Lesson 9 – Process languages (Part II)

Service Oriented Architectures

Module 1 - Basic technologies

Unit 5 – BPEL

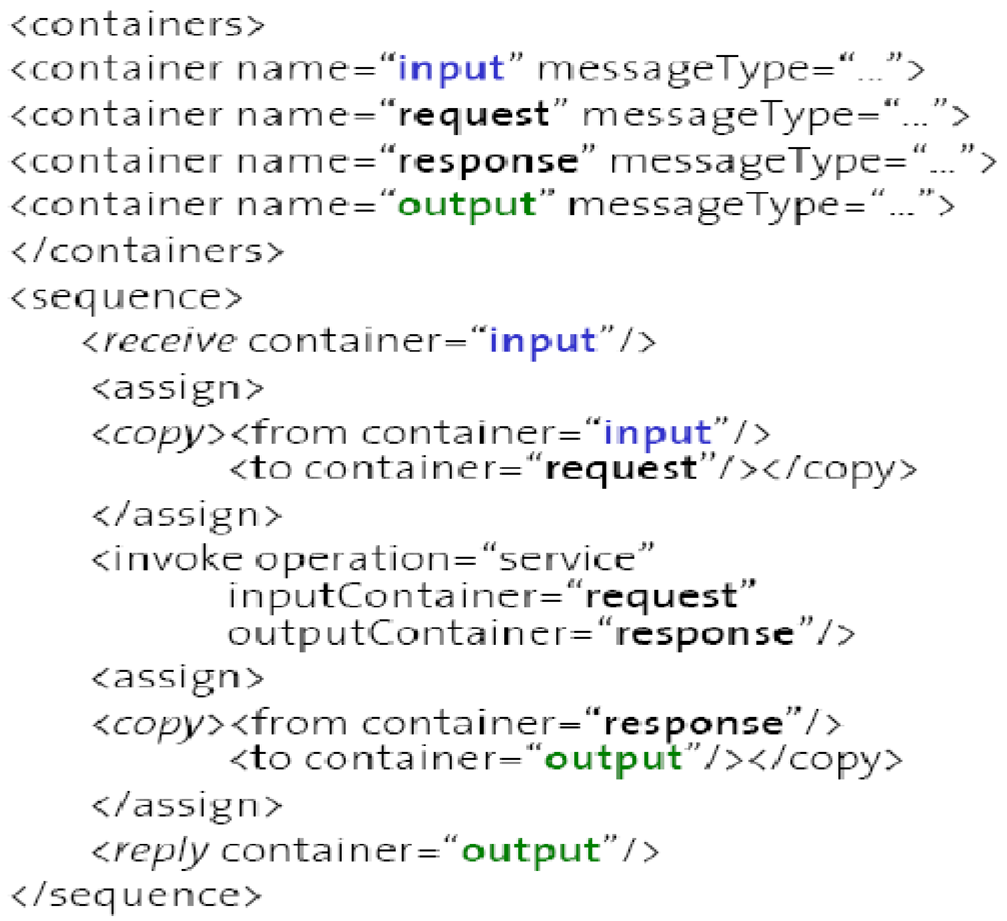
Ernesto Damiani

Università di Milano

Data Transfer model (1)

