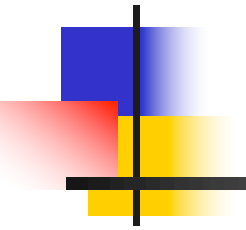


Unit 2

Software Development by Composition and Integration

Lecture 2-5

BPEL Case Studies

Dr. Yinong Chen

Case Study, also read Text Section 8.2.5

Source: Matjaz B., Juric, et al., Business Process Execution Language for Web Services BPEL and BPEL4WS, Packt Publishing; 2nd edition Jan 30, 2006.

- Design a BPEL Process for a company's travel request, which interacts with the following services:
  - Client: Access the BPEL process as a Web service
  - Employee Travel Status Web service: provides the regulations on travel;
  - Two airlines companies: are used for requesting ticket prices
- More case studies online and in YouTube video:
  - Last slide in this lecture

Where to Find Airlines Services? E.g., FlightStats Web Services

https://developer.flightstats.com/api-docs

Airlines

The *Airlines* API provides basic reference information about one or more airlines.

Base URI	https://api.flightstats.com/flex/airlines/rest/
Version	v1
WADL	https://api.flightstats.com/flex/airlines/rest/v1/schema/wadl
WSDL	https://api.flightstats.com/flex/airlines/soap/v1/airlinesService?wsdl
XSD	https://api.flightstats.com/flex/airlines/rest/v1/schema/xsd
JSON	https://api.flightstats.com/flex/airlines/rest/v1/schema/json

Entry Points	Name	Description
active/	Active airlines	*Airline* information for active airlines
all/	All airlines	*Airline* information for all airlines, active and inactive.
fs/	Airlines by FlightStats code	*Airline* information for airlines selected by FlightStats code and optionally date.
iata/	Airlines by IATA code	*Airline* information for airlines selected by IATA code and optionally date.
icao/	Airlines by ICAO code	*Airline* information for airlines selected by ICAO code and optionally date.

Airports

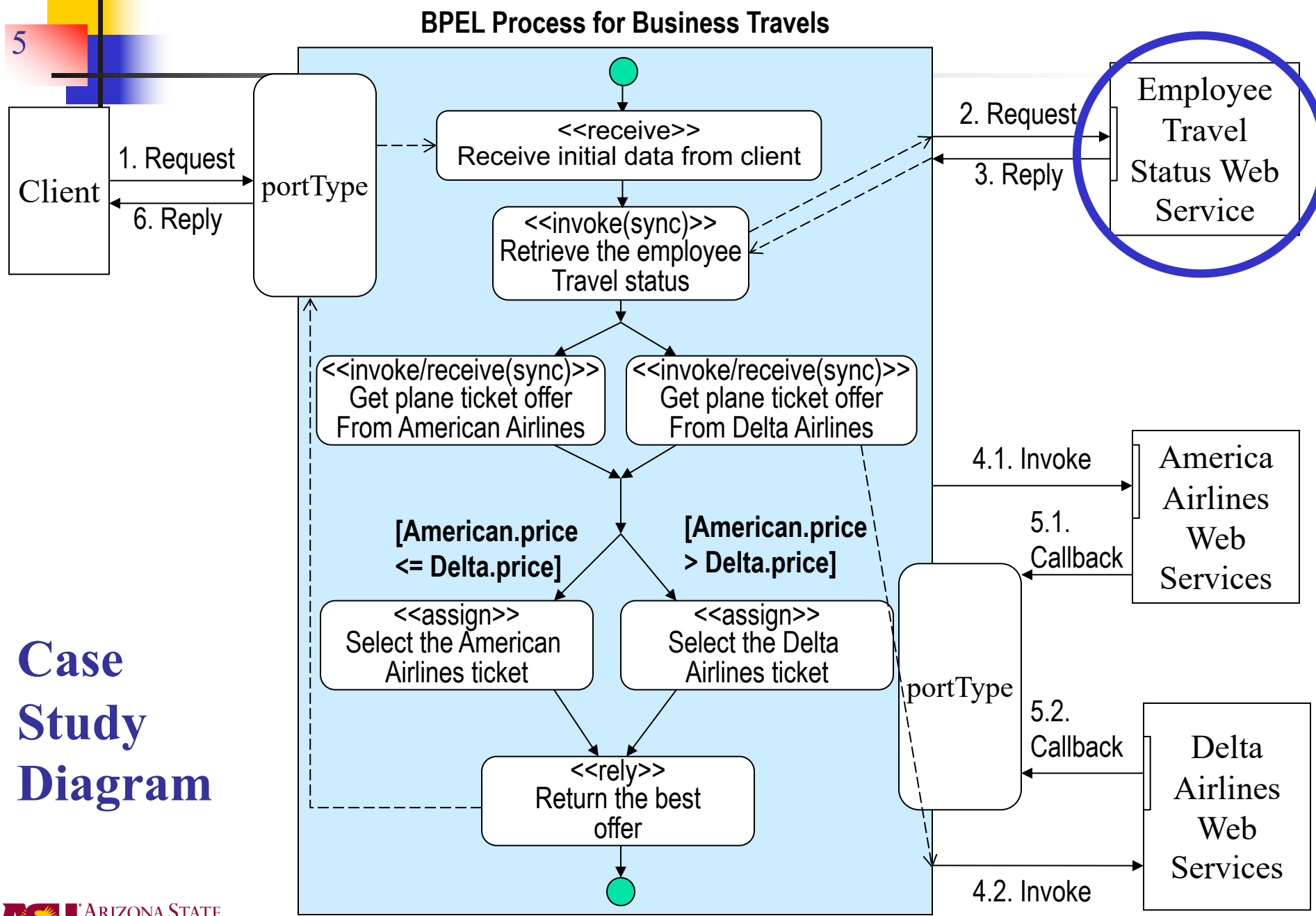
The *Airports* API provides basic reference information about one or more airports.

Base URI	https://api.flightstats.com/flex/airports/rest/
Version	v1
WADL	https://api.flightstats.com/flex/airports/rest/v1/schema/wadl
WSDL	https://api.flightstats.com/flex/airports/soap/v1/airportsService?wsdl
XSD	https://api.flightstats.com/flex/airports/rest/v1/schema/xsd
JSON	https://api.flightstats.com/flex/airports/rest/v1/schema/json

