Oracle Technologies Primer

An Oracle Fusion Middleware and iPaaS blog!

Tag Archives: Retrying Faults in SOA Suite

11.10

Fault Handling in Oracle SOA Suite : Advanced Concepts

Posted on July 18, 2011 by Arun Pareek

This tutorial is meant to cover extensively the mechanism that we can adopt for Fault Management for a SOA Suite composite. It will deal with a fairly overall strategy for handling faults and dealing with them in various ways.

Before diving more into advanced concepts of Handling Faults let me present a small introduction covering the basics of a Service composite.

Basic Architecture of a Service Composite in Oracle SOA Suite

Oracle ACE

Oracle ACE

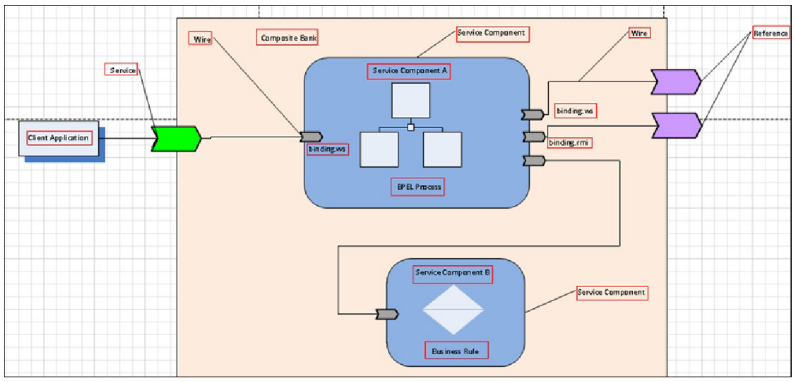
Email Subscription

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 1,185 other followers

Enter yo

Sign me up!



1. **Service components** – BPEL Processes, Business Rule, Human Task, Mediator. These are used to construct a SOA composite application. A service engine corresponding to the service component is also available.
2. **Binding components** – Establish connection between a SOA composite and external world.
3. **Services** provide an entry point to SOA composite application.
4. **Binding** defines the protocols that communicate with the service like SOAP/HTTP, JCA adapter etc.
5. **WSDL** advertises capabilities of the service.
6. **References** enables a SOA composite application to send messages to external services
7. **Wires** enable connection between service components.

Coming to Fault Handling in a composite there are primarily two types of faults

1. **Business faults** – Occurs when application executes THROW activity or an invoke receives fault as response. Fault name is specified by the BPEL process service component. This fault is caught by the fault handler using Fault name and fault variable.
2. **Runtime faults** – Thrown by system. Most of these faults are provided out of the box. These faults are associated with RunTimeFaultMessage and are included in http://schemas.oracle.com/bpel/extension namespace.

Oracle SOA Suite gives us an option of configuring fault and fault actions using policies. This means that we can create policies in response to a specific type of exception. Policies are defined in a file that by default is called **fault-policies.xml**

Policies for fault handling consist of two main elements:

Port Type in Oracle PCS
Accelerating Innovation through Device, Human and Process Centric Integration
Suppress Approval Controls in Oracle BPM – Hidden Workspace Feature

Blog Stats

551,172 hits

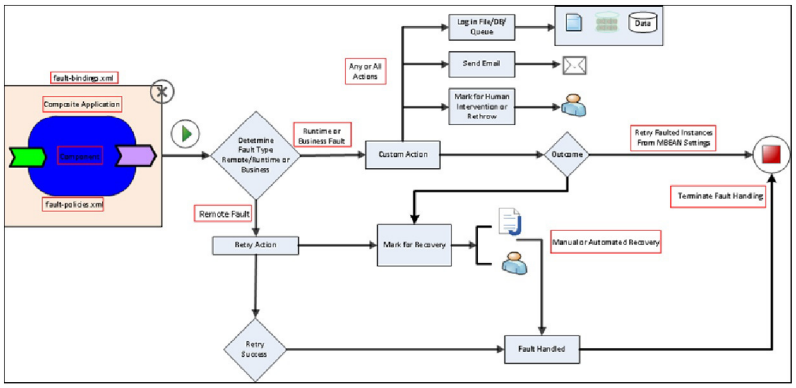
Top Posts

Change Logging Level of IntegratedWeblogicServer in JDeveloper
Stripping XML Namespaces using XQuery
Dynamic XQuery in Oracle Service Bus
Oracle BPM Human Task Management : Using Human Task Events to Invoke Microsoft Exchange Web Services
Eliminating

1. The **fault condition** that activates the policy block—we specify what type of fault(s) the policy is relevant for. We can then apply even more finer grained policy and actions based on error codes, error messages etc. 11.10
2. The **action(s)** that should be performed when the condition is satisfied. An action for an fault may be to retry it for a certain number of time at a specified interval, or to mark it in recovery for human intervention, use a custom Java code or simply to throw the fault back. If the fault is rethrown then if we have specified any explicit **'catch'** block in our BPEL process that will be executed.

