Software Service Engineering

Lecture 4:

Service Modeling

Doctor Guangyu Gao

Some contents and notes selected from “Service Oriented Architecture” by Michael McCarthy
1. Place in Service Lifecycle

- Service Innovation
  - Service Design
  - Value Proposition
- Service Optimization
  - Service Execution
  - Value Design
- Value Optimization

Gap 1: Value Optimization
Gap 2: Value Optimization

Value Co-production
Content

✓ Concept of Service Model

✓ Service Modeling Language

✓ Service Modeling Method
Content

✓ Concept of Service Model

✓ Service Modeling Language

✓ Service Modeling Method
1) Service Modeling Language

- **UML**: Unified Modeling Language
- **BPMN**: Business Process Modeling Notation
- **Service Blueprint**
- **WSDL**: Web Services Description Language
- **BPEL**: Business Process Execution Language
- **WS-CDL**: Web Service Choreography Description Language
1) Service Modeling Language

- **UML**: Unified Modeling Language
- **BPMN**: Business Process Modeling Notation
- **Service Blueprint**
- **WSDL**: Web Services Description Language
- **BPEL**: Business Process Execution Language
- **WS-CDL**: Web Service Choreography Description Language
4. Business Process Execution Language

- Web Service Composition Methods - **Orchestration**

- A central process takes control over the involved web services and coordinates the execution of different operations on the web service involved in the operation.
- Involved web services do not (and do not need to) know that they are involved into a composition and they are a part of a higher business process.
- Only the central coordinator knows that.
- Centralized with explicit definitions of operations and the order of invocation of web services.
4. Business Process Execution Language

- Web Service Composition Methods - **Choreography**
  - Does not rely on a central coordinator.
  - Each web service involved in the choreography knows exactly when to execute its operations and whom to interact with.
  - A collaborative effort focused on exchange of messages.
  - All participants of the choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges.
  - A peer-to-peer approach.
4. Business Process Execution Language

- Composing web services to execute business processes
  
  - Orchestration is the more flexible approach compared to choreography:
    - We know exactly who is responsible for the execution of the whole business process.
    - We can incorporate web services, even those that are not aware that they are a part of a business process.
    - We can also provide alternative scenarios when faults occur.
  
  - BPEL follows the orchestration paradigm.

- Choreography is covered by other standards, such as WSCI (Web Services choreography Interface) and WS-CDL (Web Services Choreography Description Language).

- Choreography has not gained support from the industry which would be comparable to BPEL.
4. Business Process Execution Language

- **BPEL**: Business Process Execution Language, service composition or service orchestration oriented service modeling language.

- Structured programming language, using **while, if else, sequence, flow**, ... programming in the large.

- Processing logic to handle synchronous and asynchronous messages.

- Quite different from programming in the small - different issues to deal with.
4. Business Process Execution Language

- Developing web services and exposing functionalities: not sufficient.
- Pay attention to: How to use existed service to construct new services.
- **Orchestration**: composing multiple smaller-grained web services to a executed business process in terms of *specific business logic*.
- A larger-grained composed web.
- Composition $\approx$ Orchestration.
4. Business Process Execution Language

- **BPEL** builds on top of XML and web services.
- An XML-based language which supports the web services technology stack, including SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination and WS-Transaction.
4. Business Process Execution Language

- **BPEL** builds on top of XML and web services.
- An XML-based language which supports the web services technology stack, including SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination and WS-Transaction.
4. Business Process Execution Language

- A typical BPEL process
  - First, the BPEL business process receives a request.
  - To fulfill it, the process then invokes the involved web services and finally responds to the original caller.
  - Since the BPEL process communicates with other web services, it relies heavily on the WSDL description of the web services invoked by the composite web service.
4. Business Process Execution Language

- Steps in a Process
  - Each step is called an **activity**.
  - BPEL supports **primitive and structure activities**.
  - Primitive activities represent basic constructs and are used for common tasks.
4. Business Process Execution Language

- **Primitive Activities**
  - Invoking other web services, using `<invoke>`.
  - Waiting for the client to invoke the business process through 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>` etc.
4. Business Process Execution Language

- Defining Processes
  
  ● Combine these and other *primitive activities* and define complex algorithms, which exactly specify the steps of business processes.
  
