Business Processes and Web Services

Workflows and Web Services
Kapitel 10

Business Processes and Web Services

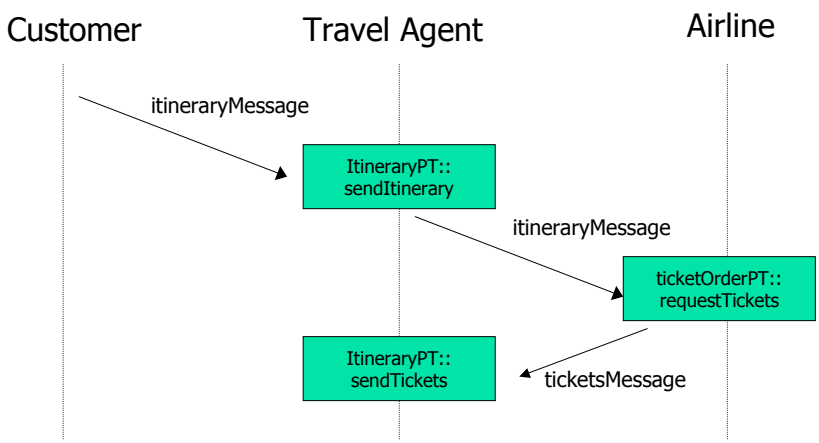
FACHBEREICH INFORMATIK

- Business Process Execution Language for Web Services (BPEL4WS)
  - XML-based workflow definition language
  - Describe business processes that both provide and consume web services
    - Steps (activities)
      - Implemented as an interaction with a web service
    - Information flow into/out of the process
      - Externalized as web service
- Complemented by
  - WS Coordination specification
    - Allows to web services involved in a process to share information that "links" them together
      - Shared coordination context
  - WS Transaction specification
    - Allows to monitor the success/failure of each coordinated activity
      - Reliably cancel the business process
        - Involves compensating activities
- Specifications proposed by IBM, Microsoft, BEA
  - Unifies XLANG, WSFL

BPEL4WS

- Business process defines
  - Potential execution order of operations (web services)
  - Data shared between the web services
  - Partners involved in business process
  - Joint exception handling for collection of web services
- Long running transactions between web services
- BPEL script
  - Fully executable specification of business process
    - Portable between BPEL-conformant environments
  - Supports specification of business protocols between partners

BPEL Example – Flow Diagram



Partners

- Partner definition
  - Specifies the web services mutually used by the partner or process
    - E.g., agents process interacts with customer, airline
  - References a service link type
    - Connects a partner to a process
    - Specifies collections of web services: roles
      - Provided and required by the connected partners
  - Defines role taken by the process itself (myRole) and role that has to be accepted by the partner (partnerRole)

```
1     <process name="ticketOrder">
2     <partners>
3       <partner name="customer"
4                serviceLinkType="agentLink"
5                myRole="agentService"/>
6       <partner name="airline"
7                serviceLinkType="buyerLink"
8                myRole="ticketRequester"
9                partnerRole="ticketService"/>
10    </partners>
```

Service link type definition

```
1  <serviceLinkType name="buyerLink">
2    <role name="ticketRequester">
3      <portType name="itineraryPT"/>
4    </role>
5    <role name="ticketService">
6      <portType name="ticketOrderPT"/>
7    </role>
8  </serviceLinkType>
```


Containers

- Container
  - WSDL messages received from or sent to partners
  - Messages that are persisted by the process
- Containers constitute the "business context" of the process
- Access to container can be serialized to some extent
  - Serializable scopes

```
11 <containers>
12     <container name="itinerary" messageType="itineraryMessage"/>
13     <container name="tickets" messageType="ticketsMessage"/>
14 </containers>
```

Flow of Activities

- Flow
  - Directed graph with
    - **Activities** as nodes
    - **Links** as edges connecting the activities
  - Each activity defines the links it is a source or a target of

```
15 <flow>
16    <links>
17      <link name="order-to-airline"/>
18      <link name="airline-to-agent"/>
19    </links>
20    <receive        partner="customer"
21                    portType="itineraryPT"
22                    operation="sendItinerary"
23                    container="itinerary">
24        <source linkName"order-to-airline"/>
25    </receive>
26    <invoke         partner="airline"
27                    portType="ticketOrderPT"
28                    operation="requestTickets"
29                    inputContainer="itinerary">
30        <target linkName"order-to-airline"/>
31        <source linkName"airline-to-agent"/>
32    </invoke>
33    <receive        partner="airline"
34                    portType="itineraryPT"
35                    operation="sendTickets"
36                    container="tickets"
37        <target linkName"airline-to-agent"/>
38    </receive>
39 </flow>
40 </process>
```


Activities

- types of (simple) activities
  - Receive
    - Wait for a message to be received from a partner
    - Specifies partner from which message is to be received, as well as
    - The port and operation provided by the process
      - Used by the partner to pass the message
  - Pick
    - Specifies a whole set of messages
      - can be received from the same or different partners
    - Activity is completed when one of the specified messages is received
    - Permits specifying a time limit after which processing continues if message is not received
    - Pick and Receive can be start activities of a process
      - Can indicate that a process instance should be created if none exists
  - Reply
    - Synchronous response to a request corresponding to a receive activity
  - Invoke
    - Issue a response asynchronously
    - Synchronously invoke an in-out operation of a web service provided by a partner

Activities (cont.)

