D Fault Handling

D.1 Introduction

Note: The solution for this chapter can be found in c:\po\solutions\apD

To run this solution, you must have completed labs in chapter 9. Alternatively you can follow the setup in Chapter 1 and use the solution from Chapter 9 located at c:\po\solutions\ch9

This lab exercise will give you a brief introduction to handling exceptions in a SOA composite.

Oracle SOA Suite provides sophisticated error handling capabilities that enable you to define fault handling easily at various levels in a composite. It allows you to handle both system generated errors, called system faults as well as application generated ones, called business faults.

The BPEL specification provides rich fault handling constructs for catching exceptions and acting on them. The Mediator on the other hand provides no such capability. Additionally, in BPEL, the code can get quite complex with fault handling code duplicated for common exceptions.

Oracle SOA Suite provides a policy-based fault handling mechanism which allows you to define how faults are handled. The policies can be bound to either the composite as a whole or can also be associated with individual components.

In this lab you will implement very simple exception handling using both the BPEL fault handling constructs as well as policy-based fault handling.

D.2 Handle Remote Faults

In this lab you will define a fault handling policy for processing a remote fault. To generate a fault you will make the getStatusByCC service unavailable and test it to see how the composite handles the exception. You will then add a fault handling policy to handle the exception

D.2.1 Prerequisites

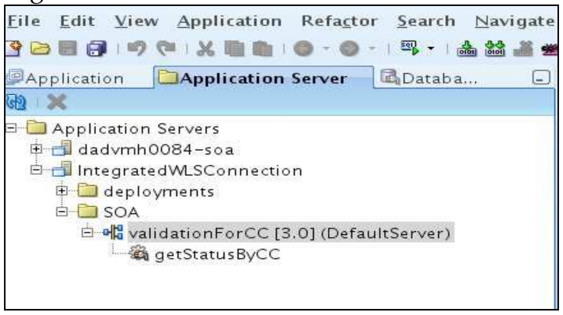
This lab requires Enterprise Manager console to execute some of the steps.

D.2.2 Test service unavailability

2. To simulate an unavailable service, shutdown the **validationForCC** composite. To do that, navigate to the **Application Server Navigator** in JDeveloper (**View→Application Server Navigator,** not **Application Navigator**).

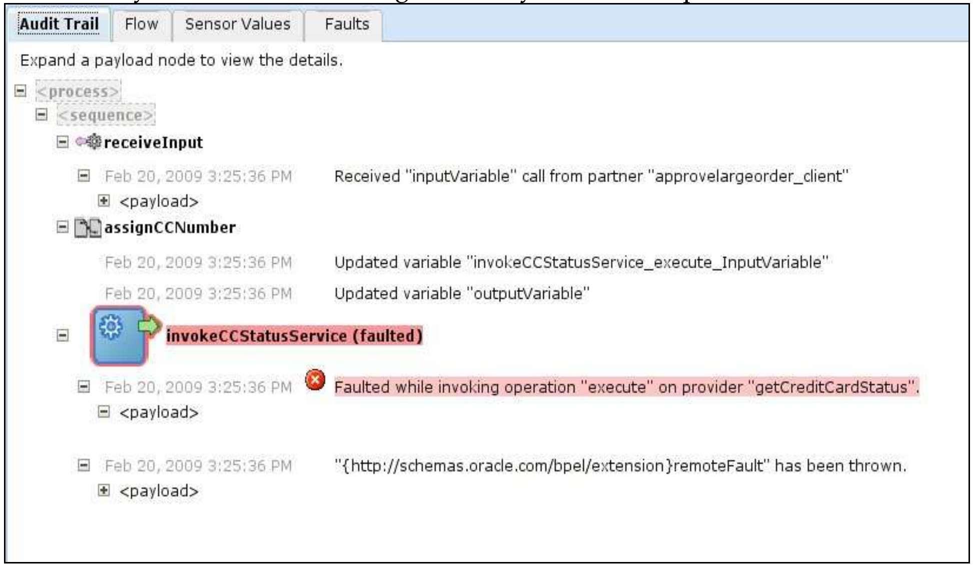
3. Expand the SOA node under the IntegratedWLSConnection and select **validationForCC**

4. Right-click and select **Turn Off**



5. From the EM Console in the web browser invoke the POProcessing composite (receivePO service operation) using the Test button on the POProcessing page. Use the po-large-iPodx30.xml from c:\po\input to submit an order that is over $1000 so the credit card validation is called.

6. In the EM Console, click on the new instance id of the *POProcessing* composite and then click *approveLargeOrder* in the **Flow Trace** to view the details. You should see that the composite has terminated with a remote fault with no way to restart it if you were to make the getStatusByCC service operation available.



Expand a payload node to view the details.

```
<process>
  <sequence>
    receiveInput
      Feb 20, 2009 3:25:36 PM    Received "inputVariable" call from partner "approvelargeorder_client"
        <payload>
    assignCCNumber
      Feb 20, 2009 3:25:36 PM    Updated variable "invokeCCStatusService_execute_InputVariable"
      Feb 20, 2009 3:25:36 PM    Updated variable "outputVariable"
    invokeCCStatusService (faulted)
      Feb 20, 2009 3:25:36 PM    Faulted while invoking operation "execute" on provider "getCreditCardStatus".
        <payload>
      Feb 20, 2009 3:25:36 PM    "{http://schemas.oracle.com/bpel/extension}remoteFault" has been thrown.
        <payload>
```

D.2.3 Add policy-based fault handler to do manual recovery

Add a fault handling policy to catch this exception and make it recoverable via manual intervention.

