



This is Unit #12 of the BPEL Fundamentals course. In past Units we've looked at ActiveVOS Designer, created a process, added our global declarations and interaction activities. Then, we looked at the Sequence, Assignments and Copies and at Correlation. The last units covered Scopes and Fault Handling. In this Unit we will take a look at a new topic, Compensation.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - What is compensation
 - Compensation handlers
 - Invoking compensation handlers

Compensation Overview

- Previous activities which had successfully completed may need to be undone
 - May have completed hours, days, or weeks ago
 - Reversing work is a form of transaction management
- Due to the nature of business process usually being long-running and asynchronous
 - Applying the concepts of ACID transactions is not applicable
 - Resource locks and isolation cannot effectively be maintained for long periods of time

3 Copyright © 2004-2008 Active Endpoints, Inc.



The purpose of a compensation handler is to undo successfully completed activities. Compensation Handlers are used to manage long-running transactions, as they can be invoked long after the compensatable activity has been completed. For example, if we created an order and we need to “undo” the order, we can use a compensation handler to do so. Unlike database transactions, long-running business processes can’t be handled as ACID transactions because it is impractical to lock resources for minutes or hours, let alone for days or even months. BPEL's Compensation Handlers provide this necessary functionality.

ACID Transaction definition (for Reference):

Atomicity: all transactions are executed properly or none are executed (DB updates are performed...)

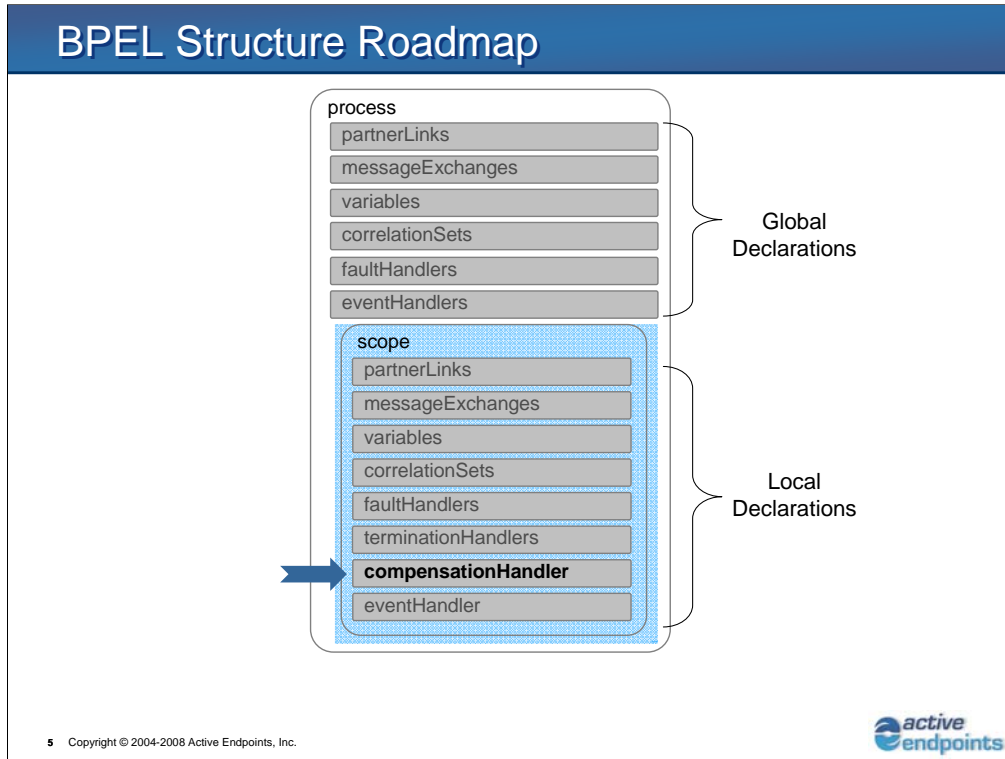
Consistency: execution of the transaction preserves the consistency of the database (sum of balances remains unchanged.)