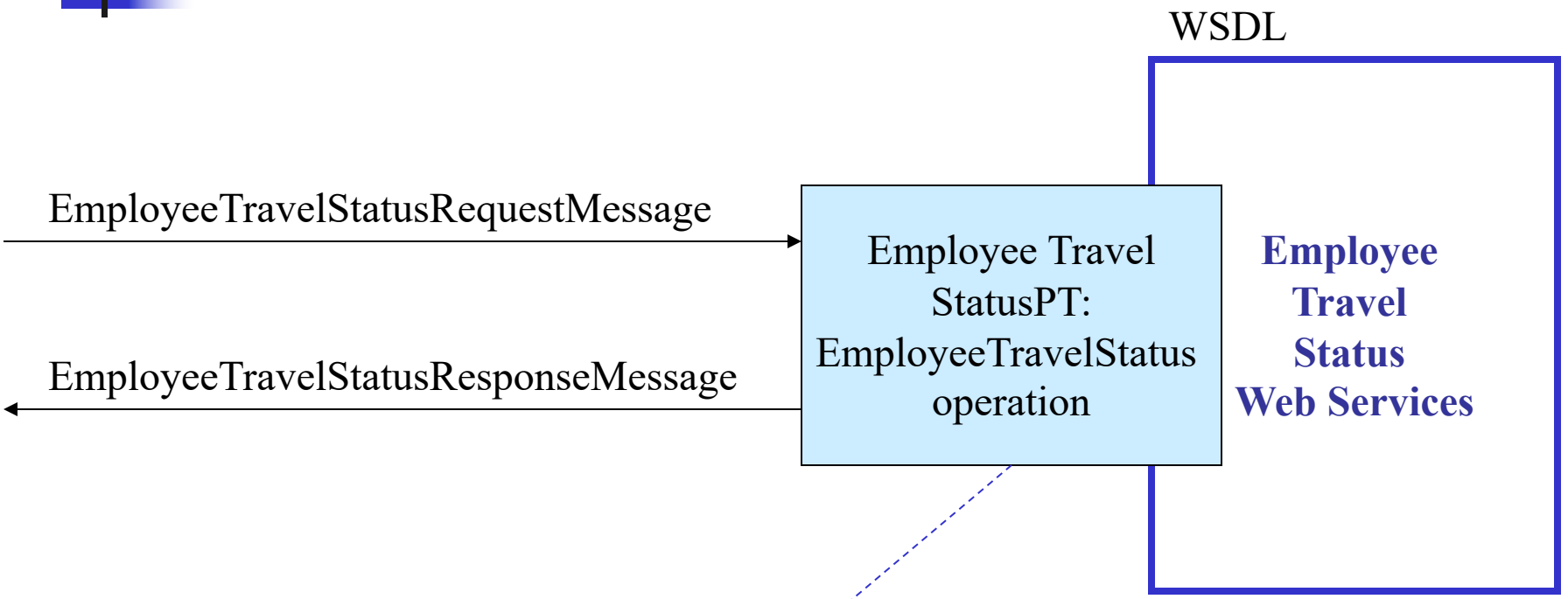
WSDL File

```
<wsdl:portType name="airlinesService">
  <wsdl:operation name="airlineByIcao">
    <wsdl:input message="tns:airlineByIcao" name="airlineByIcao" wsam:Action="http://v
    wsaw:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlineBy
    <wsdl:output message="tns:airlineByIcaoResponse" name="airlineByIcaoResponse"
    wsam:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlineBy
    wsaw:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlineBy
    <wsdl:fault message="tns:APIException" name="APIException" wsam:Action="http://v1.
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="airlinesByIata">
    <wsdl:input message="tns:airlinesByIata" name="airlinesByIata" wsam:Action="http:/
    wsaw:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlinesBy
    <wsdl:output message="tns:airlinesByIataResponse" name="airlinesByIataResponse"
    wsam:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlinesBy
    wsaw:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlinesBy
    <wsdl:fault message="tns:APIException" name="APIException" wsam:Action="http://v1.
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="airlineByFsCode">
    <wsdl:input message="tns:airlineByFsCode" name="airlineByFsCode" wsam:Action="http
    wsaw:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlineBy
    <wsdl:output message="tns:airlineByFsCodeResponse" name="airlineByFsCodeResponse"
    wsam:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlineBy
    wsaw:Action="http://v1.airlines.cache.conducivetech.com/airlinesService/airlineBy
    <wsdl:fault message="tns:APIException" name="APIException" wsam:Action="http://v1.
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="airlinesByIcao">
```

Case Study Diagram

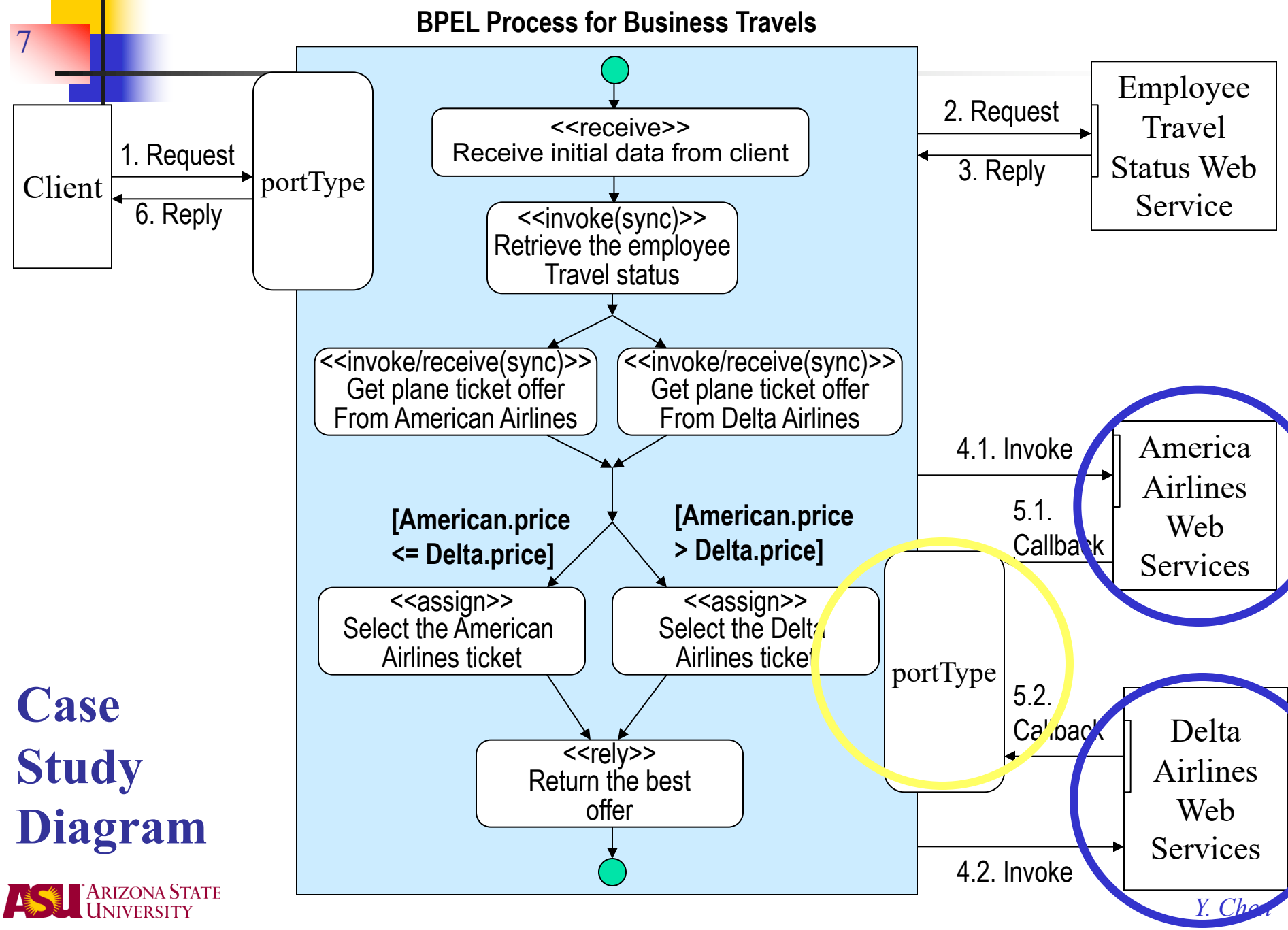


Employee Travel Status WS (Synchronous)



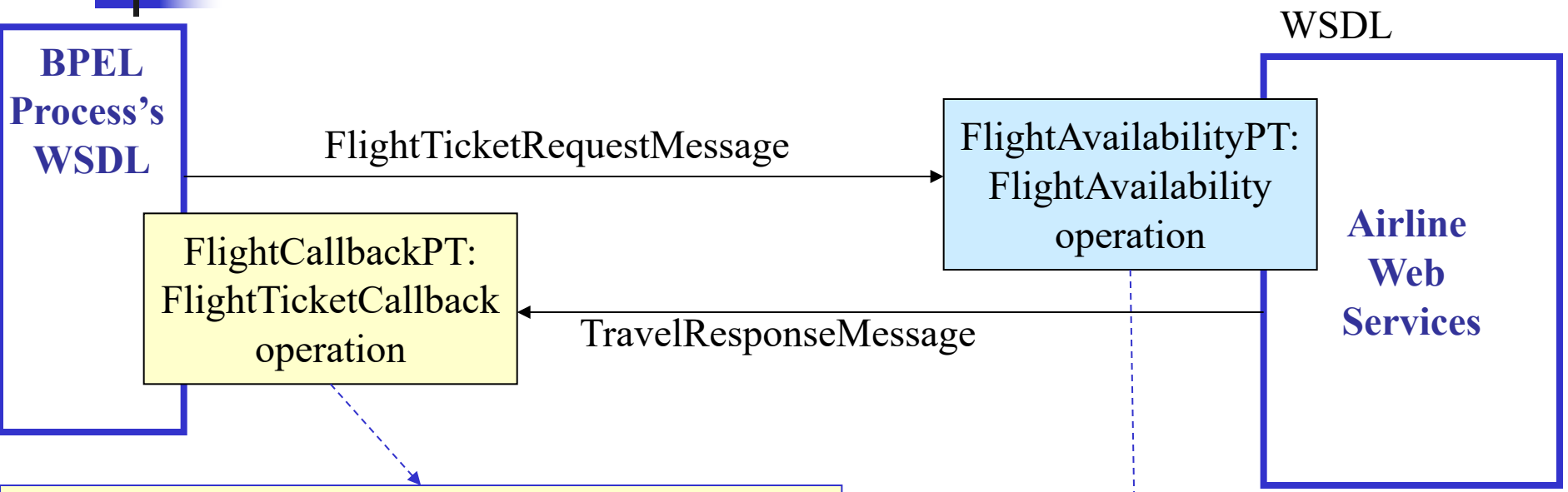
in WSDL

```
<portType name="EmployeeTravelStatusPT">
    <operation name= "EmployeeTravelStatus">
        <input message= "tns : EmployeeTravelStatusRequestMessage" />
        <output message= "tns: EmployeeTravelStatusResponseMessage" />
    </operation>
</portType>
```



