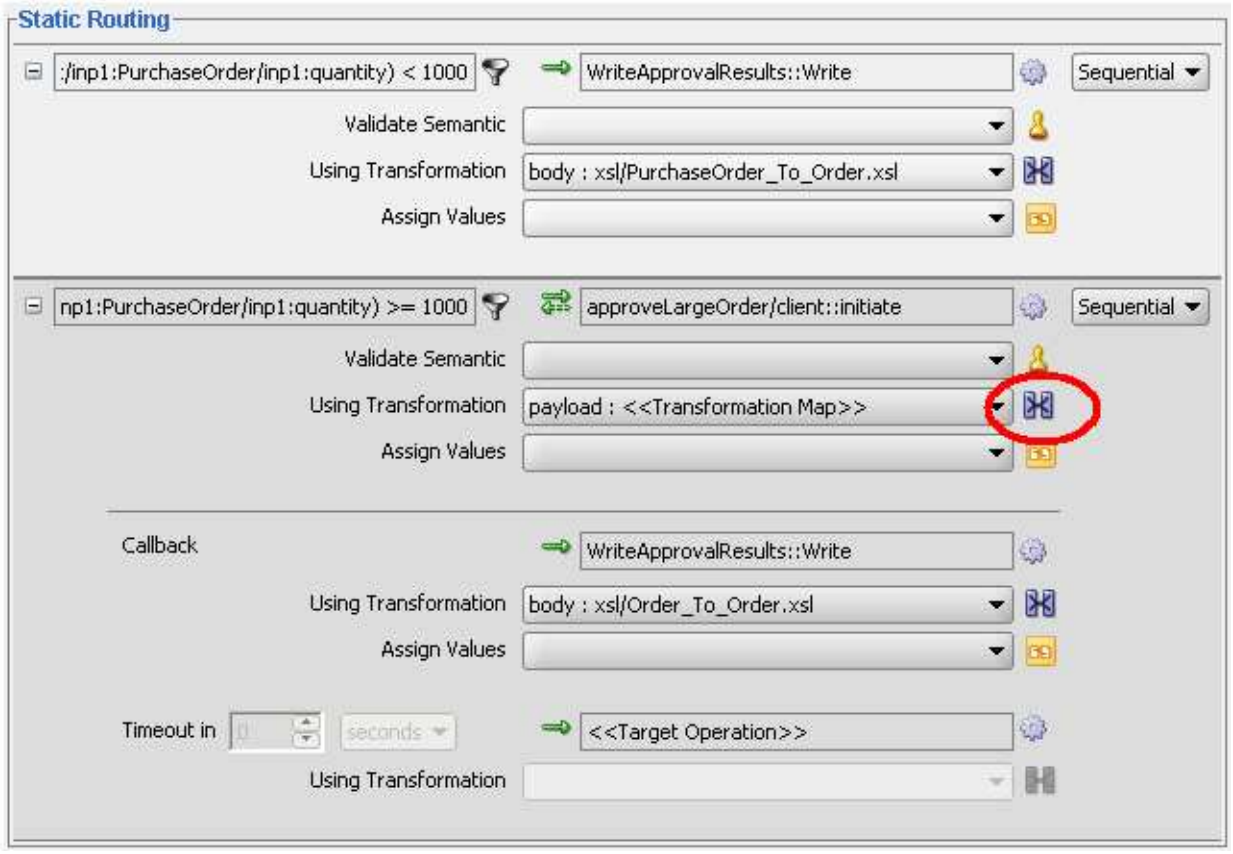
12. In the **Target Services** dialog, select **POProcessing > References > WriteApprovalResults > Write**.