It should also be noted that fault policies need to be explicitly associated with composites, components, or references. This is done in a **fault-bindings.xml** file. Fault bindings link the composite, specific components in the composite, or specific references in the components on the one hand to one of the fault policies on the other.

Have a look at the diagram below to understand a mechanism to throw a fault from a service composite, identify the fault type and then take necessary action.



The following post will try and cover all aspects of what is covered in the diagram above.

Consider the following fault-policies.xml. Read the comments in the XML to understand what each of the condition, action and property is about.

```
1  <faultPolicies xmlns="http://schemas.oracle.com/b
2  <faultPolicy version="2.0.1" id="CompositeFaultPo
3  <Conditions>
4  <!-- Conditions can be fine grained to include Ac
5  <faultName xmlns:bpelx="http://schemas.oracle.com
```

using XQuery and XPath
Eliminating Duplicate Element set using XSLT and XPath
Oracle Human Workflow Web Service APIs
Using Java APIs for Oracle Human Workflows
Oracle SOA Suite 11g Administrator's Handbook
Restrict Reassignments in Oracle BPM

Tweets

RT @BoulderWide: Big news coming from Boomi today! twitter.com/boomi/status/1...
4 months ago

Really #Melbourne? Is this what we are

```
<condition>
<action ref="ora-retry"/>
</condition>
</faultName>
<faultName xmlns:bpelx="http://schemas.oracle.com
<condition>
<action ref="java-fault-handler"/>
</condition>
</faultName>
<faultName xmlns:bpelx="http://schemas.oracle.com
<condition>
<action ref="java-fault-handler"/>
</condition>
</faultName>
</Conditions>
<Actions>
<!-- This Action will invoke a Custom Java Class
<Action id="java-fault-handler">
<javaAction className="com.beatech.faultapp.Custo
<returnValue value="Manual" ref="ora-human-interv
</javaAction>
</Action>
<!-- This Action will mark the instance as "Pendi
<Action id="ora-human-intervention">
<humanIntervention/>
</Action>
<!--This is an action will bubble up the fault to
<Action id="ora-rethrow-fault">
<rethrowFault/>
</Action>
<!--This action will attempt 3 retries with inter
<Action id="ora-retry">
<retry>
<retryCount>3</retryCount>
<retryInterval>120</retryInterval>
<retryFailureAction ref="java-fault-handler"/>
</retry>
</Action>
<!--This action will cause the instance to termin
<Action id="ora-terminate">
<abort/>
</Action>
</Actions>
<!--Properties can be used to pass values to the
<Properties>
<propertySet name="properties">
<property name="myProperty1">propertyValue1</prop
<property name="myProperty2">propertyValue2</prop
<property name="myPropertyN">propertyValueN</prop
</propertySet>
</Properties>
</faultPolicy>
</faultPolicies>
```

Inside the **custom Java fault handler we can also use a switch** that acts on the returnValue to chain another **Action.**

```
<javaaction classname="com.beatech.faultapp.Custom
<returnValue ref="ora-rethrow" value="Rethrow"/>
<returnvalue ref="ora-terminate" value="Abort"/>
<returnvalue ref="ora-retry" value="Retry"/>
<returnvalue ref="ora-human-intervention" value="M
</javaaction>
```


have 12 MPs

The next step will be to create a **fault-bindings.xml** file to simply bound the fault policy file to the composite.

```
<faultPolicyBindings version="2.0.1" xmlns="http:/
<composite faultPolicy="CompositeFaultPolicy"/>
</faultPolicyBindings>
```

Finally we have to add two properties in the composite.xml to let the composite know about them

<property name="oracle.composite.faultPolicyFile">fault-policies.xml></property>

< property name="oracle.composite.faultBindingFile">fault-bindings.xml></property>

We can use different names and locations for the fault policies and fault bindings files, by setting the properties **oracle.composite.faultPolicyFile** and **oracle.composite.faultBindingFile** in the **composite.xml** to configure these custom files.

