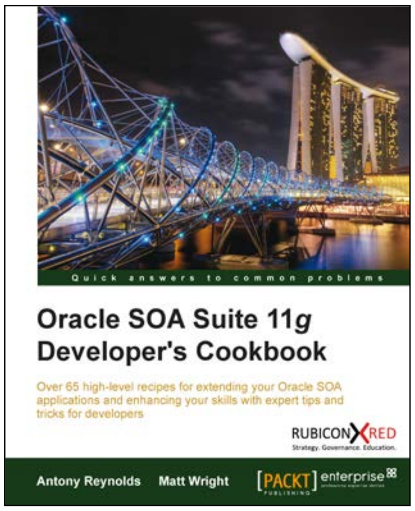
Oracle SOA Suite 11g Developer's Cookbook

Antony Reynolds

Matt Wright


Quick answers to common problems
Oracle SOA Suite 11g Developer's Cookbook
Over 65 high-level recipes for extending your Oracle SOA applications and enhancing your skills with expert tips and tricks for developers
RUBICON RED
Antony Reynolds
Matt Wright
PACKT enterprise

Chapter No. 3
"Working with Transactions"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.3 "Working with Transactions"

A synopsis of the book’s content

Information on where to buy this book

About the Authors

Antony Reynolds has worked in the IT industry for more than 25 years, first getting a job to maintain yield calculations for a zinc smelter while still an undergraduate. After graduating from the University of Bristol with a degree in Mathematics and Computer Science, he worked first for a software house, IPL in Bath, England, before joining the travel reservations system Galileo as a development team lead. Galileo gave him the opportunity to work in Colorado and Illinois where he developed a love for the Rockies and Chicago style deep pan pizza.

Since joining Oracle in 1998, he has worked in sales consulting and support. He currently works as a Sales Consultant helping customers across North America realize the benefits of standards based integration and SOA. While at Oracle he has co-authored *Oracle SOA Suite Developer's Guide, Packt Publishing* and *Oracle SOA Suite 11g R1 Developer's Guide, Packt Publishing.*

Antony lives in Colorado with his wife and four children who make sure that he is gainfully employed playing games, watching movies, and acting as an auxiliary taxi service. He is a slow but steady runner and can often be seen jogging up and down the trails in the shadow of the Rocky Mountains.

I would like to thank my wife Rowan, and my four very patient children, who have put up with my staying at home on family trips and working late nights in my basement office as I completed this book. My managers Ed Lee and Troy Hewitt were very supportive and many of my colleagues contributed knowingly or unknowingly to the recipes.

I am appreciative of Michael Weingartner and his team for their continued enhancement and development of the SOA Suite which has enabled Matt and myself to write this book. The reviewers provided valuable guidance and corrections and any errors still remaining are entirely mine. Finally, the team at Packt Publishing constantly nagged and cajoled Matt and myself to keep some sort of schedule. Without them this book would still be on the drawing board.

Matt Wright is a director at Rubicon Red, an independent consulting firm helping customers enable enterprise agility and operational excellence through the adoption of technologies such as Service-Oriented Architecture (SOA), Business Process Management (BPM), and Cloud Computing.

With over 20 years of experience in building enterprise scale distributed systems, Matt first became involved with SOA shortly after the initial submission of SOAP 1.1 to the W3C in 2000, and has worked with some of the early adopters of BPEL since its initial release in 2002. Since then, he has been engaged in some of the earliest SOA-based implementations across EMEA and APAC.

Prior to Rubicon Red, he held various senior roles within Oracle, most recently as Director of Product Management for Oracle Fusion Middleware in APAC, where he was responsible for working with organizations to educate and enable them in realizing the full business benefits of SOA in solving complex business problems.

As a recognized authority on SOA, he is a regular speaker and instructor at private and public events. He also enjoys writing and publishes his own blog (`http://blogs.bpel-people. com`). He holds a B.Sc. (Eng) in Computer Science from Imperial College, University of London.

He has worked on *Oracle SOA Suite Developer's Guide, Packt Publishing* and *Oracle SOA Suite 11g R1 Developer's Guide, Packt Publishing*.

I would like to express my deep appreciation to everyone who has reviewed this book. Their invaluable feedback and advice not only helped to validate the overall accuracy of the content, but more importantly ensure its clarity and readability.

A book like this doesn't make it into print without a lot of work from the publisher. I would like to thank the team at Packt Publishing for all their support; especially Stephanie Moss, Leena Purkait, and Susmita Panda.

A special mention must go to John Deeb for his continual encouragement, input, and above all support in ensuring that I found time to write the book; I couldn't ask for a more supportive friend and business partner.

Finally, I would like to say a very, very special thank you to my wife Natasha and my children Elliot and Kimberley, who have been incredibly patient and supportive in allowing me to spend far too many evenings and weekends stuck away in my office writing this book.

