

Chapter 15 - Web Service Composition



Middleware for Heterogenous and Distributed Information Systems - WS05/06

Motivation

- Complex web services
 - Need to interact with business partners through web services
 - May combine/utilize existing web services
 - Web services composition
 - Ability to create new web services out of existing (web service) components
 - Requirements similar to BPM, Workflow Management
 - separate function from composition logic, ...
 - Composition can be iterated
 - Composition result is again a web service
 - Can be used as a building block for further composition steps
- ⇒ Middleware for web service composition



Web Services Composition Middleware

- Main elements
 - composition model and language
 - composed WS is expressed by a composition schema (script)
 - development environment
 - graphical end user tools
 - run-time environment
 - composition "engine"
- Composition vs. coordination middleware
 - composition: focus is on implementation of operations in a web service
 - internal, private
 - for automation of the execution of a composite web service
 - coordination: focus is on conversation protocols
 - public, standardized protocols
 - external coordination for verifying compliance



Web Services vs. WFMS

- Limitations of conventional composition middleware (e.g., WFMS)
 - Significant effort to integrate existing applications
 - application-specific adapters, wrappers
 - no standard model for component description, interoperability
 - Limited success of composition model standardization
 - WfMC standard is not widely implemented
- Opportunities for Web Services
 - Web Services seem to be adequate components
 - well-defined interfaces, described using WSDL
 - standardized invocation (SOAP)
 - Significant efforts in standardizing WS composition languages
 - Reuse of existing WS "infrastructure" (directory, service selection, ...)
 - WS composition tools are less expensive to develop



Dimensions of a Web Service Composition Model

- Component model
 - nature of the elements to be composed
- Orchestration model
 - abstractions, language used to define order in which services are invoked
- Data and data access model
 - how data is specified, exchanged between components
- Service selection model
 - how a specific service is selected as a component (static, dynamic binding)
- Transactions
 - transactional semantics that can be associated with the composition
- Exception handling
 - how exceptional situations are handled during execution of a composite service



Business Processes and Web Services

- Business Process Execution Language for Web Services (BPEL4WS)
 - XML-based language for specifying business process behavior based on web services
 - 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
- Standardization is in progress (OASIS)
 - based on specification proposed by IBM, Microsoft, BEA (and Siebel for BPEL 1.1)
 - BPEL unifies XLANG (Microsoft), WSFL (IBM)



BPEL4WS

- BPEL can support specification of both, composition schemas and coordination protocols
 - can be used in both composition and coordination middleware
- Two types of processes
 - executable process (-> composition)
 - defines implementation logic for a composite web service
 - portable between BPEL-conformant environments
 - abstract process (-> coordination)
 - service-centric perspective on coordination protocols
 - describe message exchange between partners
- Business process defines
 - potential execution order of operations (web services)
 - data shared between the web services
 - correlation information
 - partners involved in business process and interfaces they need to implement
 - joint exception handling for collection of web services

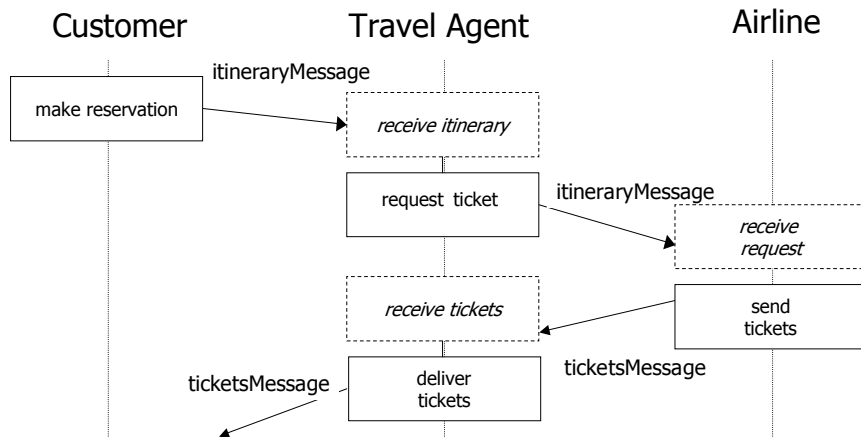


BPEL Component Model

- Components are web services described using WSDL
 - abstract WSDL interfaces are referenced in BPEL scripts
 - no reference to bindings, endpoints, or services
- Basic activities in BPEL represent components, correspond to WSDL operations
 - Invoke
 - Issue an asynchronous request, or
 - Synchronously invoke a request/reply operation of a web service provided by a partner
 - 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
 - Reply
 - Synchronous response to a request corresponding to a receive activity
 - Combination of Receive/Reply corresponds to request-response operation in WSDL