1. Copy files fault-policies.xml and fault-bindings.xml from c:\po\schemas to POProcessing directory (directory where the composite.xml for POProcessing is located). Click the **Refresh** button in the **Application Navigator** in JDeveloper to have the files show up in the list.

   The fault-bindings.xml file has been pre-created for you and binds the POProcessing composite with a fault policy called *POProcessingFaults* defined in the fault-policies.xml. The fault-policies.xml file is partially complete. You will add fault handlers here as you step through this lab.

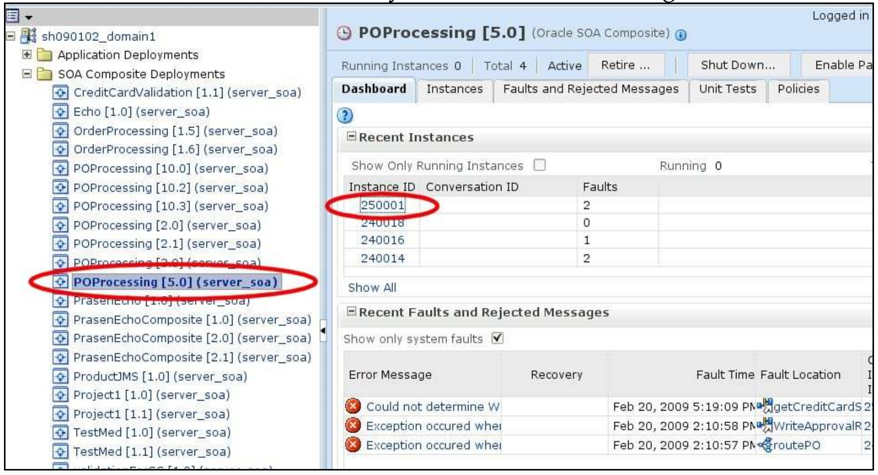
2. Open the fault-policies.xml file in JDeveloper and add the following to handle all remote faults that occur in any component in the POProcessing composite under comment marked *Step D.2.2.2*

```
<faultName xmlns:bpelx=”http://schemas.oracle.com/bpel/extension” name="bpelx:remoteFault">
   <condition>
      <action ref="ora-human-intervention"/>
  </condition>
</faultName>
```

3. Save all and deploy the POProcessing composite.

4. Invoke the POProcessing composite using the web service tester in the EM Console.

5. Select the instance id of the newly invoked POProcessing to view the flow trace


sh090102_domain1
Application Deployments
SOA Composite Deployments
CreditCardValidation [1.1] (server_soa)
Echo [1.0] (server_soa)
OrderProcessing [1.5] (server_soa)
OrderProcessing [1.6] (server_soa)
POProcessing [10.0] (server_soa)
POProcessing [10.2] (server_soa)
POProcessing [10.3] (server_soa)
POProcessing [2.0] (server_soa)
POProcessing [2.1] (server_soa)
POProcessing [3.0] (server_soa)
POProcessing [5.0] (server_soa)
PrasenEcho [1.0] (server_soa)
PrasenEchoComposite [1.0] (server_soa)
PrasenEchoComposite [2.0] (server_soa)
PrasenEchoComposite [2.1] (server_soa)
ProductJMS [1.0] (server_soa)
Project1 [1.0] (server_soa)
Project1 [1.1] (server_soa)
TestMed [1.0] (server_soa)
TestMed [1.1] (server_soa)

Logged in
POProcessing [5.0] (Oracle SOA Composite)
Running Instances 0 | Total 4 | Active | Retire ... | Shut Down... | Enable Pa
Dashboard | Instances | Faults and Rejected Messages | Unit Tests | Policies

Recent Instances
Show Only Running Instances ☐ Running 0

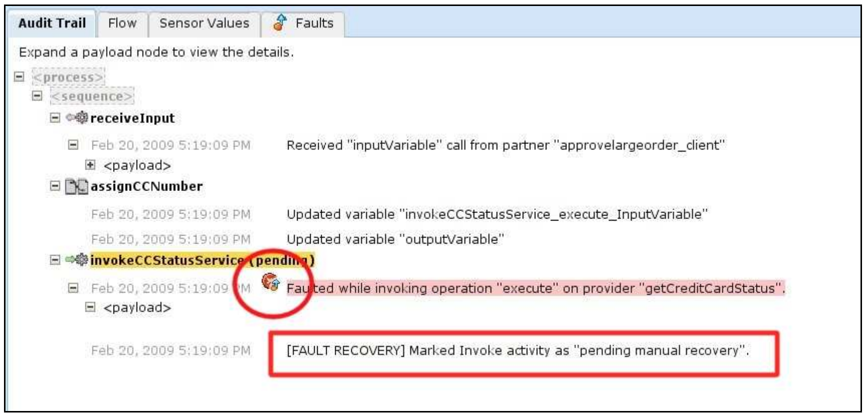
Instance ID	Conversation ID	Faults
250001		2
240018		0
240016		1
240014		2

Show All

Recent Faults and Rejected Messages
Show only system faults ☑

Error Message	Recovery	Fault Time	Fault Location
Could not determine W		Feb 20, 2009 5:19:09 PM	getCreditCardS
Exception occured whe		Feb 20, 2009 2:10:58 PM	WriteApprovalR
Exception occured whe		Feb 20, 2009 2:10:57 PM	routePO

In the flow trace window, click on **approveLargeOrder** to view the BPEL Audit Trail. You will see that the activity has faulted as before but this time it is waiting for a manual recovery. The circled icon indicates that the fault is recoverable.