Oracle SOA Suite 11g Developer's Cookbook

Service Oriented Architecture (**SOA**) provides the architectural framework needed to integrate diverse systems together and create new composite applications. Oracle SOA Suite 11gR1 provides the tools needed to turn an SOA architecture into a working solution. SOA Suite provides the developer with several high level components such as:

- **Oracle Service Bus** (**OSB**), an enterprise strength service bus for full support of service bus patterns including validation, enrichment, transformation, and routing (the VETRO pattern)
- **Service Component Architecture** (**SCA**) that hosts a number of components
- **Business Activity Monitoring** (**BAM**) that provides real-time reporting on SOA Suite activities

SCA components include:

- Mediator for light weight transformation and routing
- Rules for abstraction of business rules
- BPEL for orchestrating long running or complex integrations
- **Human workflow** (**HWF**) for allowing human interaction with long running processes
- Spring for integrating Java Spring components

This book looks at many common problems that are encountered when integrating systems and provides solutions to them in the form of more than 67 cookbook recipes. The solutions explain the problem to be solved alongside clear step by step instructions to implement a solution using SOA Suite components. Each recipe also includes a discussion of how it works and what additional problems may be tackled by the solution presented.

What This Book Covers

Chapter 1, Building an SOA Suite Cluster, explains how to prepare the environment to follow Oracle's Enterprise Deployment Guide. The Enterprise Deployment Guide is Oracle's blueprint for building a highly available SOA Suite cluster. The chapter includes key questions to ask the network storage team, the networking team, and the Database Administrators before the actual SOA Suite installation and deployment begins.

Chapter 2, Using the Metadata Service to Share XML Artifacts, explains how we can use MDS to share XML artifacts, such as XML schemas, WSDL's fault policies, XSLT Transformations, EDLs for event EDN event definitions and Schematrons between multiple composites.

Chapter 3, Working with Transactions, looks at the different ways to use transactions within SOA Suite. This includes enrolling a BPEL process in an existing transaction, forcibly committing or aborting a transaction within BPEL and catching faults that have caused the transaction to be rolled back. It also covers how to apply reversing transactions when a system does not support transaction functionality in its public interface.

Chapter 4, Mapping Data, covers how to copy and transform data using the SCA container. It includes how to deal with missing XML elements and how to control the mapping of Java objects to XML including dealing with abstract Java classes. It also covers how to process arrays of data in both BPEL and XML stylesheet transforms (XSLT).

Chapter 5, Composite Messaging Patterns, explores some of the more complex but relatively common message interaction patterns used in a typical SOA deployment. It includes recipes for implementing patterns around message aggregation, singletons, and the dynamic scheduling of BPEL processes and services.

Chapter 6, OSB Messaging Patterns, explores some common message processing design patterns for delegation of execution to downstream services and provides recipes for implementing them using Oracle Service Bus. It includes recipes for dynamic binding to services, splitting out messages, as well as dynamic Split-Joins.

Chapter 7, Integrating OSB with JSON, covers how we can use the Service Bus to integrate with RESTful web services that exchange data using JavaScript Object Notation (JSON) instead of XML. It also looks at how to expose OSB Services as RESTful JSON web services.

Chapter 8, Compressed File Adapter Patterns, explains how to use the file/FTP adapter to compress/uncompress the contents of exchanged files. This is particularly common in Business-to-Business scenarios, where network bandwidth is more of a constraint.

Chapter 9, Integrating Java with SOA Suite, explains different ways to integrate Java code into SOA Suite. This is demonstrated through creating a custom XPath function for use in SCA and OSB, as well as re-using EJBs and Spring Beans in SOA Suite. It also shows how to access the SOA runtime environment from within a BPEL process.

Chapter 10, Securing Composites and Calling Secure Web Services, shows the developer how to restrict access to a composite by applying a security policy, as well as showing how to create a new security policy. It also explains how to make a call to a security protected service and how to manage security stores.

Chapter 11, Configuring the Identity Service, details how to configure the Oracle Platform Security Services (OPSS) to use various LDAP providers for authentication and authorization within the Oracle SOA Suite. It covers configuration for Active Directory, Oracle Internet Directory, Sun iPlanet, and Oracle Virtual Directory.

Chapter 12, Configuring OSB to use Foreign JMS Queues, covers how to configure the Service Bus to read/write messages from various JMS providers, including OC4J, JBoss, and across WebLogic domains.

Chapter 13, Monitoring and Management, includes recipes to monitor the completion status of SOA composites through the EM dashboard, measuring their message throughput in real time. It also covers setting up the SOA environment to use the SOA Suite provided Monitor Express reports to take advantage of pre-built BAM dashboards.

3
Working with Transactions

In this chapter we will cover:

- Modifying a BPEL process to use the callers transaction context
- Committing a transaction
- Aborting transaction
- Catching rollback faults
- Applying a reversing or compensating transaction

Introduction

In this chapter we will examine recipes that allow us to control the transactional behavior of composites.

Transactions defined

A transaction may be thought of as a set of changes to the state of a system. All the changes must be applied together or none of the changes must be applied. For example, a transfer between two bank accounts involves two operations, debiting the payer's account and crediting the payee's account. In this case, if the credit operation fails we don't want to debit the payer's account because the money was not deposited in the payee's account.

We may have more than two changes in a transaction. For example, in addition to transferring the funds in our example, which requires two operations, we may also wish to notify the payer and payee that the transfer has occurred. Again, we want this to be part of the transaction because we do not want to send a notification unless the transfer of funds has also occurred, so we have now extended our transaction to four operations.