- Example:

```
14 <receive partner="hotel",
15        portType="roomPT",
16        operation="sendBooking",
17        container="stayInfo”
18        createInstance="yes"/>
19
20 <receive partner="rentalCar",
21        portType="carPT",
22        operation="sendBooking",
23        container="rentalInfo“
24        createInstance="yes"/>
25 <receive partner="customer",
26        portType="itineraryPT",
27        operation="sendItinerary",
28        container="itinerary”
29        createInstance="yes"/>
30
31 <reply partner="customer",
32        portType="travelPT",
33        operation="sendTickets",
34        container="tickets"/>
```


Activities (cont.)

- More simple activities
  - Wait
    - Process should wait for a specified time period or until a point in time
  - Empty
    - No action
      - Can serve as a means to synchronize parallel processing within the process
  - Terminate
    - Business process should be terminated immediately
  - Throw
    - Signal occurrence of an error
  - Assign
    - Copies fields from containers into other containers
  - Compensate
    - Undo the effects of completed activities

Activities (cont.)

- Structured activities
  - Flow
    - Defines sets of activities wired together via links
      - Parts of flow can be executed in parallel
        - Links used to "synchronize" them
      - Activities can again be flows
    - Links can be associated with transition conditions
    - Target of link has (implicit or explicit) join condition
  - Scope
    - Builds a group of activities
    - Definition of group characteristics
      - Fault handler
      - Compensation handler
      - Scope acts as a compensation sphere
  - Sequence
    - Enclosed activities are carried out in listed order
  - Switch
    - Selects one of several activities based on selection criteria
  - While
    - Carry out enclosed activities as long as the while condition is true

Properties and Correlation

- Property
  - Data to correlate a message with a specific process instance
    - E.g., order number
    - Usually included in the message
    - Often the same property is used in different messages
  - Can be defined in BPEL as a separate entity:

```
9  <property name="orderNumber" type="xsd:int"/>
```

- Property alias
  - Allows to point to a dedicated field of the message that represents the property
    - Usually different for each message type

```
10 <propertyAlias propertyName="orderNumber"
11                messageType="ticketsMessage"
12                part="orderInfo"
13                query="/orderID"/>
```

Fault Handlers

- Fault handlers catch and deal with faults
  - Process interacts with WSDL port, WSDL port may send fault message back to a process
  - Internal fault (throw activity)
- Catch element
  - Specifies fault to be handled
  - Includes activity (simple or structured) to be performed if fault occurs

```
35 <faultHandlers>
36       <catch faultName="noSeatsAvailable">
37                <invoke partner="customer"
38                          portType="travelPT"
39                          operation="sendRejection"
40                          inputContainer="rejection"/>
41       </catch>
42 </faultHandlers>
```

- A scope (set of activities) can define its own fault handlers
  - May make use of compensation handlers to undo completed activities

Compensation Handlers

- Long-Running (Business) Transactions (LRTs)
  - Define fault handling and compensation in an application-specific manner
    - Explicitly specified as part of the business protocol
      - E.g., order of compensation steps may be different from reverse order of completion
  - LRT within single business process, i.e., no support for LRT that spans
    - Distributed business process
    - Multiple vendors or platforms
  - -> WS-Transaction spec
- Compensation handler
  - Can be defined for each scope
    - Scopes can be arbitrarily nested
    - Syntactic shortcut for invoke activity
      - Inline definition of compensation handler
      - Equivalent to scope with comp. handler and invoke activity
  - Compensation activity can be any activity

Compensation Handlers – Example

```
<scope>
  <compensationHandler>
       <invoke partner="Seller"
              portType="SP:Purchasing”
              operation="CancelPurchase"
              inputContainer="getResponse”
              outputContainer="getConfirmation">
       </invoke>
  </compensationHandler>
  <invoke partner="Seller"
       portType="SP:Purchasing”
       operation="SyncPurchase"
       inputContainer="sendPO”
       outputContainer="getResponse">
  </invoke>
</scope>
```


Compensation Handler Invocation

- Compensate activity
  - Invokes compensation handler for named scope
    - Only possible if scope completed its execution normally
  - Can be invoked only from the fault handler or compensation handler of the immediately enclosing scope
- Default compensation handler
  - Invokes compensation handlers of immediately enclosed scopes in the reverse order of the completion of the scopes
  - Is used if a (enclosing) scope does not explicitly define a compensation handler
  - Can also be invoked explicitly
    - Useful if comp. action = “compensate enclosed scope in reverse order” + “additional activities”
- Default fault handler
  - Invokes compensation handlers of immediately enclosed scopes in the reverse order of the completion of the scopes
  - Rethrows the exception
- Container semantics
  - When invoked, compensation handler sees frozen container snapshot
    - All containers in the state they were at completion time of the scope being compensated
    - Compensation handlers live in a snapshot world
      - Cannot update “live” container data
      - Can only affect external entities
      - Input/output parameters for compensation handler are future direction