13. Click **OK**.

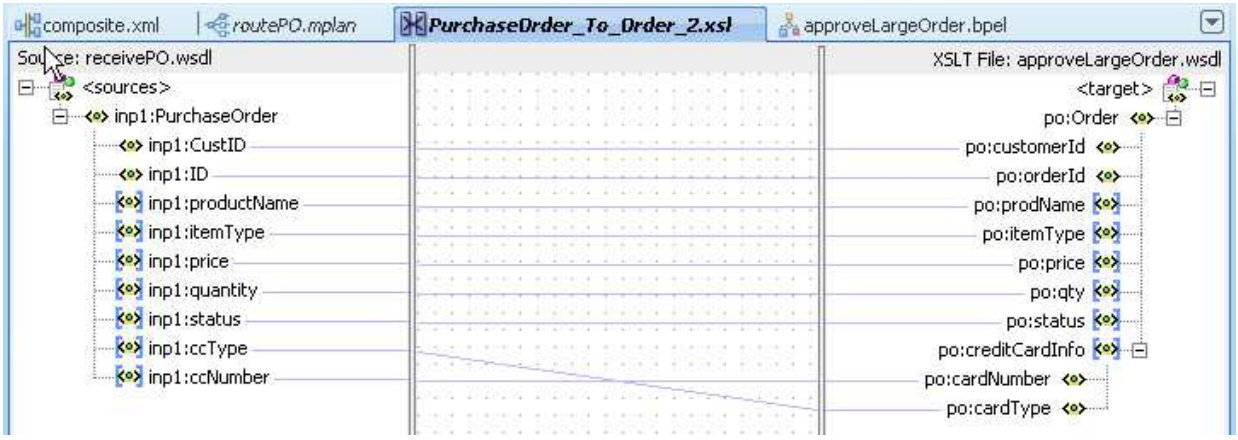
14. A transformation needs to be added for this operation. It's the same as the transformation done earlier, but the namespaces are different so a new transformation will need to be created.

    Click the transformation icon in the request section.