Transaction managers

A transaction manager is responsible for coordinating the operations in a transaction. If all the operations are in the same resource, such as the same database, then the resource may manage the transaction itself. If the transaction is spread across multiple resources, such as the database and message queue, then an XA transaction manager is required to co-ordinate the operations across different resources.

SOA Suite by default will use the XA transaction manager in the application server to co-ordinate its transactions. When a message arrives in SOA Suite, an XA transaction is started.

Compensating transactions

Not all transactions are managed by a transaction manager. Sometimes we want the benefits of a transaction but the services we are using are non-transactional, for example basic SOAP over HTTP services. In this case, we need to manage the transactional behavior within our composites.

In our example, if the two accounts are held at two separate banking institutions, things could become more complicated. This does not change the transaction requirements; it just makes implementing a transaction more complicated. We must now provide explicit reversing transactions to undo unwanted work when we are unable to complete all the operations in our transaction. These are called compensating transactions.

Within SOA Suite, the BPEL engine has built-in support for compensating transactions that allow us to register and invoke reversing operations (compensating transactions).

Hints on working with SOA Suite transactions

Always have a clear plan of where you want transaction boundaries to occur. Determine if you want BPEL processes to be part of existing transactions or if you want to execute them within their own transaction. Transactions can be committed by using a dehydrate statement or by calling a non-idempotent service. It is often helpful to create a diagram showing transaction boundaries within your composite.

Modifying a BPEL process to use the callers transaction context

We often want to include a BPEL process in the calling transaction, and this recipe shows how to modify the BPEL process to do this.

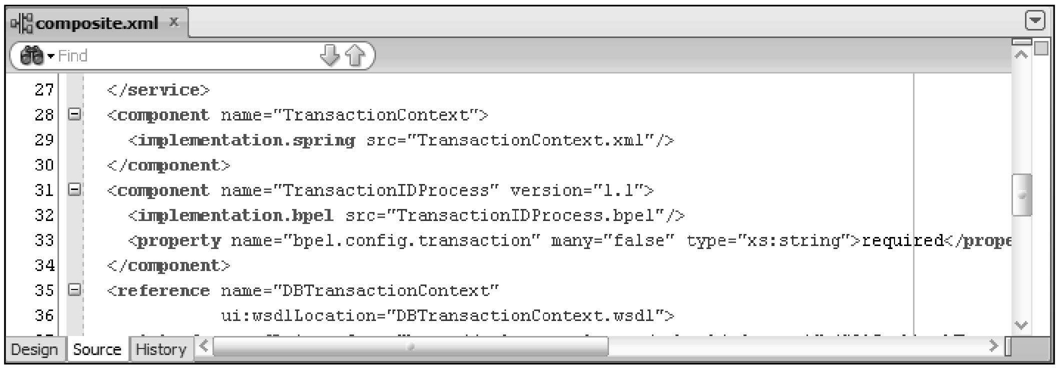
Getting ready

In JDeveloper, open the project that has the BPEL process that we want to make part of the calling transaction.

How to do it...

1. Switch to Source View.

   In JDevelper, open `composite.xml` that contains the BPEL process and click on the **Source** tab at the bottom of the diagram:



```
</service>
<component name="TransactionContext">
  <implementation.spring src="TransactionContext.xml"/>
</component>
<component name="TransactionIDProcess" version="1.1">
  <implementation.bpel src="TransactionIDProcess.bpel"/>
  <property name="bpel.config.transaction" many="false" type="xs:string">required</prope
</component>
<reference name="DBTransactionContext"
           ui:wsdlLocation="DBTransactionContext.wsdl">
```

2. Add a Transaction Required property.

   Find the component that corresponds to the BPEL process in Source View; the component name attribute will be the same as the name of the BPEL process. Add a property called `bpel.config.transaction` with the value `required` to the component, as shown in the following code:

```
<component name="TransactionIDProcess" version="1.1">
  <implementation.bpel src="TransactionIDProcess.bpel"/>
  <property name="bpel.config.transaction"
            many="false"
            type="xs:string">required</property>
</component>
```

The BPEL process will now participate in the same transaction as the caller of the process.

How it works...

When creating a BPEL process with `sync` delivery in SOA Suite 11.1.1.6 and higher, we can specify the transaction attributes of the BPEL process as `required` or `requiresNew`. This sets the `bpel.config.transaction` property.

The `bpel.config.transaction` property has two values:

- `required`: This makes the BPEL process execute as part of the calling transaction. If no calling transaction exists, it will create a new one.
- `requiresNew`: This is the default value and makes the BPEL process execute as a separate transaction. Any existing transaction will be suspended.

These properties define the transaction semantics of the BPEL process to which they are applied. Any JCA adapters, such as a database or JMS adapter, can also be executed in the same transaction context as the BPEL process.

There's more...

When executing, the BPEL engine keeps track of which activities have occurred by updating the state in the dehydration database. This updating of the process state is done in the same transaction context in which the BPEL process is being executed. This keeps the state of the BPEL process in sync with the state of the resources used by the BPEL process. These updates are only committed when the process is dehydrated or a `RequiresNew` process is completed. One way in which this can occur is following a call to a non-idempotent partner link.

