BPEL for Orchestration and Choreography

BPEL

- BPEL (or BPEL4WS) is a language used for the definition and execution of business processes using Web services.
- BPEL enables the top-down realization of Service Oriented Architecture (SOA) through composition, orchestration, and coordination of Web services.
- BPEL provides a relatively easy and straightforward way to compose several Web services into new composite services called *business processes*.

BPEL Technologies

- BPEL builds on the foundation of XML and Web services;
- It uses an XML-based language that supports the Web services technology stack, including SOAP, UDDI , WSDL, WS-Reliable Messaging, WS-Addressing, WS-Coordination, and WS-Transaction.

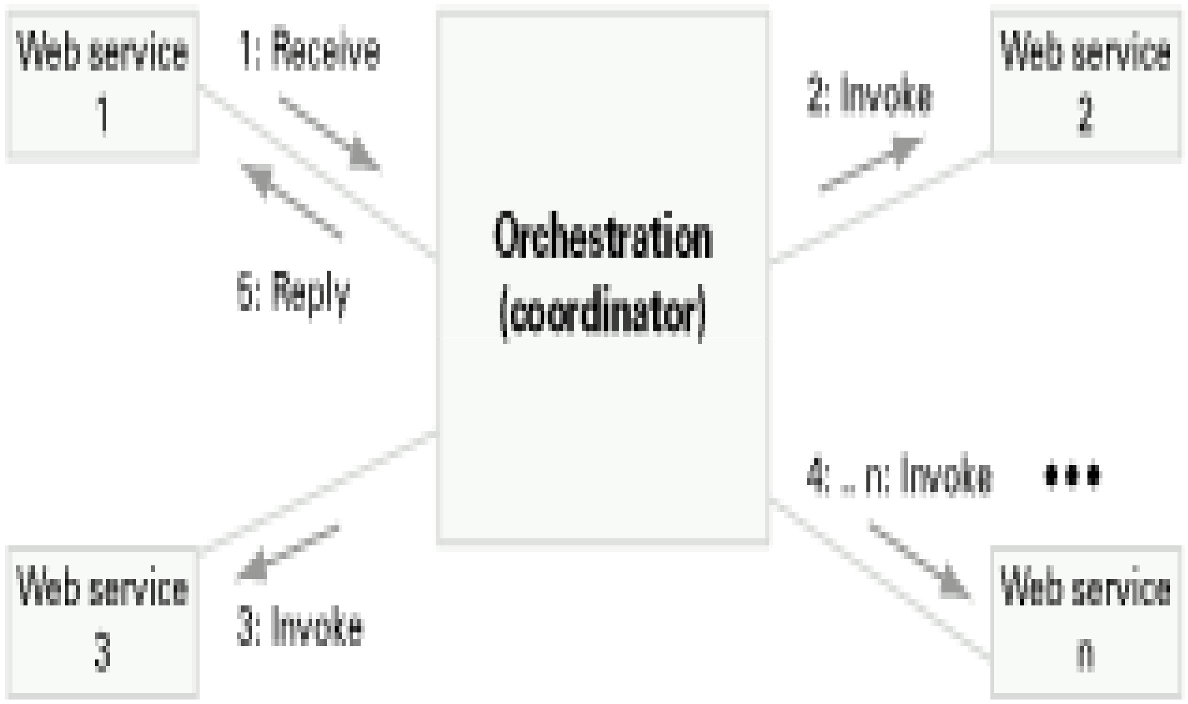
Services Integration

- Web services can be combined in two ways:
  - Orchestration
  - Choreography

Orchestration

- Used in private business processes;
- A central process takes control of the involved Web services and coordinates the execution of different operations on the Web services involved in the operation.
- The involved Web services do not "know" (and do not need to know) that they are involved in a composition process and that they are taking part in a higher-level business process.
- Only the central coordinator of the orchestration is aware of this goal, so the orchestration is centralized with explicit definitions of operations and the order of invocation of Web services.

