Business Process Digitalization and Cloud Computing

11. Service Composition with BPEL in Detail

Andrea Morichetta, Phd

December 22, 2016

Computer Science Division

slides are based on the WS-BPEL 2.0 for SOA Composite Applications with Oracle SOA Suite

Table of contents

Introduction

Developing Business Process

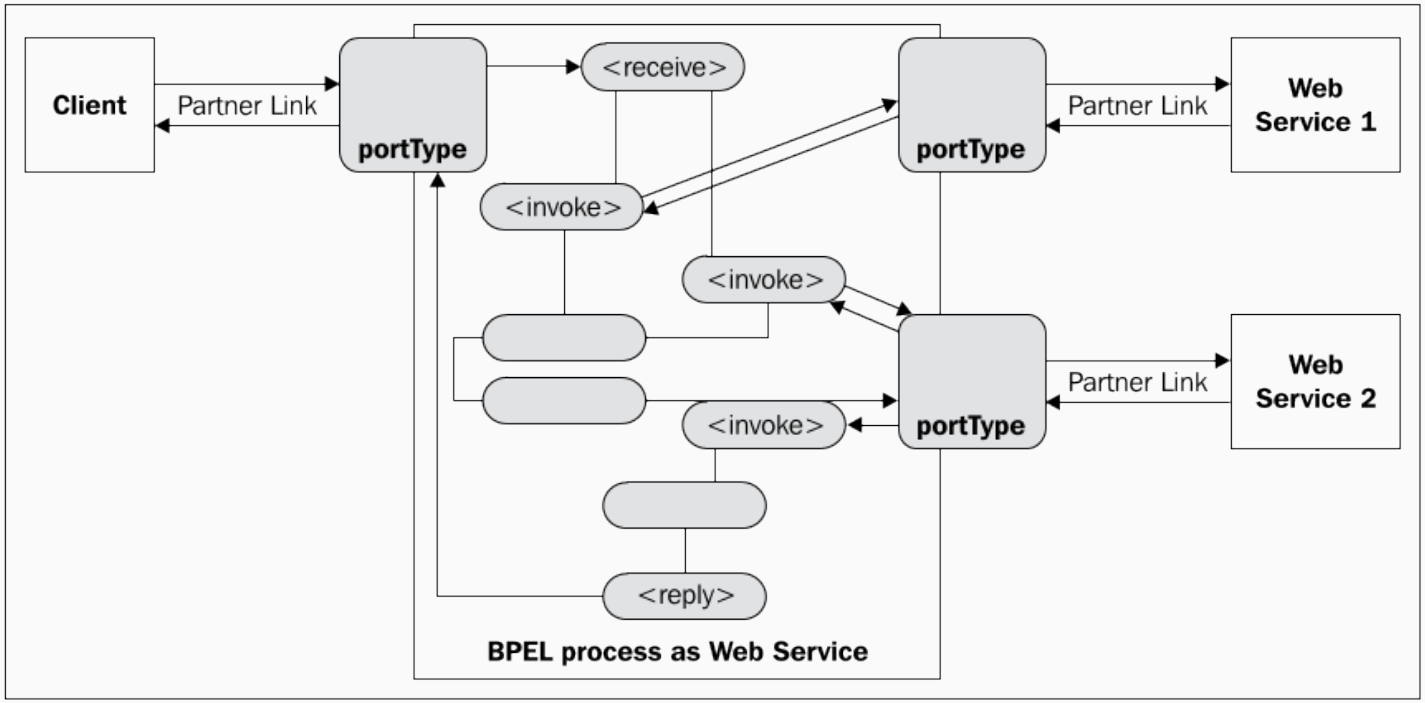
- Business processes are described using **XML-vocabulary**
- Most known **development environment** are:
  - Oracle JDeveloper
  - IBM WebSphere
  - Eclipse
- Most of them permit to switch in **source view**, in order to enter code directly

Business Process in BPEI

- **Executable BP**: describe the exact details of a **executable** BP
- **Abstract BP**: describe **templates** or **public messages** exchanged between parties. They are not executable.
- **BPEL Process**: is an executable BP and provides operations like any other service.

Partner Link

BPEL introduce WSDL extension, which enable us to specify **relation between several services** in the BP called **partner links**.



- The BPEL specify the **exact order** in which participants should be invoked.

Core Concepts

Activities

BPEL is a collection of steps, called **activity**.

- **Basic** activities:
  - `invoke`
  - `receive`
  - `reply`
  - `assign`
  - `throw`
  - `wait`
  - `exit`
- **Structured** activities:
  - `sequence`
  - `flow`
  - `if`
  - `while`, `repeatUntil`, `forEach`
  - `pick`

Partner Links Example

```
<?xml version="1.0" encoding="utf-8"?>
<process name="InsuranceSelectionProcess"
   targetNamespace="http://packtpub.com/bpel/example/"
   xmlns=
        "http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:ins="http://packtpub.com/bpel/insurance/"
   xmlns:com="http://packtpub.com/bpel/company/" >
<partnerLinks>
   <partnerLink name="client"
      partnerLinkType="com:selectionLT"
          myRole="insuranceSelectionService"/>
   <partnerLink name="insuranceA"
          partnerLinkType="ins:insuranceLT"
          myRole="insuranceRequester"
          partnerRole="insuranceService"/>
   <partnerLink name="insuranceB"
          partnerLinkType="ins:insuranceLT"
          myRole="insuranceRequester"
          partnerRole="insuranceService"/>
</partnerLinks>
```

Variables Example

```
<variables>
       <!-- input for BPEL process -->
       <variable name="InsuranceRequest"
      messageType="ins:InsuranceRequestMessage"/>
       <!-- output from insurance A -->
       <variable name="InsuranceAResponse"
          messageType="ins:InsuranceResponseMessage"/>
       <!-- output from insurance B -->
       <variable name="InsuranceBResponse"
          messageType="ins:InsuranceResponseMessage"/>
       <!-- output from BPEL process -->
       <variable name="InsuranceSelectionResponse"
          messageType="ins:InsuranceResponseMessage"/>
</variables>
```

Process Example

```
<sequence>
   <!-- Receive the initial request from client -->
   <receive partnerLink="client"
      portType="com:InsuranceSelectionPT"
          operation="SelectInsurance"
          variable="InsuranceRequest"
          createInstance="yes" />
        <!-- Make concurrent invocations to Insurance A and B -->
        <flow>
          <!-- Invoke Insurance A service -->
          <invoke partnerLink="insuranceA"
             portType="ins:ComputeInsurancePremiumPT"
             operation="ComputeInsurancePremium"
             inputVariable="InsuranceRequest"
             outputVariable="InsuranceAResponse" />
            <!-- Invoke Insurance B service -->
          <invoke partnerLink="insuranceB"
             portType="ins:ComputeInsurancePremiumPT"
             operation="ComputeInsurancePremium"
             inputVariable="InsuranceRequest"
             outputVariable="InsuranceBResponse" />
        </flow>
```

Process Example

```
    <if>
          <condition>
             $InsuranceAResponse.confirmationData/ins:Amount &lt;=
             $InsuranceBResponse.confirmationData/ins:Amount
          </condition>
          <!-- Select Insurance A -->
          <assign>
            <copy>
              <from variable="InsuranceAResponse" />
          <to variable="InsuranceSelectionResponse" />
            </copy>
          </assign>
        <else>
          <!-- Select Insurance B -->
          <assign>
            <copy>
              <from variable="InsuranceBResponse" />
              <to variable="InsuranceSelectionResponse" />
            </copy>
          </assign>
        </else>
    </if>
    <!-- Send a response to the client -->
    <reply partnerLink="client"
      portType="com:InsuranceSelectionPT"
          operation="SelectInsurance"
          variable="InsuranceSelectionResponse"/>
</sequence>
</process>
```

Invoking service

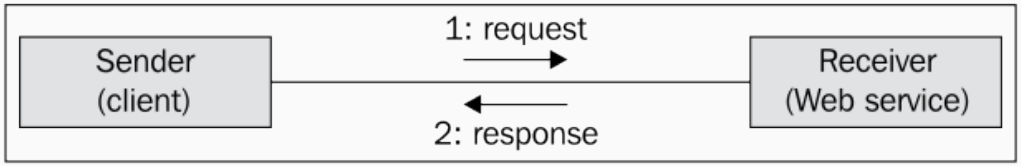
```
<process ...>
   <sequence>

   <!-- Wait for the incoming request to start the process -->
   <receive ... />

   <!-- Invoke a set of related services, one by one -->
   <invoke .../>
   <invoke .../>
   <invoke .../>
   ...
   </sequence>
</process>
```