  ● To combine primitive activities BPEL supports several structured activities
4. Business Process Execution Language

- Structured Activities
  - Sequence (<sequence>), which allows us to define 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
  - Case-switch construct (<switch>) for implementing branches
  - While (<while>) for defining loops
  - The ability to select one of a number of alternative paths, using <pick>
4. Business Process Execution Language

Definitions and Declarations

- Typically declare variables using `<variable>`
- Define partner links using `<partnerLink>`
- A BPEL process can be synchronous or asynchronous.
  - A synchronous BPEL process blocks the client (the one which is using the process) until the process finishes and returns a result to the client.
  - An asynchronous process does not block the client. Rather it uses a callback to return the result (if any)
4. Business Process Execution Language

Example Process

- For its clients a BPEL process looks like any other web service.
- When define a BPEL process, actually define a new web service that is a composition 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, have to invoke the resulting composite web service.
4. Business Process Execution Language

- Example Process

Figure: Example BPEL process
Typical Structure (1)

```xml
<process>
  <partnerLinks>
    <partnerLink>
    </partnerLink>
    <partnerLink>
  </partnerLinks>
  <variables>
    <variable>
    </variable>
    <variable>
  </variable>
  <sequence>
    <receive>
    </receive>
  </sequence>
</process>
```

the initial client request
4. Business Process Execution Language

- **Partner Links**
  - BPEL calls the links to all parties it interacts with as partner links.
  - Partner links can be links to web services that are invoked by the BPEL process.
  - Partner links can also be links to clients which invoke the BPEL process.
  - Each BPEL process has *at least one* client partner link, because there has to be a client that invokes the BPEL process.
4. Business Process Execution Language

- Typical Structure (2)

```xml
<flow>
  make calls in parallel
  <invoke> a web service </invoke>
  <invoke> another web service </invoke>
</flow>
```
4. Business Process Execution Language

- Typical Structure (3)

```xml
<switch>
  make decisions
  <case condition =“…”>
    <assign>
      <copy>
        <from>…<to>
      </copy>
    </assign>
  </case>
  <otherwise>
    <assign>
      <copy>
        <from>…<to>
      </copy>
    </assign>
  </otherwise>
</switch>
<reply> reply to synchronous caller
</reply>
</sequence>
</process>
```
4. Business Process Execution Language

- Sequential Order of Activities

```xml
<process>
  ....
  <sequence>
    <receive>
    <invoke>
    <assign>
    <invoke>
    <receive>
    <invoke>
  </sequence>
</process>
```

Do activities in sequential order.
4. Business Process Execution Language

- **Parallel Activities**

```xml
<process>
  ....
  <sequence>
    <receive>
    <flow>
      <invoke>
      <invoke>
      <invoke>
    </flow>
    </sequence>
  </process>
```

- Wait to start process
- The three invokes are carried out in parallel.
4. Business Process Execution Language

- Parallel Sequences

  ```xml
  <process> ....
  <sequence>
    <receive>
    <flow>
      <sequence>
        <invoke>
        <invoke>
      </sequence>
      <sequence>
        <invoke>
        <invoke>
      </sequence>
    </flow>
  </sequence>
  <sequence>
  </sequence>
  </process>
  
  Wait to start process.
  Both sequences may run in parallel.
  These two ‘invokes’ go in order.
  These two ‘invokes’ go in order
  ```
4. Business Process Execution Language

- **Synchronous Web Services**
  - The sender blocks and waits for a reply.
  - The service should run fast.
  - The `<receive>` and `<reply>` form a pair on B.

B is a synchronous web service and uses a BPEL reply.
4. Business Process Execution Language

Quiz on Synchronous Web Services

- What does A need to know about B? In other words, what is required in B’s WSDL?

B is a synchronous web service and uses a BPEL reply.
Quiz on Synchronous Web Services

- What does A need to know about B? In other words, what is required in B’s WSDL?
- A needs to know the message types and the available operations as well as B’s location.

B is a synchronous web service and uses a BPEL reply.
4. Business Process Execution Language

- Asynchronous Web Services
  - Most real-world processes are long running and
    - if callbacks are needed, message correlation may be used.
    - If callbacks are not needed, B need not perform an `<invoke>`.