Case Study Diagram

Airline Web Services (Asynchronous)



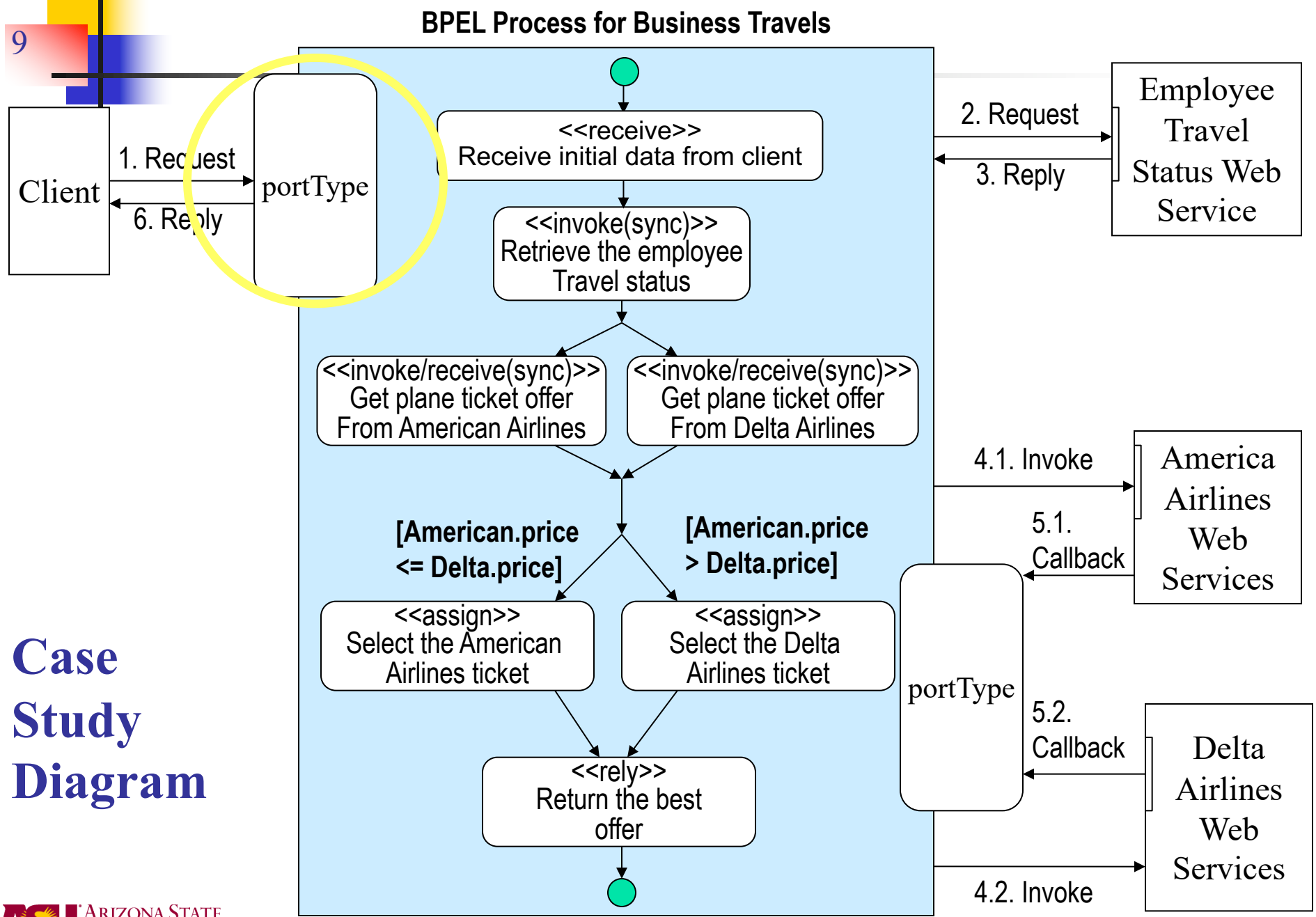

```
<portType name="FlightCallbackPT">
    <operation name= "FlightTicketCallback">
        <input message=
            "tns : TravelResponseMessage"/>
    </operation>
</portType>
```

in WSDL

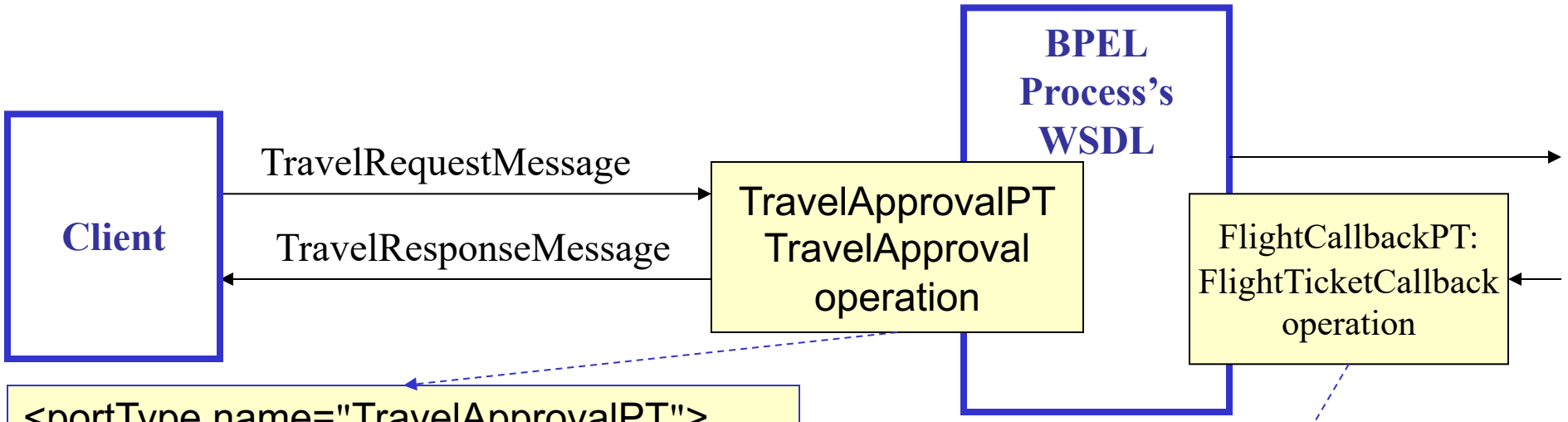
in WSDL

```
<portType name="FlightAvailabilityPT">
    <operation name= "FlightAvailability">
        <input message=
            "tns : FlightTicketRequestMessage"/>
    </operation>
</portType>
```