For example we can refer to these files even from the MDS.

<property name="oracle.composite.faultPolicyFile">oramds://apps/policy/fault-policies.xml></property>

< property name="oracle.composite.faultBindingFile">oramds://apps/policy/fault-bindings.xml></property>

Once we hit a fault in our composite that has a custom Java Action the java class **CustomFaultHandler** will be instantiated. Here is one example of a Java Class.

Follow
@arrunpareek

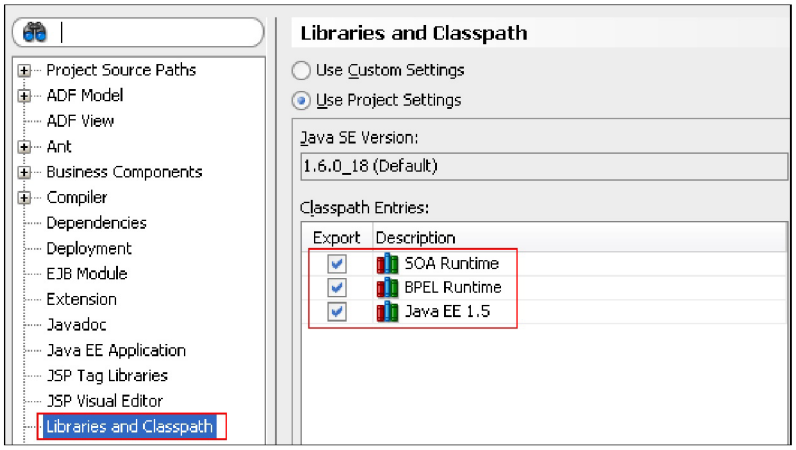
The custom Java class has to implement the interface IFaultRecoveryJavaClass that defines two methods i.e **handleRetrySuccess** and **handleFault**. The custom Java class **CustomFaultHandler** has access to the **IFaultRecoveryContext** containing information about **the composite, the fault, and the policy**.

If the fault is thrown by a **BPEL** process we can check if it's an **instanceof BPELFaultRecoveryContextImpl** to get further fault details.

```
package com.beatech.faultapp;
import com.collaxa.cube.engine.fp.BPELFaultRecove
import com.oracle.bpel.client.config.faultpolicy.
import java.util.Map;
import oracle.integration.platform.faultpolicy.IF
import oracle.integration.platform.faultpolicy.IF

public class CustomFaultHandler implements IFault
public CustomFaultHandler() {
super();
}

public void handleRetrySuccess(IFaultRecoveryCont
System.out.println("Retry Success");
handleFault(iFaultRecoveryContext);
}

public String handleFault(IFaultRecoveryContext i
//Print Fault Meta Data to Console
System.out.println("****************Fault Metadat
System.out.println("Fault policy id: " + iFaultRe
System.out.println("Fault type: " + iFaultRecover
System.out.println("Partnerlink: " + iFaultRecove
System.out.println("Port type: " + iFaultRecovery
System.out.println("****************************
//print all properties defined in the fault-polic
System.out.println("Properties Set for the Fault"
Map props = iFaultRecoveryContext.getProperties()
for (Object key: props.keySet())
{
System.out.println("Key : " + key.toString()  + "
}
//Print Fault Details to Console if it exists
System.out.println("****************Fault Details
if(iFaultRecoveryContext instanceof BPELFaultReco
{
BPELFaultRecoveryContextImpl bpelCtx = (BPELFault
System.out.println("Fault: " + bpelCtx.getFault()
System.out.println("Activity: " + bpelCtx.getActi
System.out.println("Composite Instance: " + bpelC
System.out.println("Composite Name: " + bpelCtx.g
System.out.println("****************************
}
//Custom Code to Log Fault to File/DB/JMS or send
return "Manual";
}
}
```

1. Log the fault/part of fault in a **flat file, database or error queue** in a specified enterprise format.
2. We can even configure to send an **Email** to the support group for remedy and action. (Use custom Java Email code or use the UMS java/ejb APIs to do so)
3. Return a flag with an appropriate **post Action.** This flag determines what action needs to be taken next