The BPEL engine uses a separate transaction context to keep a record of which steps were attempted; this is used to update the database with logging information and means that even if the BPEL process transaction rolls back, it will be possible to see what activities were executed before the rollback. This aids in debugging a failing BPEL process.

See also

- The *Aborting a transaction* recipe in this chapter.

Committing a transaction

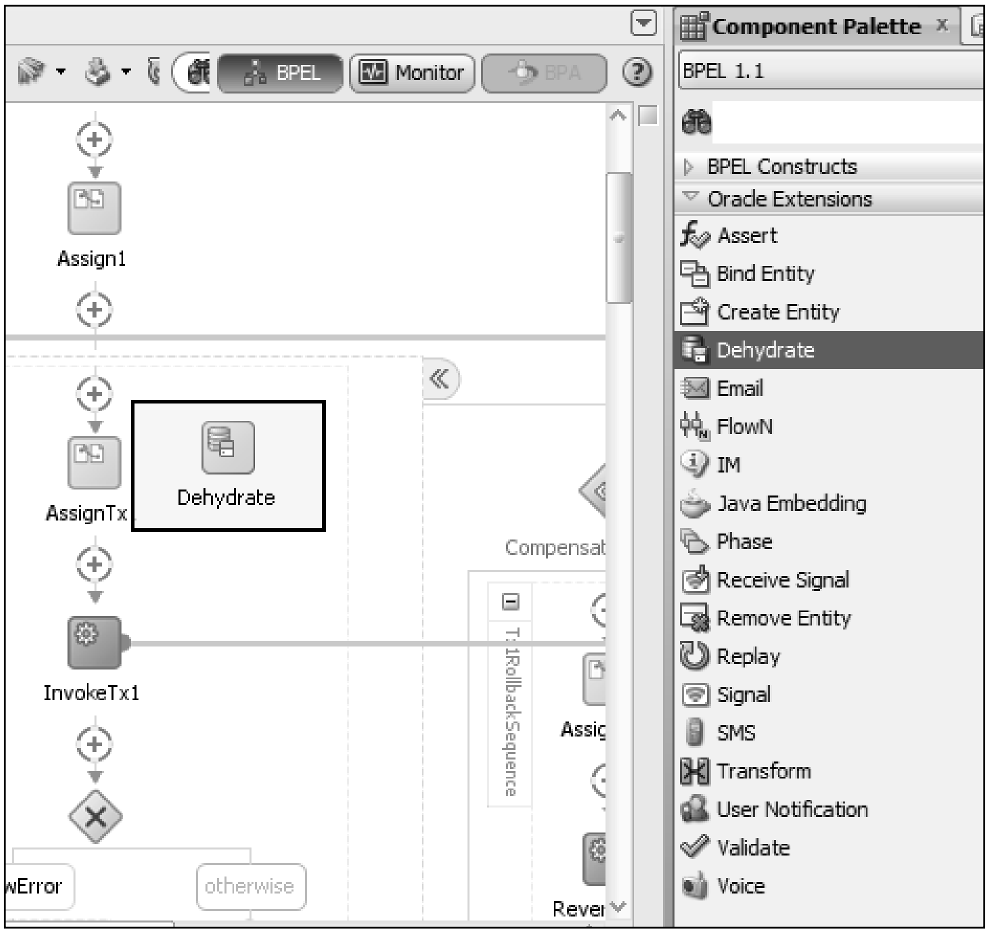
We may wish to explicitly commit a transaction in our BPEL process. This recipe describes how to achieve this.

Getting ready

In JDeveloper, open the project containing the BPEL process that you wish to explicitly commit a transaction to.

How to do it...

1. Add a **Dehydrate** activity to the process.
2. Open the BPEL process that needs to explicitly commit the transaction.
3. From the **Component Palette** expand the **Oracle Extensions** section; drag a **Dehydrate** activity onto the BPEL process:



When executed, this will cause the current transaction to be committed and a new transaction to be started.

How it works...

The **Dehydrate** activity causes the current state of the BPEL process to be saved in the dehydration database. This also causes the current transaction context to be committed. Because the BPEL process is still active, a new transaction context is immediately created.

There's more...

A **Dehydrate** activity can be very useful in an asynchronous process, but should be avoided in a synchronous process unless the BPEL process's `bpel.config.transaction` property is not set or set to `requiresNew`.

> Synchronous BPEL processes that have the required transaction attribute should always leave the committing of the transaction to the caller; they should never force the committing of a transaction.

See also

- The *Catching rollback faults* recipe in this chapter.

Aborting a transaction

If an error occurs while we are in a transaction, we may wish to abort the transaction, thus rolling back any work that has already been done. This recipe shows how to rollback the currently executing transaction.