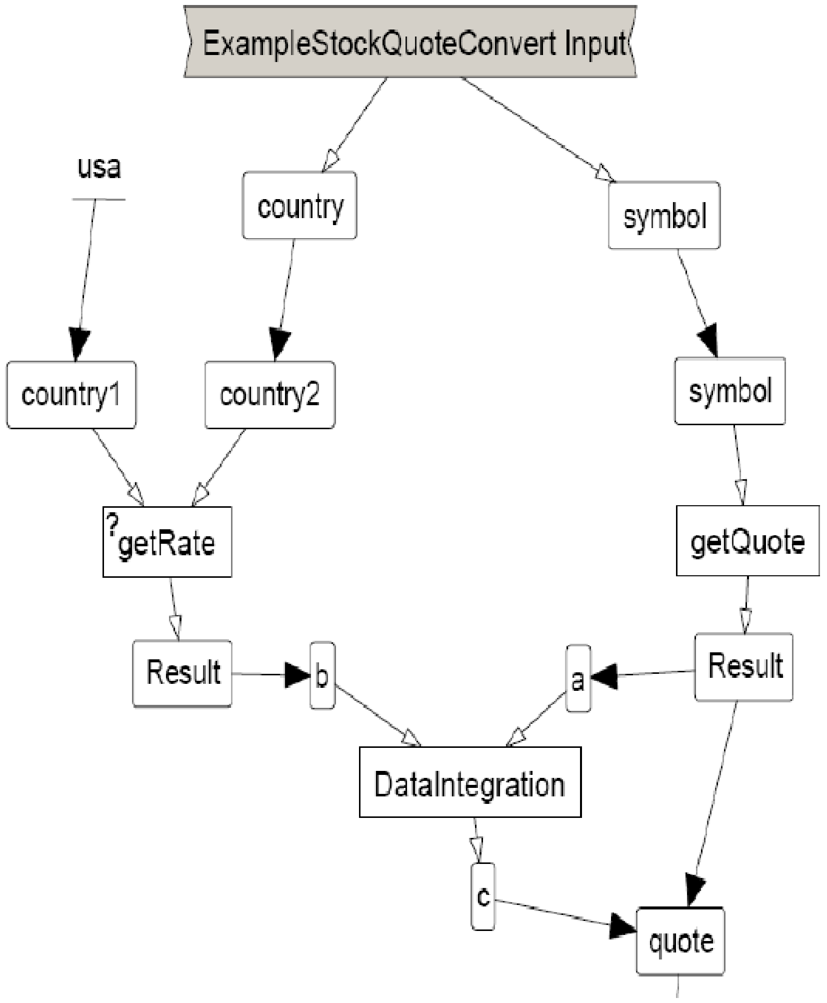
- Data transfers define how (and when) services are supposed to exchange data
  - Whiteboard
    - Service invocations read their input and deposit their results in a collection of variables. Each service invocation defines a mapping to and from the whiteboard. Data can also be copied explicitly among variables
  - Data flow graph
    - Side-effects free, declarative model where services are connected with data flow dependencies, which define the source of their input data which is fetched at the appropriate time

Data Transfer model (2)



```
<containers>
<container name=“input” messageType=“…”>
<container name=“request” messageType=“…”>
<container name=“response” messageType=“…”>
<container name=“output” messageType=“…”>
</containers>
<sequence>
    <receive container=“input”/>
     <assign>
     <copy><from container=“input”/>
           <to container=“request”/></copy>
     </assign>
     <invoke operation=“service”
           inputContainer=“request”
           outputContainer=“response”/>
     <assign>
     <copy><from container=“response”/>
           <to container=“output”/></copy>
     </assign>
     <reply container=“output”/>
</sequence>
```