Isolation: each transaction is unaware of other transactions executed “concurrently” (one is finished before the other is started.)

Durability: changes performed by the transaction persist in the database (DB recovery)

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ What is compensation
 - Compensation handlers
 - Invoking compensation handlers



In BPEL, a Compensation Handler is a local declaration, done at the scope level only, and is not normally available for the Process itself. However, ActiveVOS Designer adds extensions to allow process level compensation handling, which is an advanced topic that we will not cover in this class.

Compensation Handler Overview and Syntax

- Used to define the activities needed to undo previously successful work
 - BPEL calls this feature Long-Running Business Transactions
 - Compensation handlers can be defined
 - For a scope
 - Inline for the `invoke` activity

```
<compensationHandler>?  
  activity  
</compensationHandler>
```

6 Copyright © 2004-2008 Active Endpoints, Inc.



Compensation Handlers define how to undo units of work that were completed successfully. BPEL refers to these as “Long-Running Business Transactions.” Compensation Handlers can also be defined “inline” for Invoke activities, much like the inline fault handlers for Invokes.

Compensation Handler Overview

- **scopes** can be used to demarcate a part of the process that is designed to be compensated
- Compensation **MUST** be explicitly defined
 - There is no concept of an automatic “rollback” in BPEL

7 Copyright © 2004-2008 Active Endpoints, Inc.



Scopes can be specifically defined to demarcate “logical units of work” that can be compensated. Note that Compensation Handling is not automatically provided. If you need to compensate, you need to then define how the compensation activities will work. Think of this as Newton’s 3rd law – “For every action, there is an equal and opposite reaction” translated to “for every activity there is an equal and opposite 'unactivity'.” Note also that Compensation is *not* the same as a rollback. There is no concept of an “automatic rollback” in BPEL.

Reference definitions for Compensation and Rollback:

- Compensation is undoing something that was successfully done, i.e., a *logical* undo
- Rollback is undoing something such that it was never done at all, i.e., an *actual* undo

Fault Handlers vs. Compensation Handlers

■ Fault handling

- Attempts to recover from an activity or set of activities that could not finish normally
- Due to a fault occurring within the same context