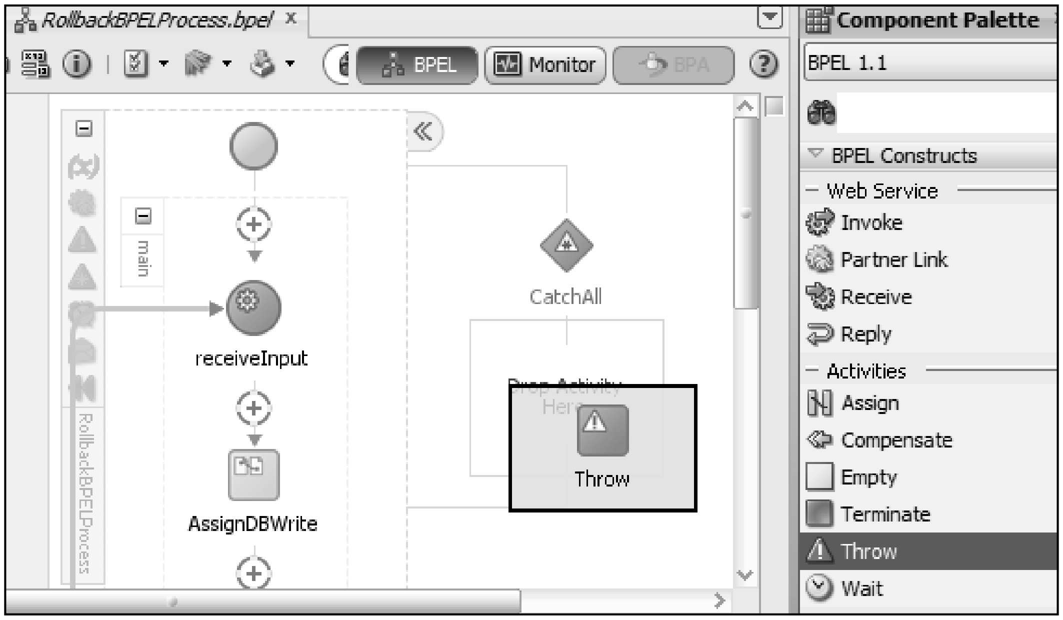
Getting ready

In JDeveloper, open the project containing the BPEL process, which may encounter errors, requiring the transaction to be rolled back.

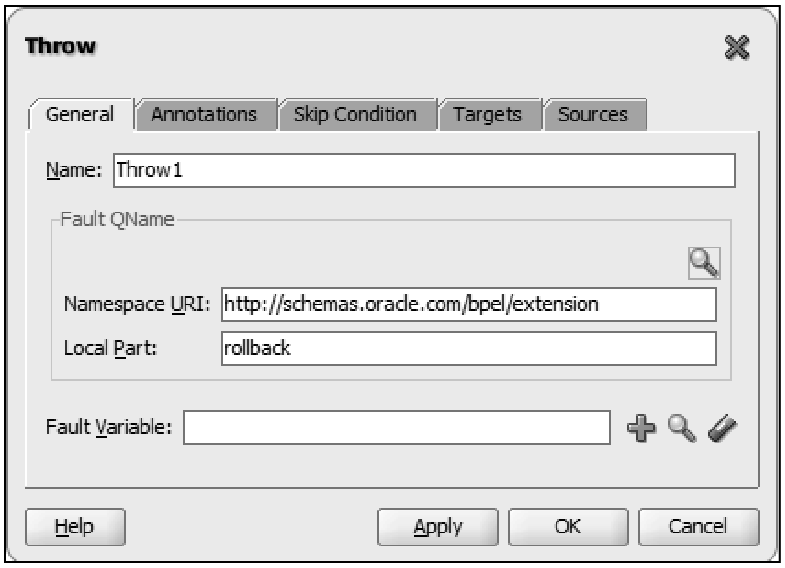
How to do it...

1. Open the BPEL process that needs to cause the transaction to be rolled back.

2. From the **Component Palette**, drag a **Throw** activity onto the BPEL process:



3. Double-click on the **Throw** activity that was created in the previous step.
4. Click on the 🔍 icon in the **Fault QName** section of the dialog to launch the **Fault Chooser** dialog:



Name: Throw1

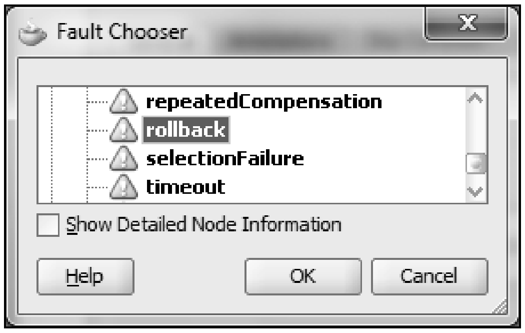
Fault QName

Namespace URI: http://schemas.oracle.com/bpel/extension

Local Part: rollback

Fault Variable:

5. Select the **rollback** fault and click on **OK**:



6. Click on **OK** to apply the changes to the **Throw** activity.

This activity will cause the current BPEL process transaction to be rolled back when executed.

How it works...

The `rollback` fault has a special meaning to the BPEL engine and causes the current transaction to be rolled back. The BPEL process will be restored to the state that it was in before the current transaction was started. A `rollback` fault can't be caught by a BPEL process in the same transaction context.

See also

- The *Catching rollback faults* recipe in this chapter

Catching rollback faults

A BPEL process may want to catch a rollback fault thrown by another BPEL process. This recipe shows how to do that.

Getting ready

Open the composites containing the caller BPEL process and the callee BPEL process.

How to do it...