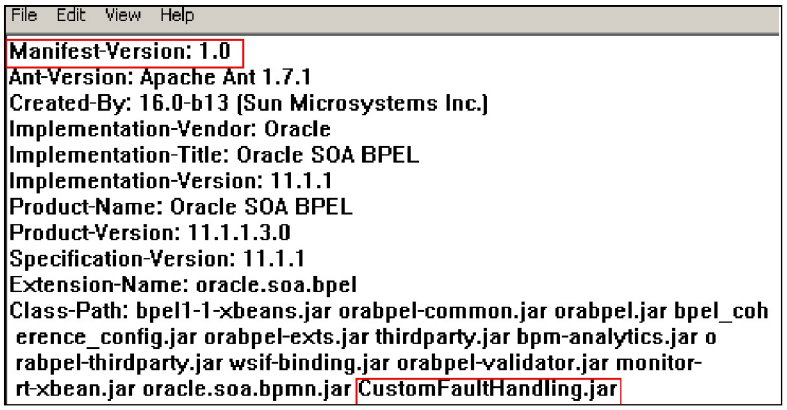
The java class would require the SOA and BPEL runtime in classpath to compile and execute.



To make sure that when the composite instances faults out and the fault-policy.xml is able to instantiate this class we have to make it available in the server's classpath.

There are a couple of ways to do that. Here is one of the way to achieve it.

1. Compile your Java Project and export it as a jar file (say **CustomFaultHandling.jar**)
2. Go to **<Middleware Home>\Oracle_SOA1 \soa\modules\oracle.soa.bpel_11.1.1** directory of your Oracle SOA Suite installation.
3. Copy the **CustomFaultHandling.jar** in the above directory
4. Unjar the **oracle.soa.bpel.jar** and edit the MANIFEST.MF to add an entry of the above jar in the **classpath**.



1. Pack the jar again and **restart both the Managed and Admin Server**.
2. Another way is to drop the **CustomFaultHandling.jar** in the **<Middleware Home>\Oracle_SOA1 \soa\modules\oracle.soa.ext_11.1.1** and run **Ant** on the **build.xml** file present in the directory.

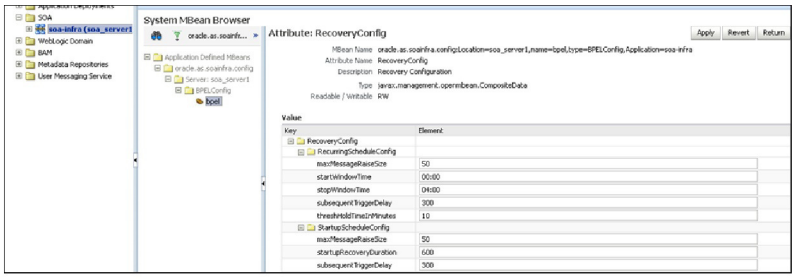
Interestingly we also have an option in the EM console to retry all faults for a composite by setting some values in custom MBeans. They are available as Advanced BPEL properties in the SOA Infra engine.

The MBean that allows recovery of **faulted instances** is **RecoveryConfig.** It has two options for retrying

1. **RecurringScheduleConfig** : A recovery window may be specified (most probably off peak hours) to recover all faulted instances. Messages being recovered can be throttled by limiting the number of messages picked up on each run by specifying the **maxMessageRaiseSize.**
2. **StartupScheduleConfig :** With this setting on all faulted instances are automatically retried when the soa server is booted up for restart.

More details on how to use **RecoverConfig** Mbean can be found here

http://download.oracle.com/docs/cd/E14571_01/relnotes.1111/e10133/soa.htm#RNLIN1052



There is a small lacuna though here. It is not always possible to recover automatically. **Auto-recovery** is subject to some conditions.

Consider the scenarios below.

1. **Scenario A:** The BPEL code uses a fault-policy and a fault is handled using the "**ora-human-intervention**" activity, then the fault is marked as **Recoverable** and the instance state is set to "**Running**".
2. **Scenario B:** The BPEL code uses a fault-policy and a fault is caught and re-thrown using the "**ora-rethrow-fault**" action, then the fault is marked as **Recoverable** and the instance state is set to "**Faulted**"; provided the fault is a recoverable one (like URL was not available).

In Scenario A, the Recoverable fault CANNOT be auto-recovered using the **RecoveryConfig** MBean.

In Scenario B, the **Recoverable** fault can be auto-recovered on server startup and/or pre-scheduled recovery.

All is not lost however. The instances can still be recovered from the console though. However for most practical purposes it isn't desirable that a huge number of composite instances that are marked for recovery for a remote fault (say end point not available) are retried automatically. It is natural that we will yearn to automate this part as well.