Audit Trail | Flow | Sensor Values | Faults

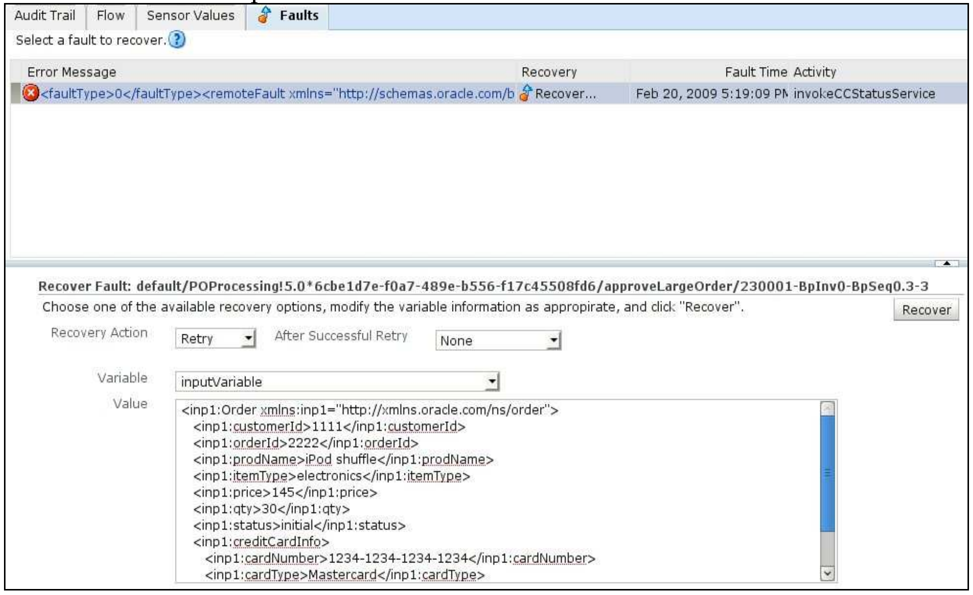
Expand a payload node to view the details.

```
<process>
  <sequence>
    receiveInput
      Feb 20, 2009 5:19:09 PM    Received "inputVariable" call from partner "approvelargeorder_client"
        <payload>
    assignCCNumber
      Feb 20, 2009 5:19:09 PM    Updated variable "invokeCCStatusService_execute_InputVariable"
      Feb 20, 2009 5:19:09 PM    Updated variable "outputVariable"
    invokeCCStatusService (pending)
      Feb 20, 2009 5:19:09 PM    Faulted while invoking operation "execute" on provider "getCreditCardStatus".
        <payload>
      Feb 20, 2009 5:19:09 PM    [FAULT RECOVERY] Marked Invoke activity as "pending manual recovery".
```

6. Before you try recovery, start the **validationForCC** composite from the **Application Server Navigator** in the JDeveloper as in step D.2.2.3. In this case select **Turn On** to startup the composite. You can also do this from **EM** by selecting the **Start Up…** button on the validationForCC page.

7. Back in the **BPEL Audit Trail** window, select the **Faults** tab.

8. Click on the row containing the fault. This will show the fault details including the contents of the input variable to the invoke.



Audit Trail | Flow | Sensor Values | Faults

Select a fault to recover.

Error Message	Recovery	Fault Time	Activity
<faultType>0</faultType><remoteFault xmlns="http://schemas.oracle.com/b	Recover...	Feb 20, 2009 5:19:09 P	invokeCCStatusService

Recover Fault: default/POProcessing!5.0*6cbe1d7e-f0a7-489e-b556-f17c45508fd6/approveLargeOrder/230001-BpInv0-BpSeq0.3-3

Choose one of the available recovery options, modify the variable information as appropirate, and click "Recover".

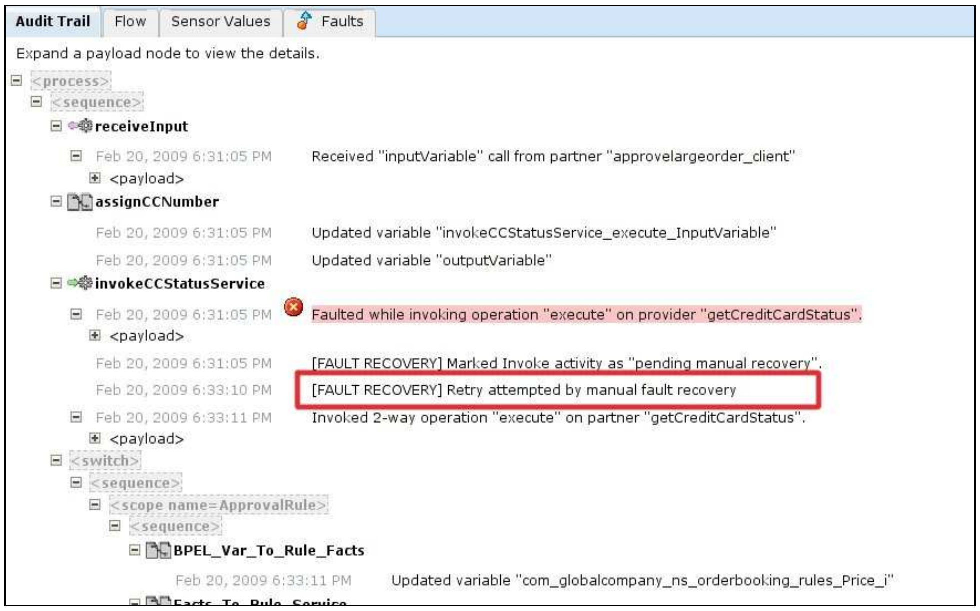
Recovery Action: Retry
After Successful Retry: None
Variable: inputVariable
Value:

```
<inp1:Order xmlns:inp1="http://xmlns.oracle.com/ns/order">
  <inp1:customerId>1111</inp1:customerId>
  <inp1:orderId>2222</inp1:orderId>
  <inp1:prodName>iPod shuffle</inp1:prodName>
  <inp1:itemType>electronics</inp1:itemType>
  <inp1:price>145</inp1:price>
  <inp1:qty>30</inp1:qty>
  <inp1:status>initial</inp1:status>
  <inp1:creditCardInfo>
    <inp1:cardNumber>1234-1234-1234-1234</inp1:cardNumber>
    <inp1:cardType>Mastercard</inp1:cardType>
```