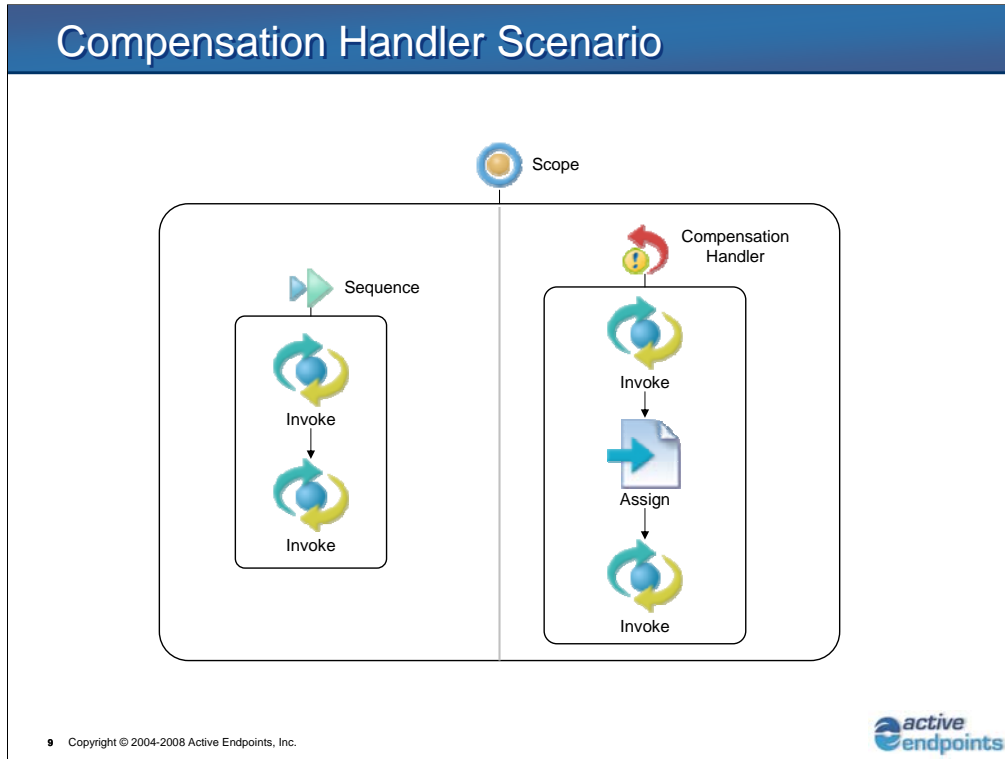
■ Compensation handling

- Attempts to undo the work of a activity or set of activities that have already been successfully completed
- In response to a fault occurring within a parent context

8 Copyright © 2004-2008 Active Endpoints, Inc.



Now, to head off any confusion, let's take a look at the difference between Fault Handlers vs. Compensation Handlers. The main difference is in their essential purposes. Fault Handlers enable the process to recover from *abnormally terminated* actions, whereas Compensation Handlers reverse *successfully completed* actions. The other difference is the context from which they are called. A BPEL Fault Handler is invoked in response to a fault in the *same* context (i.e., the same scope), whereas Compensation Handlers are invoked in response to a fault in a *higher* context (i.e., the parent scope).



Let's look at a typical scenario for a Compensation Handler. On the left we see the Sequence with the two invokes is this scope's primary activity. If the scope completes successfully and we need to undo its actions, we invoke the scope's Compensation Handler, which is defined on the right.

Compensation Handler Example

```
<scope>
  <compensationHandler>
    <sequence>
      <invoke ... />
      <assign ... />
      <invoke ... />
    </sequence>
  </compensationHandler>
  <sequence>
    <invoke ... />
    <invoke ... />
  </sequence>
</scope>
```

10 Copyright © 2004-2008 Active Endpoints, Inc.



Here is the syntax for the previous example. Inside the scope a Compensation handler is declared, and it contains a sequence of activities – invoke, assign, invoke. Then the scope’s primary Sequence follows, containing two invokes.

Compensation Handler Semantics

- Compensation is performed only on **scopes** which have completed normally
- Order in which compensation handlers are performed is usually important
 - By default, compensation is performed in the reverse order of the completion of those *scopes*

11 Copyright © 2004-2008 Active Endpoints, Inc.



To review some key points, Compensation Handling is only performed on scopes that complete normally. The *order* in which the compensation is performed usually makes a difference. Compensation is usually performed in *reverse* order of the completion of the scopes involved.

Variable Access in Compensation Handlers

- Compensation handler activities have access to the same set of variables that regular activities can access

12 Copyright © 2004-2008 Active Endpoints, Inc.



Compensation Handlers have the same variable access as the scope in which they are defined. So, if I can see a variable from inside the Scope, then the Compensation Handler defined for that Scope can also see it.

Default Compensation Handling

- Whenever a compensation handler is not present for a given `scope` then one is implicitly created
 - Behavior is to invoke the compensation handlers for immediately enclosed `scopes`
 - In the reverse order of their completion
- One additional behavior added for default fault handling
 - Behavior is to invoke each compensation handler for the immediately enclosed `scopes`
 - In the reverse order of their completion
 - In addition to re-throwing the fault to the parent `scope`

