

02267: Software Development of Web Services

Week 6

Hubert Baumeister

huba@dtu.dk

Department of Applied Mathematics and Computer Science
Technical University of Denmark

Fall 2015

Recap

- ▶ Business Processes
- ▶ Composite Web Services: BPEL
- ▶ BPEL: Calling a Web Service

Contents

Mid-term evaluation

Process instances and Correlation Sets

Traditional Programming Language Support

Dealing with Errors

Reacting on Events

Order Process Example

Mid-term evaluation

- ▶ Started: last Friday
- ▶ Ends: Sunday this week

Contents

Mid-term evaluation

Process instances and Correlation Sets

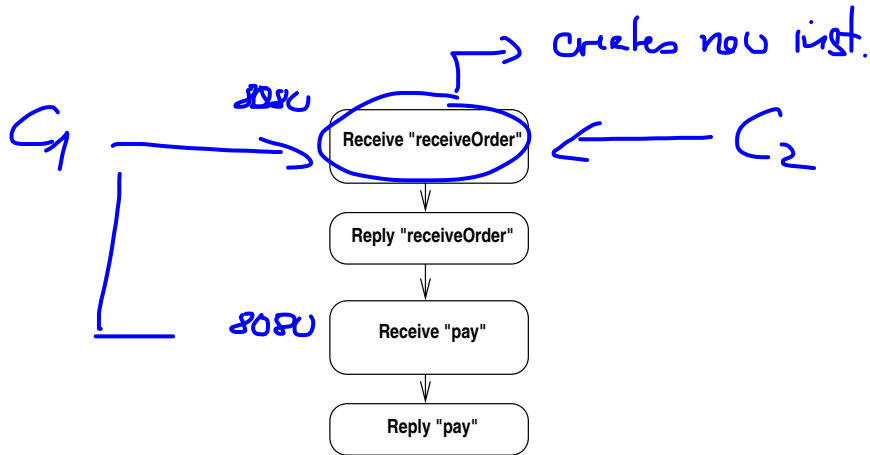
Traditional Programming Language Support

Dealing with Errors

Reacting on Events

Order Process Example

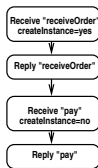
Process Lifecycle: Example I



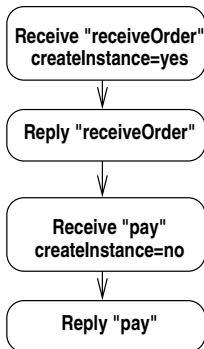
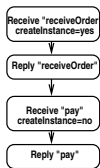
Process Lifecycle

- ▶ To deal with several requests at the same time, a process needs to be instantiated
 - ▶ Local program counter
 - ▶ Copy of the variables and other state information
- ▶ The first non-trivial activity of a process must have either a receive or pick where createInstance=yes (default is createInstance=no)

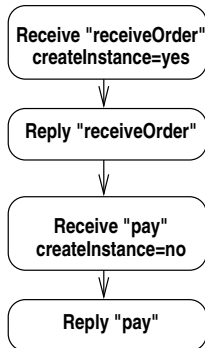
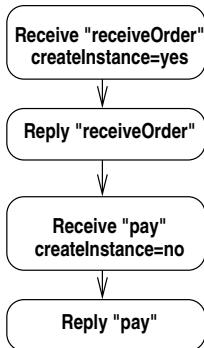
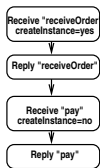
Process Lifecycle: Example II