9. Select **Retry** as the recovery action and click on **Recover.**

10. Select **Yes**

11. You should see the fault clear up from the **Faults** tab. Click on the **Audit Trail** tab to view the BPEL flow. You should see that the BPEL has completed the execution successfully (you may need to refresh the page).



Audit Trail | Flow | Sensor Values | Faults

```
Expand a payload node to view the details.
⊟ <process>
  ⊟ <sequence>
    ⊟ receiveInput
      ⊟ Feb 20, 2009 6:31:05 PM    Received "inputVariable" call from partner "approvelargeorder_client"
        ⊞ <payload>
    ⊟ assignCCNumber
        Feb 20, 2009 6:31:05 PM    Updated variable "invokeCCStatusService_execute_InputVariable"
        Feb 20, 2009 6:31:05 PM    Updated variable "outputVariable"
    ⊟ invokeCCStatusService
      ⊟ Feb 20, 2009 6:31:05 PM    Faulted while invoking operation "execute" on provider "getCreditCardStatus".
        ⊞ <payload>
        Feb 20, 2009 6:31:05 PM    [FAULT RECOVERY] Marked Invoke activity as "pending manual recovery".
        Feb 20, 2009 6:33:10 PM    [FAULT RECOVERY] Retry attempted by manual fault recovery
      ⊟ Feb 20, 2009 6:33:11 PM    Invoked 2-way operation "execute" on partner "getCreditCardStatus".
        ⊞ <payload>
    ⊟ <switch>
      ⊟ <sequence>
        ⊟ <scope name=ApprovalRule>
          ⊟ <sequence>
            ⊟ BPEL_Var_To_Rule_Facts
                Feb 20, 2009 6:33:11 PM    Updated variable "com_globalcompany_ns_orderbooking_rules_Price_i"
            ⊟ Facts_To_Rule_Service
```

D.2.4 Handle faults in BPEL

BPEL provides comprehensive error handling constructs for catching exceptions and handling them appropriately. You can choose to handle exceptions within the process itself rather than using fault-handling framework.

For example, in the current implementation of **validateForCC**, while doing the credit card validation, if a non-existing credit card number is passed to the service, it just returns an empty response. If the service were to throw specific exception for such unknown credit card numbers, you would want to catch that exception and set the status in the Order appropriately.

This lab illustrates this scenario.

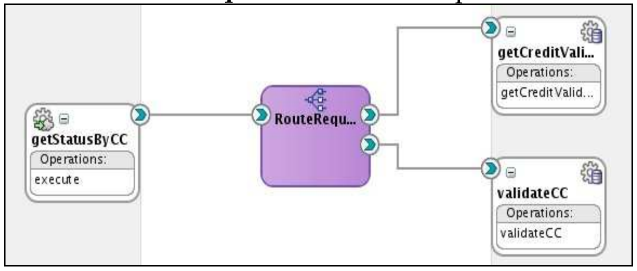
You will modify the validateForCC composite to call a database store function for validating credit cards starting with a specific number. This store function throws a PL/SQL application exception with an error code of 20001 for credit card numbers not in the database. In the BPEL process you will catch and handle the fault.

1. Create the store function in the *SOADEMO* schema by running the SQL script create_validate_cc.sql available in c:\po\sql using the *soademo* user.

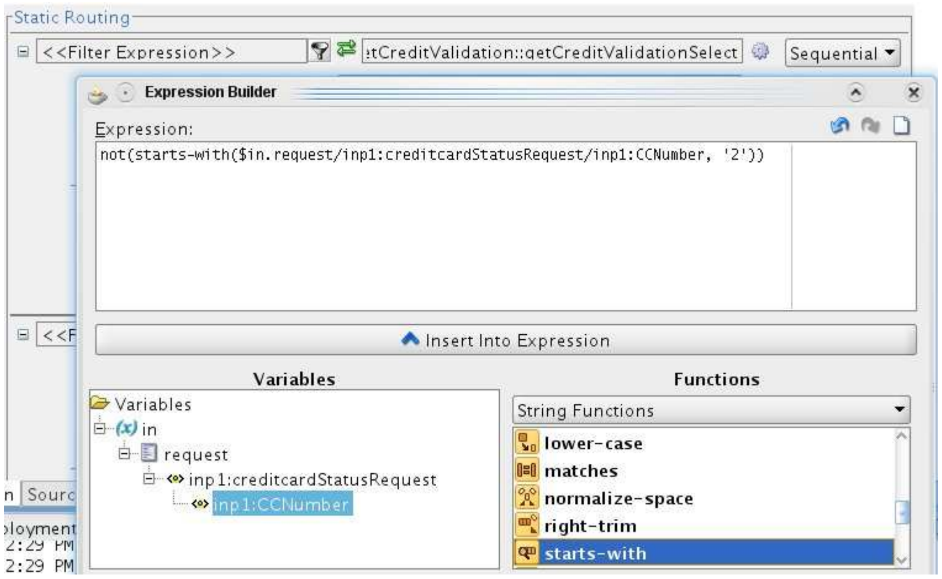
2. Open the **validateForCC** composite and drop a Database adapter to the **External References** swim lane. Step through the wizard to create the reference to the store function just created using the following values.

