Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

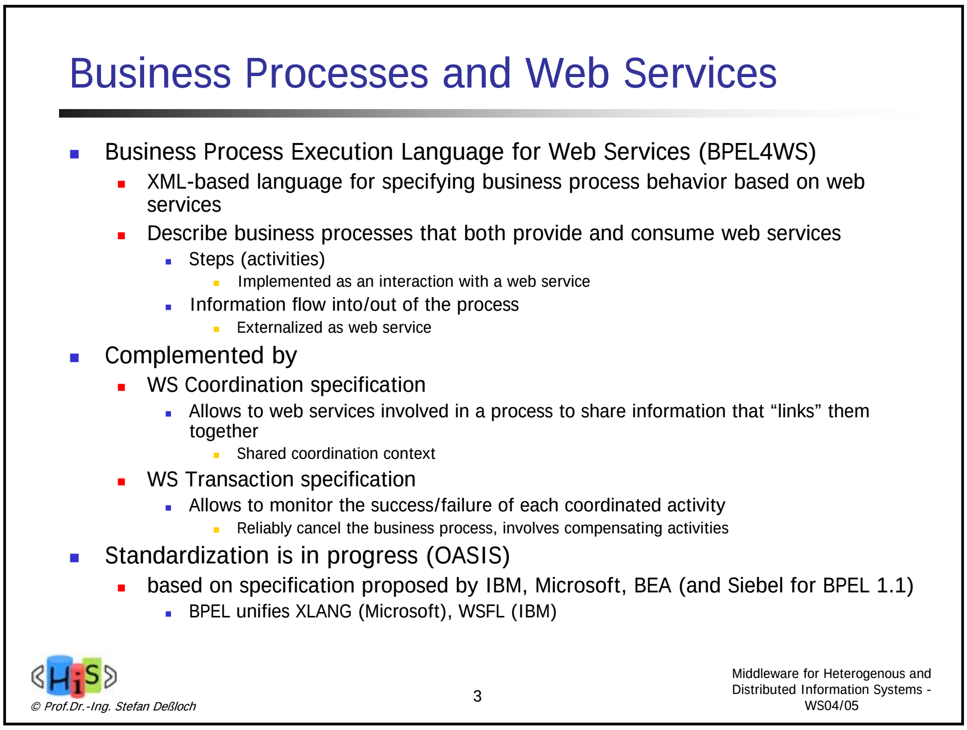
TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Chapter 13 - Business Processes and Web Services

Middleware for Heterogenous and Distributed Information Systems - WS04/05

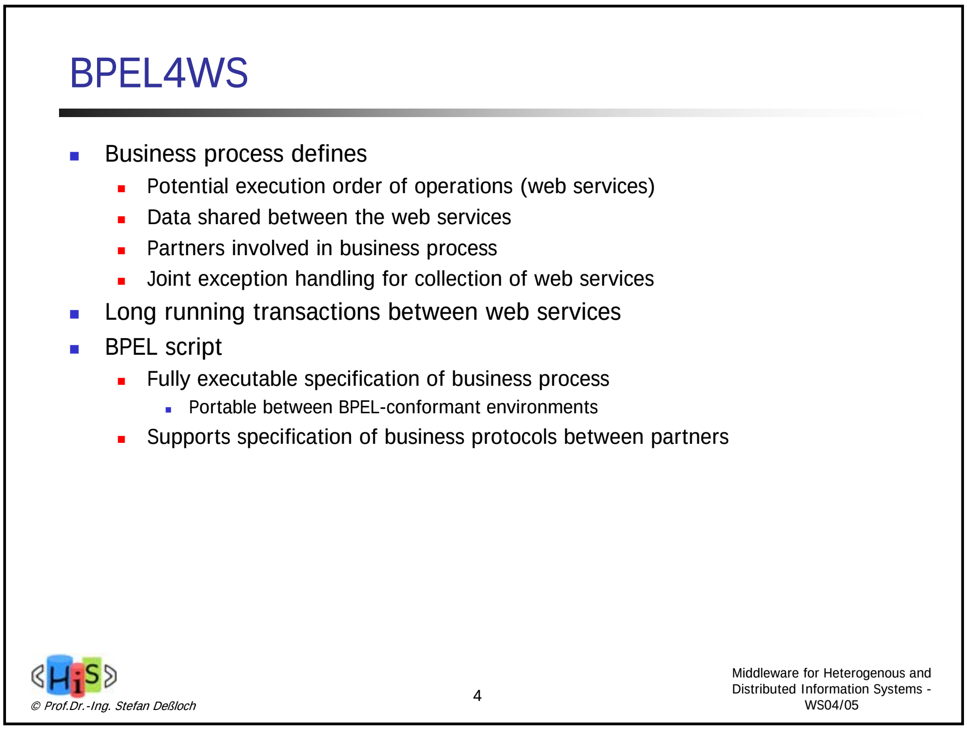
Introduction

- 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, ...
- 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