13 Copyright © 2004-2008 Active Endpoints, Inc.

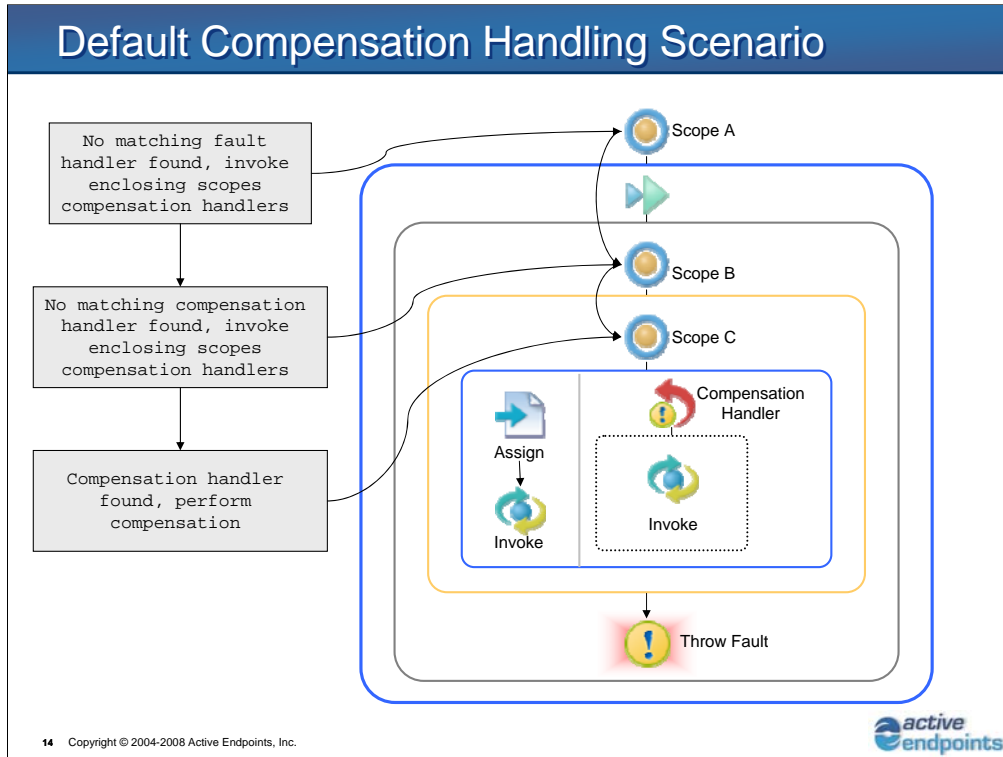


The BPEL language provides Default Compensation Handling functionality if there is no Compensation Handler defined for a given scope. To do this, BPEL invokes the Compensation Handler for any immediately enclosed scopes (i.e., any direct child scopes) in reverse order of their completion and then Rethrows the fault to the parent scope (or process.)

For Reference:

BPEL2.0 specification, sec. 12.5.1 = Default Fault, Compensation, and Termination Handlers

BPEL2.0 specification, sec. 12.5.2 = Default Compensation Order



Here we have a Process containing a Scope A. Inside Scope A is a Sequence and also notice that there is a activity at the bottom of Scope A's Sequence that will throw a fault. Inside the Sequence is a Scope B and inside Scope B is a Scope C. This Scope C has two activities – an Assign and an Invoke – and has defined a Compensation Handler. Let's throw a fault and trace the compensation logic for the process.

- 1.) There is a Fault thrown in Scope A, after Scopes B and C complete successfully.
- 2.) The Fault Handler for Scope A is invoked, but there isn't one.
- 3.) Default fault handler for Scope A invokes the Compensation Handler for Scope B, but there isn't one.
- 4.) The Default fault handler for Scope B is invoked.
- 5.) Default fault handler for Scope B invokes the Compensation Handler for Scope C.
- 6.) Compensation Handler for Scope C is found, it is invoked and the Compensation is performed.