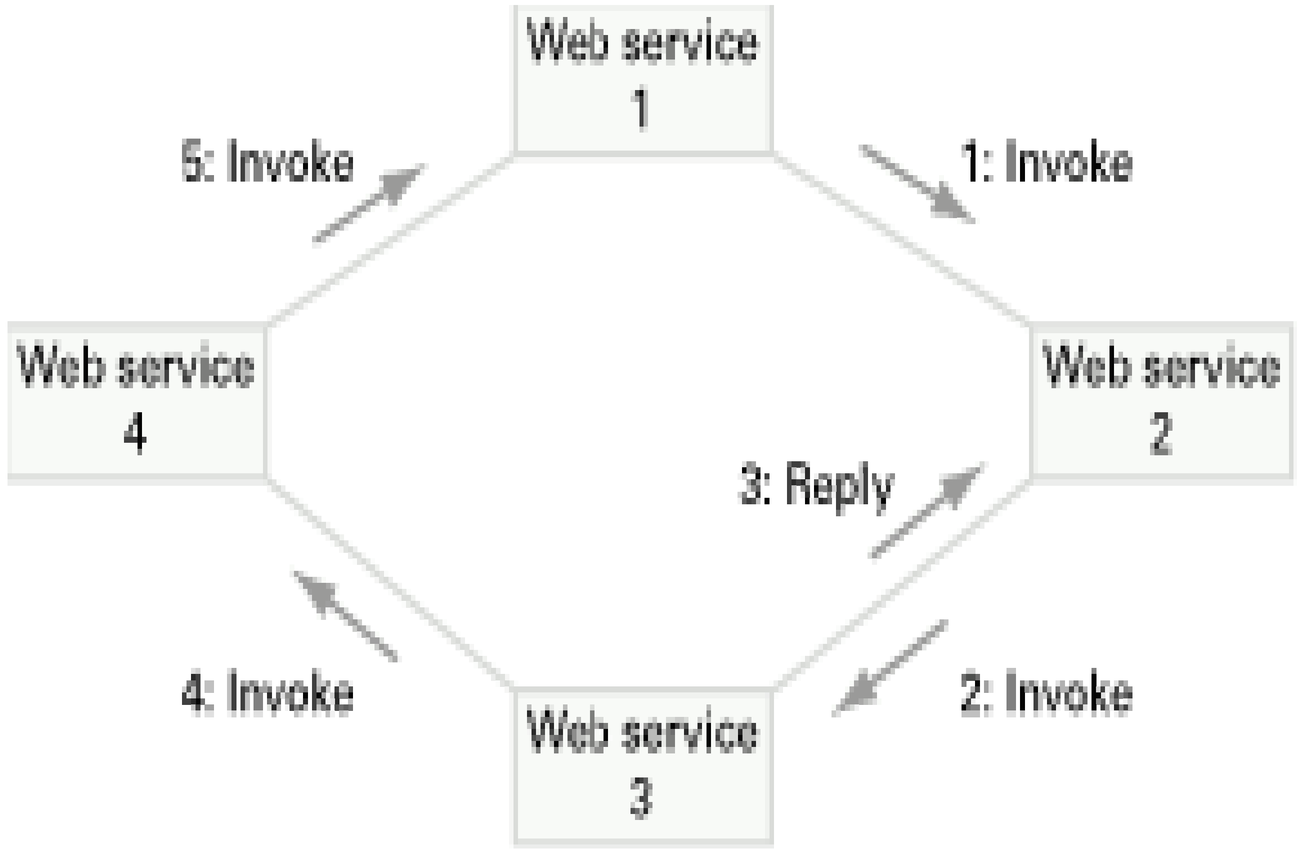
Orchestration (contd.)



Choreography

- Choreography does not rely on a central coordinator.
- Each Web service involved in the choreography knows exactly when to execute its operations and with whom to interact.
- Choreography is a collaborative effort focusing on the exchange of messages in public business processes.
- All participants in the choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges.

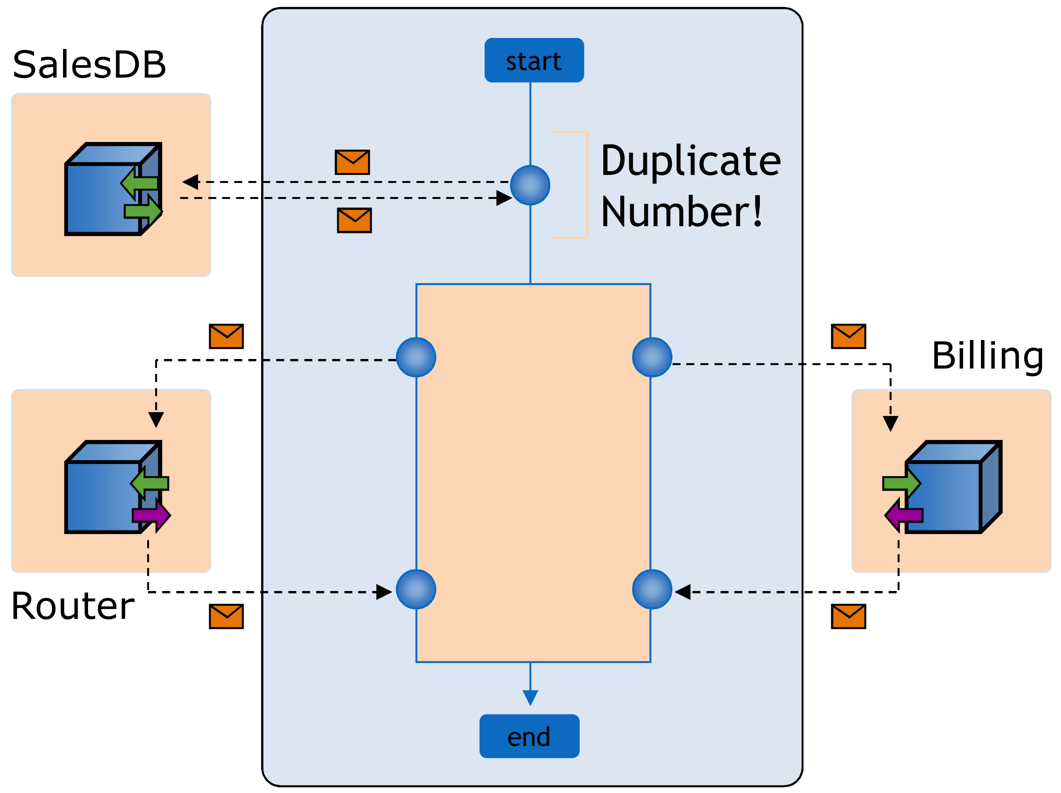
Choreography (contd.)