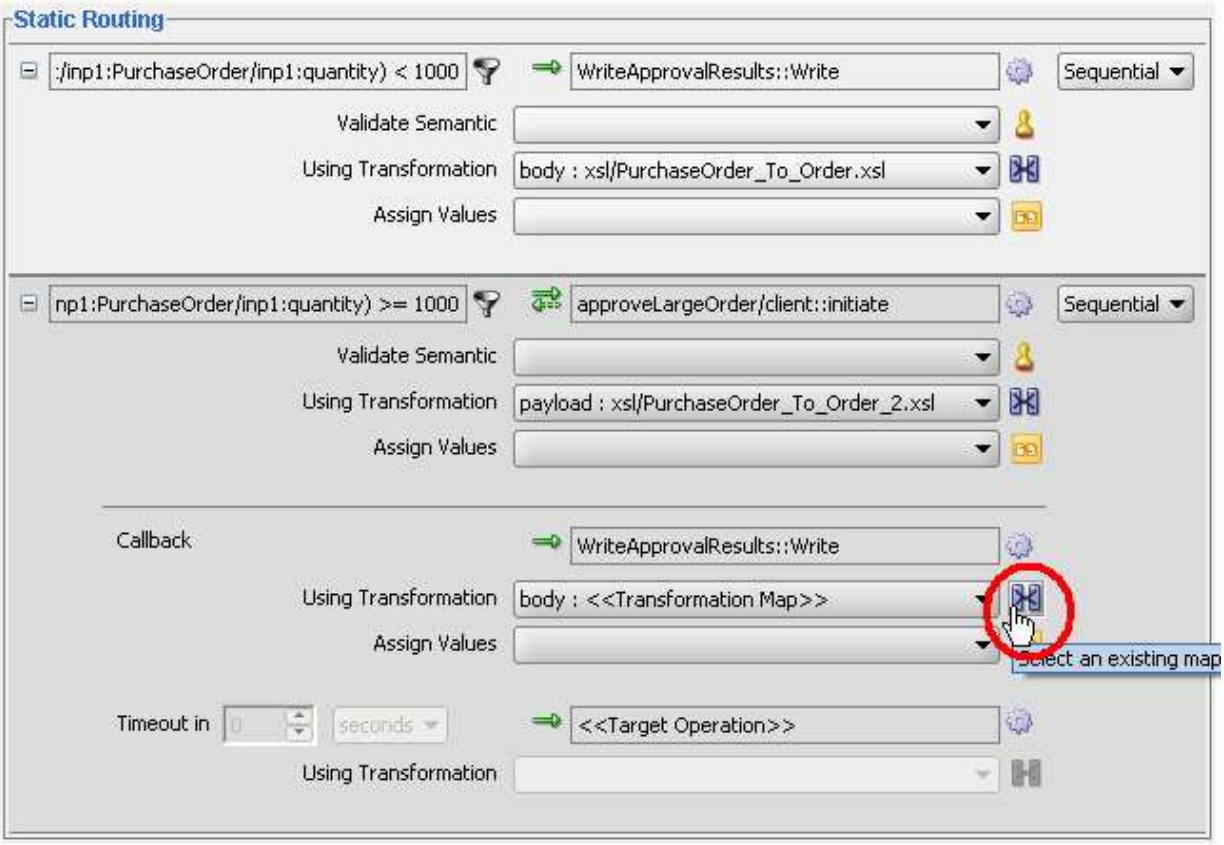


15. Select Create New Mapper File and click OK.
16. Drag-and-drop **PurchaseOrder** from the source to **Order** in target.
17. In the **Auto Map Preferences** dialog, click **OK** since you already added the dictionary earlier.

    The resulting transformation looks like this:

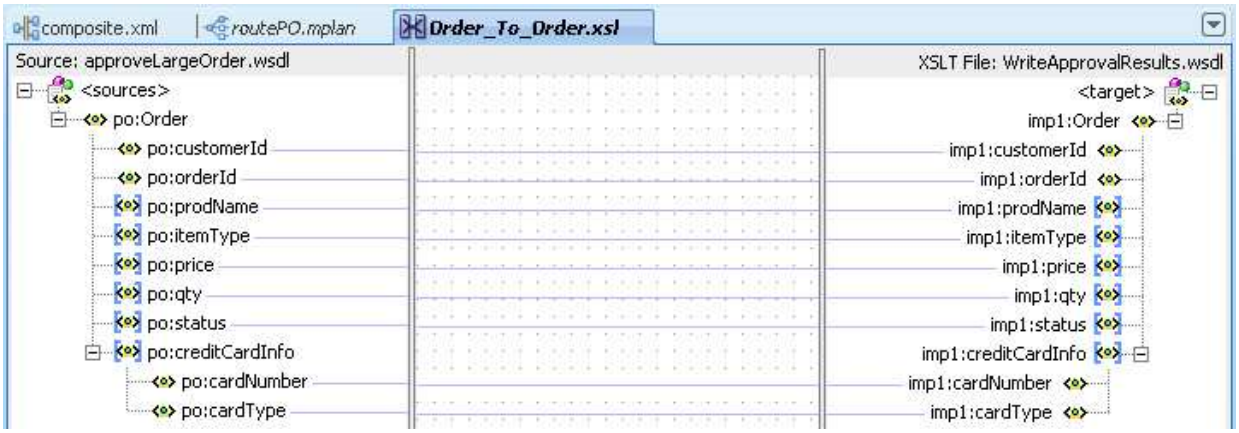


18. Save and close the mapper to return to the Mediator editor.
19. You must also add a transformation for the callback. Click the transformation icon in the callback section.



20. Select **Create New Mapper File** and click **OK**.
21. Drag-and-drop **Order** from the source to **Order** in the target.
22. In the **Auto Map Preferences** dialog, click **OK**.

    The resulting transformation looks like this:



23. Save and close the mapper.
24. Save and close the Mediator editor to return to the composite.
25. Choose **File > Save All** from the menu or the toolbar just to make sure everything is saved.

4.3 Deploying the application

Deploy the application in the same way as before using the **Deploy** command on the Application Menu. This time you can select the connection you created earlier instead of creating a new one.

Read **Appendix A Deploying and Running a Composite Application** to refresh your memory on how to deploy if you need to.