Dataflow graph

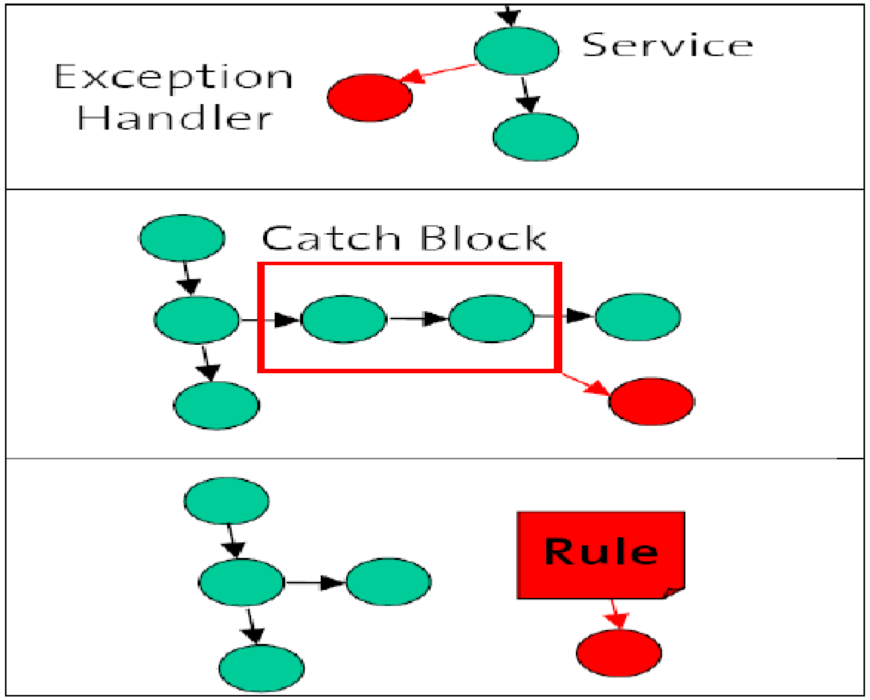


Exception handling (1)

- Important as service compositions should explicitly model what to do if a service is not available (timeout) or if its invocation fails
- Flow-based exception handling uses normal control flow constructs to branch after a service invocation has failed
- Try/Catch blocks are used in a similar way to associate an exception handler to a set of service invocations

Exception handling (2)

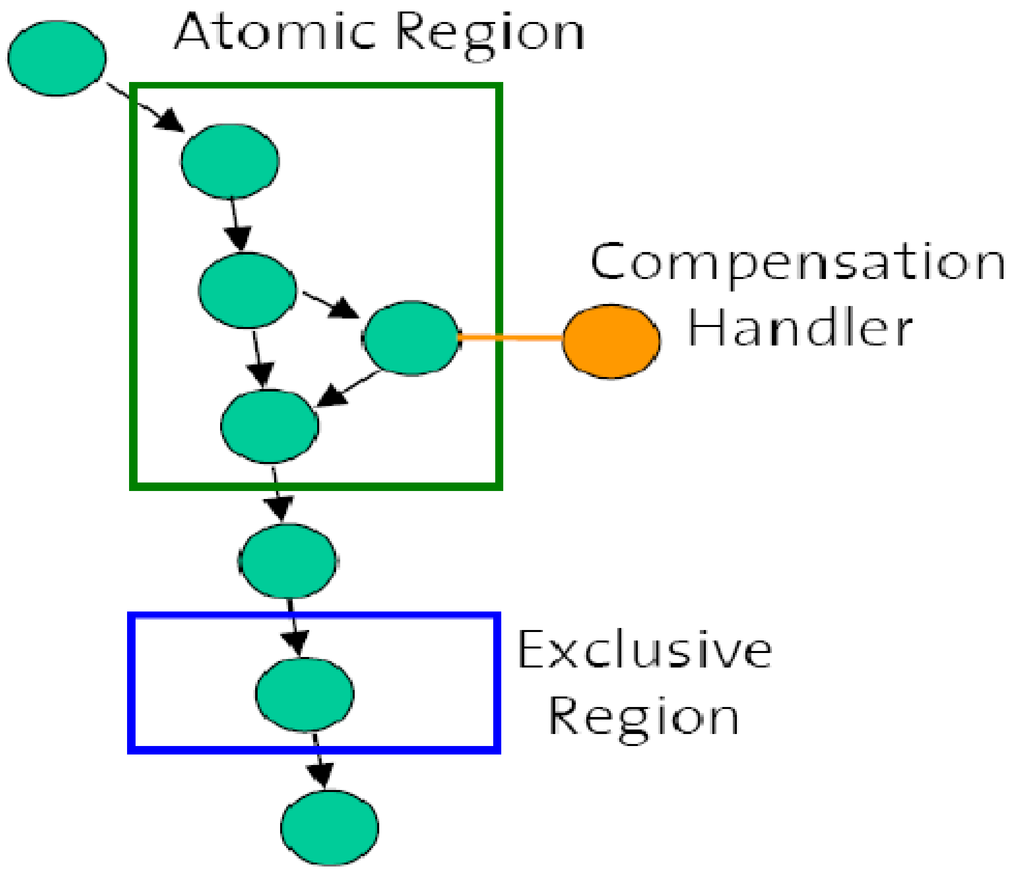
- Rules may also be associated to a composition in order to detect global exceptional conditions