Business Processes and Web Services

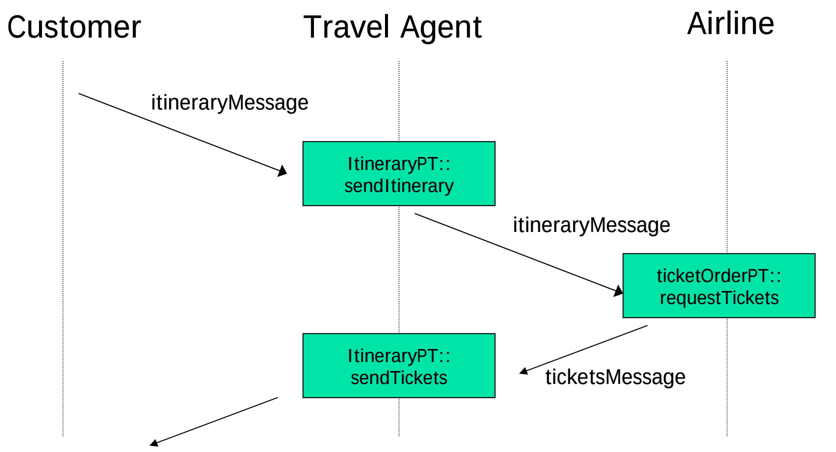
- Business Process Execution Language for Web Services (BPEL4WS)
  - XML-based language for specifying business process behavior based on web services
  - Describe business processes that both provide and consume web services
    - Steps (activities)
      - Implemented as an interaction with a web service
    - 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