Process Lifecycle: Example II



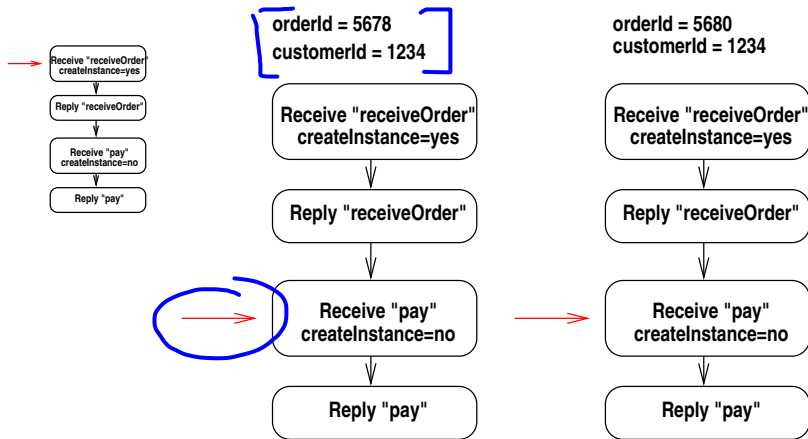
Process Lifecycle: Example II



Process Lifecycle: Identifying process instances

- ▶ Which process receives the next Receive "pay" message?

→ Solution: Correlation sets



Process Lifecycle: Correlation sets

► Correlation sets

- identify processes using properties like customer ID and order ID

```
<correlationSet name="PaymentCorrelation"  
  properties="ns0:customerID ns0:orderID" />
```

► Properties

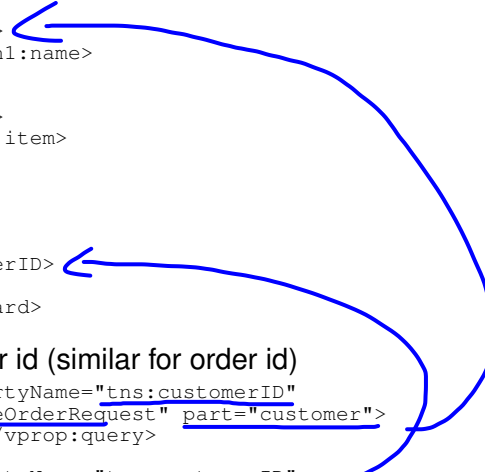
- Properties are extracted from messages → via property aliases
- Properties have a type and are abstract
- Properties are defined in the WSDL file(s)

```
<vprop:property name="customerID" type="xsd:string" />  
<vprop:property name="orderID" type="xsd:string" />
```

Property Aliases

► Message receive order:

```
<urn:receiveOrder>
  <customer>
    <urn1:id>1234</urn1:uid>
    <urn1:name>No Name</urn1:name>
  </customer>
  <order>
    <urn1:id>5678</urn1:uid>
    <urn1:uitem> ... </urn1:uitem>
  </order>
</urn:receiveOrder>
```



► Message pay:

```
<urn:pay>
  <customerID>1234</customerID>
  <orderID>5678</orderID>
  <creditcard>...</creditcard>
</urn:pay>
```

► Property alias for customer id (similar for order id)

1)

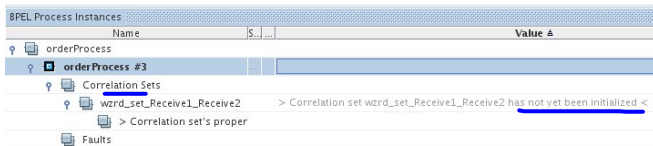
```
<vprop:propertyAlias propertyName="tns:customerID"
  messageType="tns:receiveOrderRequest" part="customer">
  <vprop:query>ns1:id</vprop:query>
```

2)

```
</vprop:propertyAlias>
<vprop:propertyAlias propertyName="tns:customerID"
  messageType="tns:payRequest" part="customerID">
```

Using Correlation Sets

- ▶ `initiate="yes"`: properties are set from received message

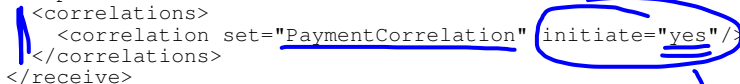


A screenshot of the 'BPEL Process Instances' window. It shows a tree view with 'orderProcess' expanded to 'orderProcess #3', which contains 'Correlation Sets'. Under 'Correlation Sets', there is an entry for 'wzrd_set_Receive1_Receive2' with a status message: '> Correlation set wzrd_set_Receive1_Receive2 has not yet been initialized <'. A blue vertical line is drawn to the left of the tree view.