Choreography vs. Orchestration

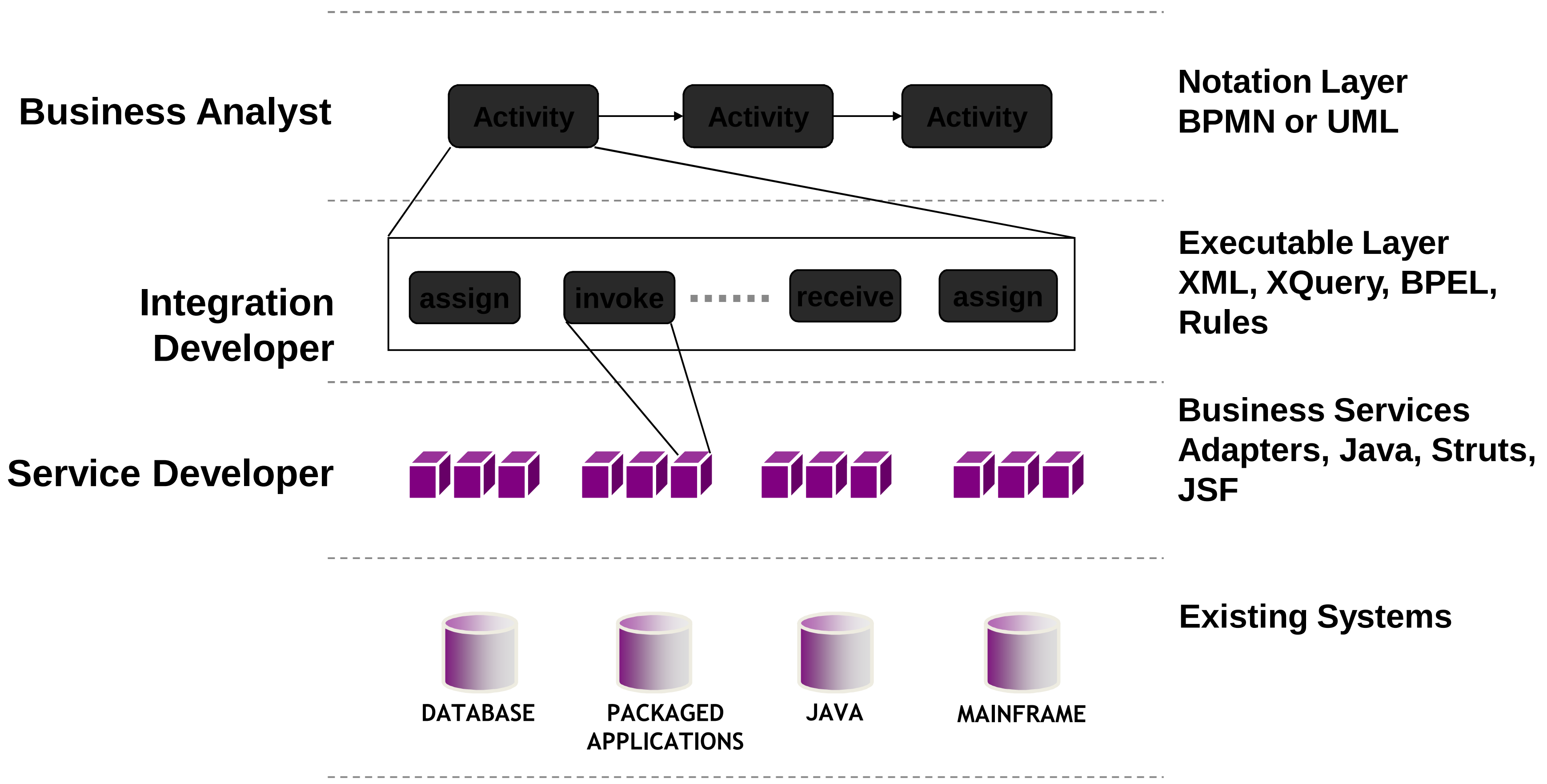
- From the perspective of composing Web services to execute business processes, orchestration is a more flexible paradigm and has the following advantages over choreography:
  - The coordination of component processes is centrally managed by a known coordinator.
  - Web services can be incorporated without their being aware that they are taking part in a larger business process.
  - Alternative scenarios can be put in place in case faults occur.