- 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



Customer

Travel Agent

Airline

itineraryMessage

ItineraryPT::
sendItinerary

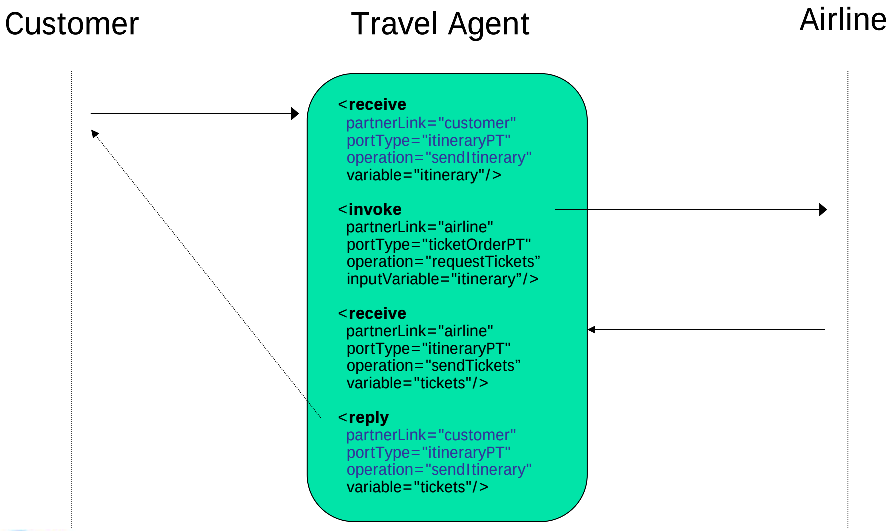
itineraryMessage

ticketOrderPT::
requestTickets

ItineraryPT::
sendTickets

ticketsMessage

Activities – Example

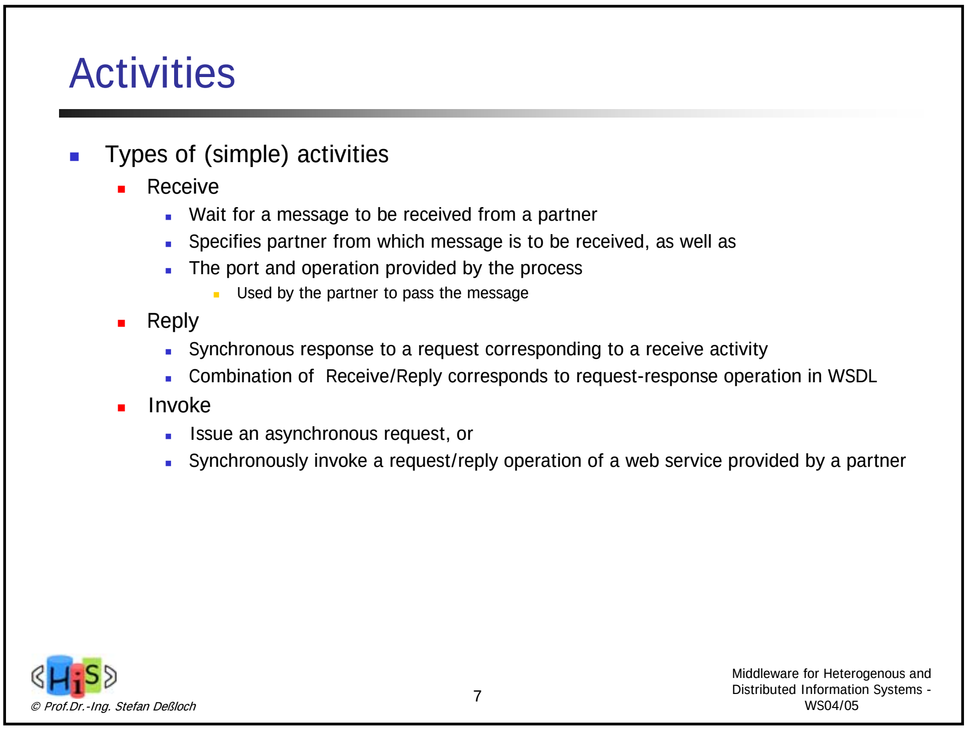


Customer

Travel Agent

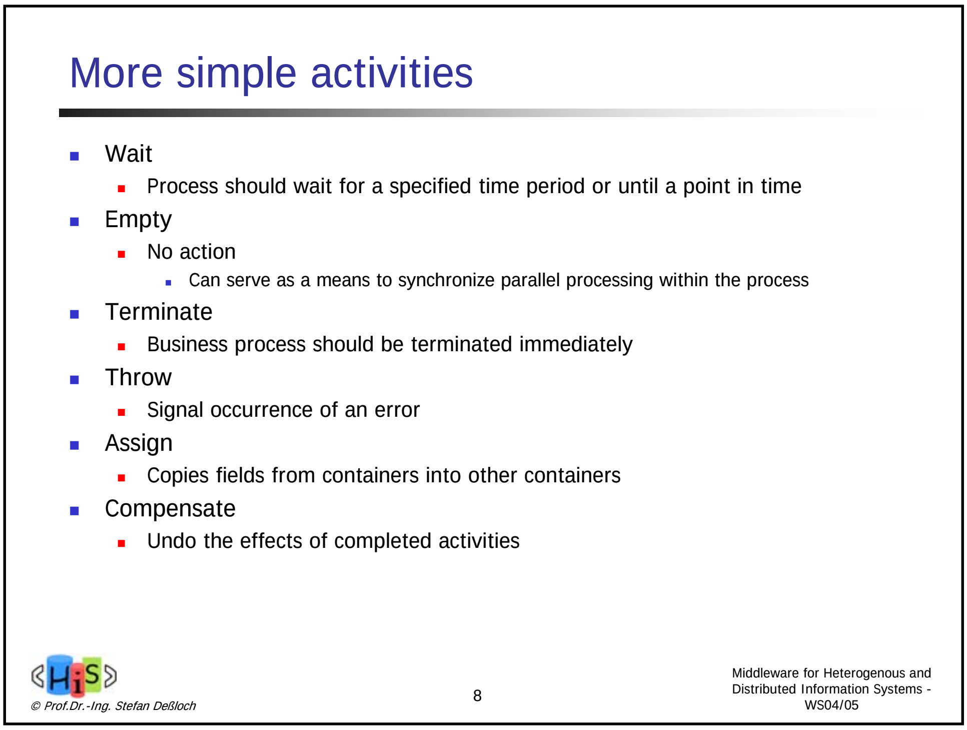
Airline

```
<receive
  partnerLink="customer"
  portType="itineraryPT"
  operation="sendItinerary"
  variable="itinerary"/>

<invoke
  partnerLink="airline"
  portType="ticketOrderPT"
  operation="requestTickets"
  inputVariable="itinerary"/>

<receive
  partnerLink="airline"
  portType="itineraryPT"
  operation="sendTickets"
  variable="tickets"/>

<reply
  partnerLink="customer"
  portType="itineraryPT"
  operation="sendItinerary"
  variable="tickets"/>
```



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
  - Reply
    - Synchronous response to a request corresponding to a receive activity
    - Combination of Receive/Reply corresponds to request-response operation in WSDL
  - Invoke
    - Issue an asynchronous request, or
    - Synchronously invoke a request/reply operation of a web service provided by a partner



More simple activities

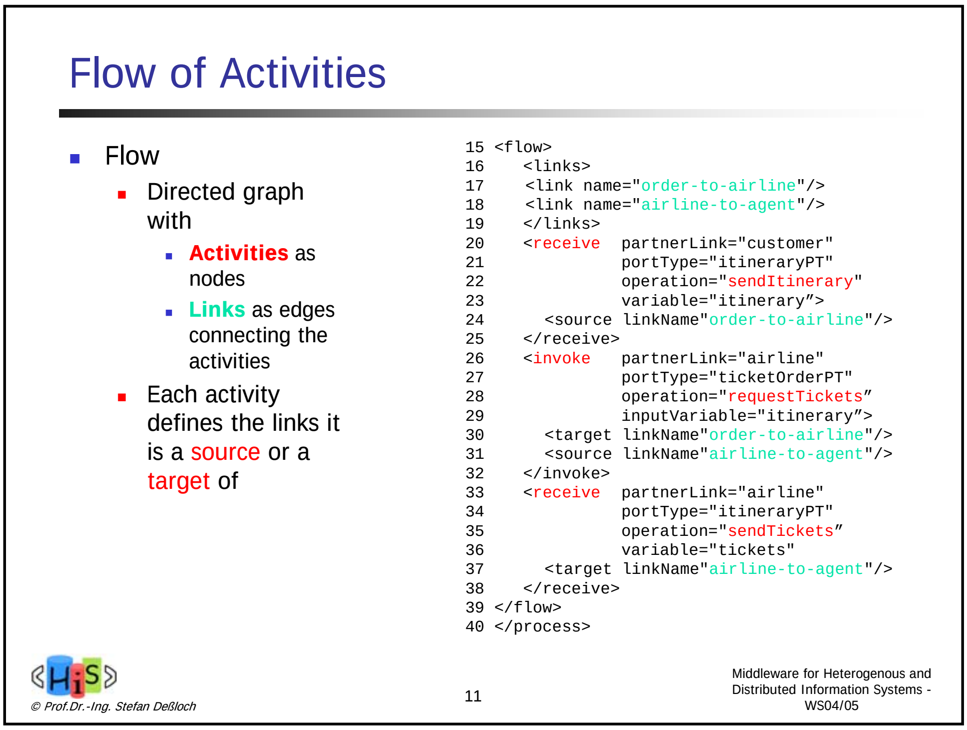
- Wait
  - Process should wait for a specified time period or until a point in time
- Empty
  - No action
    - Can serve as a means to synchronize parallel processing within the process
- Terminate
  - Business process should be terminated immediately
- Throw
  - Signal occurrence of an error
- Assign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