B is an asynchronous web service and replies with an optional “invoke” not a “reply”.
Quiz on Asynchronous Web Services

- What does A need to know in order to use B? In other words, what information must be available in B’s WSDL?

B is an asynchronous web service and replies with an optional “invoke” not a “reply”.

4. Business Process Execution Language
Quiz on Asynchronous Web Services

- What does A need to know in order to use B? In other words, what information must be available in B’s WSDL?
- A needs to know the message types and the available operations as well as B’s location. In order to use B, A must also know exactly what operations it needs to provide and what messages will be received.

B is an asynchronous web service and replies with an optional “invoke” not a “reply”. 

4. Business Process Execution Language

```xml
<receive>
Do other things...
</receive>

<invoke>
B
</invoke>

<invoke>..<invoke>
</invoke>
```

```xml
A

<receive>
</receive>

<invoke>
</invoke>
```

B is an asynchronous web service and replies with an optional “invoke” not a “reply”.
4. Business Process Execution Language

- Example Business Process
  - Collect employee information (name, id, travel plans, etc.).
  - Determine an employee’s flying status (first class or coach) and then determine the cheaper of two airlines.
  - Return suggested flight to the employee.
4. Business Process Execution Language

- Modified Example from Juric Text

Asynchronous web service

<invoke>

<receive>

<invoke>

<invoke>

<receive>

<invoke>

<invoke>

<invoke>

Coach or first class

price

price

price

Employee Travel Status WS synchronous

American Airlines WS asynchronous

Delta Airlines WS asynchronous

Asynch Process for Business Travels
4. Business Process Execution Language

- Partner Links
  - Partner links describe links to partners.
  - Partners might be:
    - (1) Services that invoke the BPEL process.
    - (2) Services invoked by the BPEL process.
    - (3) Services that play both roles - the BPEL process invokes the service and the service invokes a callback on the BPEL process.
PartnerLinkTypes

- PartnerLinkTypes represent interactions between the parties.

- We have three **types** of interactions in the airline example:
  - (1) The client interacts with the BPEL process.
  - (2) The BPEL process calls the employee status web service.
  - (3) The BPEL process calls the two airline web services and expects callbacks from both.
4. Business Process Execution Language

- **PartnerLinkTypes**
  - Within the BPEL process WSDL, we have two roles defined for one of the links:

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

The interface of the BPEL service is implemented at the service.

The interface of the client callback is implemented on the client.
4. Business Process Execution Language

- Modified Example from Juric Text

Asynchronous web service

Asynch Process for Business Travels

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

<invoke>

Coach or first class

Employee Travel Status WS synchronous

American Airlines WS asynchronous

Delta Airlines WS asynchronous

price

price

price
4. Business Process Execution Language

- **PartnerLinkTypes**
  - The employee status WS is synchronous so within the employee status WS WSDL we have **one role** defined:

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

**employeeLT Employee Link Type**

Interface of employee status web service.
4. Business Process Execution Language

- Modified Example from Juric Text

Asynch Process for Business Travels

<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
<invoke>
4. Business Process Execution Language

- **PartnerLinkTypes**

  - The airline web services are asynchronous and so within the airline WS WSDL’s we have two roles defined:

    ```xml
    <partnerLinkType name="flightLT">
        <role name="airlineService">
            <portType name="tns:FlightAvailabilityPT" />
        </role>
        <role name="airlineCustomer">
            <portType name="tns:FlightCallbackPT" />
        </role>
    </partnerLinkType>
    ```
4. Business Process Execution Language

 Modiﬁed Example from Juric Text

Asynch Process for Business Travels

Employee Travel Status WS synchronous

American Airlines WS asynchronous

Delta Airlines WS asynchronous

Asynchronous web service
4. Business Process Execution Language

- **PartnerLinks in the BPEL (1)**

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

  <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>
```
4. Business Process Execution Language

- **PartnerLinks in the BPEL (2)**

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

These names are defined in the partner link type section.

This partner link is of type `travelLT`. So, two interfaces are involved. This process is the travel service part. The partner implements the client callback interface.
4. Business Process Execution Language

- **PartnerLinks in the BPEL (3)**
  
  ```xml
  <partnerLinks>
  
  <partnerLink name="employeeTravelStatus"
   partnerLinkType="emp:employeeLT"
   partnerRole="employeeTravelStatusService"/>
  