Invoking service concurrently

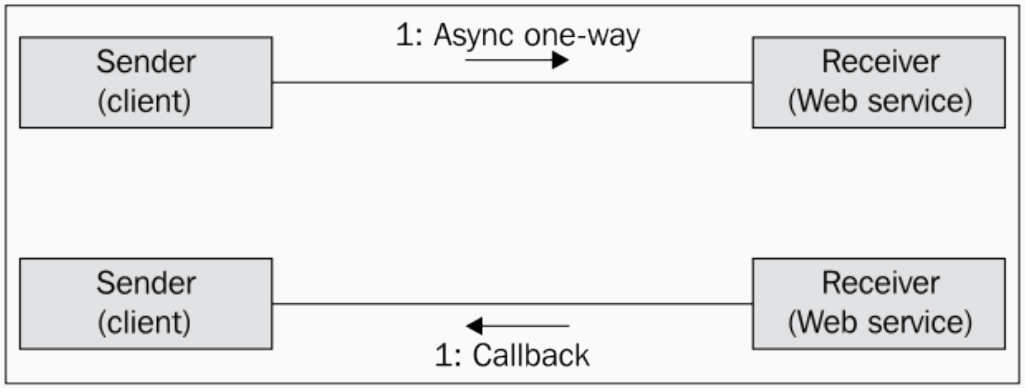
```
<process ...>
  <sequence>

  <!-- Wait for the incoming request to start the process -->
  <receive ... />

  <!-- Invoke two sequences concurrently -->
    <flow>

      <!-- The three invokes below execute sequentially -->
      <sequence>
        <invoke ... />
        <invoke ... />
        <invoke ... />
      </sequence>

      <!-- The two invokes below execute sequentially -->
      <sequence>
        <invoke ... />
        <invoke ... />
      </sequence>

    </flow>
  </sequence>
</process>
```

Invoking Service Operations

- **Synchronous request/reply operations**: send a request and **wait for the reply** (operation not require much time).



- **Asynchronous operations**: The **operation don't block the sender** for the duration of the operation. Information are sent back using the callback. Callback should be related to original request (correlation sets).



Synchronous/Asynchronous constructs

- **Synchronouse** `invoke`: the system wait for the reply without the need of an explicit construct
- **Asynchronous** `invoke`: take care only of the first part. To receive the result we need a separate construct `receive`. With receive the process waits for the incoming message.

```
<process ...>
  <sequence>

    <!-- Wait for the incoming request to start the process -->
    <receive ... />

    <!-- Invoke an asynchronous operation -->
    <invoke ... />

    <!-- Do something else... -->

    <!-- Wait for the callback -->
    <receive ... />
  </sequence>
</process>
```

Synchronous/Asynchronous business process

- Synchronous: reply

```
<process ...>
  <sequence>
    <!-- Wait for the incoming request to start the process -->
    <receive ... />
    <!-- Invoke a set of related services -->
       ...
    <!-- Return a synchronous reply to the caller (client) -->
    <reply ... />
  </sequence>
</process>
```

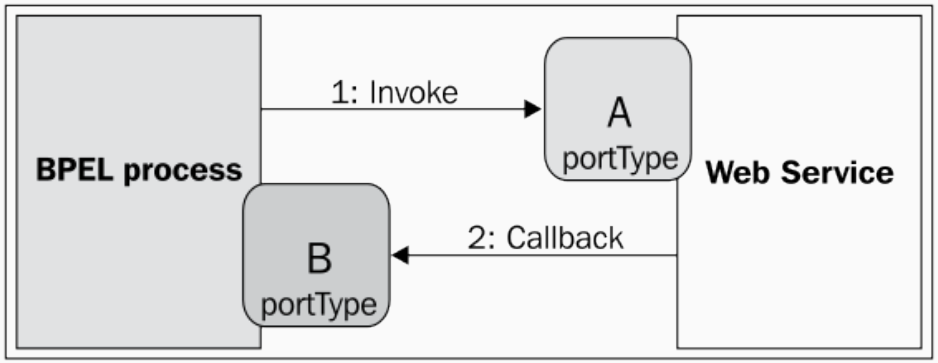
- Asynchronous: invoke

```
<process ...>
  <sequence>
       <!-- Wait for the incoming request to start the process -->
    <receive ... />
    <!-- Invoke a set of related services -->
    <!-- Invoke a callback on the client (if needed) -->
    <invoke ... />
  </sequence>
</process>
```

Links between partners

BPEL process **interact with external services** in two ways:

- Invokes operations on others services
- Receive invocations from clients (callback for replies)
- Both invoke and receive invocations



Link to all partners are called partner links.

NOTE: each BPEL process has at least one client partner link.

Partner Links Type

BPEL use partner link for:

- **Support asynchronous interactions** and permits to invoke the callback on the initial caller.
- BPEL process can offer service using the port types, and so partner link is useful for **distinguish between different clients** and offer them different functionalities.

Describing the interactions between two service require to **define the perspective**, using the partner link we have the possibility to model the **relationships as a third party**.

Partner link type must have:

- at least one role
- at most two roles

For each role we must specify a **portType** used for the

Partner Link Type Example

- Asynchronous (we require that the service support the callback)

```
<partnerLinkType name="insuranceLT"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
  <role name="insuranceService"
    portType="ins:ComputeInsurancePremiumPT"/>
  <role name="insuranceRequester"
    portType="com:ComputeInsurancePremiumCallbackPT"/>
</partnerLinkType>
```

- Synchronous

```
<partnerLinkType name="insuranceLT"
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
   <role name="insuranceService"
     portType="ins:ComputeInsurancePremiumPT"/>
</partnerLinkType>
```

Partner link type definition

- Partner link **are not part of the BPEL** (but WSDL) because is part of the service specification.
- Partner link type use the **WSDL extensibility mechanism**

WSDL and Partner Link Types

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ins="http://packtpub.com/bpel/insurance/"
  xmlns:com="http://packtpub.com/bpel/company/"
  targetNamespace="http://packtpub.com/bpel/company/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype" >
  <import ... />
  <types>
    <xs:schema ... >
      ...
    </xs:schema>
  </types>
  <message ... >
    <part ... />
    ...
  </message>
```

```
  <portType name="ComputeInsurancePremiumPT">
    <operation name="...">
      <input message="..." />
    </operation>
  </portType>
  <portType name="ComputeInsurancePremiumCallbackPT">
    <operation name="...">
      <input message="..." />
    </operation>
  </portType>
  ...
  <plnk:partnerLinkType name="insuranceLT">
    <plnk:role name="insuranceService"
       portType ="ins:ComputeInsurancePremiumPT"/>
    <plnk:role name="insuranceRequester"
       portType ="ins:ComputeInsurancePremiumCallbackPT"/>
    </plnk:partnerLinkType>
  </definitions>
```

Defining partner links

Partner links are **concrete references** to services that a BPEL business process interact with.

```
<process ...>
  <partnerLinks>
    <partnerLink ... />
    <partnerLink ... />
    ...
  </partnerLinks>
  <sequence>
  ...
  </sequence>
</process>
```

For each partner link we have to specify:

- name: is a reference for interactions via that partner link
- partnerLinkType: defines the type of the partner link
- myRole: Indicates the role of the BPEL process itself
- partnerRole: indicates the role of the partner
- initializePartnerRole: indicates whether the BPEL engine should initialize the partner link's partner role value. Should only be used with partner links that specify partner role.

Partner link example

```
<partnerLinks>
  <partnerLink name="insurance"
     partnerLinkType="tns:insuranceLT"
     myRole="insuranceRequester"
     partnerRole="insuranceService"/>
</partnerLinks>
```

- If the partnerLinkType define only one role also the partner link specify only one role.

BPEL process tag