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 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

Flow Activity

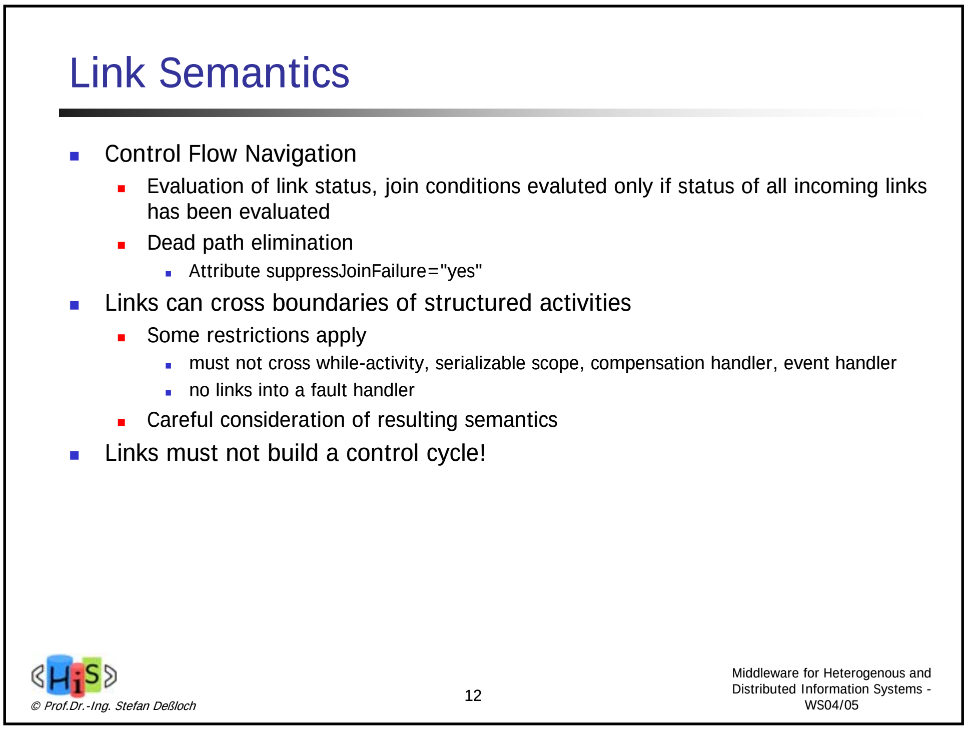
- Defines sets of activities plus (optional) control flow
  - All activities can (potentially) execute in parallel
  - Activities can be "wired together" via links
    - Links used to “synchronize” them
  - Activities can again be flows
- Links can be associated with transition conditions
  - Specified 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

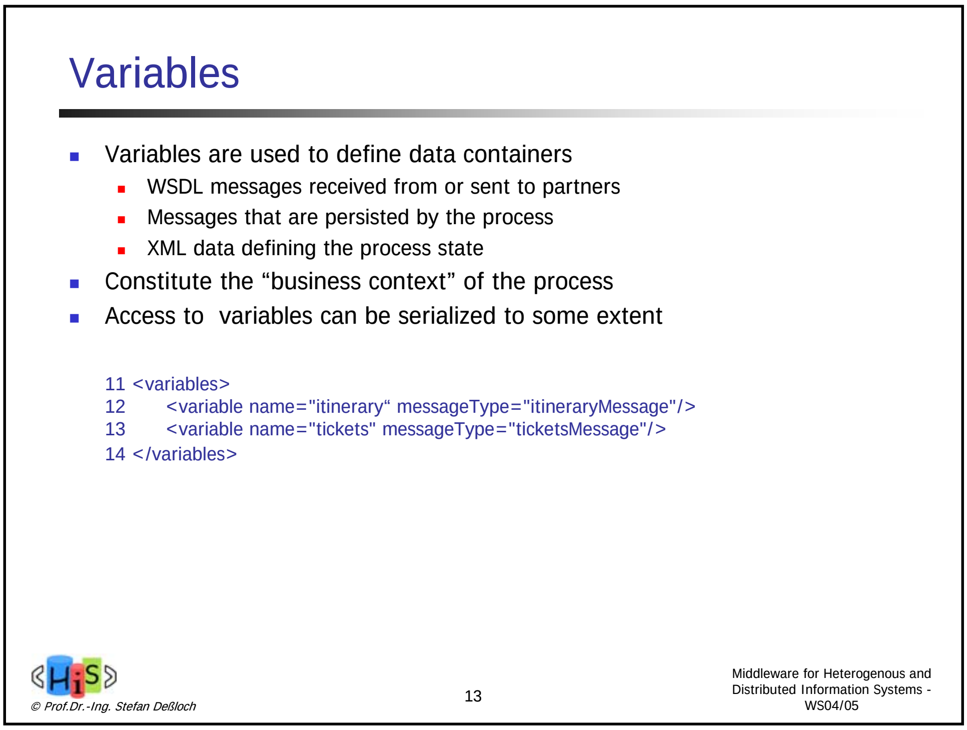
- Flow
  - Directed graph with
    - **Activities** as nodes
    - **Links** as edges connecting the activities
  - Each activity defines the links it is a source or a target of

```
<flow>
   <links>
   <link name="order-to-airline"/>
   <link name="airline-to-agent"/>
   </links>
   <receive  partnerLink="customer"
             portType="itineraryPT"
             operation="sendItinerary"
             variable="itinerary”>
     <source linkName"order-to-airline"/>
   </receive>
   <invoke   partnerLink="airline"
             portType="ticketOrderPT"
             operation="requestTickets”
             inputVariable="itinerary”>
     <target linkName"order-to-airline"/>
     <source linkName"airline-to-agent"/>
   </invoke>
   <receive  partnerLink="airline"
             portType="itineraryPT"
             operation="sendTickets”
             variable="tickets"
     <target linkName"airline-to-agent"/>
   </receive>
</flow>
</process>
```