Transaction (1)

- Transactional behavior is modeled by grouping service invocations and declaring the atomicity and isolation properties for the group
- In order to support long running transactions, each service operation can be also associated with a compensation handler, which is invoked only if the operation should be undone
- When no failures occur, the composition engine runs a 2PC protocol with all services of the atomic region. If a failure occurs, the engine invokes the compensation handlers of the services which could be invoked successfully

Transaction (2)



WS BPEL (1)

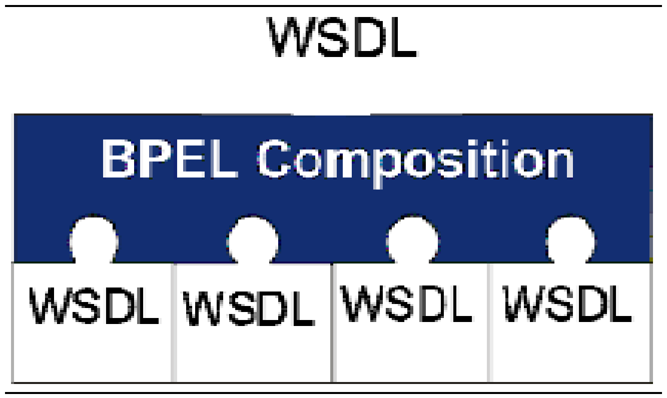
- WS-BPEL is a standard proposal for specifying business process behavior based exclusively on Web services
- WS-BPEL is a language based on the XML syntax
- It does not directly deal with implementation of the language but only with the semantics of the primitives it provides
- The latest version of the specification is 2.0

WS BPEL (2)

- Originated as the fusion of XLANG (Microsoft) and WSFL (IBM) = as a result, there are formal problems with the language
- The language is used to define
  - executable processes, with the actual interactions among different services. These processes can implement the internal business logic of a Web service (Composition)
  - abstract processes, modeling the messages exchanged by the parties involved in a business protocol without revealing details about the internal implementation (Coordination)
- The goal is to define coordination and composition of Web services in a portable way

Composing Executable Processes

- Executable Processes describe the composition (or orchestration) of different Web service interfaces (WSDL Port Types)
- The result of a composition using BPEL is meant to be recursively published as a Web service