```
<process name="InsuranceSelectionProcess"
  targetNamespace="http://packtpub.com/bpel/example/"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:ins="http://packtpub.com/bpel/insurance/"
  xmlns:com="http://packtpub.com/bpel/company/" >
```

- Name : Specifies the name of the BPEL business process
- TargetNamespace : Specifies the target namespace for the business process definition
- xmlns : The namespace used by BPEL is http://docs.oasis-open.org/wsbpel/2.0/process/executable

Attributes for the process tag

- QueryLanguage : Specifies which query language is **used for node selection** in assignments, properties, and other uses (Xpath 2.0 or XQuery).
- ExpressionLanguage : Specifies which **expression language** is used in the process (XPath 1.0).
- SuppressJoinFailure : Determines **whether to suppress join failures** (yes or no). The default is no .
- ExitOnStandardFault : Defines how the **process should behave** when a standard fault occurs. We can specify yes if we want the process to exit on a standard fault, or no if we want to handle the fault using a fault handler. The default is no.

Variables

The results of an invoke operation could be useful for a **subsequent invocations**. BPEL provide variables for maintain the **state**.

Variables can store:

- WSDL Messages
- XML schema elements
- XML schema simple types

We have to specify:

- **messageType**: A variable that can hold a WSDL message
- **element**: A variable that can hold an XML schema element
- **type**: A variable that can hold an XML schema simple type

Variables examples

Variables Example:

```
<variables>
  <variable name="InsuranceRequest"
     messageType="ins:InsuranceRequestMessage"/>
  <variable name="PartialInsuranceDescription"
     element="ins:InsuranceDescription"/>
  <variable name="LastName" type="xs:string"/>
</variables>
```

Global variables declaration:

```
<process ...>
   <partnerLinks>
   ...
   </partnerLinks>
   <variables>
       <variable ... />
       <variable ... />
       ...
   </variables>
   <sequence>
   ...
   </sequence>
</process>
```

Providing the interface to BPEL processes

The three activities `reply`, `invoke`, `receive` have the same basic attributes:

- PartnerLink: Specifies which partner link will be used
- PortType: Specifies the used port type
- Operation: Specifies the name of the operation to invoke (`invoke`), to wait to be invoked (`receive`), or the name of the operation which has been invoked but is synchronous and requires a reply (`reply`)

Invoke

- The send messages in the `invoke` operation are modeled as input messages (**inputVariable**).
- The results of a request/response operation is modeled as output message stored in a (**outputVariable**)

A synchronous `Invoke` example:

```
<invoke partnerLink="insuranceA"
   portType="ins:ComputeInsurancePremiumPT"
   operation="ComputeInsurancePremium"
   inputVariable="InsuranceRequest"
   outputVariable="InsuranceAResponse" >
</invoke>
```

Receive

`Receive` operation waits for the incoming message and use the following attributes:

- The receive waits for the incoming message and it can use the **variable attribute**.
- **createInstance** which is related to the business process lifecycle. Create a new instance of the process (`createInstance="YES"`).
- **MessageExchange** which is used to disambiguate the relationship between messages.

```
<receive partnerLink="client"
   portType="com:InsuranceSelectionPT"
   operation="SelectInsurance"
   variable="InsuranceRequest"
   createInstance="yes" >
</receive>
```

Reply

`Reply` return the response for synchronous BPEL Process.

- It is used for return the **answer** or a **fault message**.
- the **variable attribute** is where the response is stored.

```
<reply partnerLink="client"
   portType="com:InsuranceSelectionPT"
   operation="SelectInsurance"
   variable="InsuranceSelectionResponse" >
</reply>
```

Assignement

The main scopes for variables are:

- Hold and maintain the data
- Specify input and output messages for invoking operations on partner service.

How to **copy data** between variables?

- `<Assign>`: The assign activity contains the copy commands
- `<Copy>`: is used for coping data between variables specifying source (`<from>`) and destination (`<to>`)

NOTE: copy can be performed only if both variables are of the same type

Assignment Example

- Structure:

```
<assign>
  <copy>
    <from ... />
    <to ... />
  </copy>
  <copy>
    <from ... />
    <to ... />
  </copy>
...
</assign>
```

- Example:

```
<assign>
  <copy>
    <from variable="InsuranceAResponse" />
    <to variable="InsuranceSelectionResponse" />
  </copy>
</assign>
```

Copy part of a message using different variables type

- Simple message of a WSDL document 1

```
<message name="InsuranceRequestMessage">
  <part name="insuredPersonData" element="ins:InsuredPersonData" />
  <part name="insuranceDetails" element="ins:InsuranceDetails" />
</message>
```

- Simple message of a WSDL document 2

```
<message name="InsuredPersonDataRequestMessage">
  <part name="insuredPersonData" element="ins:InsuredPersonData" />
</message>
```

- BPEL variables declaration

```
<variables>
  <variable name="InsuredPersonRequest" messageType="ins:InsuredPersonDataRequestMessage"/>
  <variable name="InsuranceRequest" messageType="ins:InsuranceRequestMessage"/>
</variables>
```

- Assignment between two WSDL variables

```
<assign>
  <copy>
     <from variable="InsuredPersonRequest" part="insuredPersonData" />
     <to variable="InsuranceRequest" part="insuredPersonData" />
  </copy>
</assign>
```

Select parts using query language

XML schema of the service

```
<xs:element name="InsuredPersonData">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FirstName" type="xs:string" />
      <xs:element name="LastName" type="xs:string" />
      <xs:element name="Address" type="xs:string" />
      <xs:element name="Age" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Copy of the LastName variable to the InsuranceRequest using query language

```
<assign>
  <copy>
    <from variable="LastName" />
    <to variable="InsuranceRequest"part="insuredPersonData">
      <query>ins:LastName</query>
    </to>
  </copy>
</assign>
```

Copy a constant string

Copying a constant string to the `LastName` variable

```
<assign>
  <copy>
    <from>string("Juric")</from>
    <to variable="LastName"/>
  </copy>
</assign>
```

Copy a literal XML complex element

We specify directly the XML to copy in the `InsuredPersonRequest`

```
<assign>
  <copy>
    <from>
      <literal>
        <insuredPersonData xmlns="http://packtpub.com/bpel/insurance/">
          <FirstName>Matjaz B.</FirstName>
          <LastName>Juric</LastName>
          <Address>Ptuj</Address>
          <Age>30</Age>
        </insuredPersonData>
      </literal>
    </from>
    <to variable="InsuredPersonRequest" part="insuredPersonData" />
  </copy>
</assign>
```

Validating variables

During the assignments, if we **don't validate the variables** using the XML Schema and WSDL. It is possible to validate **explicitly** using the `validate` activity.

It is sufficient to list the variables separated by spaces.

```
<validate variables=" InsuredPersonRequest InsuranceRequest
                      PartialInsuranceDescription " />
```

Accessing variables using expressions (messageType)

Using Xpath is possible to access nested elements variables

- Definition of `messageType` variables

```
<variables>
   <variable name="InsuredPersonRequest"
      messageType="ins:InsuredPersonDataRequestMessage"/>
</variables>
```

- WSDL and the XML schema look as follow:

```
<message name="InsuredPersonDataRequestMessage">
   <part name="insuredPersonData" element="ins:InsuredPersonData" />
</message>

<xs:element name="InsuredPersonData">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FirstName" type="xs:string" />
      <xs:element name="LastName" type="xs:string" />
      <xs:element name="Address" type="xs:string" />
      <xs:element name="Age" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- BPEL example (access the LastName)

```
$variableName.messagePart/ns:node/ns:node...
$InsuredPersonRequest.insuredPersonData/ins:LastName
```

Accessing variables using expressions (XML element)

- Definition of `XML element` variables

```
<variables>
   <variable name="PartialInsuranceDescription"
      element="ins:InsuranceDescription"/>
</variables>
```

- WSDL and the XML schema look as follow:

```
<xs:element name="InsuranceDescription">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Code" type="xs:string" />
         <xs:element name="Description" type="xs:string" />
         <xs:element name="ValidFrom" type="xs:date" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

- BPEL example

```
<!-- default pattern-->
$ variableName/ns:node/ns:node...
$ PartialInsuranceDescription /ins:Description
```

Accessing variables using expressions (XML Type)

- Definition of `XML element` variables

```
<variables>
   <variable name="Address" type="ins:AddressType"/>
</variables>
```

- WSDL and the XML schema look as follow:

```
<xs:complexType name="AddressType">
   <xs:sequence>
      <xs:element name="Street" type="xs:string" />
      <xs:element name="Number" type="xs:int" />
      <xs:element name="City" type="xs:string" />
   </xs:sequence>
</xs:complexType>
```

- BPEL example

```
<!-- default pattern-->
$variableName/ns:node/ns:node...
$ Address /ins:Street
```

Conditions

Choices based on conditions

```
<if>
  <condition> boolean-expression </condition>
  <!-- some activity -->
  <elseif>
    <condition> boolean-expression </condition>
    <!-- some activity -->
  </elseif>
  <elseif>
    <condition> boolean-expression </condition>
    <!-- some activity -->
  </elseif>
    ...
  <else>
    <!-- some activity -->
  </else>
</if>
```

```
<if>
  <condition>
    $InsuranceRequest.insuredPersonData/ins:Age &gt; 50
  </condition>
  <!-- perform activities for age 51 and over -->
  <elseif>
    <condition>
      $InsuranceRequest.insuredPersonData/ins:Age &gt; 25
    </condition>
    <!-- perform activities for age 26-50 -->
  </elseif>
  <else>
    <!-- perform activities for age 25 and under -->
  </else>
</if>
```

- The **expressions** for `<condition>` elements are expressed in the **XPath query language**
- **Variable are usually used in conditions** and accessed in the same way of the assignment

Activity name

`<if>`, `<invoke>`, `<reply>`,`<sequence>` can specify name using name attributes.