Case Study Diagram



WSDL of the BPEL Process

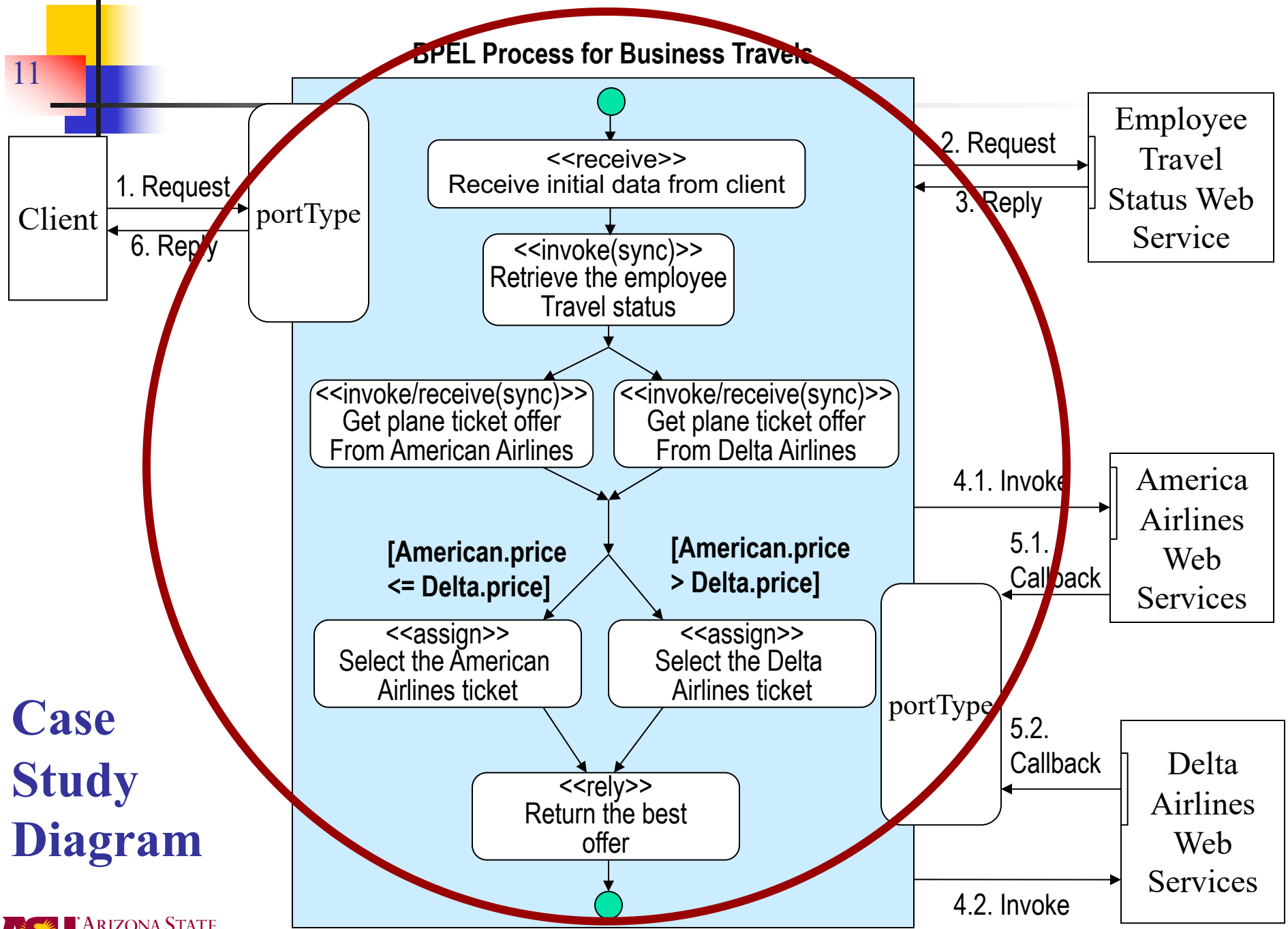



```
<portType name="TravelApprovalPT">
    <operation name= "TravelApproval">
        <input message=
            "tns: TravelRequestMessage"/>
        <output message=
    "tns: TravelResponseMessage"/>
    </operation>
</portType>
```

in WSDL

in WSDL

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



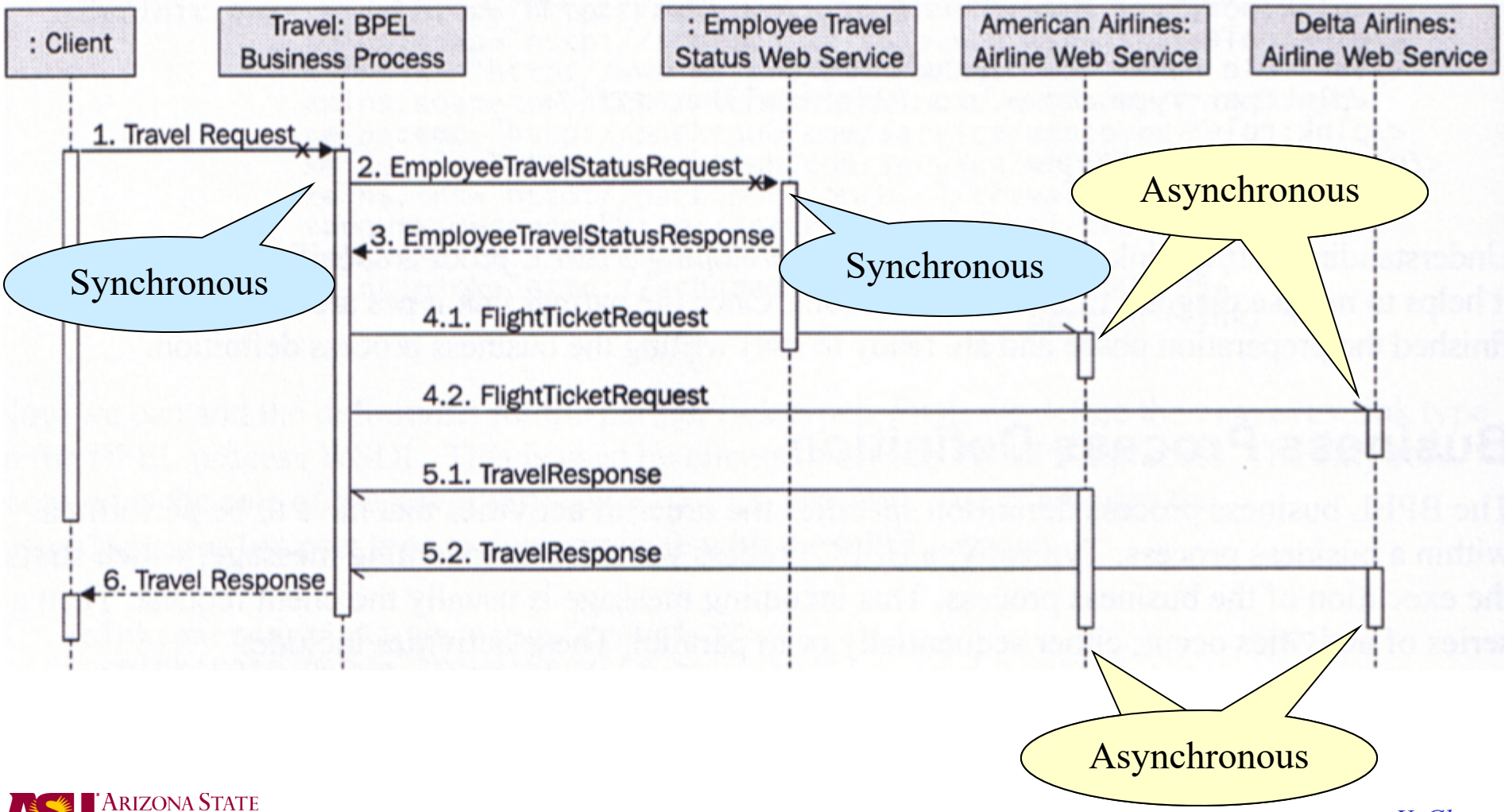
Case Study Diagram

BPEL Process Definition

Workflow-First

- BPEL process definition specifies the workflow / order of activities to be performed within a business process.
- As a WS, a BPEL process typically waits, using <receive> activity, for an incoming message, which is the client request;
- Then a series of activities occur, either sequentially or in parallel. These activities include:
  - <invoke> operations on other web services
  - <receive> results from other web services
  - Conditional branching, which influences the flow of the business process
  - Looping
  - Fault handling
  - Waiting for certain events to occur

Process Diagram with Synchronous Request



Partner Link Types (LT) of the Process

For each partner of the Process, we need to define a partner link type in the BPEL process's WSDL:

- travelLT: used to describe the interaction between the BPEL process and the client. This interaction is synchronous.
- employeeLT: used to describe the interaction between the BPEL process and the Employee Travel Status. This interaction is synchronous too.
- flightLT: used to describe the interaction between the BPEL process and the Airline web services. This interaction is asynchronous; We assume airlines have the same type of services. If different, need different types.