Inline Compensation Handling

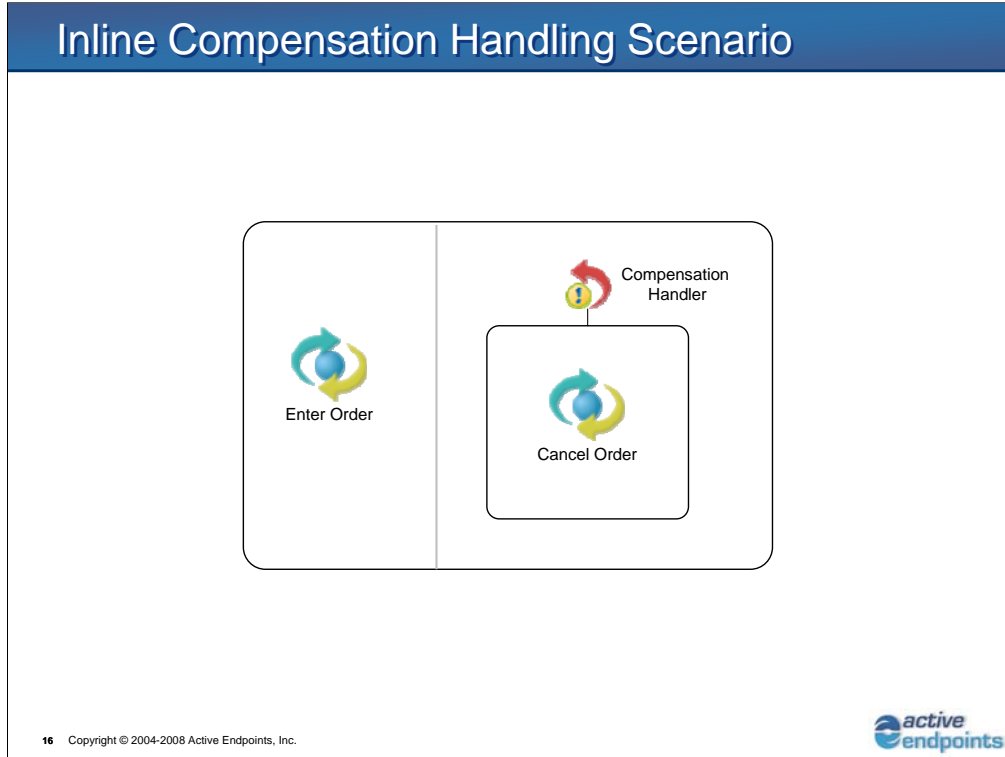
- The **invoke** activity provides a special shortcut to directly handle compensation activities related to the invoke

```
<invoke partnerLink="ncname" portType="qname"? operation="ncname"
        inputVariable="ncname"? outputVariable="ncname"?
        standard-attributes>
  standard-elements
  ...
  <compensationHandler?>
    activity
  </compensationHandler>
  ...
</invoke>
```

15 Copyright © 2004-2008 Active Endpoints, Inc.



Just as we have inline Fault Handling for our Invoke activities, we also have inline Compensation Handling for Invoked services. And, just as with our Inline Fault Handlers, our inline Compensation Handlers are defined inside the Invoke activity itself. In this example we see the declaration for the invoke, followed by the partnerLink, portType, operation and input and output variables and all of the standard attributes and elements. Below these, we see the declaration for the Compensation Handler and its primary activity.



Here is a graphic of the example, as it would appear in the ActiveVOS Designer's Process Editor. Just as with Inline Fault Handlers, we have a Scope enclosing one activity - an Invoke – and then its Compensation Handler, which contains the “CancelOrder” Invoke activity.

Inline Compensation Handling Example

```
<invoke partnerLink="erpSystem" portType="..."
  operation="EnterOrder"
  inputVariable="..." />
  <compensationHandler>
    <invoke partnerLink="erpSystem" portType="..."
      operation="CancelOrder"
      inputVariable="..." />
    </compensationHandler>
  </invoke>
```

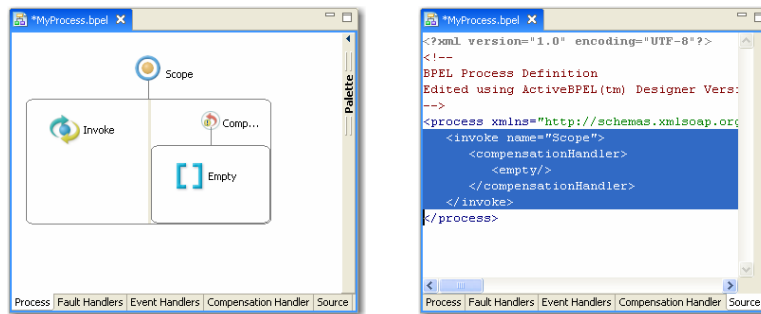
17 Copyright © 2004-2008 Active Endpoints, Inc.