What is BPEL?



Markup language for composing a set of discrete services into an end-to-end process flow

The Top Down Perspective



Why use BPEL?

- Used to standardize enterprise application integration as well as to extend the integration to previously isolated systems
- Between enterprises, BPEL enables easier and more effective integration with business partners
- Definitions of business processes described in BPEL do not affect existing systems
- Is the key technology in environments where functionalities are already or will be exposed via Web services.

Orchestration versus Choreography

- BPEL supports two different ways of describing business processes that support orchestration and choreography:
  - **Executable processes** allow you to specify the exact details of business processes. They follow the orchestration paradigm and can be executed by an orchestration engine.
  - **Abstract business protocols** allow specification of the public message exchange between parties only. They do not include the internal details of process flows and are not executable. They follow the choreography paradigm.

Building a Business Process

- Specifies the exact order in which participating Web services should be invoked
  - sequentially or in parallel.
- Express conditional behaviors.
  - an invocation of a Web service can depend on the value of a previous invocation.
- Construct loops, declare variables, copy and assign values, define fault handlers, etc.

Building a BP (2)

- By combining all these constructs, you can define complex business processes in an algorithmic manner

- Because business processes are essentially graphs of activities, it might be useful to express them using Unified Modeling Language (UML) activity diagrams.

BPEL Steps

- consists of steps; each step is called an "activity."
- supports primitive as well as structure activities.

Primitive Activities

- *Primitive* activities represent basic constructs and are used for common tasks, such as the following:
  - Invoking other Web services, using <invoke>
  - Waiting for the client to invoke the business process by sending a message, using <receive> (receiving a request)
  - Generating a response for synchronous operations, using <reply>
  - Manipulating data variables, using <assign>
  - Indicating faults and exceptions, using <throw>
  - Waiting for some time, using <wait>
  - Terminating the entire process, using <terminate>

Combining Primitives

- Combine these and other primitive activities to define complex algorithms that specify exactly the steps of business processes

- To combine primitive activities, BPEL supports several *structure* activities. The most important are:
  - Sequence (<sequence>), which allows us definition of a set of activities that will be invoked in an ordered sequence
  - Flow (<flow>) for defining a set of activities that will be invoked in parallel