BPEL Process Instances	
Name	Value
orderProcess	
orderProcess #3	
Correlation Sets	
wzrd_set_Receive1_Receive2	> Correlation set wzrd_set_Receive1_Receive2 has not yet been initialized <
Faults	

```
<receive name="ReceiveOrderRequest" createInstance="yes"  
  partnerLink="orderRequest"  
  operation="receiveOrder" ... >
```

```
<correlations>  
  <correlation set="PaymentCorrelation" initiate="yes" />  
</correlations>  
</receive>
```



A blue arrow originates from the `initiate="yes"` attribute in the XML code block and points to the 'orderid' and 'customerid' values in the BPEL instance table below.



A screenshot of the 'BPEL Process Instances' window, similar to the one above but with data populated. Under 'Correlation Sets', the entry for 'wzrd_set_Receive1_Receive2' now shows two sub-entries: 'orderid' with the value '5678' and 'customerid' with the value '1234'. Both values are circled in blue. A blue arrow points from the 'initiate="yes"' attribute in the XML code block above to these values.

BPEL Process Instances	
Name	Value
orderProcess	
orderProcess #3	
Correlation Sets	
wzrd_set_Receive1_Receive2	{7000000}((http://www.w3.org/2001/XMLSchema:string){5678})((http://www.w3.org/2001/XMLSchema:string){1234})
orderid	5678
customerid	1234
Faults	

Using Correlation Sets

- ▶ initiate="no": *receive* waits until message with right properties arrives

The screenshot shows a process instance view with two instances of 'orderProcess'. Instance 1 (labeled 'Instance 1' in blue) has a correlation set 'wzrd_set_Receive1_Receive' with values 'orderId' 5678 and 'customerId' 1234. Instance 2 (labeled 'Instance 2' in blue) has a correlation set 'wzrd_set_Receive1_Receive' with values 'orderId' 5680 and 'customerId' 1234. The values 5678, 5680, and 1234 are underlined in blue. Arrows point from the XML message to the corresponding values in the correlation sets.

```
<urn:pay>
  <customerID>1234</customerID>
  <orderID>5678</orderID>
  <creditcard>...</creditcard>
</urn:pay>
```

```
<receive name="ReceivePayRequest" createInstance="no"
  partnerLink="orderRequest"
  operation="pay" ... >
  <correlations>
    <correlation set="PaymentCorrelation" initiate="no"/>
  </correlations>
</receive>
```

Correlation with reply / invoke

- ▶ Correlation sets can be used with invoke, receive, reply, pick and eventHandlers.
- ▶ **Reply**
 - ▶ Initializes a correlation set from output variable (e.g. with if the process generates a order id instead of the customer supplying one)
- ▶ Invoke
 - ▶ paramter pattern="request||response||request-response"
 - ▶ Pattern=request: initialize correlation set from input variable
 - ▶ process identification information comes from the process
 - ▶ Pattern=response: initialize correlation set from output variable
 - ▶ process identification information comes from the web service being called
 - ▶ Pattern=request-response: initialize correlation set from input and output variable

Contents

Mid-term evaluation

Process instances and Correlation Sets

Traditional Programming Language Support

Dealing with Errors

Reacting on Events

Order Process Example

Traditional Programming Language Support

- ▶ Scope
- ▶ Sequence
- ▶ Conditional
- ▶ Loops
 - ▶ While
 - ▶ RepeatUntil
 - ▶ ForEach
 - ▶ sequential execution
 - ▶ parallel execution

Scopes

```
<scope ...>
<variables>...</variables> <!-- optional -->
<correlationSets>...</correlationSets> <!-- optional -->
<faultHandlers>...</faultHandlers> <!-- optional -->
<compensationHandler>...</compensationHandler> <!-- optional -->
<eventHandlers>...</eventHandlers> <!-- optional -->
..... <!-- one activity; mandatory -->
</scope>
```