1. Open `composite.xml` containing the callee BPEL process (the BPEL process that throws a `rollback` fault) and switch to the **Source View** tab.
2. Locate the `component` element corresponding to the callee BPEL process and verify that either:

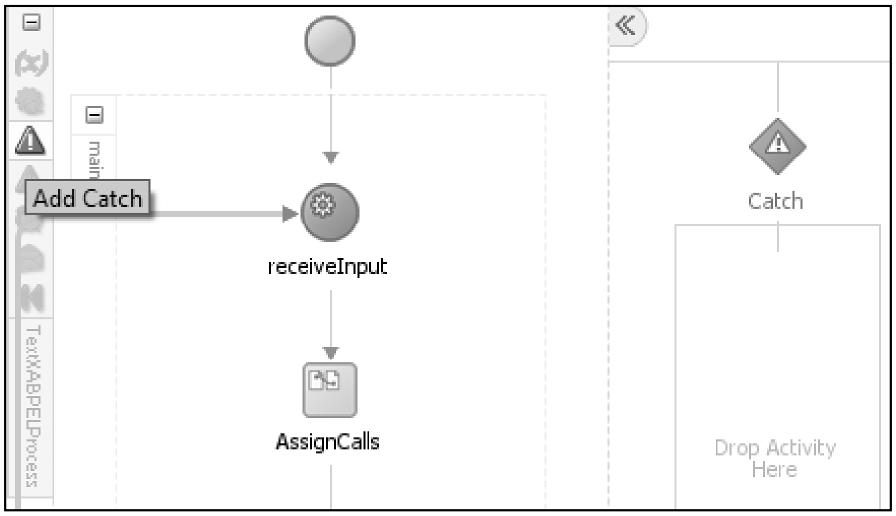
   There is no `bpel.config.transaction` property:

   ```
   <component name="BPELProcess1" version="1.1">
     <implementation.bpel src="BPELProcess1.bpel"/>
     <!—No bpel.config.transaction property -->
   </component>
   ```

   Or the `bpel.config.transaction` property is set to `requiresNew`:

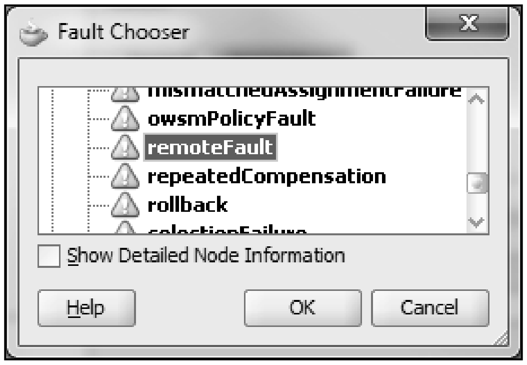
   ```
   <component name="BPELProcess2" version="1.1">
     <implementation.bpel src="BPELProcess2.bpel"/>
     <property name="bpel.config.transaction"
               many="false"
               type="xs:string">requiresNew</property>
   </component>
   ```

3. Open the caller BPEL process and add a catch block by selecting the triangular (⚠) icon on a scope containing the `invoke` activity to the callee BPEL process:

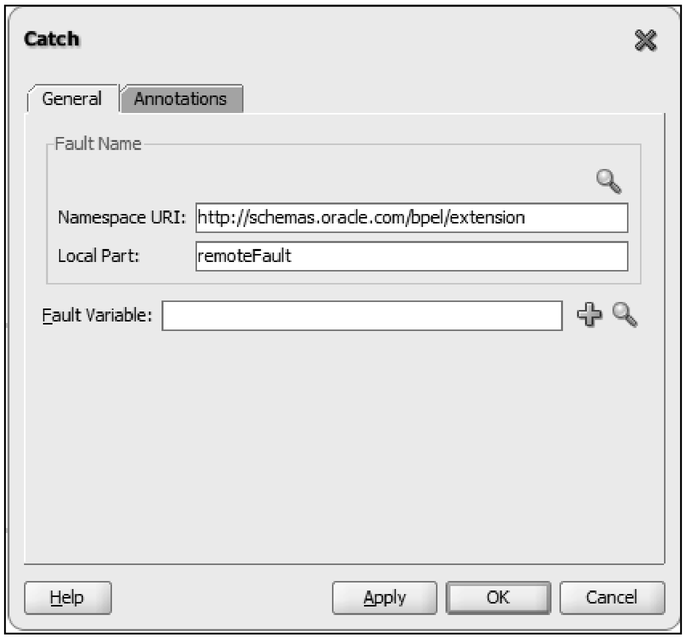


4. Double click on the **Catch** to bring up the **Catch** dialog.

5. Click on the 🔍 icon in the **Fault Name** section of the dialog to launch the **Fault Chooser** dialog:



6. Select the **remoteFault** fault and click on **OK**:



Namespace URI: http://schemas.oracle.com/bpel/extension
Local Part: remoteFault
Fault Variable:

7. Click on **OK** to apply the changes to the catch.

How it works...

The catch block is to be executed when a rollback fault is thrown in the callee BPEL process.