- Name can be used for **all basic and structured activities**
- Name are useful for:
  - **invoking** inline compensation handlers
  - **synchronizing** activity

```
...
<invoke name="EmployeeTravelStatusSyncInv"
   partnerLink="employeeTravelStatus"
   portType="emp:EmployeeTravelStatusPT"
   operation="EmployeeTravelStatus"
   inputVariable="EmployeeTravelStatusRequest"
   outputVariable="EmployeeTravelStatusResponse" />
...
```

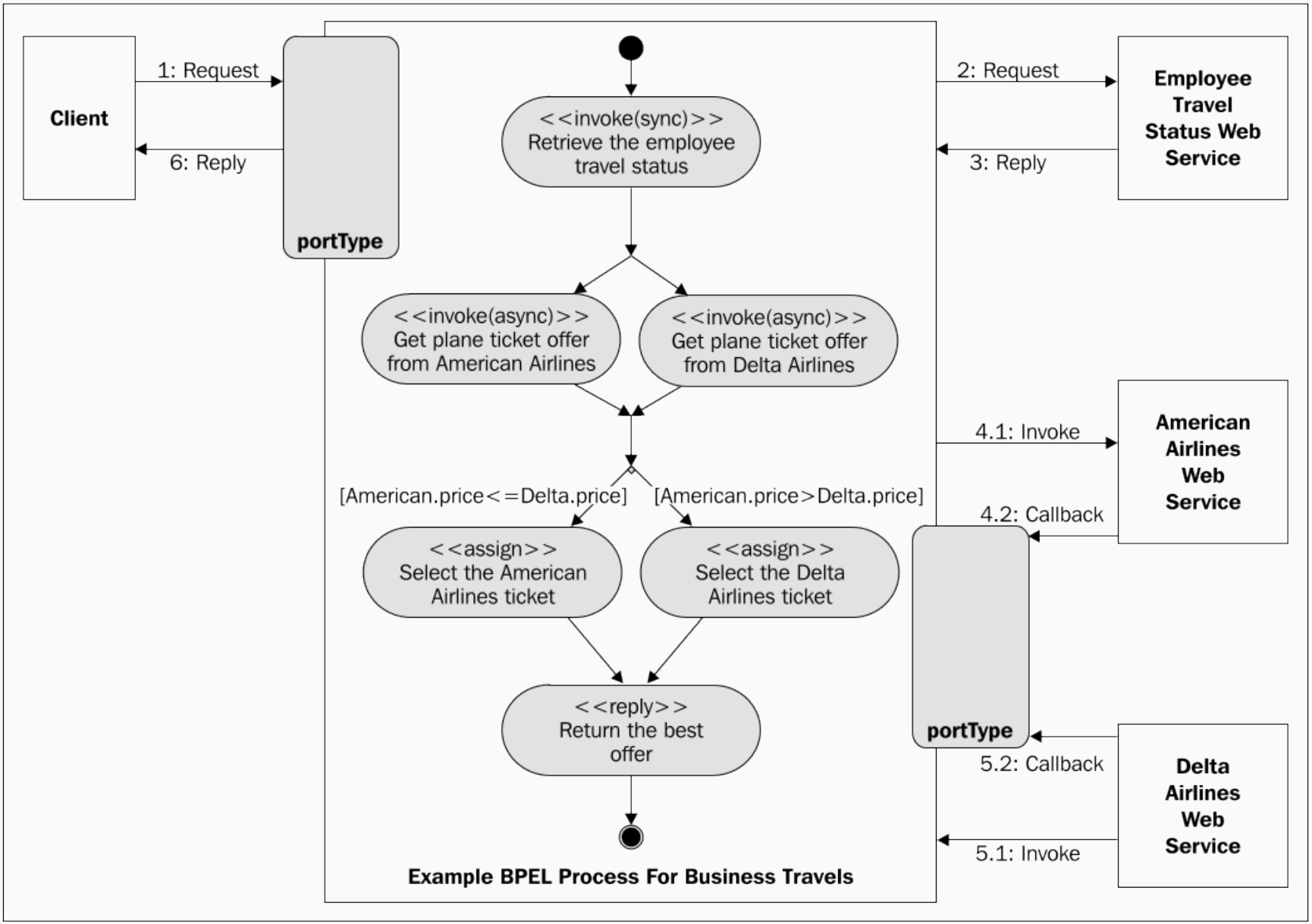
Documentation

In BPEL is possible to include documentation for each activity.

```
...
<invoke name="EmployeeTravelStatusSyncInv"
  partnerLink="employeeTravelStatus"
  portType="emp:EmployeeTravelStatusPT"
  operation="EmployeeTravelStatus"
  inputVariable="EmployeeTravelStatusRequest"
  outputVariable="EmployeeTravelStatusResponse">
     <documentation>
        Invoking the Employee Travel Status service to get the travel class for an employee.
     </documentation>
</invoke>
...
```

Business Process Case Study

BPEL example



Example details

- The **client invoke the BP** specifying; name of employee, destination, departure and return date.
- **Check the employee travel status** (economy, business, first)
- The BP **check the price** of the flight with two airlines
- The BP **select the lower price** and return the travel plan to the client

Invocations:

- The **check for the price** is developed **asynchronously** the rest of the operations are synchronous.

Development steps

Although the presented example is very symple. For reaching the final goal we should complete the following steps:

- Get familiar with the **involved services**
- **Define the WSD**L for the BPEL process
- **Define partner link types**
- **Define partner links**
- **Declare variables**
- **Write the process logic** definition

Involved Services

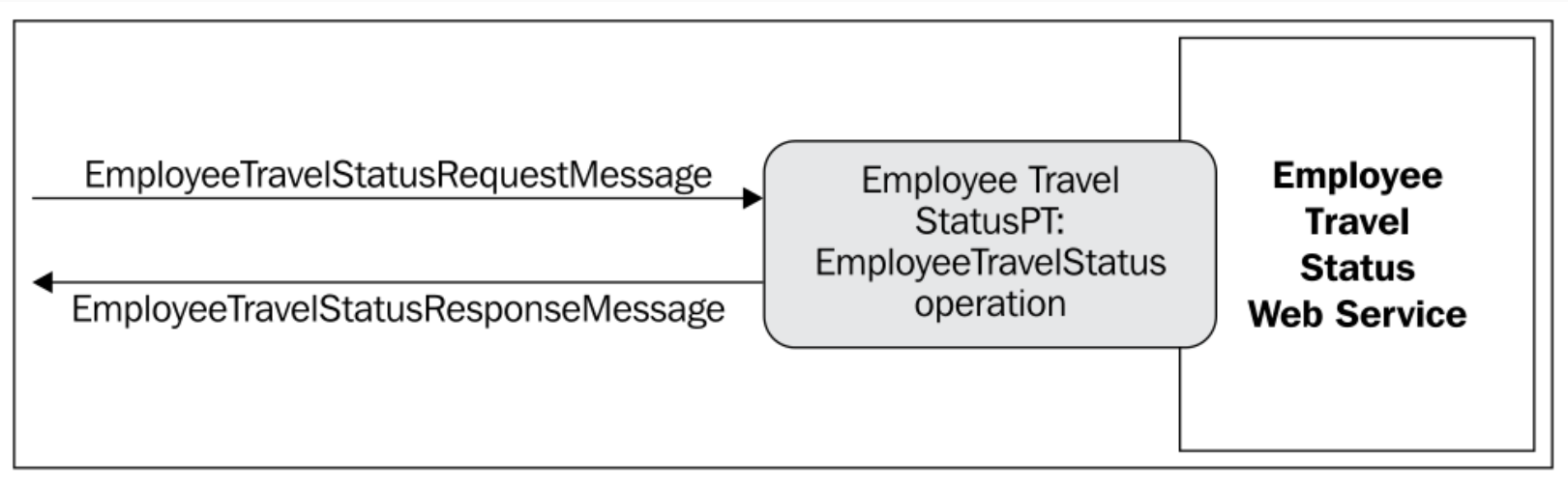
Before to start with the BP we **get to be familiar** with the **involved service**.