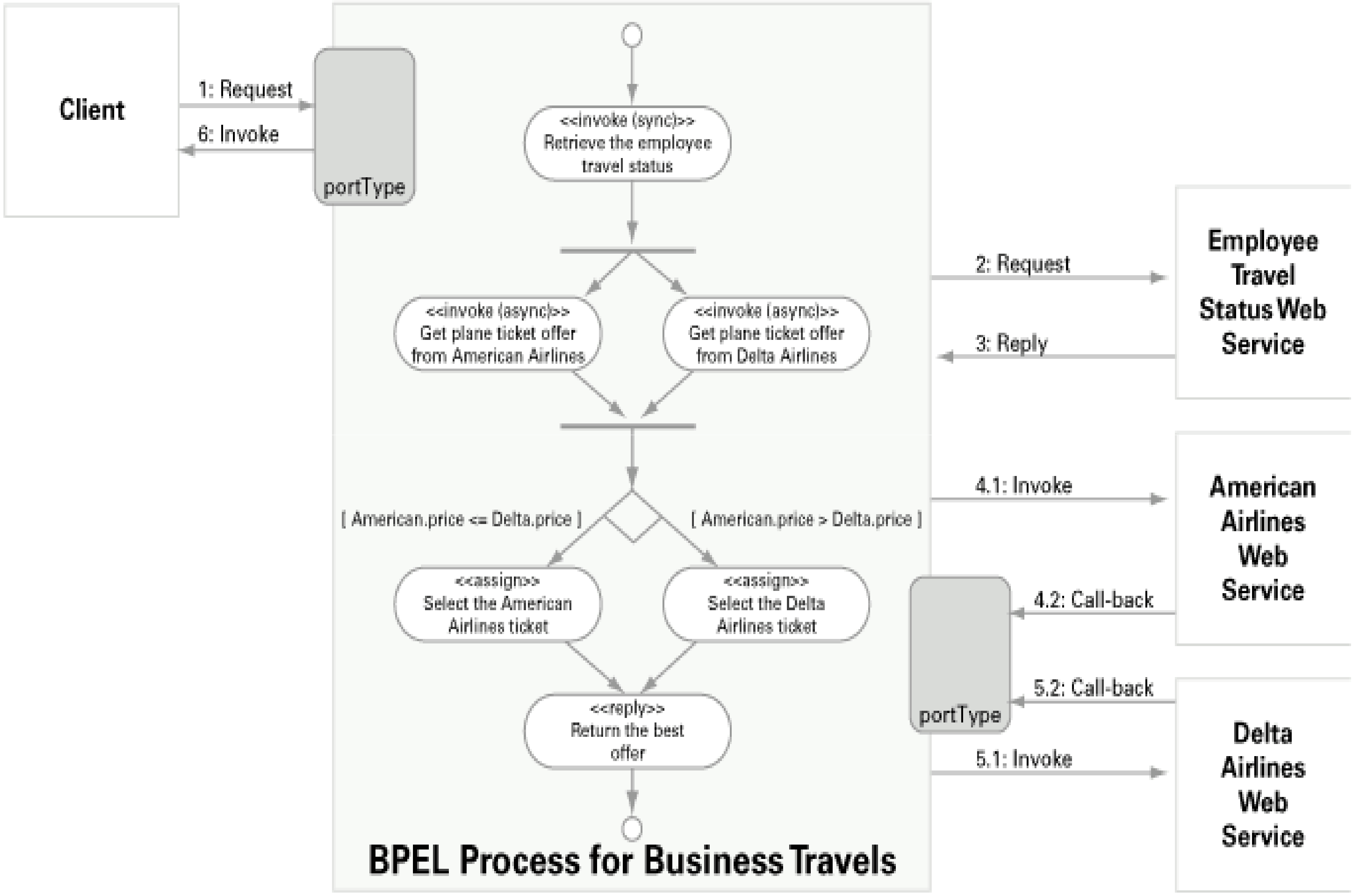
Structure Activities

- Case-switch construct (<switch>) for implementing branches
- While (<while>) for defining loops
- The ability to select one of several alternative paths, using <pick>
- Each BPEL process will also define partner links, using <partnerLink>, and declare variables, using <variable>.

Defining a process

- You define a new Web service that is a composite of existing services.
- The interface of the new BPEL composite Web service uses a set of port types through which it provides operations like any other Web service.
- To invoke a business process described in BPEL, you have to invoke the resulting composite Web service.

Schematic view


Client
1: Request
6: Invoke
portType
<<invoke (sync)>> Retrieve the employee travel status
<<invoke (async)>> Get plane ticket offer from American Airlines
<<invoke (async)>> Get plane ticket offer from Delta Airlines
[American.price <= Delta.price]
[American.price > Delta.price]
<<assign>> Select the American Airlines ticket
<<assign>> Select the Delta Airlines ticket
<<reply>> Return the best offer
BPEL Process for Business Travels
2: Request
3: Reply
Employee Travel Status Web Service
4.1: Invoke
4.2: Call-back
American Airlines Web Service
portType
5.2: Call-back
5.1: Invoke
Delta Airlines Web Service

Causes of Faults

- Faults in BPEL can be from various sources:
  - A BPEL process can explicitly signal (throw) a fault.
  - A fault can occur when the BPEL process invokes a Web service operation. The operation might return a WSDL fault message, which results in a BPEL fault.
  - A fault can be thrown automatically by the BPEL runtime environment, either due to a certain condition in the BPEL process itself (such as a join failure), as a consequence of error conditions in the runtime environment, or related to network communication or other reasons. For such situations, BPEL defines several standard faults.

Signaling

- Signaling Faults
  - A business process sometimes needs to signal a fault explicitly. Therefore, BPEL provides the <throw> activity, which has the following syntax:
  - <throw faultName="*name*" />

Handling Faults

- When a fault occurs within a business process, the process may not complete successfully.
- It can complete successfully if the fault is handled within a *scope*, which enables you to divide a complex process into several parts;

Handling Faults

- The business process can handle the fault through one or more fault handlers.
- Within a fault handler, the business process defines custom activities that should recover from the fault and possibly reverse the partial (unsuccessful) work of the activity where the fault has occurred.

Scopes

- Scopes are hierarchically organized parts into which a complex business process can be divided.

- They provide behavioral contexts for activities.
  - scopes enable you to define different fault handlers for different activities (or sets of activities gathered under a common structured activity such as <sequence> or <flow>).