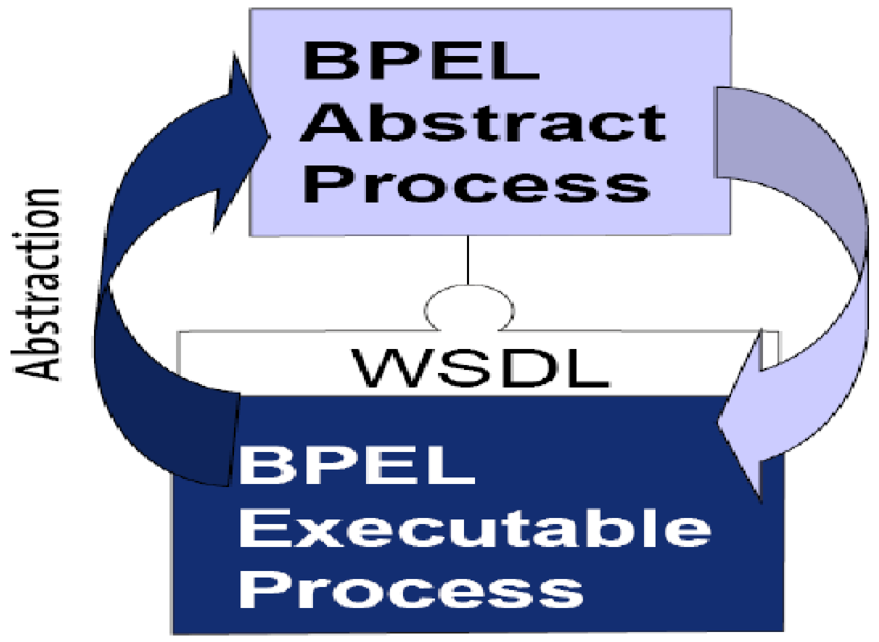


Using Abstract Processes (1)

- Abstract Processes define constraints on the WSDL interface of a Web service so that it can be used correctly
- Application Example: RosettaNetPIPs(Partner Interface Processes) standardize interfaces and protocols along an e-Commerce supply chain
- The abstract (public) process for a Web service can be generalized from the concrete (private) executable implementation, if this is constructed using BPEL

Using Abstract Processes (2)

- The abstract process definition can also drive the implementation of the private executable process behind a Web service



Elements of WS-BPEL (1)

- Partners
- Properties
- Correlation sets
- Variables Scopes
- Fault Handlers
- Compensation Handlers
- Event Handlers
- Activities
  - Structured activities
  - Simple activities

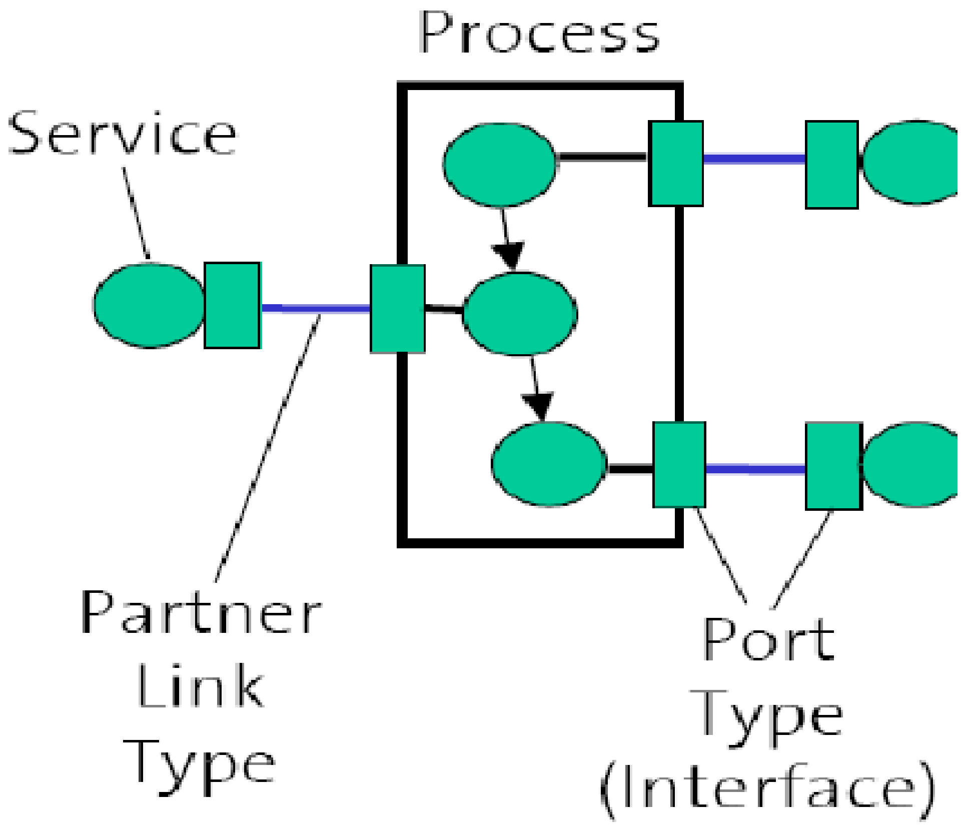
Elements of WS-BPEL (2)