- The Employee Travel Status service
- The American Airlines service
- The Delta Airlines service

The services are exposed through WSDL.

Employee Travel Status service

The employee travel status provides the following opertion:



The synchronous operation **return the travel class** an employee can use: economy, business, first.

Employee WSDL

```
<?xml version="1.0" encoding="utf-8" ?>
  <definitions
     xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:tns="http://packtpub.com/service/employee/"
     targetNamespace="http://packtpub.com/service/employee/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
   ...
  <portType name="EmployeeTravelStatusPT">
    <operation name="EmployeeTravelStatus">
      <input message="tns:EmployeeTravelStatusRequestMessage" />
      <output message="tns:EmployeeTravelStatusResponseMessage" />
    </operation>
  </portType>
   ...
```

The `employeeTravelStatus` operation consists of an input/output message

```
...
<message name="EmployeeTravelStatusRequestMessage">
   <part name="employee" element="tns:Employee" />
</message>
<message name="EmployeeTravelStatusResponseMessage">
   <part name="travelClass" element="tns:TravelClass" />
</message>
...
```

Employee WSDL

The `EmployeeTravelStatusRequestMessage` has a single part employee of element `Employee` with type `EmployeeType`

```
...
<types>
  <xs:schema elementFormDefault="qualified"
    targetNamespace="http://packtpub.com/service/employee/">
  <xs:complexType name="EmployeeType">
    <xs:sequence>
      <xs:element name="FirstName" type="xs:string" />
      <xs:element name="LastName" type="xs:string" />
      <xs:element name="Department" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Employee" type="EmployeeType"/>
```

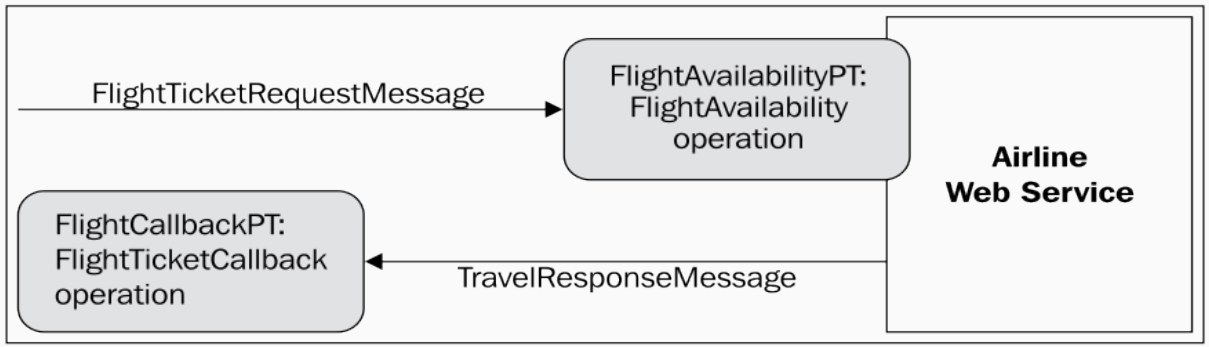
The `TravelClassType` is a simple type that uses enumeration to list

```
    <xs:simpleType name="TravelClassType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="Economy"/>
        <xs:enumeration value="Business"/>
        <xs:enumeration value="First"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:element name="TravelClass" type="TravelClassType"/>
  </xs:schema>
</types>
...
```

Arirline service

The airline is an **asynchronous** web service.

- `FlightAvailabilityPT` used to check the flight availability using the `FlightAvailability` operation
- `FlightCallbackPT` specifies the FlightCallback operation



Airline Request WSDL

The asynchronous flight service contain only the input message.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:emp="http://packtpub.com/service/employee/"
  xmlns:tns="http://packtpub.com/service/airline/"
  targetNamespace="http://packtpub.com/service/airline/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
...
<portType name="FlightAvailabilityPT">
  <operation name="FlightAvailability">
     <input message="tns:FlightTicketRequestMessage" />
  </operation>
</portType>
...
```

Airline Request WSDL

The definition of synchronous message consist of two parts `flightData` and `travelClass`.

```
<message name="FlightTicketRequestMessage">
   <part name="flightData" element="tns:FlightRequest" />
   <part name="travelClass" element="emp:TravelClass" />
</message>
```

The `flightData` is element of `FlightRequest` that is a complex type `FlightRequestType` composed by four elements.

```
...
<types>
  <xs:schema elementFormDefault="qualified" targetNamespace="http://packtpub.com/service/airline/">
  <xs:complexType name="FlightRequestType">
    <xs:sequence>
      <xs:element name="OriginFrom" type="xs:string" />
      <xs:element name="DestinationTo" type="xs:string" />
      <xs:element name="DesiredDepartureDate" type="xs:date" />
      <xs:element name="DesiredReturnDate" type="xs:date" />
    </xs:sequence>
  </xs:complexType>
<xs:element name="FlightRequest" type="FlightRequestType"/>
...
```

Airline Callback WSDL

The FlightCallbackPT has the FlightTicketCallback operation with the TravelResponseMessage input message

```
      ...
<portType name="FlightCallbackPT">
  <operation name="FlightTicketCallback">
    <input message="tns:TravelResponseMessage" />
  </operation>
</portType>
  ...
```

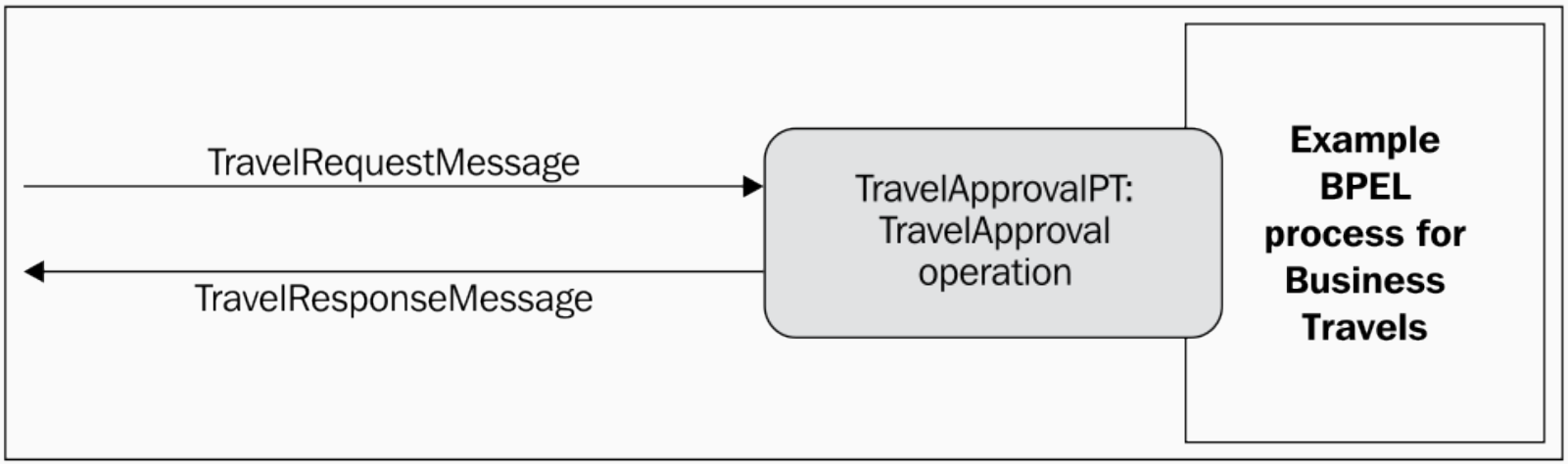
```
...
<message name="TravelResponseMessage">
  <part name="confirmationData"
      element="tns:FlightConfirmation" />
</message>
...
```

The FlightConfirmation is a complex type FlightConfirmationType