Service Name	validateCC
JNDI Name	eis/DB/soademoDatabase
Operation Type	Call a Stored Procedure or Function
Procedure	Browse and select VALIDATECC

3. Wire the **RouteRequest** mediator component to **validateCC**



4. Open the **RouteRequest** mediator and add a filter expression for **getCreditValidationSelect** to route all requests for credit card numbers that **don't** start with 2.



```
not(starts-with($in.request/inp1:creditcardStatusRequest/inp1:CCNumber, '2'))
```

5. Add a filter expression for validateCC to route only those requests that have credit card numbers starting with 2 to the database stored function.

6. Create a new transformation for the validateCC route, mapping **CCNumber** to **db:CC_NUMBER.** You don't need a transformation for the reply because this service only raises a fault, it doesn't reply.

7. Save all and deploy the composite

8. You can test it using web service tester in the SOA Console. Use credit card number 1234-1234-1234-1234. You should see a response with status as **VALID** back. Try the test again, except this time change the credit card number to 2234-1234-1234-1234. You should see an error – ORA-20001 UNKNOWN CREDIT CARD

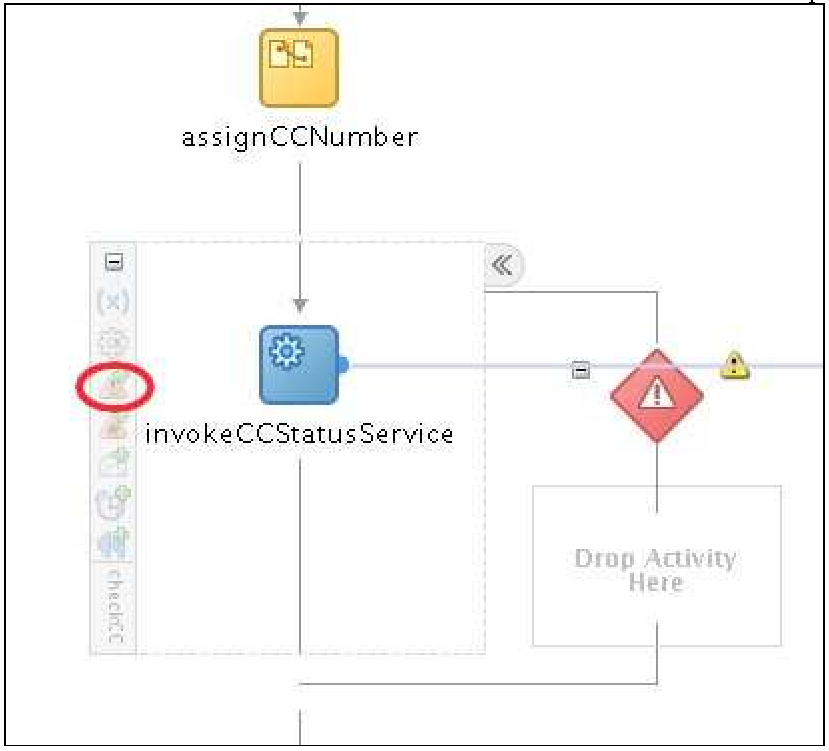
   Now, update the BPEL process to catch the fault.

9. Open the POProcessing composite

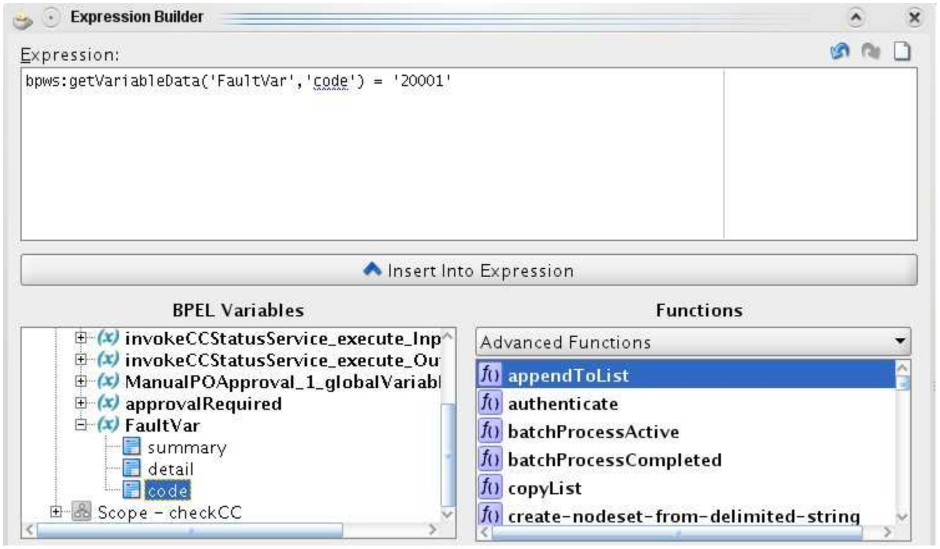
10. Open the **approveLargeOrder** BPEL process.

11. Add new Scope activity above the **invokeCCStatusService** activity. Rename it to **checkCC**.

12. Expand the checkCC scope and move the **invokeCCStatusService** into the scope.
13. Click on **Add Catch Branch** to add catch branch and click on **+** to expand it.