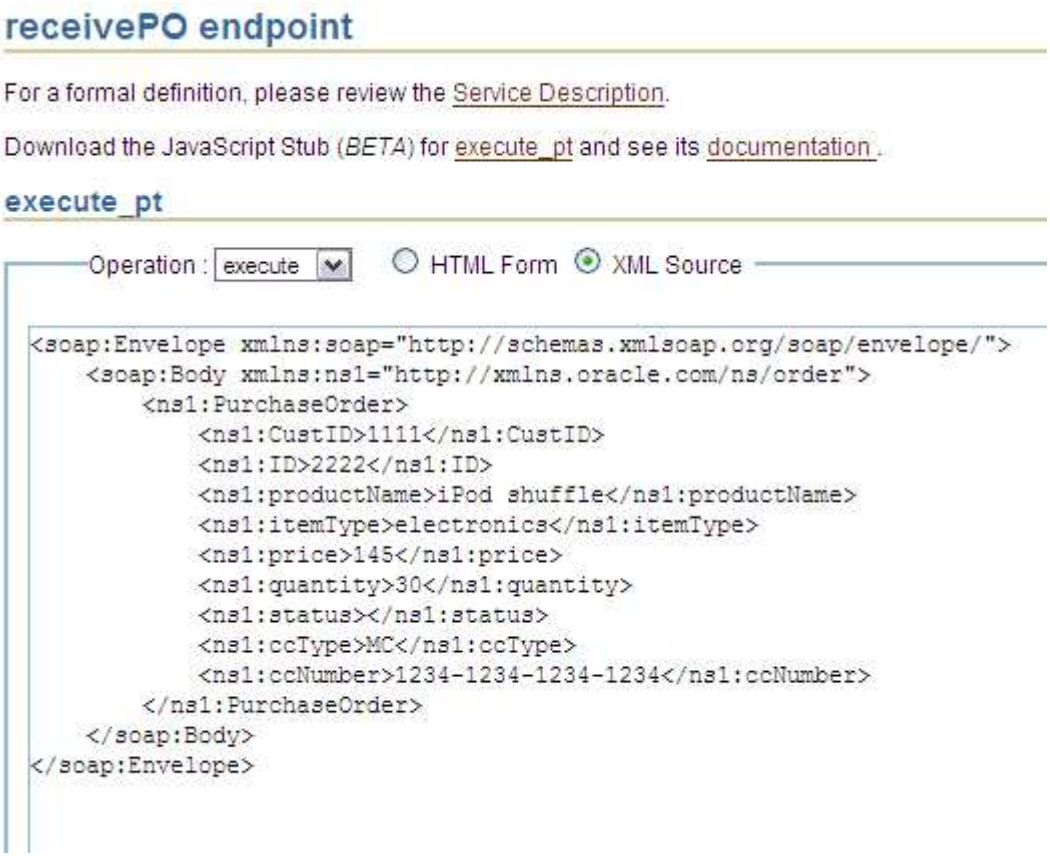
4.4 Testing the application

1. In the SOA Console, click on the **POProcessing** application and then open the tester.
2. Click **XML Source**.
3. In the previous chapter you submitted a small order which created an order file directly. This time you'll create a large order which the Mediator will route to the BPEL approval process.

   Open the following file in a text editor:

   `c:\po\input\po-small-Headsetx1.xml`

   Copy the entire contents and paste them into the large text field in your browser:



4. Click **Invoke**.
5. As before, the **Test Result** screen won't have any response because this is a one-way invocation with no reply or callback. However a new `order_n.txt` file will have

been created in `c:\temp`. You can open it in a text editor and view the results (shown in Figure 1). Notice that the value of <status> on line 8 has been set to **approved**.

Figure 1 Output from order_n.txt file

```
1 <?xml version="1.0" ?><imp1:Order xmlns:imp1="http://xmlns.oracle.com/ns/order">
2    <imp1:customerId>1111</imp1:customerId>
3    <imp1:orderId>2222</imp1:orderId>
4    <imp1:prodName>iPod shuffle</imp1:prodName>
5    <imp1:itemType>electronics</imp1:itemType>
6    <imp1:price>145</imp1:price>
7    <imp1:qty>30</imp1:qty>
8    <imp1:status>approved</imp1:status>
9    <imp1:creditCardInfo>
10      <imp1:cardNumber>1234-1234-1234-1234</imp1:cardNumber>
11      <imp1:cardType>MC</imp1:cardType>
12   </imp1:creditCardInfo>
13 </imp1:Order>
```

6. Press the back button in the browser to return to the tester.
7. Re-run the application using the same input data from , but this time change the credit card number to **4321-4321-4321-4321** which represents an invalid credit card.