Process's Partner Link Types Definition in Process's WSDL

travelLT (sync)

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

Synchronous: one role

employeeLT (sync)

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

Asynchronous: two roles

flightLT (async)

```
<plnk:partnerLinkType name= "flightLT">
    <plnk:role name= "airlineService">
        <plnk:portType name="tns:flightAvailabilityPT" />
    </plnk:role>
    <plnk:role name= "airlineCustomer">
        <plnk:portType name="tns:flightCallbackPT" />
    </plnk:role>
</plnk:partnerLinkType>
```

BPEL Process Definition Outline

```
<process name="Travel" ... >

    <partnerLinks>
       <!-- The declaration of partner links -->
    </partnerLinks>

    <variables>
       <!-- The declaration of variables -->
    </variables>

   <sequence>
         <!-- The definition of the BPEL process main body -->
    </sequence>

</process>
```

The Root Element "process"

```
<process name="Travel"
    targetNameSpace="http://packtpub.com/bpel/travel/"
    xmlns=http://schemas.xmlSoap.org/ws/2003/03/business-process/
    xmlns:bpws= "http://schemas.xmlsoap.org/ws/2003/03/business-process!"
    xmlns:trv="http://packtpub.com/bpel/travel/"
    xmlns:emp="http://packtpub.com/service/employee/"
    xmlns:aln="http://packtpub.com/Service/airline/" >
```

partnerLinks of the BPEL Process

```
<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>
```

One role

One role

Two roles

Two roles

There is one portType defined between the two partners. Thus, sync communication is applied

There are two portTypes defined between the two partners: One is from BPEL to airlines, and the other is from airlines to BPEL, and thus, async communication will be applied.

Variables

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

The Main Body of the Process

```
<sequence>
    <!-- Receive the initial request for from client -->
1   <receive partnerLink="client"
        portType="trv:TravelApprovalPT"
        operation="TravelApproval"
        variable="TravelRequest"
        createInstance="yes" />
    <!-- Prepare input for the Employee Travel Status WS -->
2   <assign>
        <copy>
            <from variable="TravelRequest" part="employee"/>
            <to variable="EmployeeTravelStatusRequest" part="employee"/>
        </copy>
    </assign>
```

In Main Body: Invoke Employee Travel Status WS

```
<!-- synchronously invoke Employee Travel Status Web Service
  -->
3 <invoke partnerLink="employeeTravelStatus"
    portType="emp:EmployeeTravelStatusPT"
    operation="EmployeeTravelStatus"
    inputvariable="EmployeeTravelStatusRequest"
    outputvariable="EmployeeTravelStatusResponse"/>
```

In Main Body: Prepare the Input

```
<!-- Prepare the input for invoking airlines -->
4 <assign>
    <copy>
4.1     <from variable="TravelRequest" part="flightData"/>
        <to variable="FlightDetails" part="flightData"/>
    </copy>
    <copy>
4.2     <from variable="EmployeeTravelstatusResponse"
                                              part="travelClass"/>
        <to variable="FlightDetails" part="travelClass"/>
    </copy>
</assign>
```

In Main Body: Invoke the Airlines

```
<!--  Make concurrent invocation to AA and DA -->
<flow>  5
    <sequence>
        <!-- Async invoke of the AA WS -->
5.1     <invoke partnerLink="AmericanAirlines"
            portType="aln:FlightAvailabilityPT"
            operation="FlightAvailability"
            inputvariable= "FlightDetails" />
        <receive partnerLink="AmericanAirlines"
            portType="aln:FlightCallbackPT"
            operation="FlightTicketCallback"
            variable="FlightResponseAA" />
    </sequence>

    <sequence>
5.2     <invoke partnerLink="DeltaAirlines"
            portType="aln:FlightAvailabilityPT"
            operation="FlightAvailability"
            inputvariable="FlightDetails" />
        <receive partnerLink="DeltaAirlines"
            portType="aln:FlightCallbackPT"
            operation="FlightTicketCallback"
            variable="FlightResponseDA" />
    </sequence>
</flow>
```

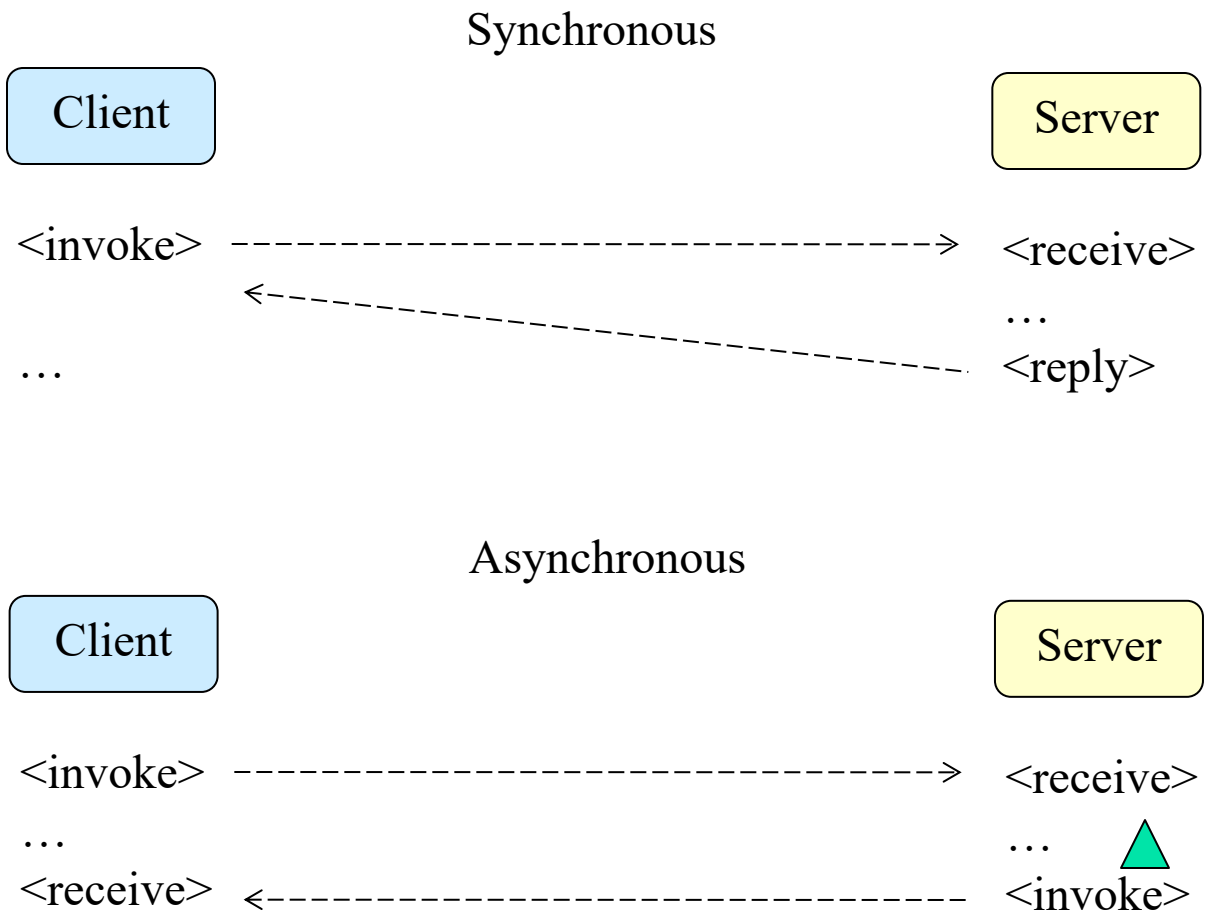
In Main Body: Selection the Best Price

```
<!-- select the best offer and construct the Travel Response -->
<switch> 6
  <case condition="bpws:getVariableData('FlightResponseAA',
                  'confirmationData' , '/confirmationData/aln:Price')
       &lt; = bpws:getVariableData('FlightResponseDA',
                  'confirmationData', '/confirmatlonData/aln:Price')">
    <!-- Select American Airlines -->
    <assign>
      <copy>
        <from variable=
            "FightResponseAA" />
        <to variable=
            "TravelResponse" />
      </copy>
    </assign>
  <otherwise>
    <!-- select Delta Airlines -->
    <assign>
      <copy>
        <from variable=
            "FlightResponseDA" />
        <to variable=
            "TravelResponse" />
      </copy>
    </assign>
  </otherwise>
</switch>
```