Scopes (2)

- In addition to defining fault handlers, you can declare variables that are visible only within a scope.
- Scopes also let you define local correlation sets, compensation handlers, and event handlers.

Events

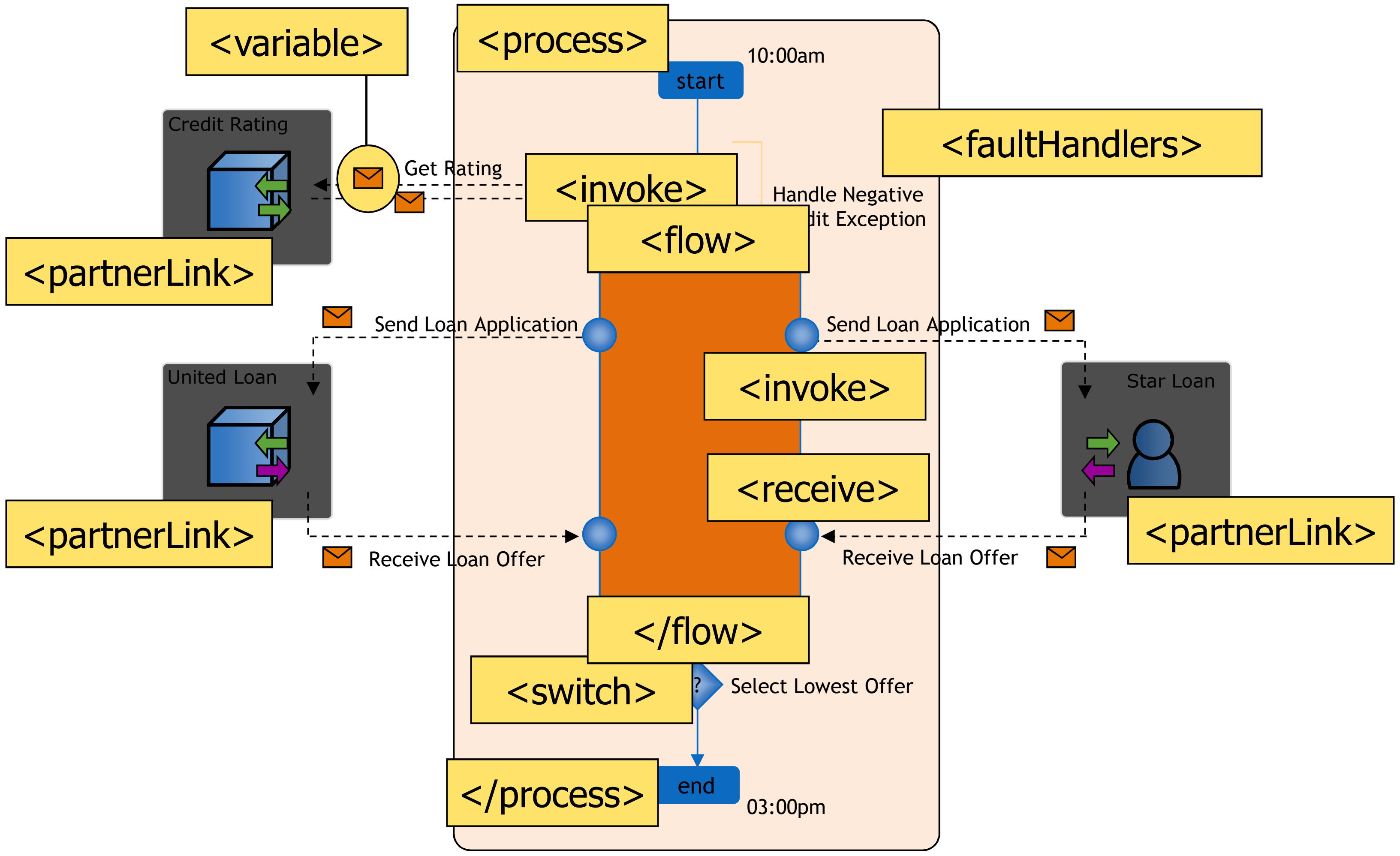
- BPEL supports two types of events:
  - **Message events** are triggered by incoming messages through operation invocation on port types.
  - **Alarm events** are time-related and are triggered either after a certain duration or at a specific time.
- Managing message events is particularly important when the business process is waiting for callbacks from partner Web services.

Events (2)

- Often, however, it is more useful to wait for more than one message, of which only one will occur.

- Alarm events are useful when you want the process to wait for a callback for a certain period of time, such as 15 minutes.
  - If no callback is received, the process flow continues as designed.
  - Useful in loosely coupled service-oriented architectures, where you cannot rely on Web services being available all the time.