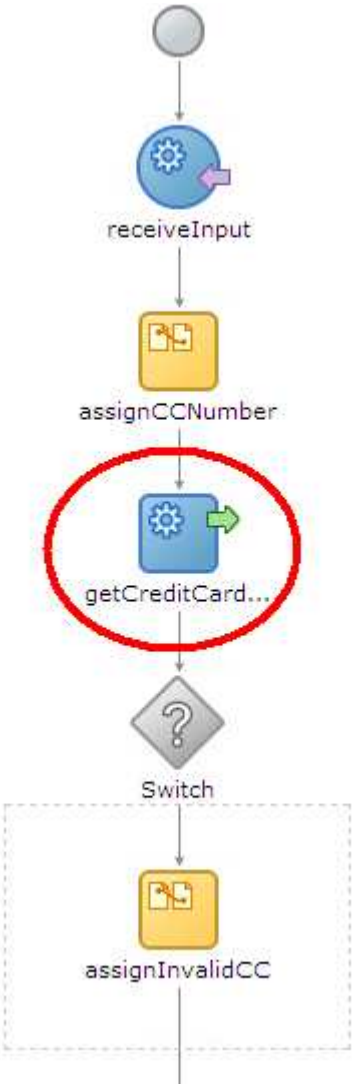
8. Click **Invoke** to run the application.
9. Open the new order file in `c:\temp` and notice what the status is this time. This is the result of the <switch> statement in your BPEL process.

```
<?xml version="1.0" ?><imp1:Order xmlns:imp1="http://xmlns.oracle.com/ns/order">
   <imp1:customerId>1111</imp1:customerId>
   <imp1:orderId>2222</imp1:orderId>
   <imp1:prodName>iPod shuffle</imp1:prodName>
   <imp1:itemType>electronics</imp1:itemType>
   <imp1:price>145</imp1:price>
   <imp1:qty>80</imp1:qty>
   <imp1:status>invalidCreditCard</imp1:status>
   <imp1:creditCardInfo>
      <imp1:cardNumber>4321-4321-4321-4321</imp1:cardNumber>
      <imp1:cardType>MC</imp1:cardType>
   </imp1:creditCardInfo>
</imp1:Order>
```

10. Close the tester window. Back in the SOA Console click on the application or click the **Refresh** link. In the **Last 5 Instances** section click on one of the most recent instances to see the execution flow.

11. You can click the approveLargeOrder link to look at the BPEL process instance, then click the Flow-Debug tab to see a visual representation of the BPEL instance that ran (as shown in Figure 2). You can click the various activities to see their results. An interesting one to click is the getCreditCardStatus invoke activity.