When a BPEL process throws a `rollback` fault, it cannot be caught in the current transaction context. When the fault leaves the current transaction context, it is converted to a `remoteFault` that can be caught in the caller BPEL process. It is necessary to make sure that the caller and callee are in separate transaction contexts; hence the need to check the value of the `bpel.config.transaction` property is not set to `required`.

When a BPEL process throws any fault that is not caught in the current transaction context, it causes the current transaction to be rolled back. If instead of throwing a fault a BPEL process returns a fault through a `reply` activity, then the current transaction is not rolled back.

Applying reversing or compensating transactions

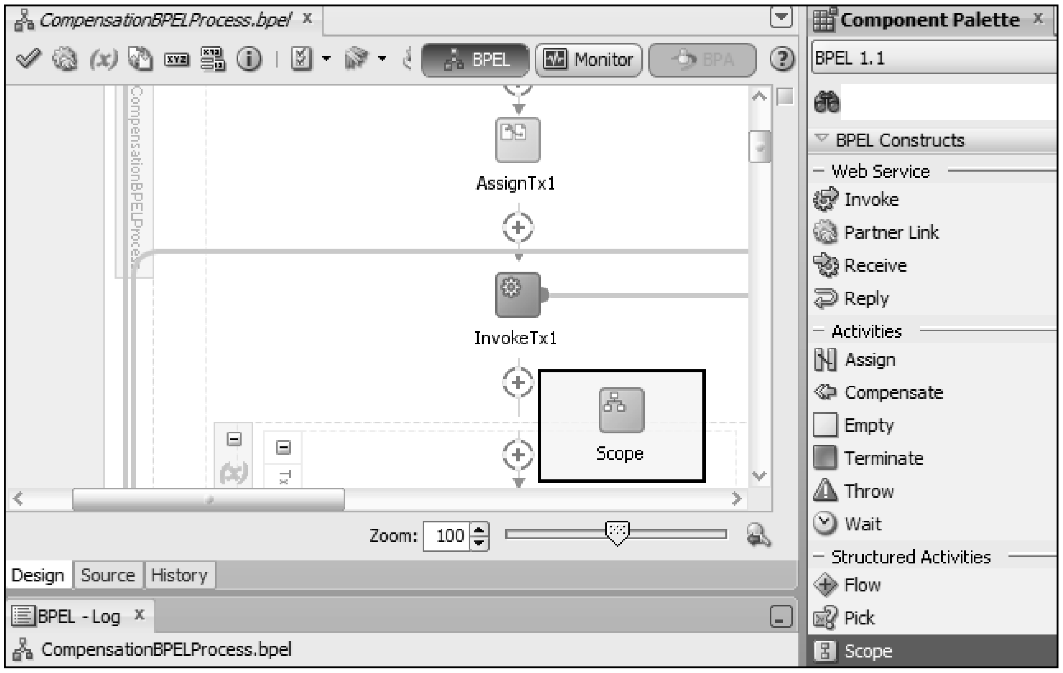
If operations occur that are not part of a transaction, then reversing operations must be applied to undo the changes. The reversing operations are performed to reverse the effects of the unwanted operations. This recipe shows how to do that.

Getting ready

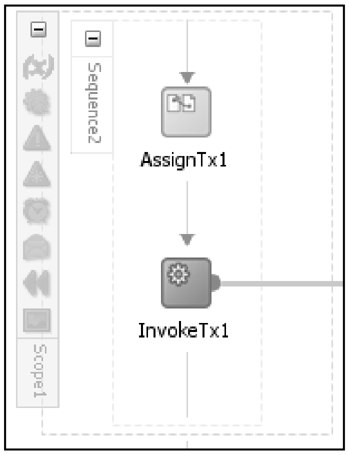
Open the BPEL process that performs operations that cannot be rolled back as part of a transaction, and instead requires reversing operations to be applied in the case of a processing failure.

How to do it...

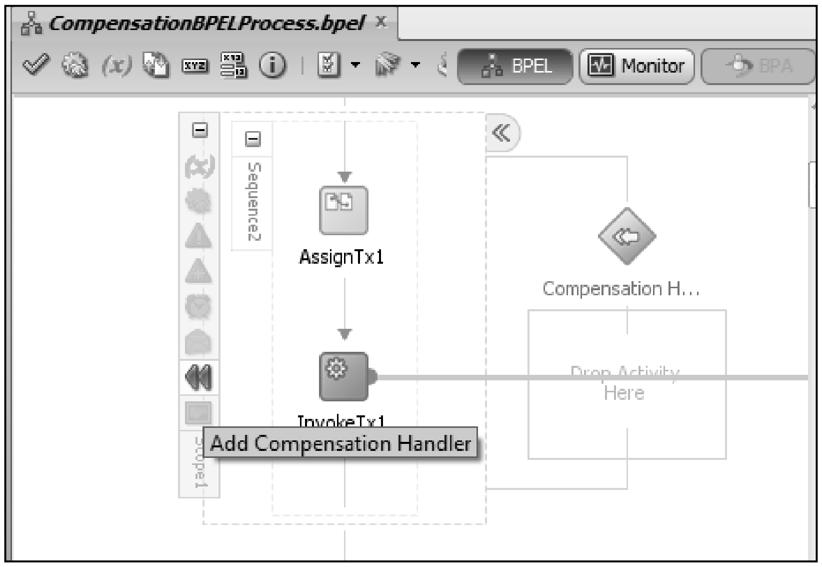
1. For each operation (usually an `invoke` activity) that has a corresponding reversing operation, wrap it in a `scope` activity by dragging a scope from the **Component Palette** and dropping it just after the operation that requires reversing:



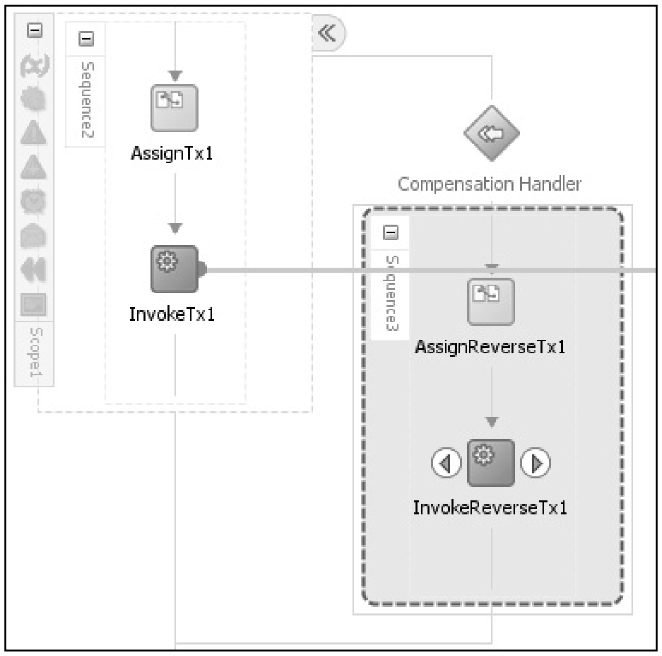
2. Move the activity or activities that require reversing into the scope by dragging-and-dropping them into the scope:



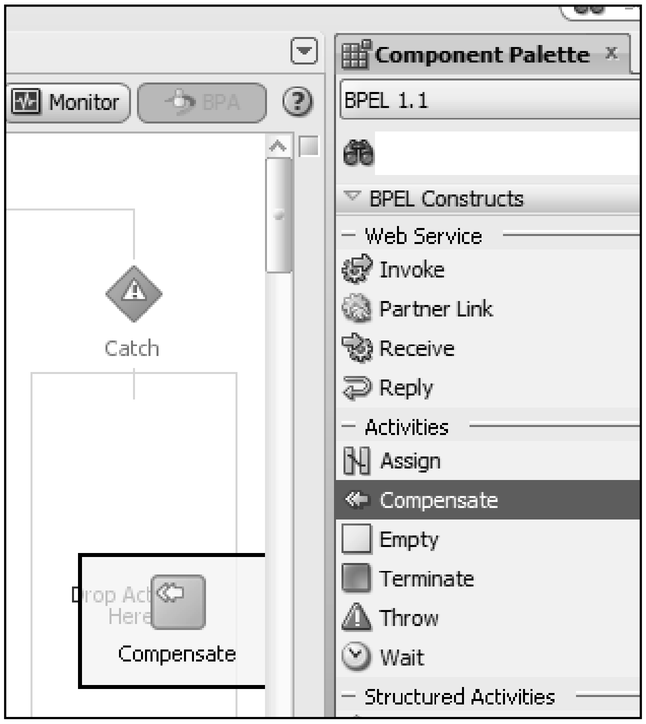
3. Click on the **Add Compensation Handler** icon in the scope to add a compensation handler:



4. Drag appropriate activities (usually an `assign` and an `invoke`) into the compensation handler to reverse the operations in the corresponding scope:



5. If a failure in an operation does not throw a fault, then create a **Throw** activity by dragging it into the scope that contains the failed operation (see step 2 of the recipe *Aborting a transaction* in this chapter).
6. Choose an appropriate fault type (see step 4 of the recipe *Aborting a transaction*).
7. Drag appropriate activities (usually an `assign` and an `invoke`) into the compensation handler to reverse the operations in the corresponding scope.
8. Add a catch block to the outermost scope (see step 3 of the recipe *Catching rollback faults*).
9. Drag a **Compensate** activity from the **Component Palette** onto the catch that was just created:



How it works...

When a `fault` is caught by `catch`, `compensate` it will cause all the operations that require reversing to be reversed.

`Compensate` can only be called from within a `catch` block. When it is called, it starts with the most recently completed `scope` and calls the `compensation handler` for that `scope` if it has one. It then looks for the previously completed `scope` and calls the `compensation handler` for that `scope`. In this way, the reversing operations are applied in the reverse order to the original operations. If a scope was not completed, its compensation handler will not be invoked.

If an operation throws a `fault`, we can `catch` it and call the **Compensate** activity. Operations may indicate failure by returning a failure status rather than throwing a `fault`. In this case, by placing a **Throw** activity inside the `scope` for which the operation failed, we can avoid the reversing operation being invoked for that `scope`.

Compensation occurs outside of transaction boundaries. So a BPEL process may be spread across several transactions, but compensation ignores this and continues to invoke compensation handlers for completed scopes regardless of the transaction context in which they were completed.