We will simulate a global transaction via a xaDataSource between a BPEL Composite and OSB that interact via native transports (SB and SOA-DIRECT) and define transaction boundaries. Global Transaction is governed by the following parameters viz. the message exchange pattern, the datasource and the composite properties for the particular components viz. bpel.config.

**Behaviour:** In a single global transaction, any fault within the Composite will result in a complete rollback of all transactions. However, if there are boundaries then in the event of a fault within the global transaction, local transactions will be committed while the global transaction is rollbacked. If fault happens in the partnerlink and it is not handled then the local transaction is rollbacked and a FabricInvocationExceptions is thrown back to the parent BPEL process. If fault happens in the partnerlink and it is handled then the local transaction is committed.

**How to create local transactions or boundaries:**
The boundaries are defined in the composite.xml of the component properties of the called process. The exact properties are dependant on the Message Exchange Pattern of the called Component:

1. If MEP is Sync Request Response, then transaction=requiresNew
2. If MEP is Async Request Response, then transaction=RequiresNew, oneWayDeliveryPolicy=sync
3. If MEP is One Way, then transaction=RequiresNew, oneWayDeliveryPolicy=sync

Note 1.1: The transaction parameters are only defined for BPEL Components. So, if you do want to create local transactions, you would have to embed adapters in bpel processes. I do not understand why this limitation should be there.

**Steps:**
1. Create a Thin XA Datasource that supports 'Two-Phase Commit' global transaction and a connection pool. We will require the JNDI of this pool for the project.
2. Create two OSB Business Services for inserting and updating a table in the datasource configured in Step 1. These are based on the JCA transport which is transactional in nature. A link is provided in resources for the same.
3. Create Proxy Services for these Business Services based on SB-transport which is also transactional.
4. Create a composite with two BPEL processes viz. GlobalTransactionBPELRequestor which makes inserts into a table NAME_ID via OSB Proxy Service and LocalTransaction_BPEL_Insert2 which inserts into a table NAME_ID2 again via OSB Proxy Service. In addition, GlobalTransactionBPELRequestor also invokes the LocalTransaction_BPEL_Insert2 and then throws an Assert Fault based on the input.

**LocalTransaction_BPEL_Insert2 snapshot:**

---

The input for the LocalTransaction_BPEL_Insert2 is:

GlobalTransactionBPELRequestor snapshot:
The input for the GlobalTransactionBPELRequestor looks like:

Name and ID will go into the NAME_ID table which the GenerateFault (values Yes or No) will decide on whether to throw the fault or not within the switch activity.

Here, the process inserts data into NAME_ID table via the Invoke_InsertDB invoke to an OSB Proxy. After this, the process makes an Async one way call to the second bpel process via Invoke_BPEL_Insert2. After this, based on the GenerateFault value of YES or NO, a deliberate Assert Fault is thrown.

The database tables, the Assert Fault and the inputs have been created merely to create this simulation. Knock yourself out with any other options you can think of.

5. Now, we wire these components together in the Composite:

6. As per the earlier discussion, we will now insert the following custom properties inside the component property of the called BPEL process viz. LocalTransaction_BPEL_Insert2.

   ```xml
   <component name="GlobalTransactionBPELRequestor">
     <implementation.bpel src="GlobalTransactionBPELRequestor.bpeli/>
   </component>
   <component name="LocalTransaction_BPEL_Insert2">
     <implementation.bpel src="LocalTransaction_BPEL_Insert2.bpeli"/>
     <property name="bpel.config.transaction" is="false" type="xs:string" required="false"/>
   </component>
   ```

   This ensures that the LocalTransaction_BPEL_Insert2 invoke will result in a separate local transaction.

7. Now, deploy the components and test the GlobalTransactionBPELRequestor with the following inputs
This means insert name “GLOBAL TRANSACTION TEST” and ID 1 into the tables NAME_ID and NAME_ID2 and don’t throw any fault i.e. This is just to check the sanity of the code.

8. The Trace in shows that the test completes successfully and the entries for the name and id show up in both the tables:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>*payload</td>
<td>payload</td>
<td></td>
</tr>
<tr>
<td>*Name</td>
<td>string</td>
<td>GLOBAL TRANSACTION TEST</td>
</tr>
<tr>
<td>*ID</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>*GenerateFault</td>
<td>string</td>
<td>YES</td>
</tr>
</tbody>
</table>

9. Now, test the same process with the following inputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>*payload</td>
<td>payload</td>
<td></td>
</tr>
<tr>
<td>*Name</td>
<td>string</td>
<td>GLOBAL TRANSACTION TEST</td>
</tr>
<tr>
<td>*ID</td>
<td>string</td>
<td>2</td>
</tr>
<tr>
<td>*GenerateFault</td>
<td>string</td>
<td>YES</td>
</tr>
</tbody>
</table>

This means, insert name “GLOBAL TRANSACTION TEST” and ID 2 into the tables NAME_ID and NAME_ID2 and throw a fault in the Parent BPEL process i.e. GlobalTransactionBPELRequestor.

10. The trace shows that the LocalTransaction_BPEL_Insert2 process completes without fault while the GlobalTransactionBPELRequestor faults as expected. The table NAME_ID does not have the new entry now (as this is part of the global transaction that has been rollbacked) while the table NAME_ID2 has the entry (as this is the local transaction that commits without any fault)

11. You can also remove the transactional parameters we set earlier in the composite.xml of the LocalTransaction_BPEL_Insert2 and deploy and test with the inputs we used in step 9. The em trace will show that the fault has happened at the end as expected and the entire global transaction is rolled back.

This tutorial shows how we can create sets of local transactions in a single global transaction and thereby have better control over what we want to achieve.