14. Double click on the **Catch** and enter the Fault details. For **Namespace URI** and **Local Part** click on the browse icon and select **System Faults → bindingFault**. Auto create the fault variable and accept the default name, *FaultVar*.
15. Click **OK**
16. Drop a **Switch** activity in the catch block
17. Open the expression editor for **<case>** and build an expression checking for **code** in **FaultVar** is '20001'.

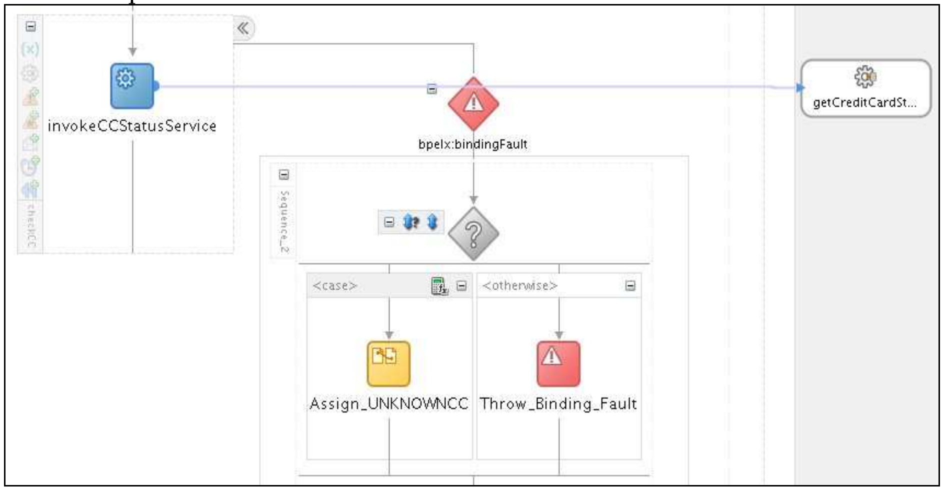


```
bpws:getVariableData('FaultVar','code') = '20001'
```

18. Add an assign activity called *Assign_UNKNOWNCC* in the **<case>** block to assign the literal string 'UKNOWN CC' to the variable **invokeCCStatusService_execute_OutputVariable→reply→creditcardStatus**.

19. In the **<otherwise>** block, drop a **Throw** activity. Open the Throw activity and name it **Throw_Binding_Fault**. Using the browse button select **System Faults→bindingFault**. For the fault variable use the browse button and select the fault variable created in the earlier step.

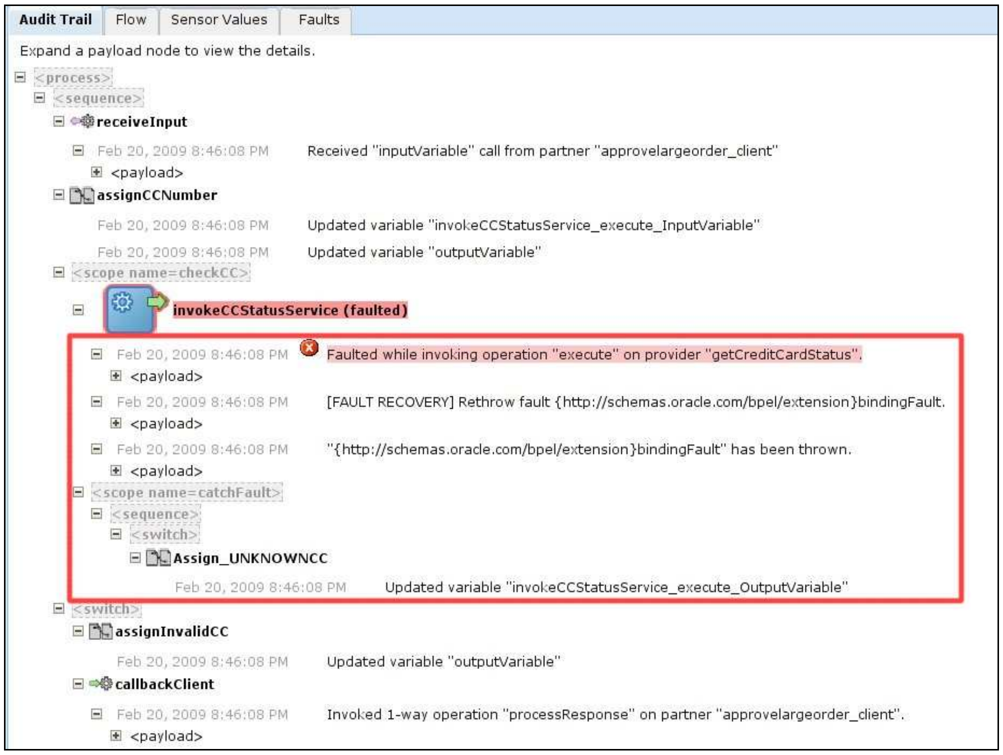
20. The new updated flow should look lik this:



21. The fault-handling framework takes precedence over the BPEL catch so you need to have the fault-handling framework re-throw the fault so that BPEL can process it. Add the following in the fault-policies.xml after the comment *Step D.2.4.20*

```
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"  name="bpelx:bindingFault">
    <condition>
        <!-- Let the component handle this specific binding fault -->
        <test>$fault.code="20001"</test>
        <action ref="ora-rethrow-fault"/>
    </condition>
    </faultName>
```