Link Semantics

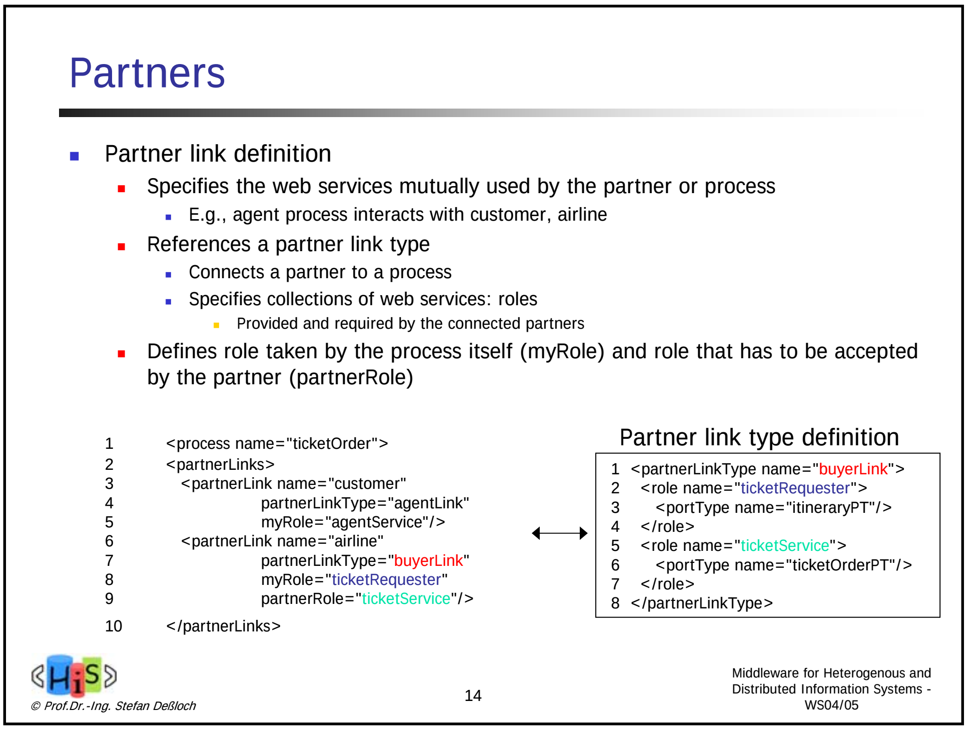
- Control Flow Navigation
  - Evaluation of link status, join conditions evaluted only if status of all incoming links has been evaluated
  - Dead path elimination
    - Attribute suppressJoinFailure="yes"
- Links can cross boundaries of structured activities
  - Some restrictions apply
    - must not 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!



Variables

- Variables are 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

```
<variables>
    <variable name="itinerary“ messageType="itineraryMessage"/>
    <variable name="tickets" messageType="ticketsMessage"/>
</variables>
```



