

JBossESBjBPMIntegration

JBossESB - jBPM Integration

-

This page contains both requirements and design examples about the way of integration.

1. ESB to jBPM

The integration layer uses the jBPM Command API, and it provides action for the following jBPM commands:

- **DeployProcessDefinition** - Deploy a Process Instance to jBPM. In reality one would most likely use the IDE, and the QuickStarts offer the deployment by sending a message to action command. For details on native jBPM options to deploy a process archive see the [jBPM documentation](#). We need to document the pros and cons about deploying classes along with the Process Instance definition to avoid class loading issues.
- **NewProcessInstanceCommand** - Start a new ProcessInstance given a process definition that was already

deployed to jBPM. The **NewProcessInstanceCommand** leaves the Process Instance in the start state, which would be needed if there is an task associated to the Start node (i.e. some task on some actor's tasklist). In most cases however you would like the new Process Instance to move to the first node, which is where the next command comes in.

- **StartProcessInstanceCommand** - Identical to the **NewProcessInstanceCommand**, but additionally the new Process Instance is moved from the Start position into the first Node.

- `CancelProcessInstanceCommand` - Cancel a `ProcessInstance`. i.e. when an event comes in which should result in the cancellation of the entire `ProcessInstance`. This action requires some jBPM context variables to be set on the message, in particular the `ProcessInstance Id`. Details on that are discussed later.
- `GetProcessInstanceVariablesCommand` - Retrieve variables from the specified process instance.

The actions should be asynchronous in the sense that they do not return information to the caller and that any jBPM processing should occur in a different context. This means that these actions return a 'null' message, and terminate the action pipeline.

2. jBPM to ESB

Right now the communication from jBPM to ESB (`EsbActionHandler`) can be synchronous or asynchronous. A synchronous invocation of the Service Invoker is used when a timeout value is specified in the process definition.

The synchronous call opens up generic issues synchronous calls have in the ESB right now, such as problems with transactions across boundaries, reliable delivery etc. More specifically in jBPM, we leave ourselves open to issues around recursion - if the Node is blocked in the `EsbActionHandler` and another request comes into this Node.

We want to replace the synchronous call with a 'async - callback' one that would NOT change the existing configuration.

2.1 Recipe for one-way call (`EsbNotifier`)

The BPM designer attaches the `EsbNotifier` to the Transition or an event, rather than the Node. The jBPM processing can move along while the request to the ESB service is

processed in the background. In the process-deployment.xml, the designer specifies the Service category and name, and adds variable mapping from jBPM to ESB.

2.2 Recipe for a two-way call (async-callback EsbActionHandler)

The BPM designer creates a node attaches the EsbActionHandler to the Node. The

designer specifies the Service category and name, and the variable mappings for both ways. Since the EsbActionHandler is attached to the Node, jBPM processing does not transition to the next Node. The Node is waiting for a signal, which will come from the ESB callback (or from somewhere else, and then the callback call should fail).

2.3 Recipe for a two-way call (async-callback, EsbActionHandler) with timeout

Same as 2.2, but additionally the designer needs to add a [Timer](#) to the Node and sets the DueDate. The DueDate is a timeout which can be set to something like $t + 5s$. Optionally the designer can add a timeout transition, to process things differently when a timeout occurs. If an old 'millisToWaitForResponse' is specified we should log a warning that this is no longer supported.

2.4 Design of the async - callback

Both 2.2 and 2.3 use the asyc-callback mechanism which is hidden from the user. The following section fleshes out how this should work.

2.4.1 jBPM -> ESB (EsbActionHandler)

- Create or increment a process version counter whose name is related

to the current node ID. The counter should be added to the ProcessVariable Map (global to the ProcessInstance) on the jBPM side. The name of the counter should be something like

-ESBInvocationCounter. The same variable should be added to the EsbMessage before it is passed onto the ESB.

- Include process version counter version, nodeId and tokenId in the EsbMessage's ReplyTo LogicalEPR (Service Category and Name) for the generic CallBack service. If there is an original ReplyTo, store that somewhere else in the message.
- Send async message into ESB, ReplyTo set to EPR from previous step.

2.4.2 ESB -> jBPM generic service 'jBPMCallback'

- Extract node id and version from EPR
- Verify that the version is still the same and that the process is still at that node. If either of these conditions are not met, a warning should be logged.
- Update variables and signal.

3. Exception Handling

3.1 ESB -> jBPM

When the ESB calls into jBPM a jBPMException can be thrown in from the jBPM Command API. This exception is not handled by the integration

we let it propagate into the ESB Action Pipeline code.

3.2 jBPM -> ESB

- When jBPM calls into ESB, the ServiceInvoker can throw a MessageDeliveryException. In the jBPM node one can add an [ExceptionHandler](#) to handle this exception.

When the EsbActionHandler is used and the node is waiting for a Callback we can set

- a timer element on the node, so that a timeout transition is taken when timeout occurs. This is an example of a timer that fires after 1 second, after which it takes the 'time-out-transition'.

```
<node name="Service1">
  <action class="org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
    <esbServiceName>MockService</esbServiceName>
  </action>
  <timer name='timeout' dueDate='1 seconds' transition='time-out-transition'></timer>
  <transition name="ok" to="Service2"></transition>
  <transition name="time-out-transition" to="ExceptionHandling"></transition>
</node>
```

- set an exceptionTransition. The exceptionTransition is used to build the faultToEPR on the ESB message. So if anything goes wrong during processing on the ESB side the faultToEPR will be used, which will use the same CallbackService described above, but the callback will signal the transition referenced in the exceptionTransition. This is an example of setting the exceptionTransition to 'exception'

```
<node name="Service2">
  <action class="org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
    <esbServiceName>MockService</esbServiceName>
    <exceptionTransition>exception</exceptionTransition>
  </action>
  <transition name="ok" to="Service3"></transition>
  <transition name="exception" to="ExceptionHandling"></transition>
</node>
```

- set a variable such as an error code in the jBPM context and add a jBPM decision element which can pick a conditional exception transition. An example of this would be

```
<node name="Service3">
  <action class="org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
    <esbServiceName>MockService</esbServiceName>
    <esbToBpmVars>
      <mapping esb="SomeExceptionCode" bpm="errorCode"></mapping>
    </esbToBpmVars>
  </action>
  <decision name="errorCode">
    <condition name="errorCode" value="SomeExceptionCode"></condition>
  </decision>
  <transition name="errorCode" to="ExceptionHandling"></transition>
  <transition name="default" to="Service3"></transition>
</node>
```

```
</action>
<transition name="ok" to="exceptionDecision"></transition>

</node>

<decision name="exceptionDecision">
  <transition name="ok" to="end"></transition>
  <transition name="exceptionCondition" to="ExceptionHandling">
    <condition>#{ errorCode!=void }</condition>
  </transition>
</decision>
```

The last three scenarios are depicted in the following diagram

Figure 1. ExceptionHandling Flows.

Resources

- A German Java Magazin article and the English showcase description about the jBPM and ESB integration can be found here: [jbpm meets esb](#)