Here is the syntax for the Invoke that we saw on the last slide. The partnerLink is “ERPSystem”, there is a portType, and then the Operation “EnterOrder” followed by an input variable. The Invoke's Inline Compensation Handler is defined next, followed by the definition of its primary activity, which is also an Invoke. That invoke activity executes the “CancelOrder” operation that reverses the creation of the original sales order.

Working with Inline Compensation Handling

- Modeling an inline Compensation Handler using ActiveVOS Designer
 - Requires a `scope` activity which contains a single `invoke` activity
 - Enable Show Compensation Handler view on `scope`



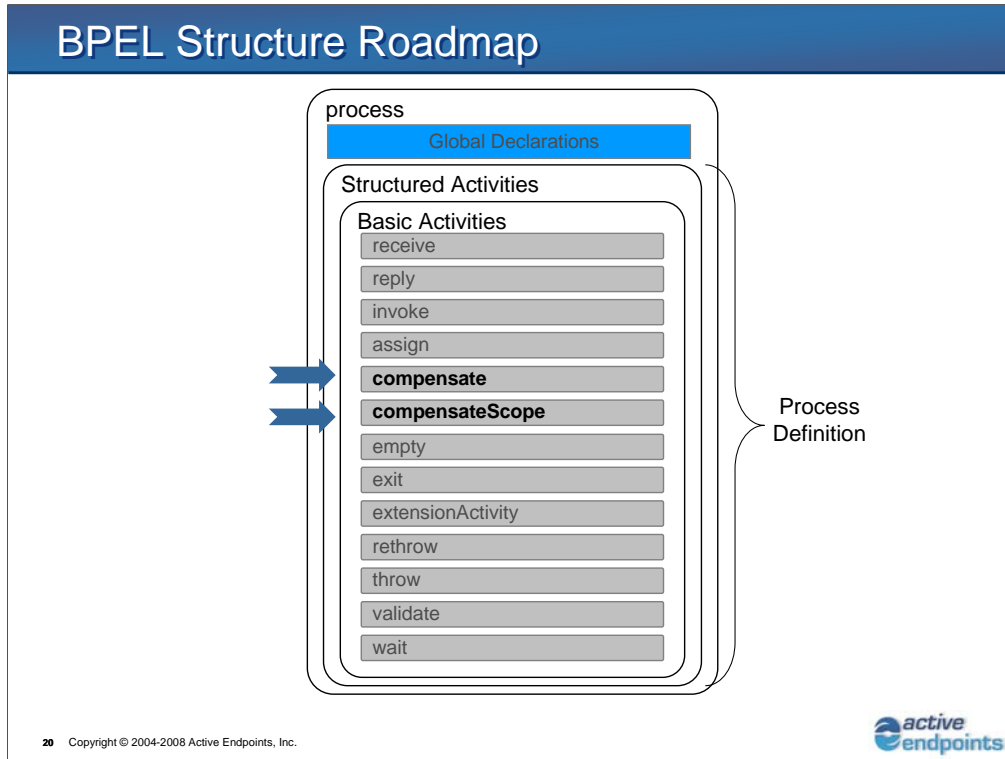
18 Copyright © 2004-2008 Active Endpoints, Inc.



Syntactically, an Inline Compensation Handler is similar to an Inline fault handler. When working in the Process Editor, place the Invoke inside a scope activity, then provide a Compensation Handler for it, just the same as you would an inline Fault Handler. Note that Compensation Handlers can be hidden/viewed (toggled) by using the Right Mouse context menu and selecting "Show Compensation Handlers."