- ▶ Scopes provide a way of localizing definitions of variables, correlation sets, fault handlers, ...
- ▶ These are only visible in the scope and all sub-scopes but not in the enclosing scope
- ▶ Like a block structure (begin end) in Pascal or the use of { } in Java/C/C++/C#

ForEach

forEach

```
<forEach counterName="BPELVariableName" parallel="yes|no">
  <startCounterValue ..>...</startCounterValue>
  <finalCounterValue ..>...</finalCounterValue>
  <completionCondition>?
    <branches expressionLanguage="anyURI"?
      <successfulBranchesOnly="yes|no"?>?
        m
      </branches>
    </completionCondition>
  <scope ...>...</scope>
</forEach>
```

- ▶ Executes only the first m completed scope activities out of the n iterations
 - ▶ Either any completion (also errornous) or only successful completions
 - ▶ Depends on `successfulBranchesOnly` attribute
- ▶ With parallel execution, the other operations are terminated
- ▶ e.g. "Send n requests (sequential or parallel), but only $m < n$ answers are needed, all other answers are discarded"

Contents

Mid-term evaluation

Process instances and Correlation Sets

Traditional Programming Language Support

Dealing with Errors

Reacting on Events

Order Process Example

Fault Handlers

- ▶ Very much like OnException in VisualBasic or try-catch in Java
- ▶ Fault Handlers are associated to scopes
- ▶ Handles exceptions occurring in that scope and
 - ▶ thrown by the throw activity
 - ▶ thrown / returned by invoke activities (e.g. via SOAP faults)

throw activity

```
<throw faultName="QName" faultVariable="BPELVariableName"? />
```

- ▶ Default Fault Handler for a scope
 - ▶ calls compensate
 - ▶ rethrows the exception

Fault Handlers

Fault Handler

```
<faultHandlers>?  
<catch faultName="qname"? faultVariable="ncname"?>*  
  activity  
</catch>  
<catchAll>?  
  activity  
</catchAll>  
</faultHandlers>
```

- ▶ For matching, first the name is matched and then the `faultVariable`
- ▶ A matching fault handler first terminates all activities
 - ▶ Then it executes its activity
 - ▶ In case this activity contains again a fault
 - ▶ handled by a scope inside the activity
 - ▶ if not, the exception is processed as a normal failure by the enclosing scope
- ▶ `catchAll` catches all faults
- ▶ Fault handlers can rethrow a fault

Fault messages defined in WSDL files and BPEL faults

► Fault types

- 1 BPEL faults, e.g., systemFault
- 2 User defined faults: Defined through faults in operation definition

► Important

- throw does not create a SOAP fault defined in the WSDL file for an operation

→ use reply activity with a fault variable of the correct type

```
<reply name="Reply2"
  partnerLink="Order"
  operation="pay" xmlns:tns="urn:ws.imm.dtu:orderWsd1"
  portType="tns:orderPortType"
  faultName="tns:fault1"
  variable="PayFault"/>
```


Contents

Mid-term evaluation

Process instances and Correlation Sets

Traditional Programming Language Support

Dealing with Errors

Reacting on Events

Order Process Example

Reacting to Events: Event driven business processes

- ▶ Sometimes it makes sense to not prescribe exactly the execution of a business process (e.g. if the process partners have multiple choices to act)
 - ▶ Instead, there can be several events happening on which to react differently
- event-driven programming
- ▶ Two constructs in BPEL
 1. event handlers: react on events while a normal activity is executed
 2. pick activity: react on events as an activity

Pick Activity

- ▶ Is an activity
- ▶ Offers a choice between
 - ▶ Reception of messages (i.e. multiple receive)
 - ▶ Alarms
 - ▶ Wait for a specific time
 - ▶ Wait until some time has passed
 - ▶ Repeat every n seconds
- ▶ Can simulate receive
 - ▶ Can be used to create a process
- ▶ Can be used to define timeouts