Example

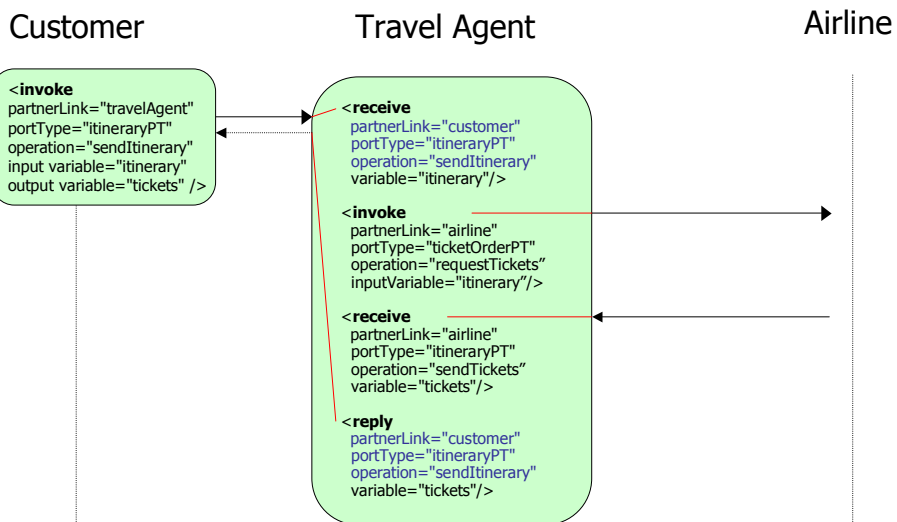


© Prof. Dr.-Ing. Stefan Deßloch

9

Middleware for Heterogenous and Distributed Information Systems - WS05/06

BPEL Activities – Example



© Prof. Dr.-Ing. Stefan Deßloch

10

Middleware for Heterogenous and Distributed Information Systems - WS05/06

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

Orchestration Model - Structured Activities

- 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
- Pick
 - Specifies a set of activities with associated events (e.g., receipt of message)
 - messages can be received from the same or different partners
 - Activity is completed when one of the events occurs
 - 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
- Flow activity

Flow Activity

- Defines sets of activities plus (optional) control flow
 - All activities can (potentially) execute in parallel
 - Activities can be "wired together" via links
 - Links used to "synchronize" them
 - Activities can again be flows
- Links can be associated with transition conditions
 - Specified at the source activity
- Target of link has join condition
 - Explicit join condition can reference the status of incoming links
 - Implicit join condition: at least one incoming link has a positive status



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 partnerLink="customer"
21 portType="itineraryPT"
22 operation="sendItinerary"
23 variable="itinerary">
24 <source linkName="order-to-airline"/>
25 </receive>
26 <invoke partnerLink="airline"
27 portType="ticketOrderPT"
28 operation="requestTickets"
29 inputVariable="itinerary">
30 <target linkName="order-to-airline"/>
31 <source linkName="airline-to-agent"/>
32 </invoke>
33 <receive partnerLink="airline"
34 portType="itineraryPT"
35 operation="sendTickets"
36 variable="tickets">
37 <target linkName="airline-to-agent"/>
38 </receive>
39 </flow>
40 </process>
```



Link Semantics

- Control Flow Navigation
 - Evaluation of link status, join conditions evaluated only if status of all incoming links has been evaluated
 - Dead path elimination
 - Attribute *suppressJoinFailure="yes"*
- Links can cross boundaries of structured activities
 - Some restrictions apply
 - must not cross while-activity, serializable scope, compensation handler, event handler
 - no links into a fault handler
 - Careful consideration of resulting semantics
- Links must not build a control cycle!

Process life-cycle

- Start activities
 - receive, pick – createInstance attribute
 - creates a new process instance, if it doesn't exist already
 - Example:

```
<receive partner="customer",
  portType="itineraryPT",
  operation="sendItinerary",
  variable="itinerary"
  createInstance="yes"/>
```
 - each process must have at least one start activity as an initial activity
- Process termination
 - process-level activity completes successfully
 - fault "arrives" at the process level (handled or not)
 - terminate activity is invoked

Data Types and Data Transfer

- **Variables** can be used to define data containers
 - WSDL messages received from or sent to partners
 - Messages that are persisted by the process
 - XML data defining the process state
- Constitute the "business context" of the process
- Access to variables can be serialized to some extent

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

- **Variable assignment**
 - Receiving a message (or a reply of an invoke activity) implicitly assigns value
 - Alternative: **assign** activity (another simple activity)
 - Copies fields from containers into other containers



Service Selection

- Partner link type (WSDL extension) defines
 - roles played by partners in a conversational relationship
 - web service interfaces that need to be implemented to assume a role
- Partner link (BEL process definition)
 - identifies the web services mutually used by the partner or process
 - e.g., agent process interacts with customer, airline
 - references a partner link type
 - defines role taken by the process itself (myRole) and role that has to be accepted by the partner (partnerRole)

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

Partner link type definition

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



Service Selection (cont.)

- Partner link names are used in all service interactions to identify partners
 - see activities for invoking/providing services
- Assignment of endpoints for partners
 - at deployment time
 - dynamically at run time

Properties

- Property
 - Globally defined types
 - Primarily used 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
 - Can be used in expression and assignments to easily use properties

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

Correlation

- Message needs to be delivered not only to the correct port, but to the correct instance of the business process providing the port (instance routing)
- Correlation Set
 - one or more properties used for correlating messages
 - example
 - ```
<correlationSets>
 <correlationSet name="Booking"
 properties="orderNumber"/>
 ...
</correlationSets>
```
    - correlation properties are like "late-bound constants"
      - binding happens through specially marked message send/receive activities
      - value must not change after the binding happens
  - Often, more than one correlation set is used for an entire process
    - example: orderNumber -> invoiceNumber
    - correlated message exchanges may nest, overlap
    - same message may carry multiple correlation sets

## Scope

- Defines the behavior context of an activity
  - simple or structured (group of activities)
- Can provide the following for a (regular) activity
  - (Local) data variables
  - Correlation Sets
  - Fault handler(s)
  - Event handler(s)
  - Compensation handler
    - Scope acts as a compensation sphere
- Scopes can be arbitrarily nested

## 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)
- Fault reaching a fault handler means that regular processing within scope can no longer proceed
  - All active work in the scope must be stopped!
- 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 inputVariable="rejection"/>
41 </catch>
42 </faultHandlers>
```
  - May make use of **compensation handlers** to undo completed nested activities
- After fault handler completes successfully, processing may continue outside the scope
  - Processing of the scope is still considered to have ended abnormally