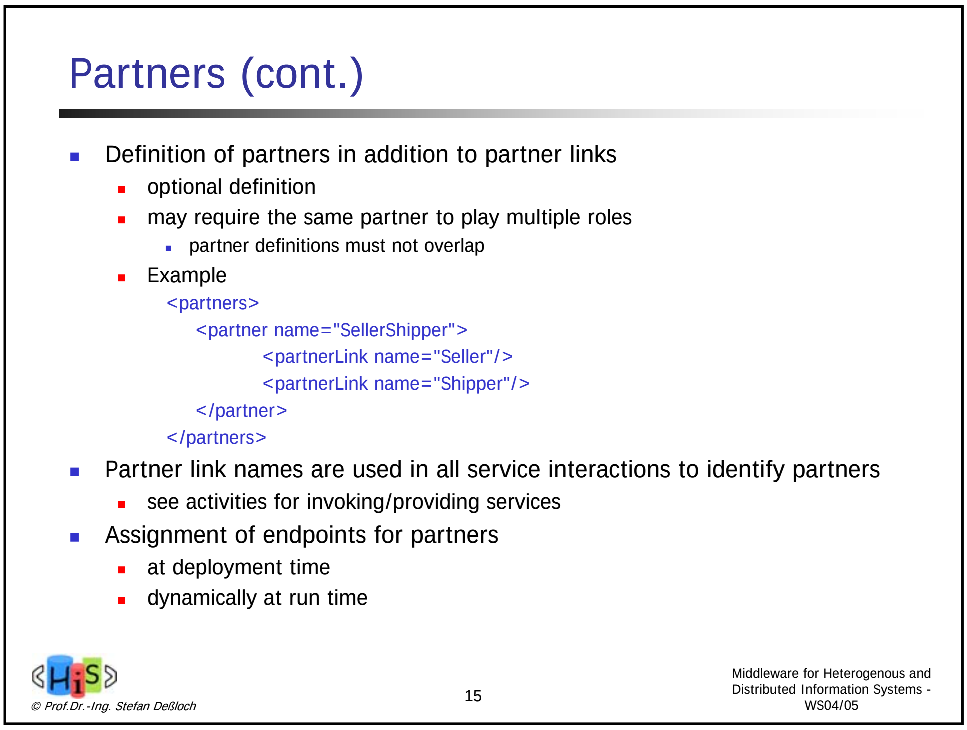
Partners

- Partner link definition
  - Specifies the web services mutually used by the partner or process
    - E.g., agent process interacts with customer, airline
  - References a partner 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)

```
<process name="ticketOrder">
<partnerLinks>
  <partnerLink name="customer"
               partnerLinkType="agentLink"
               myRole="agentService"/>
  <partnerLink name="airline"
               partnerLinkType="buyerLink"
               myRole="ticketRequester"
               partnerRole="ticketService"/>
</partnerLinks>
```

Partner link type definition

```
<partnerLinkType name="buyerLink">
  <role name="ticketRequester">
     <portType name="itineraryPT"/>
  </role>
  <role name="ticketService">
     <portType name="ticketOrderPT"/>
  </role>
</partnerLinkType>
```

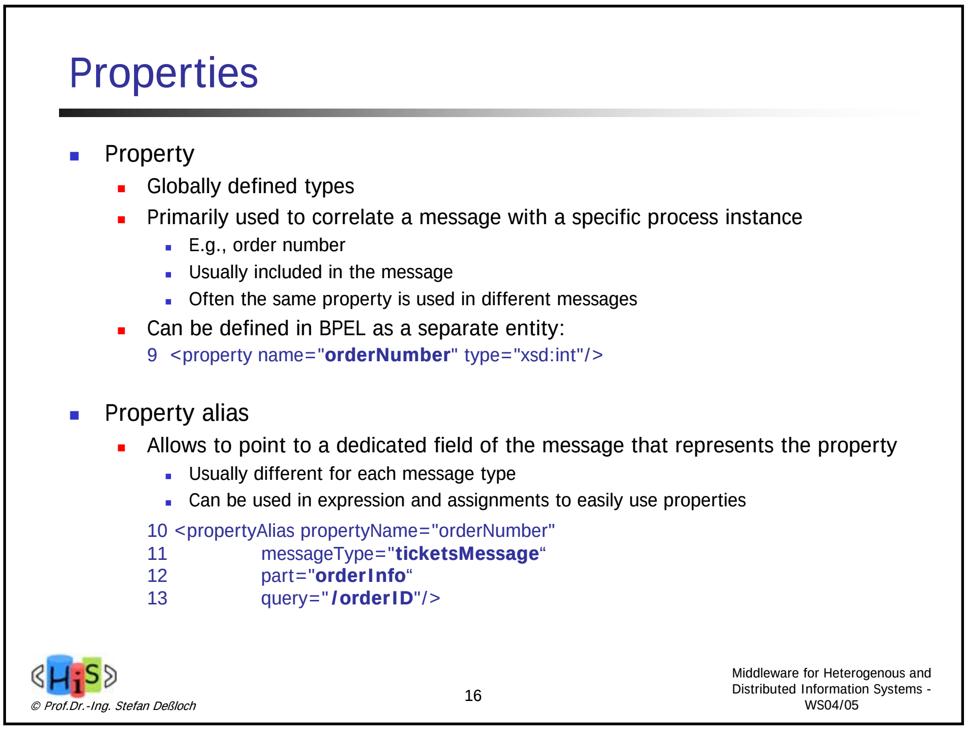


Partners (cont.)

- Definition of partners in addition to partner links
  - optional definition
  - may require the same partner to play multiple roles
    - partner definitions must not overlap
  - Example

```
<partners>
    <partner name="SellerShipper">
              <partnerLink name="Seller"/>
              <partnerLink name="Shipper"/>
    </partner>
</partners>
```

- 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:

```
<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