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ What is compensation
 - ✓ Compensation handlers
 - Invoking compensation handlers



So, now that we understand why we need Compensation Handlers and we know how they work, how do we actually implement them? We have two Basic BPEL activities for this, **Compensate** and **CompensateScope**. Both are part of our process definition.

Compensate Activities Overview

- Used to explicitly invoke a compensation handler on the immediately enclosing **scope**
- Can only be invoked from the following areas:
 - Fault handler, compensation handler or termination handler of the parent `scope` for which compensation should be performed
- Compensation handler for a **scope** can only be invoked when the **scope** has completed normally
 - Invoking a compensation handler for a `scope` that completed abnormally results in a “no-op” `empty` activity being performed
- Can be performed using either one of these activities:
 - `compensate`
 - `compensateScope`

21 Copyright © 2004-2008 Active Endpoints, Inc.



First, we'll cover the **Compensate** Activity. `Compensate` is used to explicitly invoke a Compensation Handler on the immediately enclosing (i.e., parent) scope. `Compensate` can be invoked from a Fault Handler, from a Compensation Handler or from a Termination Handler of the parent scope. If we try to invoke a Compensation Handler for an abnormally terminated (i.e., for a faulted) scope it results in a “no operation.” Compensation can only be performed on scopes that completed successfully.

The `Compensate` and `CompensateScope` activities are used to call the Compensation Handlers for scopes

- `Compensate` = a general/generic call to any defined Compensation Handlers
- `CompensateScope` = executes the Compensation Handler for a specifically targeted scope

compensate Activity Syntax

- **compensate** - use to start compensation for all inner **scopes** that have already completed successfully in default order

```
<compensate standard-attributes>
  standard-elements
</compensate>
```

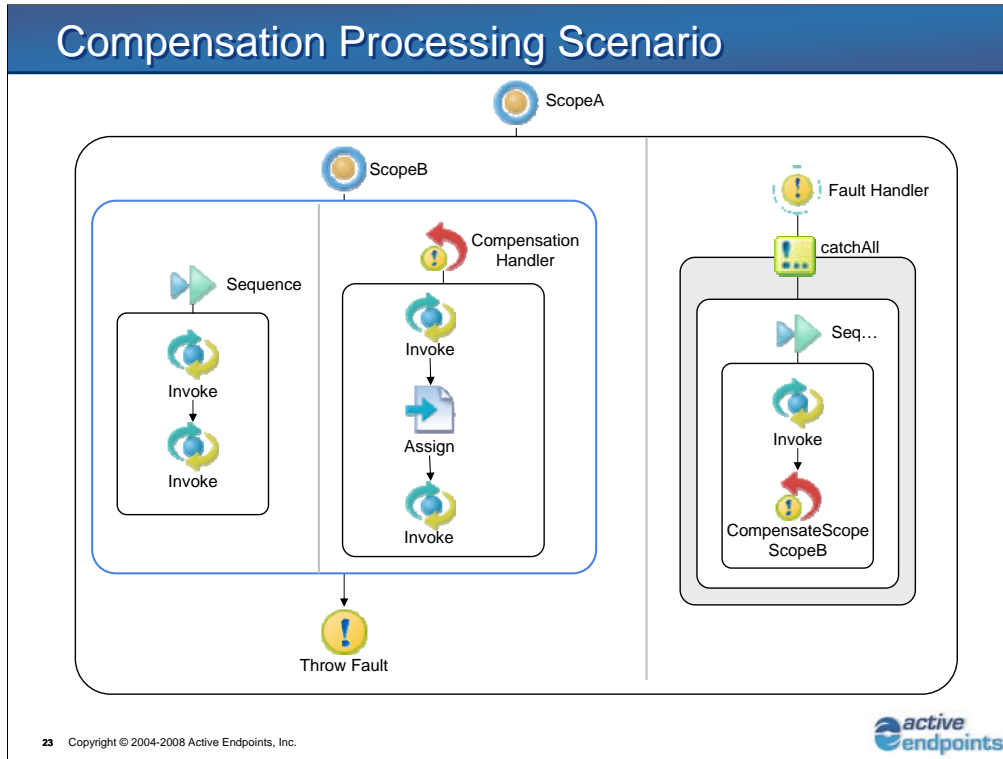
- **compensateScope** - use to start compensation for a specified inner **scope** that has already completed successfully

```
<compensateScope target="NCName" standard-attributes>
  standard-elements
</compensateScope>
```