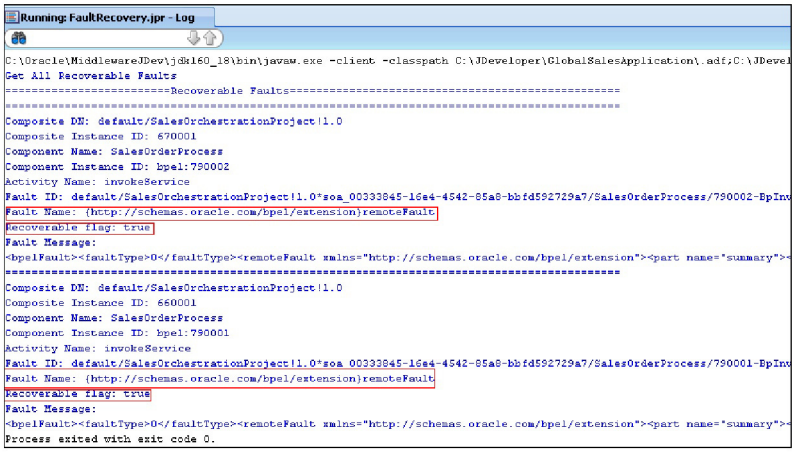
Here is a sample code that gets all remote faults that are marked as recoverable from the Custom Java Class and retries them.

```
1  package com.beatech.salapp;
2
3  import java.util.Hashtable;
4  import java.util.List;
5
6  import javax.naming.Context;
```

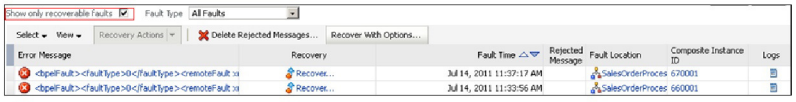
```
import oracle.soa.management.facade.LocatorFactor
import oracle.soa.management.facade.bpel.BPELServ
import oracle.soa.management.util.FaultFilter;

public class FaultRecovery {

private Locator locator = null;
private BPELServiceEngine mBPELServiceEngine;

public FaultRecovery() {
locator = this.getLocator();
try {
mBPELServiceEngine =
(BPELServiceEngine)locator.getServiceEngine(Locat
} catch (Exception e) {
e.printStackTrace();
}
}

public Hashtable getJndiProps() {
Hashtable jndiProps = new Hashtable();
jndiProps.put(Context.PROVIDER_URL,"t3://localhos
jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,"we
jndiProps.put(Context.SECURITY_PRINCIPAL, "weblog
jndiProps.put(Context.SECURITY_CREDENTIALS, "welc
jndiProps.put("dedicated.connection", "true");
return jndiProps;
}

public Locator getLocator() {

try {
return LocatorFactory.createLocator(getJndiProps(
} catch (Exception e) {
e.printStackTrace();
}
return null;
}

public void recoverFaults() {
try {
System.out.println("Get All Recoverable Faults");
/* Set Search Filters like composite Name, Instan
Here I am setting the setRevoverable filter to tr
Also I am setting filter on faultName as i want t
*/
FaultFilter filter = new FaultFilter();
filter.setFaultName("{http://schemas.oracle.com/b
filter.setRecoverable(true);

//Get faults using defined filter
List<Fault> faultList = mBPELServiceEngine.getFau
System.out.println("=======================Reco
for (Fault fault : faultList) {
System.out.println("============================
System.out.println("Composite DN: " + fault.getCo
System.out.println("Composite Instance ID: " + fa
System.out.println("Component Name: " + fault.get
System.out.println("Component Instance ID: " + fa
System.out.println("Activity Name: " + fault.getL
System.out.println("Fault ID: " + fault.getId());
System.out.println("Fault Name: " + fault.getName
System.out.println("Recoverable flag: " + fault.i
System.out.println("Fault Message: " + fault.getM
}
```

```
80  e.printStackTrace();
81  }
82
83  }
84
85  public static void main(String[] args) {
86  FaultRecovery faultRecovery = new FaultRecovery()
87  faultRecovery.recoverFaults();
88  }
89  }
```

Replace the values in the property map with the ones in your server. **Remember to give the managed server port in the Provider URL.** Run the Java Class and you would see that the recoverable faults are printed.



Verify the same from the console



Run the Java program again but this time uncomment the line below

```
1  //mBPELServiceEngine.recoverFaults(faultList.toArr
```

This will result in all faults marked with Recovery icon to be retried. So if the remote endpoint is responding and active now the processes will complete.