In Main Body: Send the Response to the Client

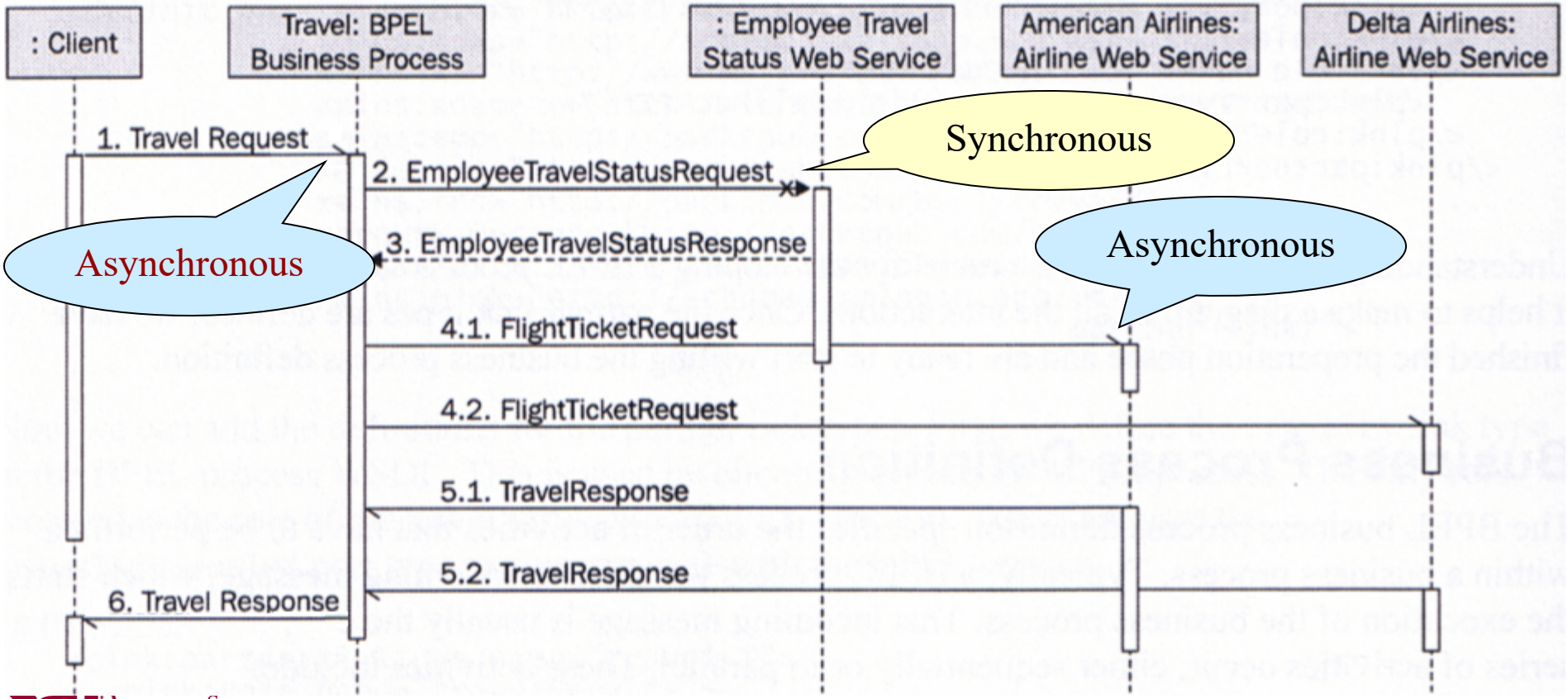
```
    <!-- send a response to the client -->
  7 <reply partnerLink="Client"
           portType="trv:TravelApprovalPT"
           operation="TravelApproval"
           variable="TravelResponse"/>
  </sequence>
</process>
```

Activities forming synchronous and asynchronous two-way communication

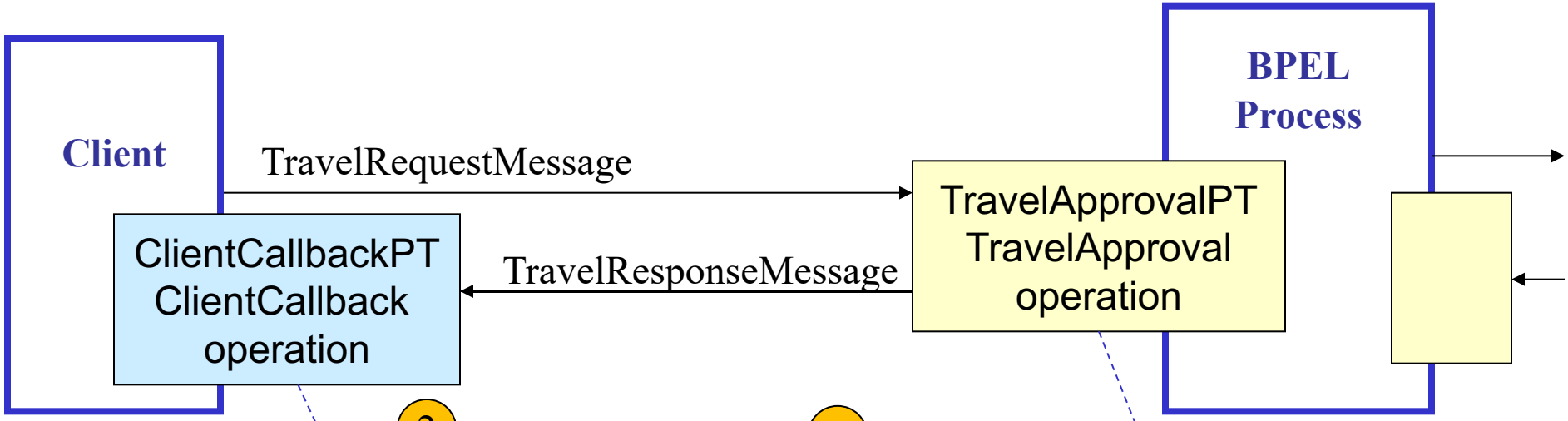


Process Diagram with Asynchronous Request

The **problem** with the combination of synchronous and asynchronous is that the client may have to wait an excessively long period of time. We may want to make the following changes.



Revise WSDL between Client BPEL Process




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

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

Revise the Partner Link Types and Link in WSDL

travelLT

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

3

Add this role

Revise the Process Definition

4

In the Partner link definition

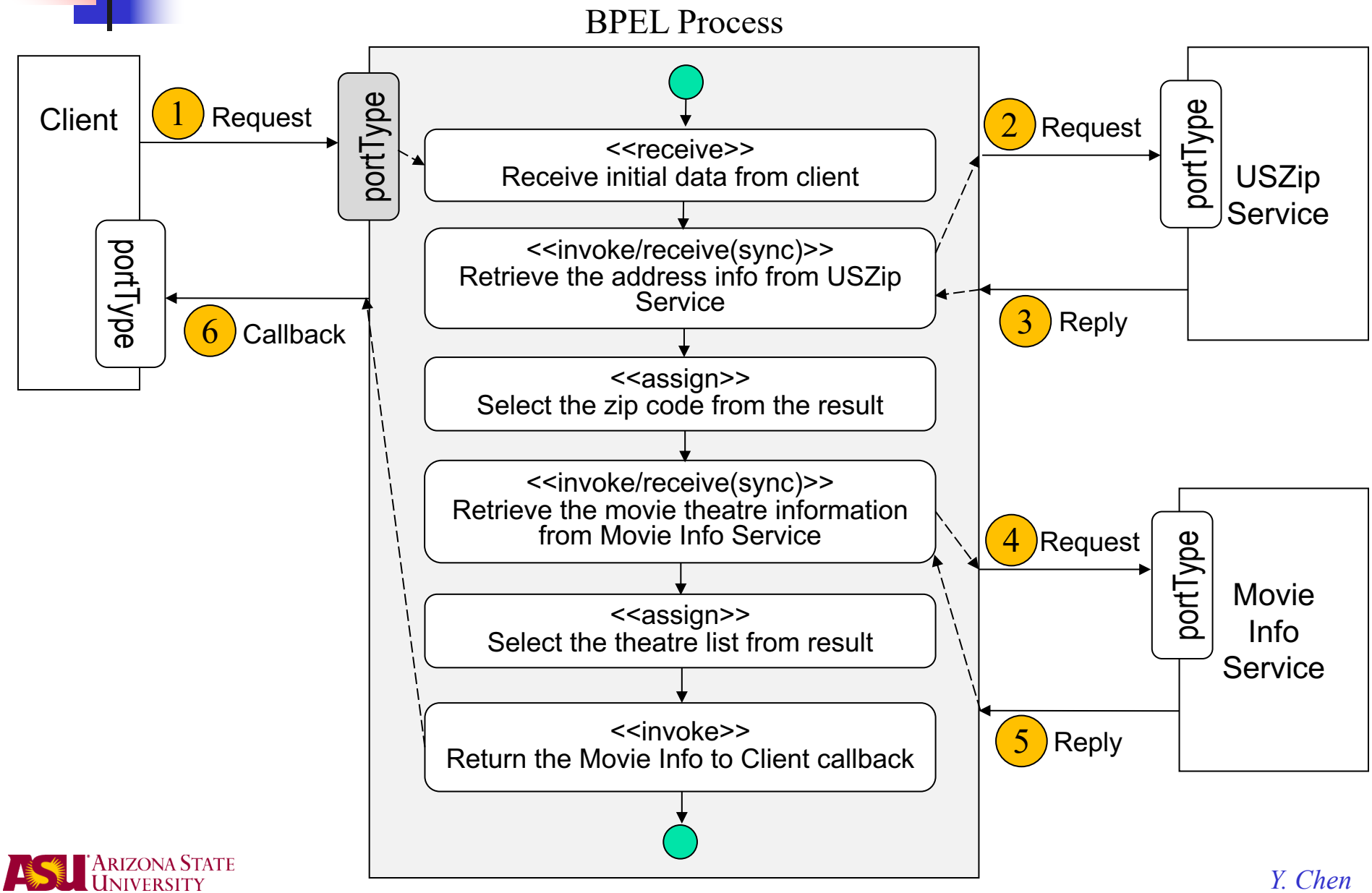
```
<partnerLinks>
    <partnerLink name="client"
        partnerLinkType="trv:travelLT"
        myRole="travelService"/>
        partnerRole = "travelServiceCustomer"/>
    …
</partnerLinks>
```