22 Copyright © 2004-2008 Active Endpoints, Inc.



Why do we have two Compensation activities? Compensate is to start compensation for all inner scopes, that is, *all* scopes enclosed by the child scopes of the current scope – to any depth. CompensateScope is to start compensation at a *specific* inner scope. You must give the name of the target scope as the target for the activity.



Now let's take a look at a typical Compensation processing scenario. We have an outer Scope A, which has a nested Scope B. Scope A has a Fault Handler defined (in the panel on the right...) Scope B has a Compensation Handler defined (in the panel in the middle...) As we execute Scope A, at some point control passes to Scope B, which calls one Invoke, and then the second. If either of these two Invokes throws a fault, or if they both complete successfully and you exit Scope B and hit the "Throw Fault," then you would execute the Fault Handler for Scope A. Scope A's Fault Handler runs all faults through the defined "catchAll", which has an enclosing Sequence activity and then an Invoke and the "CompensateScope" call, which uses "Scope B" as the target for the action.

Note: Yes, this process - as defined – will *a/ways* throw a fault, because every execution path ends up at the "Throw Fault" activity.

compensate Processing Example

```

<scope name="ScopeA">
  <faultHandlers>
    <catchAll>
      <compensateScope target="ScopeB" />
    </catchAll>
  </faultHandlers>
  <sequence>
    <scope name="ScopeB">
      <compensationHandler>
        <sequence>
          <invoke ... />
          <assign ... />
          <invoke ... />
        </sequence>
      </compensationHandler>
      <sequence>
        <invoke ... />
        <invoke ... />
      </sequence>
    </scope>
    <throw faultName="ns:foo" />
  </sequence>
</scope>

```

24 Copyright © 2004-2008 Active Endpoints, Inc.



Now, let's take a look at the syntax for the previous example. We are inside Scope A, and inside that scope we have a Fault Handler defined. Inside that fault Handler is a catchAll that calls "compensateScope" with Scope B as its target. Inside Scope A is its primary activity, which is the outer Sequence. Inside the Sequence is a Scope B and it has a Compensation Handler defined that has a nested sequence: Invoke, Assign, Invoke. The Scope B has a primary activity that is a Sequence, defined with two Invoke activities. Now we are outside Scope B and back into Scope A, and here we see the Throw activity, which throws a fault named "ns:foo."

compensate Activity Semantics

- A **compensate** and **compensateScope** activity may be defined only
 - Inside a fault handler (`catch`, `catchAll`)
 - Within a compensation handler
 - Within a termination handler

25 Copyright © 2004-2008 Active Endpoints, Inc.



Compensate and the Compensate Scope activities can only be defined: in a Fault Handler (i.e., inside the Catch or catchAll), in a Compensation Handler or in a Termination Handler.

Lab 7 – scope, Fault & Compensation Handlers

■ Overview of Lab Exercises

- Add a `scope` for Order Entry section
- Add a Fault Handler to the `scope` to catch faults that may occur
- Add a Compensation Handler to undo the Order Entry if something downstream goes wrong

26 Copyright © 2004-2008 Active Endpoints, Inc.



The next Lab in the BPEL Fundamentals class is Lab #7. In this lab we will create two Scopes, a Fault Handler and a Compensation Handler:

- 1.) We'll put the OrderEntry sequence inside a scope
- 2.) We'll add a Fault Handler to the scope
- 3.) Then we'll add a Compensation Handler to the scope to undo scope activity, if necessary.

Once we have made these changes, then we add the proper activities to each container.

Unit Summary

- Now you are familiar with:
 - What is compensation
 - Compensation handlers
 - Invoking compensation handlers