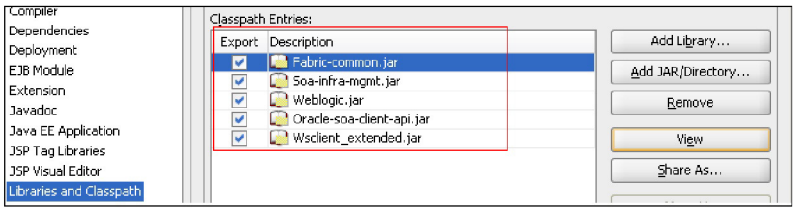
There are a host of other things that we can do in this Java Class.

Using the BPELServiceEngine object we can write messages to the BPEL audit trail, inspect the current

The following code snippet if inserted in the code will replace any process variable with a new value before retrying. (May be used in case of Binding or Business Faults)

```
//Get faults using defined filter
List<Fault> faultList = mBPELServiceEngine.getFau
System.out.println("========================Reco
for (Fault fault : faultList)
{
System.out.println("========================Read
//Get Process Instance variables from fault objec
String[] variables = mBPELServiceEngine.getVariab
System.out.println("Process Instance Variables:")
for (int i = 0; i < variables.length; i++)
{
System.out.println("Variable Name: " + variables[
}
//Get Input Variable Data from the Activity, Modi
System.out.println("========================Repl
System.out.println("Activity Input Variable Data:
String value =mBPELServiceEngine.getVariable(faul
System.out.println("Present value: " + value);
value = value.replace("remoteFault", "Modified Va
System.out.println("New value: " + value);
mBPELServiceEngine.setVariable(fault,"activityReq

// Recover Faults one by one
mBPELServiceEngine.recoverFault(fault,FaultRecove
}
```

The following JARS would be required in the classpath for the above Java Code



<MWHOME>\oracle_common\modules\oracle.fabriccommon_11.1.1

\fabric-common.jar

<MWHOME>\jdeveloper\soa\modules\oracle.soa.mgmt_11.1.1\soa-

infra-mgmt.jar

<MWHOME>\wlserver_10.3\server\lib\weblogic.jar

<MWHOME>\jdeveloper\soa\modules\oracle.soa.fabric_11.1.1 \oracle-soa-client-api.jar
<MWHOME>\oracle_common\webservices\wsclient_extended.jar

The post discussed the different approaches and strategies for handling faults in a composite in SOA Suite. Let me conclude this article by describing a few best practices around Fault Handling.

Oracle SOA Suite Fault Handling Best Practices

1. Create **fault** (catch block) for each partner link. For each partner link, have a catch block for all possible errors. Idea is not to let errors go to **catchAll** block.
2. **CatchAll** should be kept for all errors that cannot be thought of during design time.
3. Classify errors into various types – runtime, remote, binding, validation, Business errors etc.
4. Notification should be setup in production, **so that, errors are sent to concerned teams by E-Mail.** Console need not be visited for finding out status of execution.
5. Use **Catch Block** for non-partner link error.
6. **Every retry defined in fault policy causes a commit of the transaction**. Dehydration will be reached and threads released.
7. Automated recovery can be created by creating a fault table, persisting the queue and having an agent to re-submit the job (For example writing a Timer agent to invoke the Java code we wrote to recover instances) . Can be achieved through scripts. **Use only PUBLISHED API of ESB or QUEUE (AQ etc.)** for re-submission. Another example would be to use **WLST** to change the **RecoveryConfig** MBean to configure recovery window to retry all faulted instances.
8. Handle Rollback fault by providing **'No Action'** in fault policy.
9. Remember – Receive, OnMessage, On Alarm, Wait, CheckPoint (Activity for forcing Java thread to store its current state to Dehydration store) will cause storing of current state to dehydration store and threads will be released.
10. Always use **MDS to store fault policies and bindings to**

All artifacts used for demonstration in this article can be found at this link.

.

Rate this:

23 Votes

Share this:

Print Email Pinterest 1 LinkedIn Tumblr WhatsApp Reddit Twitter Facebook

Like

7 bloggers like this.

Posted in BPEL, BPM, Fault Handling, Fault Management, Oracle SOA Suite, SOA, Weblogic | Tagged BPEL, BPM, Fault Bindings, Fault Handling and Management in Oracle SOA Suite 11g, Fault Management through Java, Fault Policies, Mediator, Retrying Faults in SOA Suite | 17 Comments