```
    <xs:complexType name="FlightConfirmationType">
      <xs:sequence>
        <xs:element name="FlightNo" type="xs:string" />
        <xs:element name="TravelClass" type="tns:TravelClassType" />
        <xs:element name="Price" type="xs:float" />
        <xs:element name="DepartureDateTime" type="xs:dateTime" />
        <xs:element name="ReturnDateTime" type="xs:dateTime" />
        <xs:element name="Approved" type="xs:boolean" />
      </xs:sequence>
    </xs:complexType>
  <xs:element name="FlightConfirmation" type="FlightConfirmationType"/>
</xs:schema>
</types>
```

BPEL process

After the definition of the involved service, we can define the BPEL Process.



Since the BPEL is a process we need to define the **WSDL that permit to communicate with the clients**. The service is composed by `TravelApprovalPT` with the `TravelApproval` operation.

BPEL WSDL

The `TravelApproval` operation will be synchronous request/response type

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:emp="http://packtpub.com/service/employee/"
  xmlns:aln="http://packtpub.com/service/airline/"
  xmlns:tns="http://packtpub.com/bpel/travel/"
  targetNamespace="http://packtpub.com/bpel/travel/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
  ...
  <portType name="TravelApprovalPT">
    <operation name="TravelApproval">
      <input message="tns:TravelRequestMessage" />
      <output message="aln:TravelResponseMessage" />
    </operation>
  </portType>
```

BPEL TravelRequestMessageWSDL

The `TravelRequestMessage` consists of two parts:

- `employee`: reuse the Employee travel status service
- `flightData`: reuse the airline service

```
...
  <import namespace="http://packtpub.com/service/employee/" location="./Employee.wsdl"/>
  <import namespace="http://packtpub.com/service/airline/" location="./Airline.wsdl"/>
...
  <message name="TravelRequestMessage">
    <part name="employee" element="emp:Employee" />
    <part name="flightData" element="aln:FlightRequest" />
  </message>
...
```

For the `TravelResponseMessage` is used the same message type used to return the flight information in the airline service.

Partner link types

- Represent the interaction between BPEL and the involved parties.
  - `travelLT` (client): defined in the WSDL of the BP. The interaction is synchronous.
  - `employeeLT` (employee travel service): defined in the WSDL of the employee service. The interaction is synchronous.
  - `flightLT` (airline services): defined in the WSDL of the airline service. The interaction is Asynchronous.

The **partner link type** can have **one or two roles** and for each role we specify the Port Type.

For **synchronous** operation there is a **single role** in the PT because the operation is invoked in a single direction.

For **asynchronous** operation we need to specify **two roles**.

Partner Link Type

Partner Link Types are defined using a special namespace (`http://docs.oasis-open.org/wsbpel/2.0/plnktype`)

```
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:emp="http://packtpub.com/service/employee/"
  xmlns:aln="http://packtpub.com/service/airline/"
  xmlns:tns="http://packtpub.com/bpel/travel/"
  targetNamespace="http://packtpub.com/bpel/travel/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
...
```

Link type in the BPEL process WSDL

- TravelLT

```
...
<plnk:partnerLinkType name="travelLT">
  <plnk:role name="travelService" portType="tns:TravelApprovalPT" />
</plnk:partnerLinkType>
...
```

- EmployeeLT

```
...
  <plnk:partnerLinkType name="employeeLT">
    <plnk:role name="employeeTravelStatusService" portType="tns:EmployeeTravelStatusPT" />
  </plnk:partnerLinkType>
...
```

- FlightLT

```
...
  <plnk:partnerLinkType name="flightLT">
    <plnk:role name="airlineService" portType="tns:FlightAvailabilityPT" />
    <plnk:role name="airlineCustomer" portType="tns:FlightCallbackPT" />
  </plnk:partnerLinkType>
...
```

For **each role** we need to **specify the a portType**

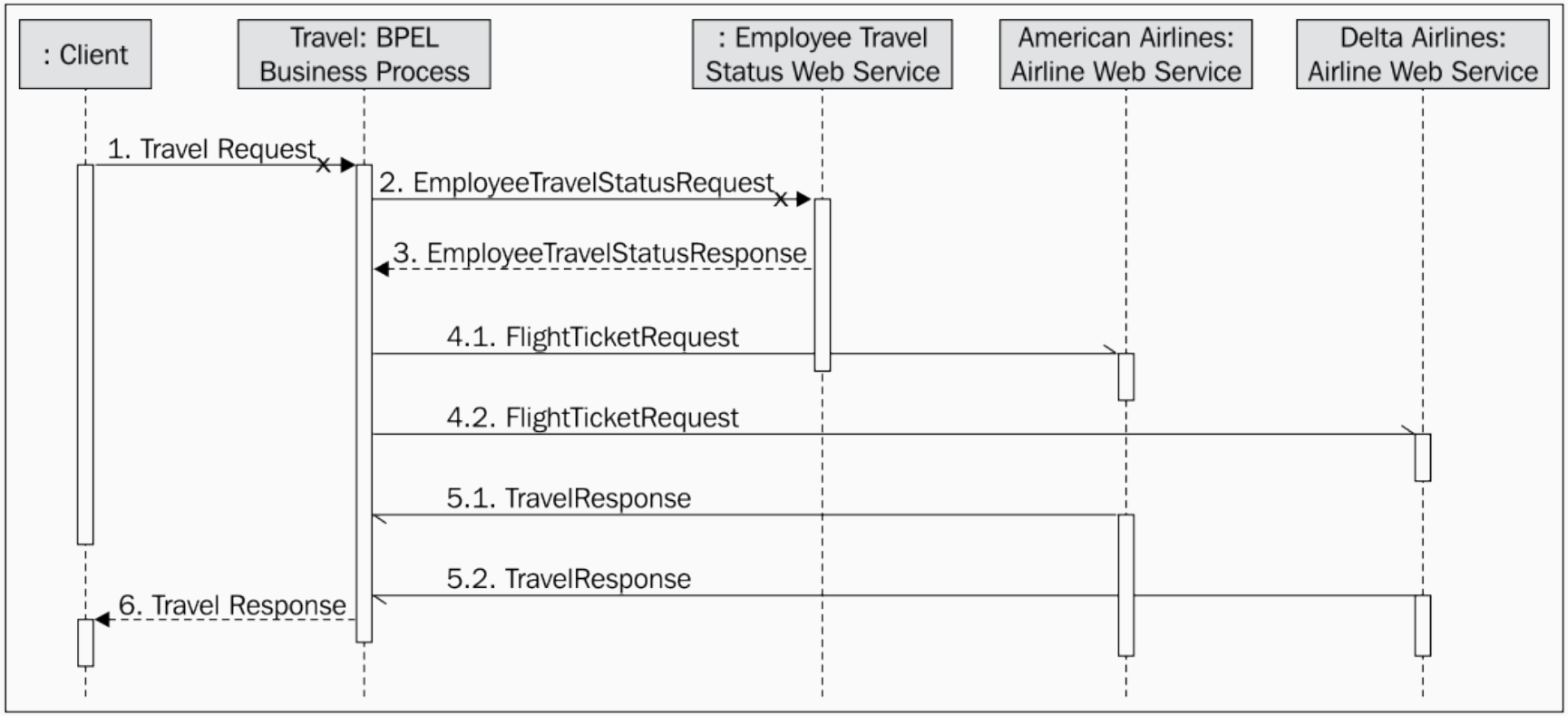
Busienss Process Definition

- Specify the **order of activities** that have to be performed
- Generally **waits for an incoming message**

The **involved parties** are:

- The client that invoke the BPEL process
- The BPEL process itself
- The employee travel status service
- Two airline web services (American and Delta)

Busienss Process Definition



Generally is not a good programming methodology define **synchronous BPEL process** that use **asynchronous web service** (4.1; 4.2).

Business Process Outline

Business Process contain at least four main parts

- The initial <process> root element with the declaration of namespaces
- The definition of partner links, using the <partnerLinks> element
- The declaration of variables, using the <variables> element
- The main body where the actual business process is defined; is the <sequence> element that specifies the flow of the process

```
<process name="Travel" ... >
  <partnerLinks>
    <!-- The declaration of partner links -->
  </partnerLinks>
  <variables>
    <!-- The declaration of variables -->
  </variables>
  <sequence>
    <!-- The definition of the BPEL business process main body -->
  </sequence>
</process>
```

Travel Business Process

In the definition we have to define the **target namespace for the BPEL process** and the **namespaces to access the involved service WSDL**.