BPEL by Example



A BPEL Process

```
<process name="purchaseOrderProcess"
      targetNamespace="..."
      xmlns="..."
      xmlns:lns="...">
...
  <sequence>
    <receive partnerLink="purchasing"
          portType="lns:purchaseOrderPT"
          operation="sendPurchaseOrder"
          variable="PO">
    </receive>
    <flow>
      <links>
        <link name="ship-to-invoice"/>
        <link name="ship-to-scheduling"/>
      </links>
      <sequence>
        <assign>
          <copy>
            <from variable="PO" part="customerInfo"/>
            <to variable="shippingRequest"
               part="customerInfo"/>
          </copy>
        </assign>
        <invoke  partnerLink="shipping"
              portType="lns:shippingPT"
              operation="requestShipping"
              inputVariable="shippingRequest"
              outputVariable="shippingInfo">
          <source linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="shipping"
              portType="lns:shippingCallbackPT"
              operation="sendSchedule"
              variable="shippingSchedule">
          <source linkName="ship-to-scheduling"/>
        </receive>
      </sequence>
      <sequence>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="initiatePriceCalculation"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="sendShippingPrice"
              inputVariable="shippingInfo">
          <target linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="invoicing"
              portType="lns:invoiceCallbackPT"
              operation="sendInvoice"
              variable="Invoice"/>
      </sequence>
      <sequence>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="requestProductionScheduling"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="sendShippingSchedule"
              inputVariable="shippingSchedule">
          <target linkName="ship-to-scheduling"/>
        </invoke>
      </sequence>
    </flow>
    <reply partnerLink="purchasing"
         portType="lns:purchaseOrderPT"
         operation="sendPurchaseOrder"
         variable="Invoice"/>
  </sequence>
</process>
```

Structured Activities

```
<process name="purchaseOrderProcess"
      targetNamespace="..."
      xmlns="..."
      xmlns:lns="...">
...
  <sequence>
    <receive partnerLink="purchasing"
             portType="lns:purchaseOrderPT"
             operation="sendPurchaseOrder"
             variable="PO">
    </receive>
    <flow>
      <links>
        <link name="ship-to-invoice"/>
        <link name="ship-to-scheduling"/>
      </links>
      <sequence>
        <assign>
          <copy>
            <from variable="PO" part="customerInfo"/>
            <to variable="shippingRequest"
               part="customerInfo"/>
          </copy>
        </assign>
        <invoke  partnerLink="shipping"
                 portType="lns:shippingPT"
                 operation="requestShipping"
                 inputVariable="shippingRequest"
                 outputVariable="shippingInfo">
          <source linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="shipping"
                 portType="lns:shippingCallbackPT"
                 operation="sendSchedule"
                 variable="shippingSchedule">
          <source linkName="ship-to-scheduling"/>
        </receive>
      </sequence>
      <sequence>
        <invoke  partnerLink="invoicing"
                 portType="lns:computePricePT"
                 operation="initiatePriceCalculation"
                 inputVariable="PO">
        </invoke>
        <invoke  partnerLink="invoicing"
                 portType="lns:computePricePT"
                 operation="sendShippingPrice"
                 inputVariable="shippingInfo">
          <target linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="invoicing"
                 portType="lns:invoiceCallbackPT"
                 operation="sendInvoice"
                 variable="Invoice"/>
      </sequence>
      <sequence>
        <invoke  partnerLink="scheduling"
                 portType="lns:schedulingPT"
                 operation="requestProductionScheduling"
                 inputVariable="PO">
        </invoke>
        <invoke  partnerLink="scheduling"
                 portType="lns:schedulingPT"
                 operation="sendShippingSchedule"
                 inputVariable="shippingSchedule">
          <target linkName="ship-to-scheduling"/>
        </invoke>
      </sequence>
    </flow>
    <reply partnerLink="purchasing"
           portType="lns:purchaseOrderPT"
           operation="sendPurchaseOrder"
           variable="Invoice"/>
  </sequence>
</process>
```