  </partnerLinks>
  ```

  This partner link is of type **employeeLT**. So, one interface is involved, that is, the interface of the employee status web service.
Partners in the BPEL (4)

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

Both of these partner links are of type `flightLT`. As such, two interfaces are mentioned. The role of this process is to provide the callback (FlightCallbackPT) and the role the partner is to provide the FlightAvailabilityPT.
4. Business Process Execution Language

- PartnerLinks in the BPEL (5)

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

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

The interface of the BPEL service is implemented at the service.

The interface of the client callback is implemented on the client.
4. Business Process Execution Language

- PartnerLinks in the BPEL (6)

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

```xml
<partnerLinkType name="employeeLT">
    <role name="employeeTravelStatusService">
        <portType name="tns:EmployeeTravelStatusPT" />
    </role>
</partnerLinkType>
```
4. Business Process Execution Language

- PartnerLinks in the BPEL (7)

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

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

<partnerLinkType name="flightLT">
    <role name="airlineService">
        <portType name="tns:FlightAvailabilityPT" />
    </role>

    <role name="airlineCustomer">
        <portType name="tns:FlightCallbackPT" />
    </role>
</partnerLinkType>
```
4. Business Process Execution Language

- **Variables in BPEL (1)**
  
  ```xml
  <variables>
    <variable name="TravelRequest"
      messageType="trv:TravelRequestMessage"/>

    <variable name="EmployeeTravelStatusRequest"
      messageType="emp:EmployeeTravelStatusRequestMessage"/>

    <variable name="EmployeeTravelStatusResponse"
      messageType="emp:EmployeeTravelStatusResponseMessage"/>
  </variables>
  ```
4. Business Process Execution Language

- Variables in BPEL (2)

  <!-- input for American and Delta Web 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>
4. Business Process Execution Language

- BPEL Main Process (1)

  <sequence>

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


4. Business Process Execution Language

- **BPEL Main Process (2)**

```xml
<!-- Prepare the input for the Employee Travel Status Web Service -->
<assign>
  <copy>
    <from variable="TravelRequest" part="employee"/>
    <to variable="EmployeeTravelStatusRequest" part="employee"/>
  </copy>
</assign>
```
4. Business Process Execution Language

- **BPEL Main Process (3)**

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

<!-- Synchronously invoke the Employee Travel Status Web Service -->
4. Business Process Execution Language

- **BPEL Main Process (4)**

```xml
<!-- Prepare the input for the airlines. The input comes from two variables. -->
<assign>
 <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>
```
4. Business Process Execution Language

- **BPEL Main Process (5)**

```xml
<!-- Make a concurrent invocation on both airlines. -->
<flow>

<sequence>
  <!-- Async invoke of the AA Web service and wait for the callback-->
  <invoke partnerLink="AmericanAirlines"
     portType="aln:FlightAvailabilityPT"
     operation="FlightAvailability"
     inputVariable="FlightDetails" />

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

</sequence>
```

The receive operation must occur after the invoke. Hence, the sequence tag is used.
4. Business Process Execution Language

- BPEL Main Process (6)

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

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

<receive partnerLink="DeltaAirlines"
    portType="aln:FlightCallbackPT"
    operation="FlightTicketCallback"
    variable="FlightResponseDA" />

</sequence>

Only the flow is done in parallel. For the sequence to complete, the airline must respond.
```
4. Business Process Execution Language

- **BPEL Main Process (7)**

<!-- The airlines have responded. Select the best offer and construct the TravelResponse -->

```xml
<switch>
  <case condition="bpws:getVariableData('FlightResponseAA', 'confirmationData','/confirmationData/Price') <= bpws:getVariableData('FlightResponseDA', 'confirmationData','/confirmationData/Price')">
    <!-- Select American Airlines -->
    <assign>
      <copy>
        <from variable="FlightResponseAA" />
        <to variable="TravelResponse" />
      </copy>
    </assign>
  </case>
</switch>
```
4. Business Process Execution Language

- BPEL Main Process (8)

```xml
<otherwise>
  <!-- Select Delta Airlines -->
  <assign>
    <copy>
      <from variable="FlightResponseDA" />
      <to variable="TravelResponse" />
    </copy>
  </assign>
</otherwise>
</switch>
```
4. Business Process Execution Language

- BPEL Main Process (9)

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

</process>
```
4. Business Process Execution Language