Example:

```
<process name="Travel"
  targetNamespace="http://packtpub.com/bpel/travel/"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:trv="http://packtpub.com/bpel/travel/"
  xmlns:emp="http://packtpub.com/service/employee/"
  xmlns:aln="http://packtpub.com/service/airline/" >
  ...
```

Partner Links

- Partner links **define different parties** that interact with the BPEL process.
- Each partner link is **related** to a specific `partnerLinkType`
- Partner links attributes:
  - `myRole`: Indicates the role of the business process itself
  - `partnerRole`: Indicates the role of the partner

Partner Links in the Travel BP

```
<partnerLinks>
  <partnerLink name="client"
    partnerLinkType="trv:travelLT"
    myRole="travelService"/>

  <partnerLink name="employeeTravelStatus"
    partnerLinkType="emp:employeeLT"
    partnerRole="employeeTravelStatusService"/>

  <partnerLink name="AmericanAirlines"
    partnerLinkType="aln:flightLT"
    myRole="airlineCustomer"
    partnerRole="airlineService"/>

  <partnerLink name="DeltaAirlines"
    partnerLinkType="aln:flightLT"
    myRole="airlineCustomer"
    partnerRole="airlineService"/>
</partnerLinks>
```

- Partnerlinks **specify a single role** in in the **synchronous** request/response operations
- Usually we describe the **role for the service that receive the invocation**
- In the **asynchronous callback communication we need two roles**

Variables for the Travel BP

Variables are:

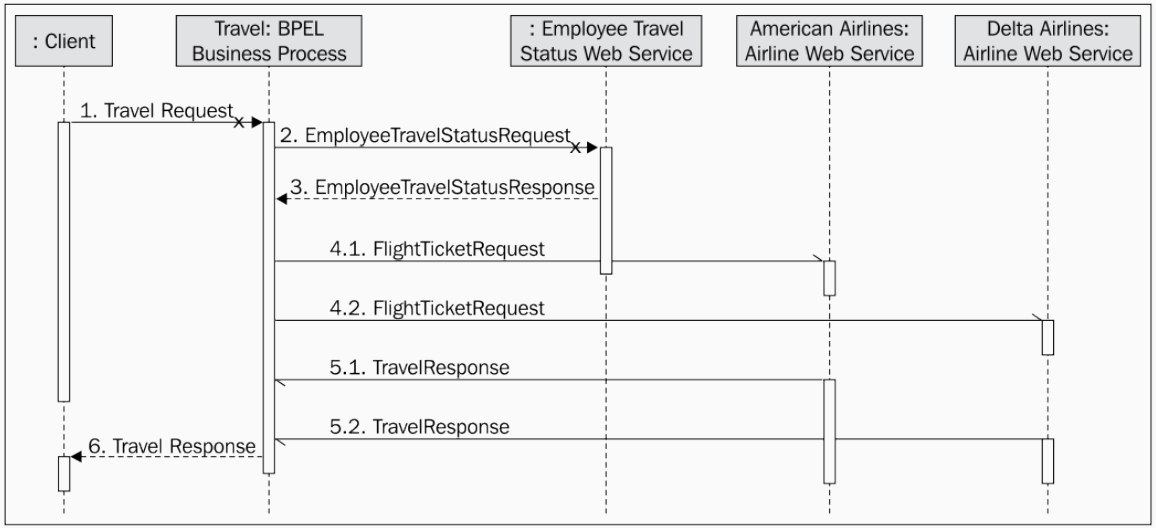
- used to **store**, **reformat** and **transform** messages.
- Usually we need **a variable for every message sent and received**
- Possible type: **WSDL message**; **XML schema**; **XML element**

```
<variables>
    <!-- input for this process -->
  <variable name="TravelRequest" messageType="trv:TravelRequestMessage"/>
    <!-- input for the Employee Travel Status service -->
  <variable name="EmployeeTravelStatusRequest" messageType="emp:EmployeeTravelStatusRequestMessage"/>
    <!-- output from the Employee Travel Status service -->
  <variable name="EmployeeTravelStatusResponse" messageType="emp:EmployeeTravelStatusResponseMessage"/>
    <!-- input for American and Delta services -->
  <variable name="FlightDetails" messageType="aln:FlightTicketRequestMessage"/>
    <!-- output from American Airlines -->
  <variable name="FlightResponseAA" messageType="aln:TravelResponseMessage"/>
    <!-- output from Delta Airlines -->
  <variable name="FlightResponseDA" messageType="aln:TravelResponseMessage"/>
    <!-- output from BPEL process -->
  <variable name="TravelResponse" messageType="aln:TravelResponseMessage"/>
</variables>
```

Travel Process main body

Contains:

- one **top-level activity** (`<sequence>`)
- **matching message** used to start the process (`<receive>`)



Travel Request message:

```
...
<sequence>
    <!-- Receive the initial request for business travel from client -->
  <receive name="ReceiveInitialRequest" partnerLink="client"
    portType="trv:TravelApprovalPT" operation="TravelApproval"
    variable="TravelRequest" createInstance="yes" />
...
```

Employee Travel Request

We **prepare the input** for **invoke** the Employee Travel status service

```
...
<!-- Prepare the input for the Employee Travel Status Service -->
<assign name="PrepareInputForEmployeeWS">
  <copy>
    <from variable="TravelRequest" part="employee"/>
    <to variable="EmployeeTravelStatusRequest" part="employee"/>
  </copy>
</assign>

<!-- Synchronously invoke the Employee Travel Status Service
using the employeeTravelStatus partnerlink -->
<invoke name="InvokeEmployeeWS"
  partnerLink="employeeTravelStatus"
  portType="emp:EmployeeTravelStatusPT"
  operation="EmployeeTravelStatus"
  inputVariable="EmployeeTravelStatusRequest"
  outputVariable="EmployeeTravelStatusResponse" />
...
```

Airline Web Service invocation

The `FlightTicketRequest` message consists of two parts:

- `flightData`: This is retrieved from the client message (TravelRequest)
- `travelClass`: This is retrieved from the EmployeeTravelStatusResponse variable

```
...
  <!-- Prepare the input for AA and DA -->
  <assign name="PrepareInputForAAandDA">
    <copy>
      <from variable="TravelRequest" part="flightData"/>
      <to variable="FlightDetails" part="flightData"/>
    </copy>
    <copy>
      <from variable="EmployeeTravelStatusResponse"part="travelClass"/>
      <to variable="FlightDetails" part="travelClass"/>
    </copy>
  </assign>
...
```

Airline Web Service invocation

The invocations to airline web services consist of two steps:

- The `<invoke>` activity is used for the asynchronous **invocation**.
- The `<receive>` activity is used to **wait** for the callback.

```
...
<!-- Make invocation to AA in DA -->
<flow name="InvokeAAandDA">
  <sequence>
  <!--Async invoke of the AA web service
      and wait for the callback-->

    <invoke name="InvokeAA"
      partnerLink="AmericanAirlines"
      portType="aln:FlightAvailabilityPT"
      operation="FlightAvailability"
      inputVariable="FlightDetails" />

    <receive name="ReceiveCallbackFromAA"
      partnerLink="AmericanAirlines"
      portType="aln:FlightCallbackPT"
      operation="FlightTicketCallback"
      variable="FlightResponseAA" />
  </sequence>
```

```
  <sequence>
    <!--Async invoke of the DA web service
        and wait for the callback-->

    <invoke name="InvokeDA"
      partnerLink="DeltaAirlines"
      portType="aln:FlightAvailabilityPT"
      operation="FlightAvailability"
      inputVariable="FlightDetails" />

    <receive name="ReceiveCallbackFromDA"
      partnerLink="DeltaAirlines"
      portType="aln:FlightCallbackPT"
      operation="FlightTicketCallback"
      variable="FlightResponseDA" />
  </sequence>
</flow>
  ...
```

Selection of tickets offer

```
...
  <!-- Select the best offer and construct the TravelResponse -->
  <if name="SelectBestOffer">
    <condition>
      $FlightResponseAA.confirmationData/aln:Price &lt;=
      $FlightResponseDA.confirmationData/aln:Price
    </condition>
    <!-- Select American Airlines -->
    <assign>
      <copy>
        <from variable="FlightResponseAA" />
        <to variable="TravelResponse" />
      </copy>
    </assign>
  <else>
  <!-- Select Delta Airlines -->
    <assign>
      <copy>
        <from variable="FlightResponseDA" />
        <to variable="TravelResponse" />
      </copy>
    </assign>
  </else>
</if>
...
```

Reply to the Client

```
...
<!-- Send a response to the client -->
  <reply name="SendResponse"
    partnerLink="client"
    portType="trv:TravelApprovalPT"
    operation="TravelApproval"
    variable="TravelResponse"/>
  </sequence>
</process>
```

Here we specify the **same partner link** as in the initial receive client. We also specify the **same portType and operation name**.