5

In the processing part

```
    <!– Make a callback to the client -->
        <invoke partnerLinl="client"
            portTypeLink="trv:ClientCallbackPT"
            operation="ClientCallback"
            inputvariable = "TravelResponse"/>
        …
    </ sequence
</process>
```

Invoking Real Web Service: Text 8.2.5



Correlations for Asynchronous Calls

- **Correlations and Stateful Service**
- BPEL Development Frameworks
  - Oracle SOA Suite: BPEL Process Manager and JDeveloper Designer

    http://www.oracle.com/technology/software/tech/webservices/index.html
    http://www.oracle.com/technology/software/products/jdev/index.html
    http://www.oracle.com/technetwork/articles/soa/index-095969.html
- BPMN and Oracle BPM Studio
- Microsoft BizTalk

  http://www.microsoft.com/BizTalk

Structure of a BPEL Process

```
<process ...>

    <partners> ... </partners>
        <!-- Web services the process interacts with -->
    <variables> ... </variables>
        <!– Global variables for storing data used by the process -->
    <correlations> ... </correlations>
        <!– Used to support stateful interactions -->
    <faultHandlers> ... </faultHandlers>
        <!–Alternate execution path to deal with faulty conditions -->
    <compensationHandlers> ... </compensationHandlers>
        <!–Code to execute when “undoing” an action -->
    (activities)*
        <!– What the process actually does -->

</process>
```

Stateless versus Stateful Web Services

- Web services are normally stateless. Each call is considered to be from a different client.
- An asynchronous service can be implemented in two different ways:
  - A one-way call to initiate the service, and a second call to retrieve the result;
  - A one-way call to initiate the service, and a callback to send the result back to the client.
- In both cases, the service needs to correlate the two calls, so that the result is delivered to the right caller, because
  - There can be multiple clients calling the service, and
  - Each client can call the service multiple times.

Different Solutions for Stateful Services

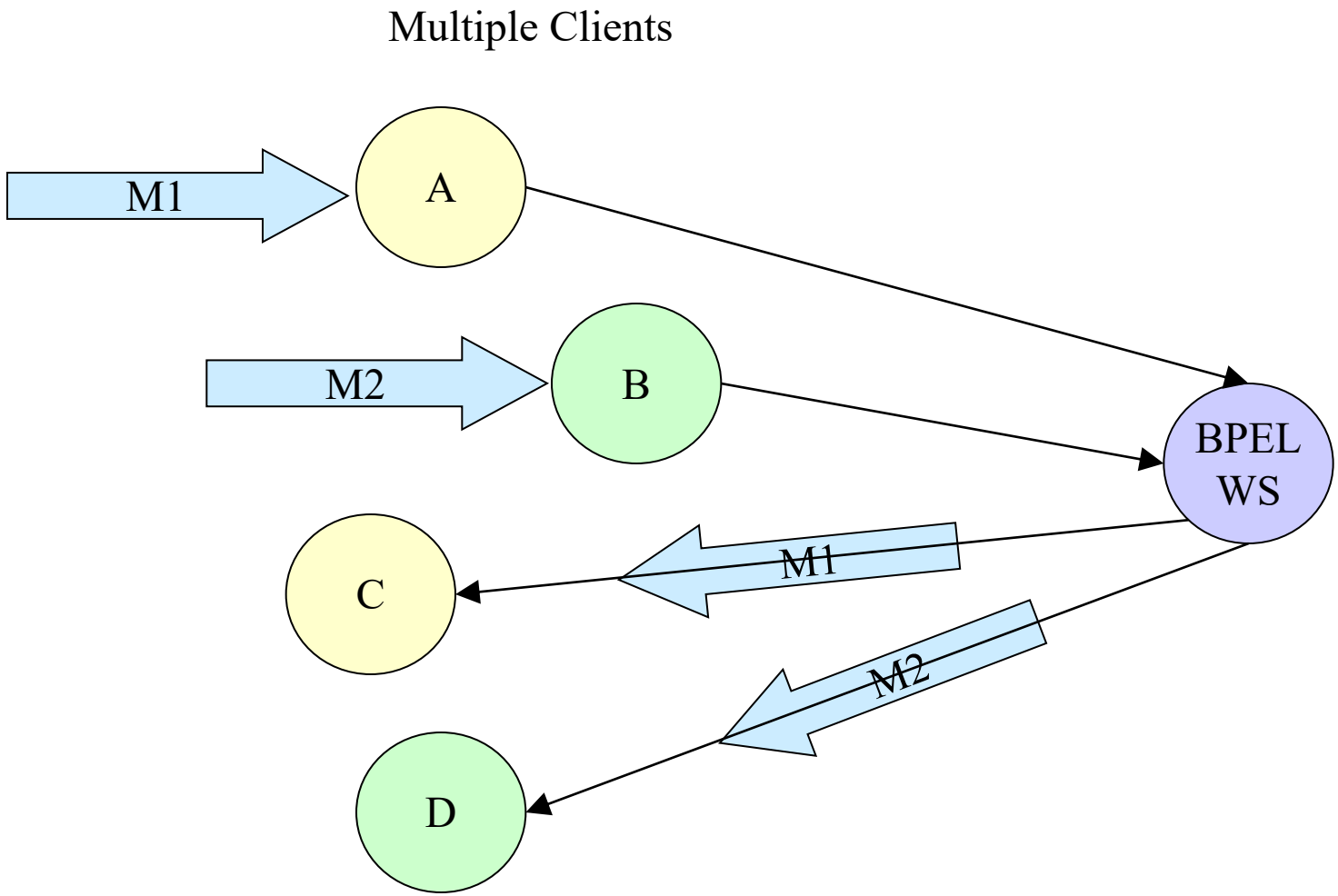
- Windows Communication Foundation (WCF)
  - *PerCall*: one instance per call (stateless)
  - *PerSession*: one instance per session
  - *Single*: one instance per application (singleton)
- BizTalk: singleton instance
- The service supporting singleton instance will need to
  - share certain information among all users, and thus
  - correlate between the requests within the instance created before.
    - Maintain a list of proxies pointing to the instance;
    - Maintain a list of accounts and states related to each user account if re-login is allowed;
    - Recognize the proxy (session information) and user account (login) when an invocation is requested.

A design pattern

BPEL State and Correlation

- BPEL process can model both types of interactions:
  - Simple stateless interactions
  - Stateful, long running (persistent), and asynchronous interactions.
- <correlations> provides support for the latter:
  - <correlations> represents the data that is used for maintaining the state of the interaction (a "conversation").
  - At the process side of the interaction, <correlations> allows incoming messages to reach the right process instance.
- What are correlations?
  - A set of shared business data fields that capture the state of the interaction ("correlating business data"). For example: a "purchase order number", a "customer id", etc.;
  - Each correlation is initialized once;
  - Its values do not change in the course of the interaction.