```
<propertyAlias propertyName="orderNumber"
             messageType="ticketsMessage“
             part="orderInfo“
             query="/orderID"/>
```

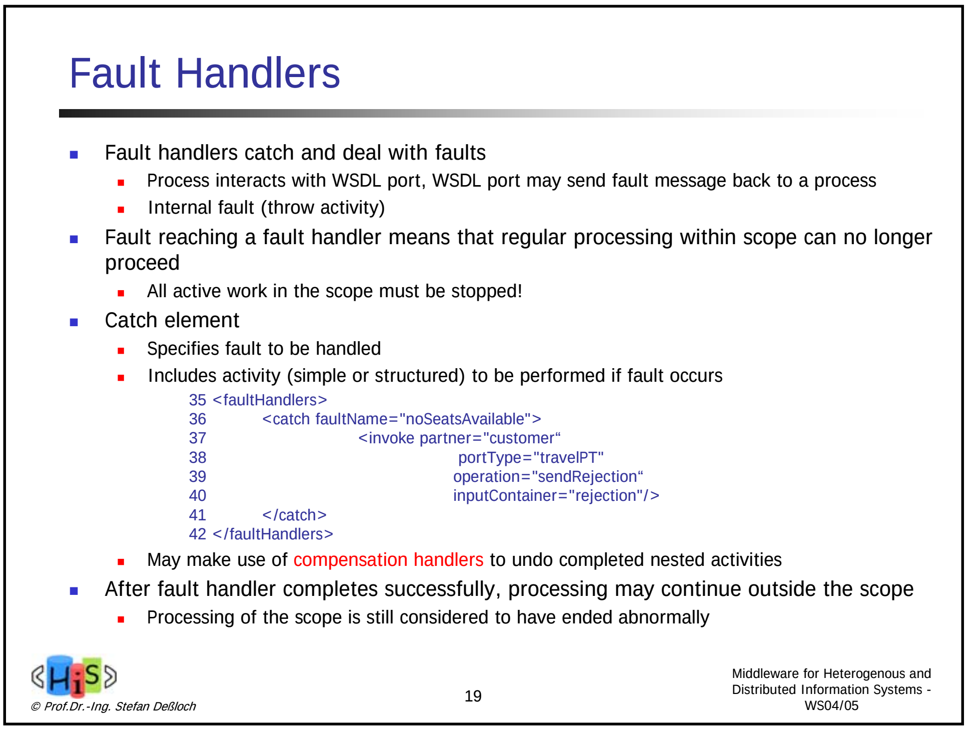
Correlation

- Message needs to be delivered not only to the correct port, but to the correct instance of the business process providing the port
- 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

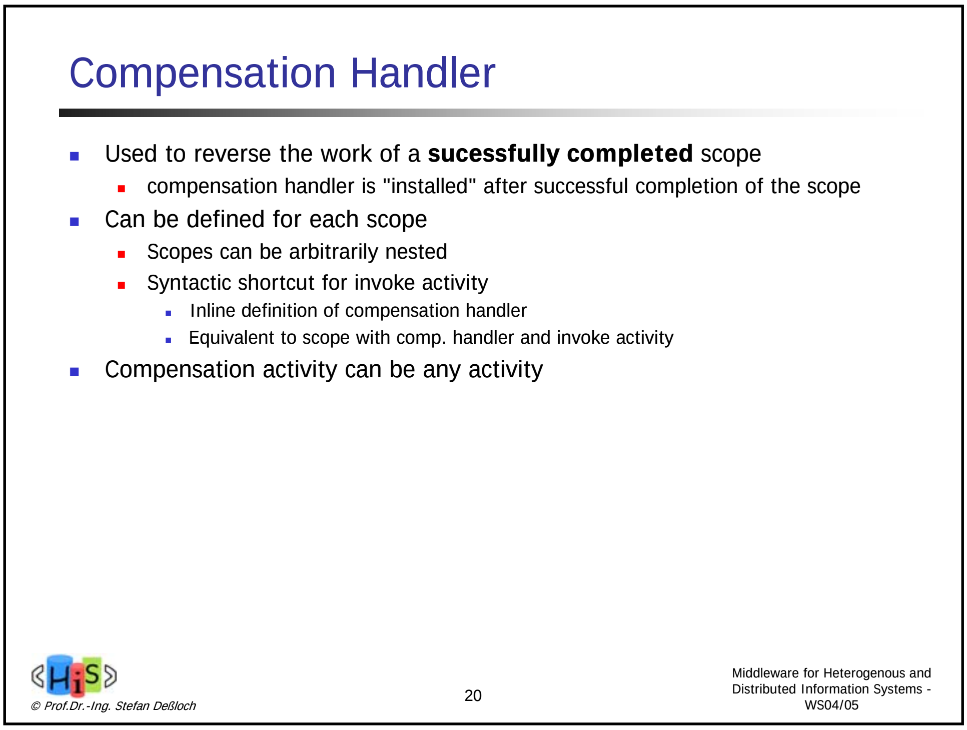


Fault Handlers

- 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

```
<faultHandlers>
        <catch faultName="noSeatsAvailable">
                <invoke partner="customer“
                        portType="travelPT"
                        operation="sendRejection“
                        inputContainer="rejection"/>
        </catch>
</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 **sucessfully completed** scope
  - compensation handler is "installed" after successful completion of the scope
- 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 name="purchase">
   <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

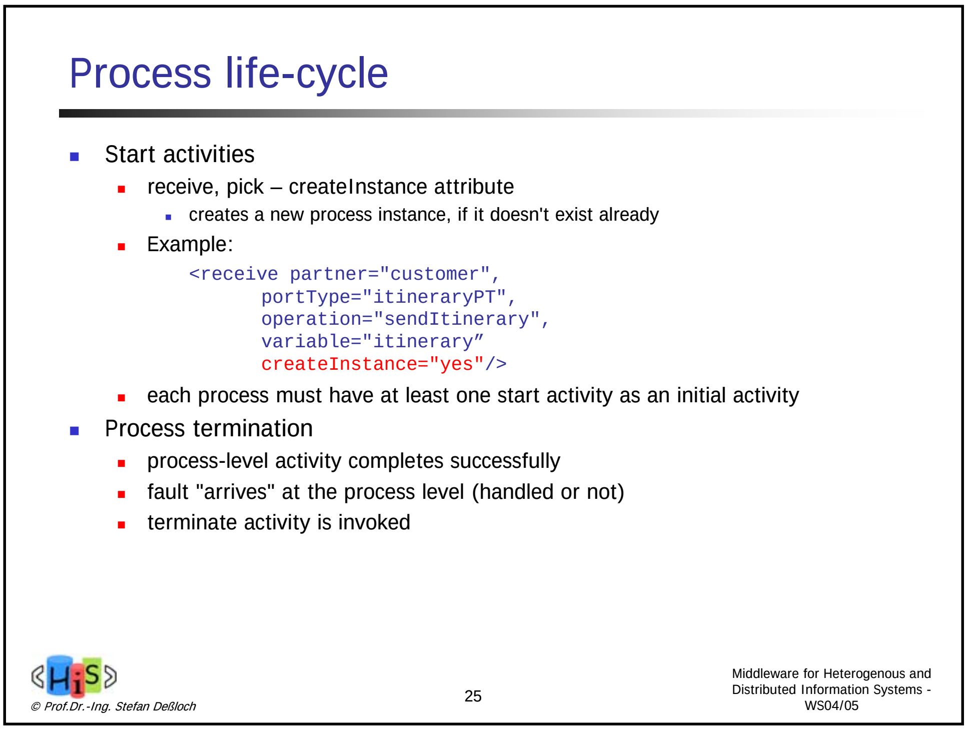
- Compensate activity
  - Invokes compensation handler for named scope
    - Example: <compensate scope="purchase"/>
  - Can be invoked only from the fault handler or compensation handler of the immediately enclosing scope
- Data semantics
  - When invoked, compensation handler sees frozen snapshot of 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 occuring in compensation handlers or fault handlers
  - Can be caught by regular fault handlers in enclosing scopes or scopes with the fault handler