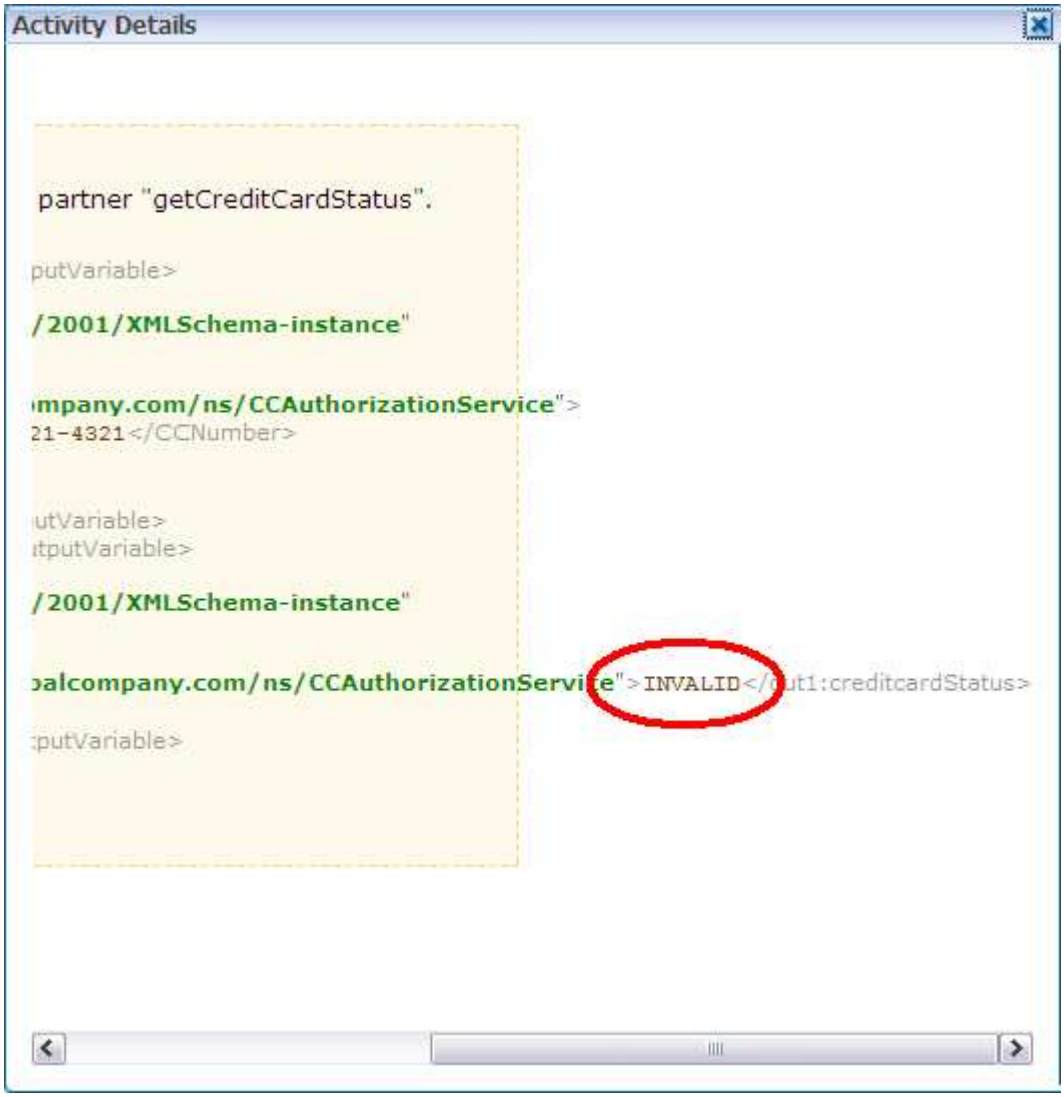
Figure 2 Visual flow of the BPEL process



12. When you click the **getCreditCardStatus** invoke activity, it's a synchronous (request-response) call, so you see both the input to the service (i.e., what you're passing to the service) and the output (i.e., what you're getting back from the service).

    In the screenshots, below, the input to the service is the credit card number **4321-4321-4321-4321** and the output returned is **INVALID**.


Activity Details
getCreditCardStatus
[2007/12/13 11:14:44]
Invoked 2-way operation "execute" on partner "getCreditCardStatus".
Input
- <messages>
- <getCreditCardStatus_execute_InputVariable>
- <part
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
name="request">
- <creditcardStatusRequest
xmlns="http://www.globalcompany.com/ns/CCAuthorizationServ
<CCNumber>4321-4321-4321-4321</CCNumber>
</part>
</getCreditCardStatus_execute_InputVariable>
- <getCreditCardStatus_execute_OutputVariable>
- <part
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
name="reply">
<out1:creditcardStatus
xmlns:out1="http://www.globalcompany.com/ns/CCAuthorization
</part>
</getCreditCardStatus_execute_OutputVariable>
</messages>
Output
Copy details to clipboard
Scroll right to see the output results



13. Back in the main screen of the SOA Console, run the tester again and this time use the small order that you used in the previous chapter, which can be found in `c:\po\input\po-small-Headsetx1.xml`

    Note that the BPEL process does not get invoked and instead you only get a new order file generated in `c:\temp`.