- Sketch of Working Process

sequence
receive information from employee
assign assign to variable
invoke invoke service to determine flying status
assign assign result to variable
flow Do sequences in parallel
  sequence
    invoke call airline A
    receive get price for ticket
  sequence
    invoke call airline B
    receive get price for ticket
  switch select cheaper flight
  invoke inform the employee
Would this work?

sequence
receive
flow
assign
invoke
assign
invoke
receive
invoke
receive
switch
invoke

No. The previous slide shows its right version. Here, we have not expressed the synchronization dependencies between activities.

However, BPEL provides for more complex concurrency scenarios using links. A single link is specified with a source and a target.
4. Business Process Execution Language

- We Need To Add Links

sequence
receive
flow
assign
invoke
assign
invoke
assign
receive
invoke
receive
switch
invoke

assign before the invoke
invoke before the assign
assign before the two invokes
invoke before receive
receive before the switch
invoke before receive
receive before the switch
receive before the switch
4. Business Process Execution Language

- Sources and Targets In BPEL

```xml
<sequence>
  <receive>
  <flow>
    <assign>...
      <source linkName = "A"/>
    </assign>
  </flow>
  <invoke>....
    <target linkName = "A"/>
    <source linkName = "B"/>
  </invoke>
  <assign>
    <target linkName = "B"/>
    <source linkName = "C"/>
    <source linkName = "D"/>
  </assign>
</sequence>
```

Assign before invoke.
Invoke before assign
Assign before the two invokes.
Link names are user defined and should be well chosen.
4. Business Process Execution Language

- Sources and Targets In BPEL

```xml
<invoke>
  <target linkName = “C”/>
  <source linkName = “E”/>
</invoke>

<receive>
  <target linkName = “E”/>
  <source linkName = “G”/>
</receive>

<invoke>
  <target linkName = “D”/>
  <source linkName = “F”/>
</invoke>
```
4. Business Process Execution Language

- Sources and Targets In BPEL

```xml
<receive>
  <target linkName = "F"/>
  <source linkName = "H"/>
</receive>
<switch>
  <target linkName = "G"/>
  <target linkName = "H"/>
</switch>
</flow>
<invoke>
```
Concurrency and Links

- The flow tag provides the ability to express synchronization dependencies between activities.
- In other words, we can specify what happens and when.
- Link definitions are placed within the flow activity. For example,
  
  ```xml
  <flow>
    <links>
      <link name="A"/>
      <link name="B"/>
    </links>
  </flow>
  ```

  - Every link must be associated with exactly one source and target.
  - A link’s target activity may only be performed after the source activity has completed.
  - Transition conditions may be added for additional confusion.
6. WS-CDL

- Service Choreography:
  - It relates to describing externally observable interactions between web services.
  - It is based on collaboration.
  - It tracks the sequence of public message exchanges that occur between multiple sources, including customers, suppliers, partners.
- Choreography does not depend on a centralized controller.
6. WS-CDL

- Choreography: Buyer and Seller

Choreography of Buyer’s & Seller’s Collaboration
6. WS-CDL

- Choreography defines the common observable behavior of two or more participants, focusing on a global, participant agnostic viewpoint; where exchanges of information, with potential information alignment, occur when jointly agreed, information driven reactive rules are satisfied.

- WS-CDL is a language in which a Choreography description is specified.
  - Initially designed by Oracle and then submitted into the W3C Choreography WG in September 2003.
6. WS-CDL

6. WS-CDL

- Modeling a Collaboration Protocol with WS-CDL

**WS-CDL Choreography Description**

**Step-1**
- **PO Request**
  - **Exchange PO**
  - **Receive PO**
  - **Send PO**

**Step-2**
- **PO Acknowledgement**
  - **Exchange PO Ack**
  - **Receive PO Ack**
  - **Send PO Ack**

**Step-3**
- **PO Response**
  - **Exchange PO Resp**
  - **Receive PO Resp**
  - **Send PO Resp**

**Buyer-Participant “A”**
- Transform
- Receive PO Ack
- Receive PO Response
- JAVA logic

**Seller-Participant “B”**
- Receive PO
- Translate
- Send PO Ack
- Receive PO Ack
- Receive PO Resp
- Send PO Resp
- Transform