22. Deploy and test the POProcessing composite. Use the po-large-iPodx30.xml from c:\po\input. But change the credit card number to start with 2. You should see the fault being re-thrown from the fault handling framework and being caught and processed in the BPEL process.



```
Expand a payload node to view the details.
<process>
  <sequence>
    receiveInput
      Feb 20, 2009 8:46:08 PM    Received "inputVariable" call from partner "approvelargeorder_client"
        <payload>
    assignCCNumber
      Feb 20, 2009 8:46:08 PM    Updated variable "invokeCCStatusService_execute_InputVariable"
      Feb 20, 2009 8:46:08 PM    Updated variable "outputVariable"
    <scope name=checkCC>
      invokeCCStatusService (faulted)
        Feb 20, 2009 8:46:08 PM    Faulted while invoking operation "execute" on provider "getCreditCardStatus".
          <payload>
        Feb 20, 2009 8:46:08 PM    [FAULT RECOVERY] Rethrow fault {http://schemas.oracle.com/bpel/extension}bindingFault.
          <payload>
        Feb 20, 2009 8:46:08 PM    "{http://schemas.oracle.com/bpel/extension}bindingFault" has been thrown.
          <payload>
      <scope name=catchFault>
        <sequence>
          <switch>
            Assign_UNKNOWNCC
              Feb 20, 2009 8:46:08 PM    Updated variable "invokeCCStatusService_execute_OutputVariable"
    <switch>
      assignInvalidCC
        Feb 20, 2009 8:46:08 PM    Updated variable "outputVariable"
      callbackClient
        Feb 20, 2009 8:46:08 PM    Invoked 1-way operation "processResponse" on partner "approvelargeorder_client".
          <payload>
```

D.2.5 Using A Custom Java Fault Handler

In addition to some of the pre-defined actions like **humanIntervention, rethrowFault, abort** etc, you can also define your own custom fault handler using Java.

In this exercise, you will change the policy for **bindingFault** and instead of re-throwing the fault; you will use a custom Java class to handle it.

1. Install the custom Java handler by copying the provided myfaulthandler.jar available in c:\po\lib to the lib directory of your Weblogic Server domain home. The JDeveloper project for this JAR is available in c:\po\solutions\fault-handling\MyFaultHandlerApp
2. Restart the managed server.
3. Modify the fault-policies.xml and change the bindingFault handling

from

```
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"  name="bpelx:bindingFault">
      <condition>
            <!-- Let the component handle this specific binding fault -->
            <test>$fault.code="20001"</test>
            <action ref="ora-rethrow-fault"/>
      </condition>
 </faultName>
```

to

```
<faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"  name="bpelx:bindingFault">
      <condition>
            <!-- Let the component handle this specific binding fault -->
            <test>$fault.code="20001"</test>
            <action ref="my-java-handler"/>
     </condition>
 </faultName>
```

4. Ensure that the directory c:\po\log exists.
5. Deploy the POProcessing composite and test using the po-large-iPodx30.xml from c:\po\input with the credit card number changed to start with 2. You should see myfaulthandler.log in c:\po\log. You should also see that the POProcessing composite handles the fault in the EM Console

D.2.6 Handle Mediator Faults

Since the Mediator doesn't provide any built-in fault handling mechanism, the policy-based fault handler is the only way to catch and handle exception occurring in the Mediator. A variety of exceptions can be caught ranging from adapter exception to transformation exceptions.

In this exercise, you will define a fault handler for catching all Mediator faults and use the custom Java handler to write a log file. To force the Mediator to fault, we will simulate a disk write error and have the File adapter throw an exception.

1. Modify fault-policies.xml and add the following after the comment containing *Step D.2.6.1*

```
<faultName xmlns:medns="http://schemas.oracle.com/mediator/faults" name="medns:mediatorFault">
  <condition>
      <action ref="my-mediator-fault-handler"/>
</condition>
</faultName>
```

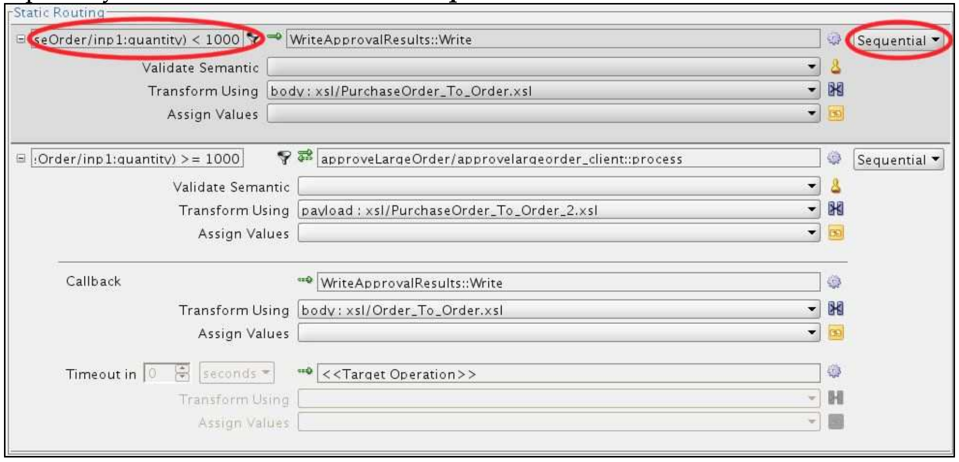
2. Add the following after the comment starting with *Step D.2.6.2*. This defines a java action using the same custom java class that you used in the earlier exercise but uses a different set of properties and also defines a different action to be performed on return from the custom Java call

```
<Action id="my-mediator-fault-handler">
  <javaAction className="soatraining.faulthandling.MyFaultHandler"
          defaultAction="ora-terminate" propertySet="myMediatorProps">
      <returnValue value="OK" ref="ora-human-intervention"/>
  </javaAction>
</Action>
```

3. Add the following after the comment starting with *Step D.2.6.3.* This defines the property set that is to be used by the new JavaAction.

```
<propertySet name="myMediatorProps">
      <property name="logFileName">mediator-faults.log</property>
      <property name="logFileDir">c:\po\log</property>
 </propertySet>
```

4. Open the **routePO** mediator component and change the route for the filter "quantity < 1000" to **Parallel** from **Sequential**.