## Compensation Handler

- Used to reverse the work of a **successfully completed** scope
  - compensation handler is "installed" after successful completion of the scope
- Can be defined for each scope
  - Inline definition of compensation handler for invoke activity
    - Syntactic shortcut, is equivalent to scope with comp. handler and invoke activity
- Compensation activity can be any activity



## Compensation Handlers – Example

```
<scope name="purchase">
 <compensationHandler>
 <invoke partner="Seller"
 portType="SP:Purchasing"
 operation="CancelPurchase"
 inputVariable="getResponse"
 outputVariable="getConfirmation">
 </invoke>
 </compensationHandler>
 <invoke partner="Seller"
 portType="SP:Purchasing"
 operation="SyncPurchase"
 inputVariable="sendPO"
 outputVariable="getResponse">
 </invoke>
</scope>
```

## Compensation Handler Invocation

- **Compensate** activity
  - Invokes compensation handler for named scope
    - Example: `<compensate scope="purchase"/>`
  - Can be invoked only from the fault handler or compensation handler of the immediately enclosing scope
- **Data semantics**
  - When invoked, compensation handler sees frozen **snapshot of data variables**
    - All variables in the state they were at completion time of the scope being compensated
  - Compensation handlers live in a snapshot world
    - Cannot update "live" data variables
    - Can only affect external entities
    - Input/output parameters for compensation handler are future direction

## Default Compensation and Fault Handlers

- 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

## More on Faults

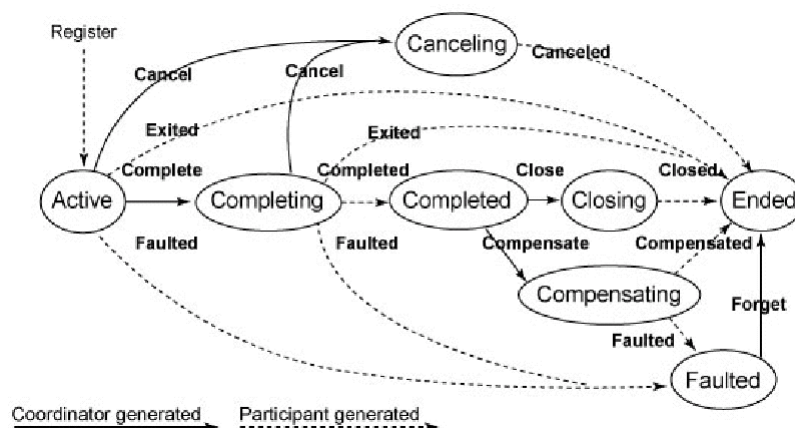
- Termination of running activities
  - Regular processing is stopped
  - If the activity is a scope, the fault handler for **forcedTermination** fault is invoked
    - Activity being terminated can react to termination
      - call compensation handlers of nested, completed activities, ...
    - Implicit fault handler is invoked otherwise
- Faults occurring in compensation handlers or fault handlers
  - Can be caught by regular fault handlers in enclosing scopes or scopes with the fault handler

## BPEL 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, local business process, i.e., no support for LRT that spans
    - Distributed business process
    - Multiple vendors or platforms
- WS-Transaction specification
- Business Activities
  - Protocol Framework can be used to model the fault and compensation relationships between a scope and its enclosing scopes

## Business Agreement Protocol

- BusinessAgreementWithComplete – State Diagram



## Summary

---

- Web service composition
  - means to implement web service by reusing/combining existing services
  - can be supported by WS composition middleware
    - borrowing concepts from WFMS
- BPEL
  - effort to standardize web service composition
  - allows definition of composition and coordination aspects
    - abstract vs. executable processes
  - main concepts
    - basic activities for web service operations
    - structured activities for defining service composition, control flow
    - blackboard approach for data flow based on variables
    - service selection based on partner link types, partner links, endpoints
    - elaborate model for transactions and exception handling
      - fault handler
      - compensation handler
  - supported by key industry players