Pick example

- ▶ "Wait either for an inputLineItem or an orderComplete message from the customer or do something other after 3 days and 10 hours"

Pick example

```
<pick>
  <onMessage partnerLink="buyer"
    portType="orderEntry"
    operation="inputLineItem"
    variable="lineItem">
    <!-- activity to add line item to order -->
  </onMessage>
  <onMessage partnerLink="buyer"
    portType="orderEntry"
    operation="orderComplete"
    variable="completionDetail">
    <!-- activity to perform order completion -->
  </onMessage>
  <!-- set an alarm to go after 3 days and 10 hours -->
  <onAlarm for="'P3DT10H'">
    <!-- handle timeout for order completion -->
  </onAlarm>
</pick>
```

Event Handler

- ▶ Event Handlers are defined within scopes and are not an activity
- ▶ Event Types
 1. Reception of messages (onEvent)
 2. Timing Events (onAlarm)
 - ▶ Wait for a specific time
 - ▶ Wait until some time has passed
 - ▶ Repeat every n time units
- ▶ Events are executed in parallel
 - ▶ With other events
 - ▶ With the activity of the scope it belongs to
 - ▶ They react to events while a main activity is executed
 - ▶ Event handler is terminated when the scope is terminated

Event Handler example

"While processing the order, allow the user to query the order status or to cancel the order"

```
<process name="purchaseOrderProcess" ...>
  ...
  <eventHandlers>
    <onEvent partnerLink="purchasing"
      operation="queryOrderStatus" ...>
      <scope>...</scope>
    </onEvent>
    <onEvent partnerLink="purchasing"
      operation="cancelOrder" ...>
      <scope>...</scope>
    </onEvent>
  </eventHandlers>
  ...
</process>
```

Pick vs Event Handler

- ▶ When to choose?
- ▶ Pick
 - ▶ Choice between events
- ▶ Event Handler
 - ▶ One main action, other events happen in parallel
 - ▶ Several events can happen at the same time

Contents

Mid-term evaluation

Process instances and Correlation Sets

Traditional Programming Language Support

Dealing with Errors

Reacting on Events

Order Process Example

Order Process Example

- ▶ We extend the Order Process example to deal with
 1. An empty list of orders should return a fault to the client
 2. If the customer does not pay within 5 seconds, the process should be terminated
 3. The customer can try to cancel his order after payment
- ▶ A detailed description of this example can be found in *Additional Course Material* on the course's Web site:
<http://www.compute.dtu.dk/courses/02267>

Empty list of orders

An empty list of orders should return a fault to the client

- ▶ Constructs

- ▶ Pick: Choice between two events; event can be reception of a message or a timer event
- ▶ Throw a fault
- ▶ If: Conditional
- ▶ Reply to a message
- ▶ Receive a message
- ▶ Invoke another Web service
- ▶ Scope=Block structure: can be used to declare faults and event handlers
- ▶ Event Handler: Handling of events (see pick)
- ▶ Fault Handler: Handling of faults

5 Seconds Timeout

- ▶ If the customer does not pay within 5 seconds, the process should be terminated
- ▶ Constructs
 - ▶ Pick: Choice between two events; event can be reception of a message or a timer event
 - ▶ Throw a fault
 - ▶ If: Conditional
 - ▶ Reply to a message
 - ▶ Receive a message
 - ▶ Invoke another Web service
 - ▶ Scope=Block structure: can be used to declare faults and event handlers
 - ▶ Event Handler: Handling of events (see pick)
 - ▶ Fault Handler: Handling of faults

Cancel Order

- ▶ The customer can try to cancel his order after payment
- ▶ Constructs
 - ▶ Pick: Choice between two events; event can be reception of a message or a timer event
 - ▶ Throw a fault
 - ▶ If: Conditional
 - ▶ Reply to a message
 - ▶ Receive a message
 - ▶ Invoke another Web service
 - ▶ Scope=Block structure: can be used to declare faults and event handlers
 - ▶ Event Handler: Handling of events (see pick)
 - ▶ Fault Handler: Handling of faults