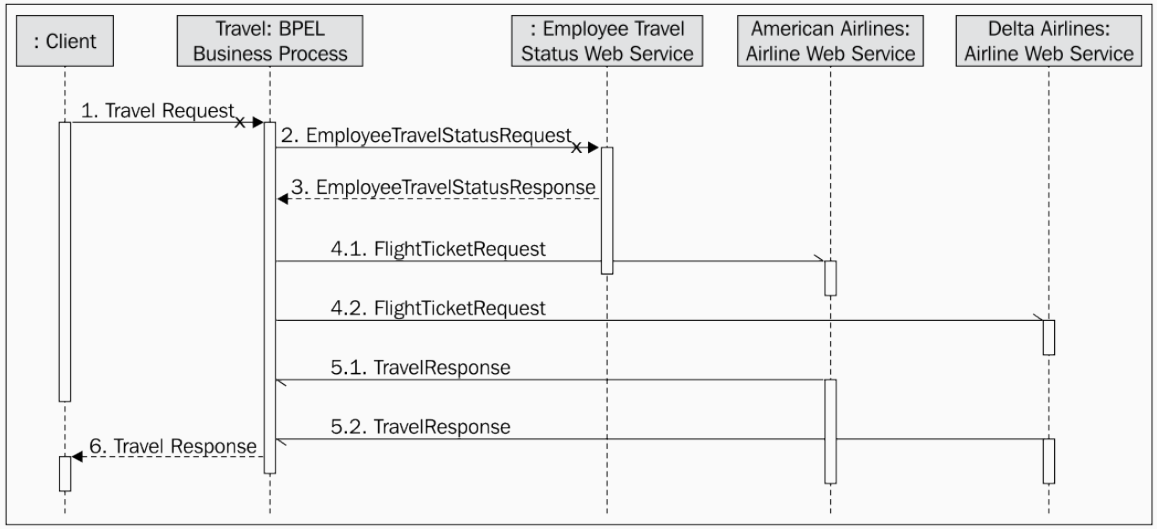
Asynchronous Process

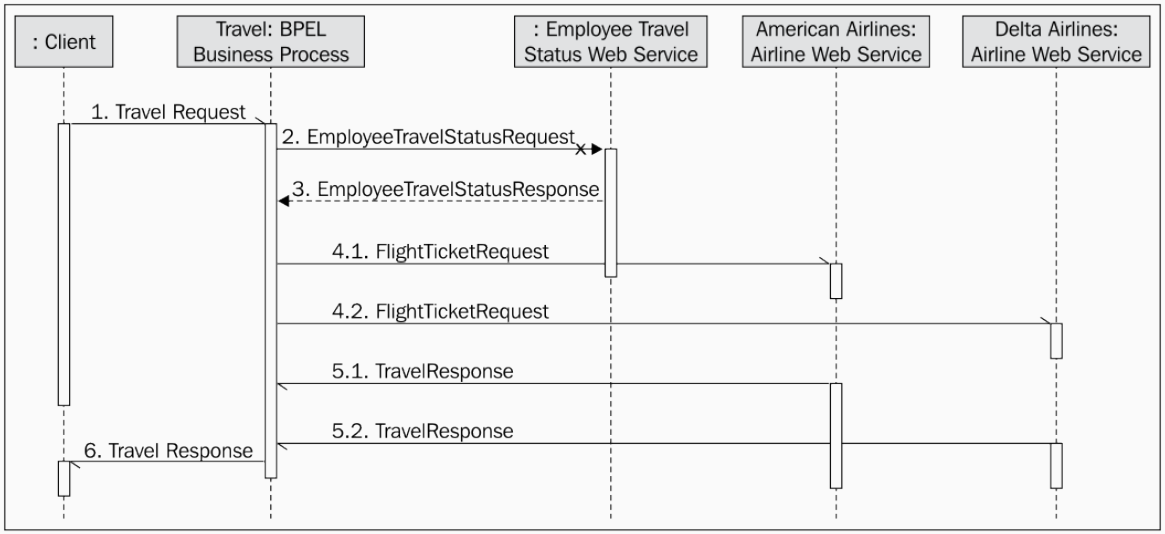
Asynchronous BPEL example

It makes no sense for a **client** to wait and be **blocked for the entire duration of the process**.
Model the process asynchronously has some consequences:

- For the BPEL process to be able to **perform a callback** to the client, the client must be a **service and implement a certain port type**
- The **partner link type** for the client will have to **specify two roles**
- The BPEL process will not `<reply>` to the client. Rather it will `<invoke>` the callback

Difference between sequence diagram

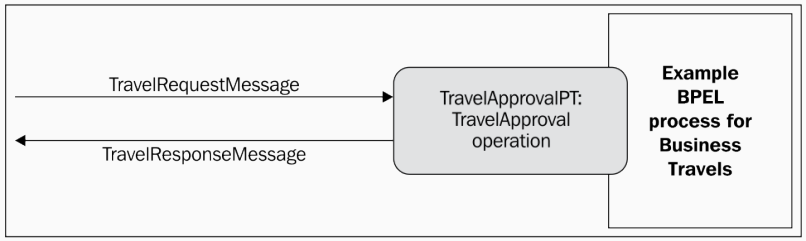


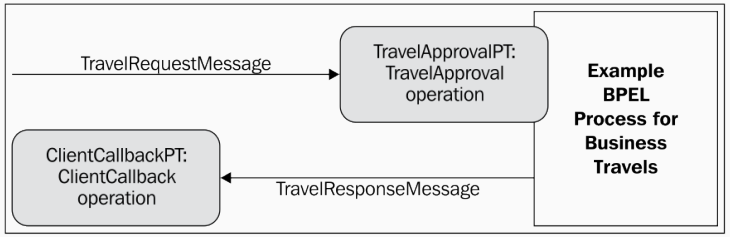


Transformation steps

1. **Modify the BPEL process WSDL**, (operation invoked by the client will now have only the input message).
2. Define the **client port type and the operation** in the WSDL of the BPEL process.
3. **Modify the partner link type**, where we will add the second role.
4. **Modify the BPEL process specification**. We have to modify the partner link and replace the `<reply>` activity with an `<invoke>` .

BPEL process WSDL





TravelApprovalPT

```
...
<portType name="TravelApprovalPT">
  <operation name="TravelApproval">
    <input message="tns:TravelRequestMessage"/>
    <output message="aln:TravelResponseMessage"/>
  </operation>
</portType>
...
```

TravelApprovalPT

```
...
 <portType name="TravelApprovalPT">
   <operation name="TravelApproval">
     <input message="tns:TravelRequestMessage" />
   </operation>
 </portType>
...
```

ClientCallbackPT operation

```
...
 <portType name="ClientCallbackPT">
   <operation name="ClientCallback">
     <input message="aln:TravelResponseMessage" />
   </operation>
 </portType>
...
```

Modify the Partner link types

TravelLT: WSDL BP

```
...
  <plnk:partnerLinkType name="travelLT">
       <plnk:role name="travelService"
         portType="tns:TravelApprovalPT" />
  </plnk:partnerLinkType>
                      ...
```

```
<plnk:partnerLinkType name="travelLT">
   <plnk:role name="travelService"
      portType="tns:TravelApprovalPT" />
   <plnk:role name="travelServiceCustomer"
      portType="tns:ClientCallbackPT" />
</plnk:partnerLinkType>
```

Modify BPEL Process Definition

Partner Link definition:

```
<partnerLink name="client"
  partnerLinkType="trv:travelLT"
  myRole="travelService"/>
```

```
<partnerLink name="client"
  partnerLinkType="trv:travelLT"
      myRole="travelService"
      partnerRole="travelServiceCustomer"/>
```

TravelResponse Invocation:

```
<!-- Send a response to the client -->
  <reply name="SendResponse"
    partnerLink="client"
    portType="trv:TravelApprovalPT"
    operation="TravelApproval"
    variable="TravelResponse"/>
</sequence>
</process>
```

```
<!-- Make a callback to the client -->
  <invoke name="SendResponse"
    partnerLink="client"
    portType="trv:ClientCallbackPT"
    operation="ClientCallback"
    inputVariable="TravelResponse" />
  </sequence>
</process>
```

JDeveloper Guide: http://docs.oracle.com/cd/E37547_01/tutorials/tut_web_services/tut_web_services.html

JDeveloper Tool:

http://www.oracle.com/technetwork/developer-tools/jdev/downloads/index.html

Questions?