Multiple Correlations



Defining Correlations

```
<correlation set="..." properties="..."/>
```

- A correlation defines a named set of properties, which are defined as WSDL extensibility elements.
- A property is “mapped” to a field in a WSDL message type.
- The property can thus be found in the messages actually exchanged.
- Typically, a property will be mapped to several different message types and carried on in many interactions, across operations and portTypes

Use Correlations in receive and invoke

```
<receive partner= "client" operation="..."
                   portType="ClientPT" container="...">
  <correlations>
    <correlation set = "Order" initiate = "yes"/>
  </correlations>
</receive>
<invoke partnerLink="Client" portType="ClientPT"
        operation="PurchaseResponse" inputVariable="PO">
  <correlations>
    <correlation set="Order" initiate="no" />
  </correlations>
</invoke>
```

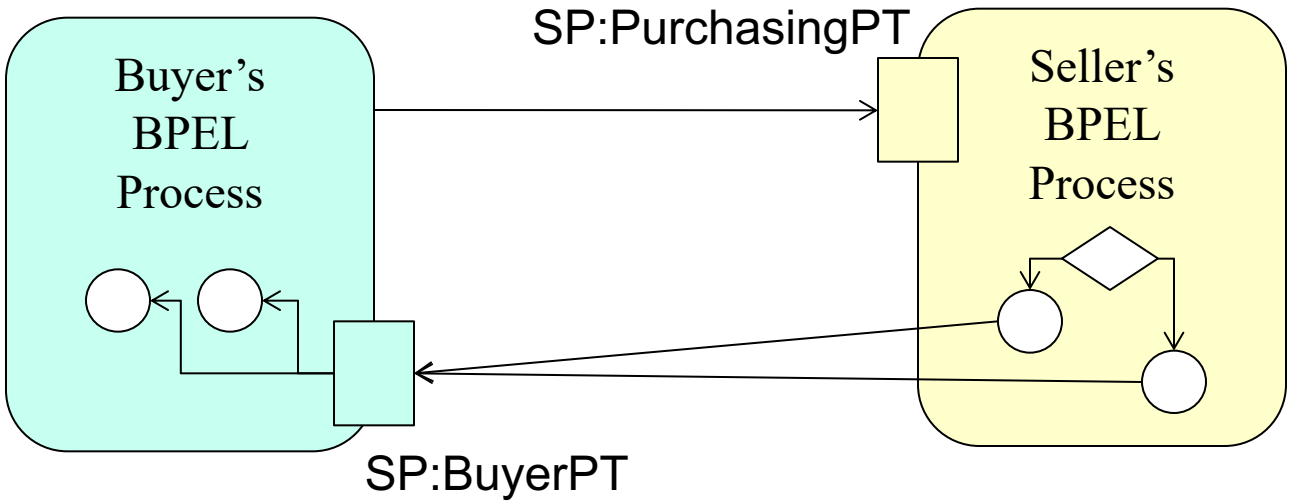
- An input or output operation identifies which correlation applies to the messages received or sent.
- That correlation will be used to assure that the message is related to the appropriate stateful interaction.

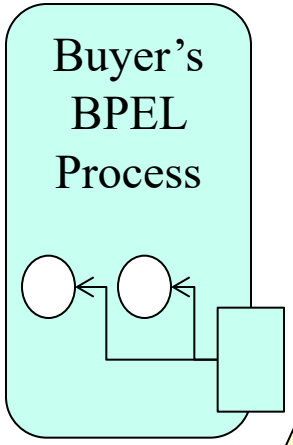
Correlation "initiate" Attribute Values

- When the initiate attribute is set to "yes":
  - The related activity must attempt to initiate the correlation element.
  - If the correlation element is already initiated, an error bpel:correlationViolation must be thrown.
- When the initiate attribute is set to "join":
  - The related activity must attempt to initiate the correlation element, if the correlation element is not yet initiated.
  - If the correlation element is already initiated and the correlation consistency constraint is violated, an error bpel:correlationViolation must be thrown.
- When the initiate attribute is set to "no" or is not explicitly set, the related activity must not attempt to initiate the correlation element.
  - If the correlation element has not been previously initiated, an error bpel:correlationViolation must be thrown.
  - If the correlation element is already initiated and the correlation consistency constraint is violated, an error bpel:correlationViolation MUST be thrown

Correlation Example

http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

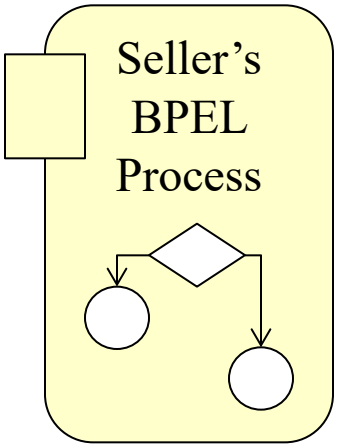





```
<invoke partnerLink="Seller" portType="SP:PurchasingPT"
                                operation="PurchaseRequest" variable="PO">
    <correlations>
        <correlation set="PurchaseOrder" initiate="yes" />
    </correlations>
</invoke>
...
<pick>
    <onMessage partnerLink="Seller" portType="SP:BuyerPT"
                operation="PurchaseResponse" variable="POResponse">
        <correlations>
            <correlation set="PurchaseOrder" initiate="no" />
            <correlation set="Invoice" initiate="yes" />
        </correlations>
        ...  <!-- handle the response message -->
    </onMessage>
    <onMessage partnerLink="Seller" portType="SP:BuyerPT"
                operation="PurchaseReject" variable="POReject">
        <correlations>
            <correlation set="PurchaseOrder" initiate="no" />
        </correlations>
        ...   <!-- handle the reject message -->
    </onMessage>
</pick>
```

<pick> executes the activity associated with first arrival of "onMessage": event-driven

Correlation Element Example (contd.)




```
<receive partnerLink="Buyer" portType="SP:PurchasingPT"
        operation="PurchaseRequest" variable="PO">
    <correlations>
        <correlation set="PurchaseOrder" initiate="yes" />
    </correlations>
</receive>
<! -- ...  Process … switch … -->
<invoke partnerLink="Buyer" portType="SP:BuyerPT"
    operation="PurchaseResponse" inputVariable="POResponse">
    <correlations>
        <correlation set="PurchaseOrder" initiate="no" />
        <correlation set="Invoice" initiate="yes" />
    </correlations>
</invoke>
```

The 1st property is used to correlate activities

```
<invoke partnerLink="Buyer" portType="SP:BuyerPT"
        operation="PurchaseReject" inputVariable="POReject">
    <correlations>
        <correlation set="PurchaseOrder" initiate="no" />
    </correlations>
</invoke>
```

OASIS WS-BPEL 2.0

http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

- <receive>
- <reply>
- <invoke>
- <assign>
- <throw>
- <exit>
- <wait>
- <empty>
- <sequence>
- <if>
- <while>
- <repeatUntil>
- <forEach>
- <pick>
- <flow>
- <scope>
- <compensate>
- <compensateScope>
- <rethrow>
- <validate>
- <extensionActivity>

Summary of BPEL

- Overview of Application/WS Composition languages
- BPEL process defines the order of activities to be performed;
- BPEL process defines a composite Web service;
- BPEL process has a service interface, which uses WSDL with extensions
- Case Study
- Correlation

<table>
<tr><td>BPEL Process
(workflow)</td><td rowspan="2">BPEL WSDL
(interface)</td><td rowspan="2">BPEL-Defined
WSDL Extension</td><td>partnerLinkType</td></tr>
<tr><td>correlations</td></tr>
</table>

Case Study Examples and Tutorials

- References
  - In textbook section 8.2.5, an example using real Web services (zip service and movie service), is given.
  - Oracle: SOA Suite Integration: Part 1: Building a Web Service: https://blogs.oracle.com/theshortenspot/soa-suite-integration:-part-1:-building-a-web-service
  - Oracle: SOA Suite Integration: Part 2: A basic BPEL process: https://blogs.oracle.com/theshortenspot/soa-suite-integration:-part-2:-a-basic-bpel-process
  - Salesforce using SOA Suite: https://www.youtube.com/watch?v=W870Zm9J_9s
- YouTube Videos
  - https://www.youtube.com/watch?v=BAc7aTZCq7U
  - https://www.youtube.com/watch?v=Uq6qS454hdU
  - https://www.youtube.com/watch?v=qCxdmsM_FHs