Primitive Activities

```
<process name="purchaseOrderProcess"
        targetNamespace="..."
        xmlns="..."
        xmlns:lns="...">
...
  <sequence>
    <receive partnerLink="purchasing"
            portType="lns:purchaseOrderPT"
            operation="sendPurchaseOrder"
            variable="PO">
    </receive>
    <flow>
      <links>
        <link name="ship-to-invoice"/>
        <link name="ship-to-scheduling"/>
      </links>
      <sequence>
        <assign>
          <copy>
            <from variable="PO" part="customerInfo"/>
            <to variable="shippingRequest"
                part="customerInfo"/>
          </copy>
        </assign>
        <invoke  partnerLink="shipping"
              portType="lns:shippingPT"
              operation="requestShipping"
              inputVariable="shippingRequest"
              outputVariable="shippingInfo">
          <source linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="shipping"
              portType="lns:shippingCallbackPT"
              operation="sendSchedule"
              variable="shippingSchedule">
          <source linkName="ship-to-scheduling"/>
        </receive>
      </sequence>
      <sequence>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="initiatePriceCalculation"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="sendShippingPrice"
              inputVariable="shippingInfo">
          <target linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="invoicing"
              portType="lns:invoiceCallbackPT"
              operation="sendInvoice"
              variable="Invoice"/>
      </sequence>
      <sequence>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="requestProductionScheduling"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="sendShippingSchedule"
              inputVariable="shippingSchedule">
          <target linkName="ship-to-scheduling"/>
        </invoke>
      </sequence>
    </flow>
    <reply partnerLink="purchasing"
          portType="lns:purchaseOrderPT"
          operation="sendPurchaseOrder"
          variable="Invoice"/>
  </sequence>
</process>
```

Data Flow

```
<process name="purchaseOrderProcess"
      targetNamespace="..."
      xmlns="..."
      xmlns:lns="...">
...
  <sequence>
    <receive partnerLink="purchasing"
          portType="lns:purchaseOrderPT"
          operation="sendPurchaseOrder"
          variable="PO">
    </receive>
    <flow>
      <links>
        <link name="ship-to-invoice"/>
        <link name="ship-to-scheduling"/>
      </links>
      <sequence>
        <assign>
          <copy>
            <from variable="PO" part="customerInfo"/>
            <to variable="shippingRequest"
               part="customerInfo"/>
          </copy>
        </assign>
        <invoke  partnerLink="shipping"
              portType="lns:shippingPT"
              operation="requestShipping"
              inputVariable="shippingRequest"
              outputVariable="shippingInfo">
          <source linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="shipping"
              portType="lns:shippingCallbackPT"
              operation="sendSchedule"
              variable="shippingSchedule">
          <source linkName="ship-to-scheduling"/>
        </receive>
      </sequence>
      <sequence>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="initiatePriceCalculation"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="sendShippingPrice"
              inputVariable="shippingInfo">
          <target linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="invoicing"
              portType="lns:invoiceCallbackPT"
              operation="sendInvoice"
              variable="Invoice"/>
      </sequence>
      <sequence>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="requestProductionScheduling"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="sendShippingSchedule"
              inputVariable="shippingSchedule">
          <target linkName="ship-to-scheduling"/>
        </invoke>
      </sequence>
    </flow>
    <reply partnerLink="purchasing"
          portType="lns:purchaseOrderPT"
          operation="sendPurchaseOrder"
          variable="Invoice"/>
  </sequence>
</process>
```

Partner Links

```
<process name="purchaseOrderProcess"
        targetNamespace="..."
        xmlns="..."
        xmlns:lns="...">
...
  <sequence>
    <receive partnerLink="purchasing"
          portType="lns:purchaseOrderPT"
          operation="sendPurchaseOrder"
          variable="PO">
    </receive>
    <flow>
      <links>
        <link name="ship-to-invoice"/>
        <link name="ship-to-scheduling"/>
      </links>
      <sequence>
        <assign>
          <copy>
            <from variable="PO" part="customerInfo"/>
            <to variable="shippingRequest"
               part="customerInfo"/>
          </copy>
        </assign>
        <invoke  partnerLink="shipping"
              portType="lns:shippingPT"
              operation="requestShipping"
              inputVariable="shippingRequest"
              outputVariable="shippingInfo">
          <source linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="shipping"
              portType="lns:shippingCallbackPT"
              operation="sendSchedule"
              variable="shippingSchedule">
          <source linkName="ship-to-scheduling"/>
        </receive>
      </sequence>
      <sequence>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="initiatePriceCalculation"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="invoicing"
              portType="lns:computePricePT"
              operation="sendShippingPrice"
              inputVariable="shippingInfo">
          <target linkName="ship-to-invoice"/>
        </invoke>
        <receive partnerLink="invoicing"
              portType="lns:invoiceCallbackPT"
              operation="sendInvoice"
              variable="Invoice"/>
      </sequence>
      <sequence>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="requestProductionScheduling"
              inputVariable="PO">
        </invoke>
        <invoke  partnerLink="scheduling"
              portType="lns:schedulingPT"
              operation="sendShippingSchedule"
              inputVariable="shippingSchedule">
          <target linkName="ship-to-scheduling"/>
        </invoke>
      </sequence>
    </flow>
    <reply partnerLink="purchasing"
         portType="lns:purchaseOrderPT"
         operation="sendPurchaseOrder"
         variable="Invoice"/>
  </sequence>
</process>
```