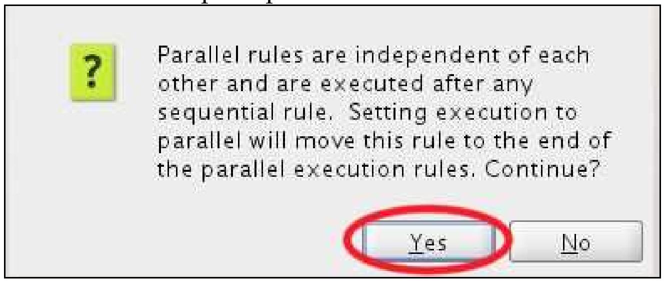


Static Routing

seOrder/inp1:quantity) < 1000 → WriteApprovalResults::Write — Sequential
Validate Semantic
Transform Using: body : xsl/PurchaseOrder_To_Order.xsl
Assign Values

:Order/inp1:quantity) >= 1000 → approveLargeOrder/approvelargeorder_client::process — Sequential
Validate Semantic
Transform Using: payload : xsl/PurchaseOrder_To_Order_2.xsl
Assign Values
Callback → WriteApprovalResults::Write
Transform Using: body : xsl/Order_To_Order.xsl
Assign Values
Timeout in 0 seconds → <<Target Operation>>
Transform Using
Assign Values

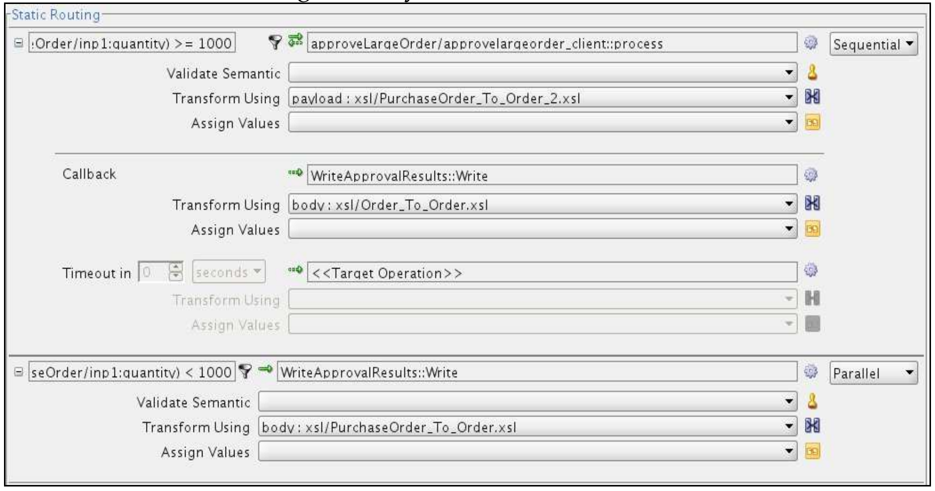
5. Select **Yes** when prompted



Parallel rules are independent of each other and are executed after any sequential rule. Setting execution to parallel will move this rule to the end of the parallel execution rules. Continue?

[✓ Yes] No

6. You should see this change when you are done



Static Routing

:Order/inp1:quantity) >= 1000 → approveLargeOrder/approvelargeorder_client::process — Sequential
Validate Semantic
Transform Using: payload : xsl/PurchaseOrder_To_Order_2.xsl
Assign Values
Callback → WriteApprovalResults::Write
Transform Using: body : xsl/Order_To_Order.xsl
Assign Values
Timeout in 0 seconds → <<Target Operation>>
Transform Using
Assign Values

seOrder/inp1:quantity) < 1000 → WriteApprovalResults::Write — Parallel
Validate Semantic
Transform Using: body : xsl/PurchaseOrder_To_Order.xsl
Assign Values

Without this change the fault policy handler will not get control when a fault occurs in the Mediator. This is because sequential routes are executed in the same thread and transaction context as the caller of the mediator service. If a fault occurs while executing a sequential route, it is thrown back to the caller and the fault handler is bypassed otherwise it could potentially break the transaction. In case of a parallel route a new thread and new transaction context is created by the Mediator, within which the route is executed. If a fault occurs while executing a parallel route, and a matching fault policy exists, the appropriate fault handler is executed.

7. Deploy the POProcessing composite.

8. Change the permission of the directory c:\temp to read-only. This will force an error while writing the po file.

- In Windows, open a command window and enter

  ```
  >attrib +r c:\temp
  ```

  to make the directory readonly, and

  ```
  >attrib –r c:\temp
  ```

  to remove the readonly attribute.

- In Linux, open a command window and enter

  ```
  chmod –w /temp
  ```

  to make the directory readonly, and

  ```
  chmod +w /temp
  ```

  to make it writable again.

9. Test the POProcessing composite using the po-small-Headsetx1.xml. The process will try to write the approval to c:\temp and fail. You should see a file mediator-faults.log in the c:\po\log directory. The EM Console should show the composite waiting for manual recovery.

10. Try submitting the po-large-iPodx30.txt. Did the fault handler execute? Refer to D.2.6.6 for explanation of this behavior.