**WS-CDL Choreography Description**
6. WS-CDL

- Example of Service Choreography
6. WS-CDL

Orchestration vs. Choreography:

**Orchestration**
- Centralized through explicit definitions of operations and the invocation order of Web Services
- A single director in control

**Choreography**
- Tracks the sequence of publicly visible message exchange between multiple services
- Participants has to know exactly when to become active
6. WS-CDL

- **Orchestration vs. Choreography:**

![Diagram of Web Service Orchestration and Web Service Choreography](image-url)
Orchestration vs. Choreography:

- **BPEL**
  - Need centralized control.
  - Recursive service composition.
  - Execution process modeling language.
  - Need to call web service.

- **WS-CDL**
  - Need not centralized control (distributed in different participator).
  - Description language, can not be executed.
  - Need not call web service.
Content

✓ Concept of Service Model

✓ Service Modeling Language

✓ Service Modeling Method
Service Modeling Method

- Basic Service Modeling Process:
  - Service modeling process: process to understand the service system.
  - Differentiate and recognize processed service contents, such as service granularity and level as well as necessity and availability.
  - Naming concept, to represent each service content with definite concept.
  - Using unified representation method to express service concept for convenient communication.
  - Thus, **understand** -> **differentiate** -> **Name** -> **Represent**
Service Modeling Method

- **Two Phases** of Service Modeling Process:
  - Modeling for current service business (AS-IS modeling).
  - Modeling for future service business (TO-BE modeling).

- Represent service system with different view.
  - Start with a specific view, and then construct other views with semantic relationship between different views.
  - Multi-view construction should maintain consistency.

Alternate Modeling Process
Service Modeling Method

- **Top-down service modeling: service decomposition method**
  - equivalence principal:
    - Union of bottom functions/activities equal to top function/activity.
- **SADT**
  - Structured Analysis and Design Technique
Service Modeling Method

- Structured Analysis and Design Technique: **IDEF**
  - ICAM DEFinition Languages.
  - In the 1970’s, IDEF0 originated in the U.S. Air Force under the Integrated Computer Aided Manufacturing (ICAM) program from a well-established graphical language, the Structured Analysis and Design Technique (SADT).
Service Modeling Method

- Structured Analysis and Design Technique: IDEF
Service Modeling Method

- **Structured Analysis and Design Technique**: IDEF
  
  - What activity is being performed?
  - What is being transformed into what (inputs and outputs)?
  - What guides or constrains the activity (controls)?
  - Who or what is performing the activity (mechanism)?
Service Modeling Method

- Structured Analysis and Design Technique: IDEF
Service Modeling Method

- **Structured Analysis and Design Technique:** IDEF

  - Step 1: define service system boundary, drawing top diagram.
  - Step 2: decompose service system function and behavior.
  - Step 3: draw decomposition diagram until particular granularity for each service function and behavior in top diagram.
  - Step 4: draw service flow chat for each smaller-granularity function and behavior.
  - Step 5: recognize correlated service element, and then draw other corresponding model views.
Service Modeling Method

- **Object-oriented Service Modeling Method**
  - Step 1: recognizing object.
  - Step 2: recognizing different object status.
  - Step 3: adding necessary action and conditions between object and its status to realize state transitions.
  - Step 4: recognizing object’s service elements and control elements in each step.
  - Step 5: transferring service elements and control elements into operable service processes and control processes.
  - Step 6: checking model completeness and consistency.
Service Modeling Method

Layered Service Modeling: MDA

- Model Driven Architecture, MDA, partition the model into four layers:
  - Computation Independent Model, CIM
  - Platform Independent Model, PIM
  - Platform Specific Model, PSM
  - Code.
Service Modeling Method

- Service Oriented Modeling and Architecture, SOMA
  - Proposed by IBM.
Service Modeling Method

- Service Oriented Modeling and Architecture, SOMA

  - Basic Ideas for SOMA:
    - Top-down idea, start with business requirement, to construct service business model, and identify, design and realize service and components (to support service) with model-driven.
    - With bottom-up idea, make most of the legacy system, to package functions can be used by service.
    - Using the existed and new designed service, to form orchestration and choreography to support top layer business realization.