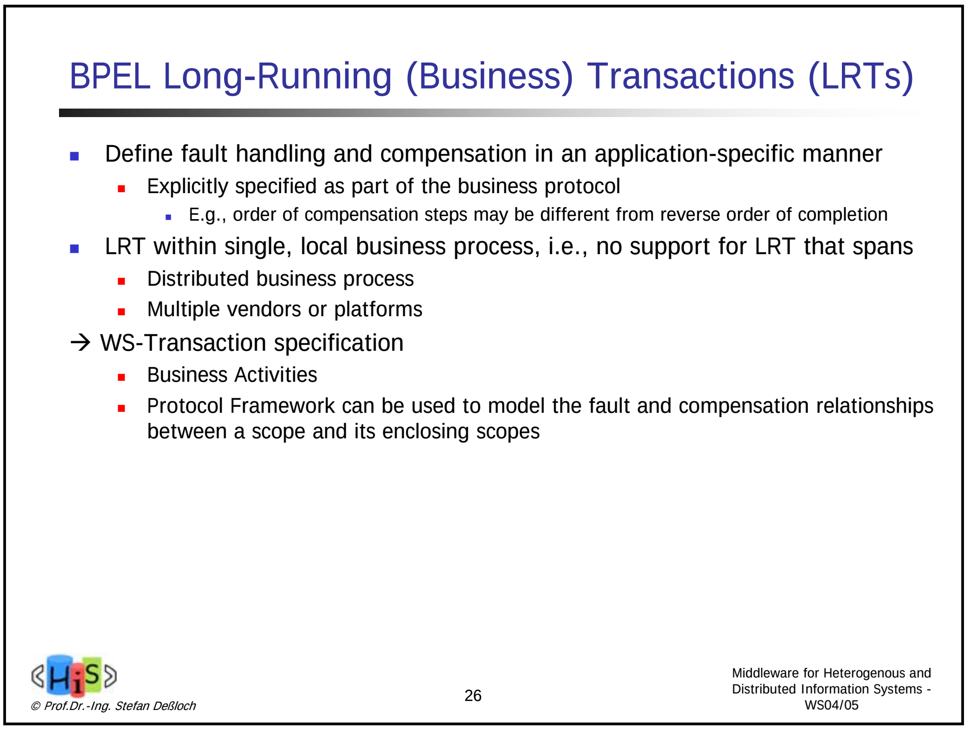


Process life-cycle

- 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



BPEL Long-Running (Business) Transactions (LRTs)

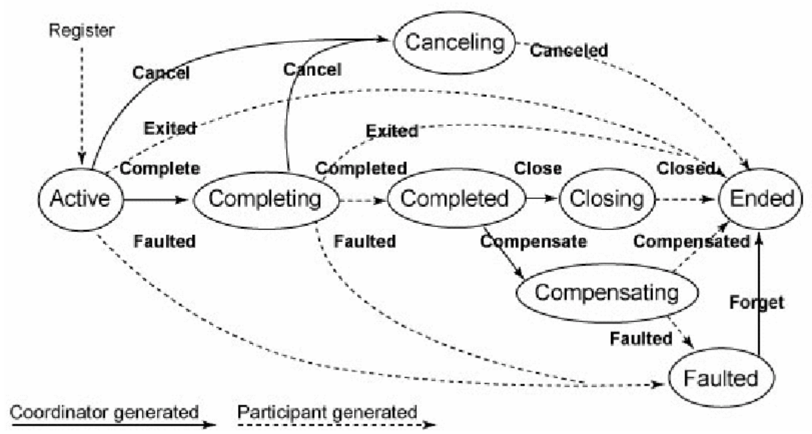
- Define fault handling and compensation in an application-specific manner
  - Explicitly specified as part of the business protocol
    - E.g., order of compensation steps may be different from reverse order of completion
- LRT within single, local business process, i.e., no support for LRT that spans
  - 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


Register
Canceling
Canceled
Cancel
Cancel
Exited
Exited
Complete
Completed
Close
Closed
Active
Completing
Completed
Closing
Ended
Faulted
Faulted
Compensate
Compensated
Compensating
Forget
Faulted
Faulted
Coordinator generated
Participant generated