- These elements are equivalent to declarations in a normal programming language. They define the way services are to be called, which data is to be used and which data is to be treated as stateful
- These elements establish what the process does, how it reacts under different circumstances (errors, message arrivals, events, etc.), and how data flows from one step to the next

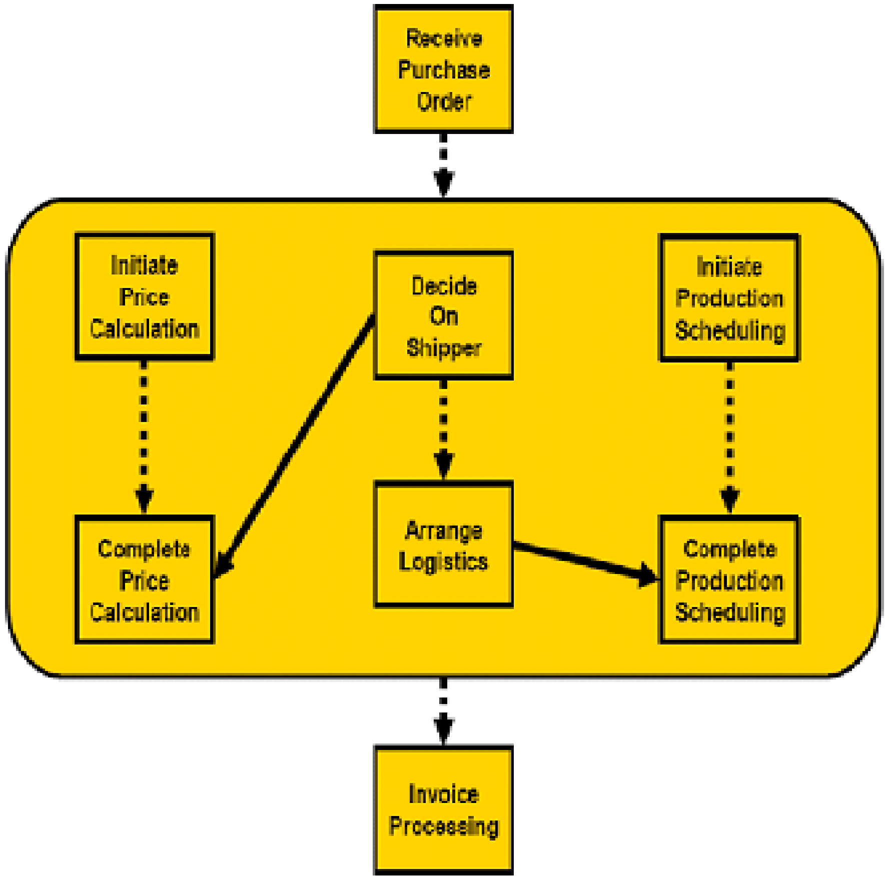
Partner Link Types (1)

- The concept of partners is used to define the Web services that are to be invoked as part of the process. It is based on three elements:
  - Partner Link Type: it contains two PortTypes(WSDL), one for each of the roles in the partner entry (i.e., one portTypebelongs to the process itself, the other one is the portTypeof the service being invoked). Partner link types are not stored in a process, but usually extend a WSDL document
  - Partner Link: the actual link to the service. This is where the actual assignment to a binding is made (outside the scope of BPEL). Several partner links may share the same partner link type
  - Partners: a group of Partner Links (this is an optional element). A partner link can only appear in one partner

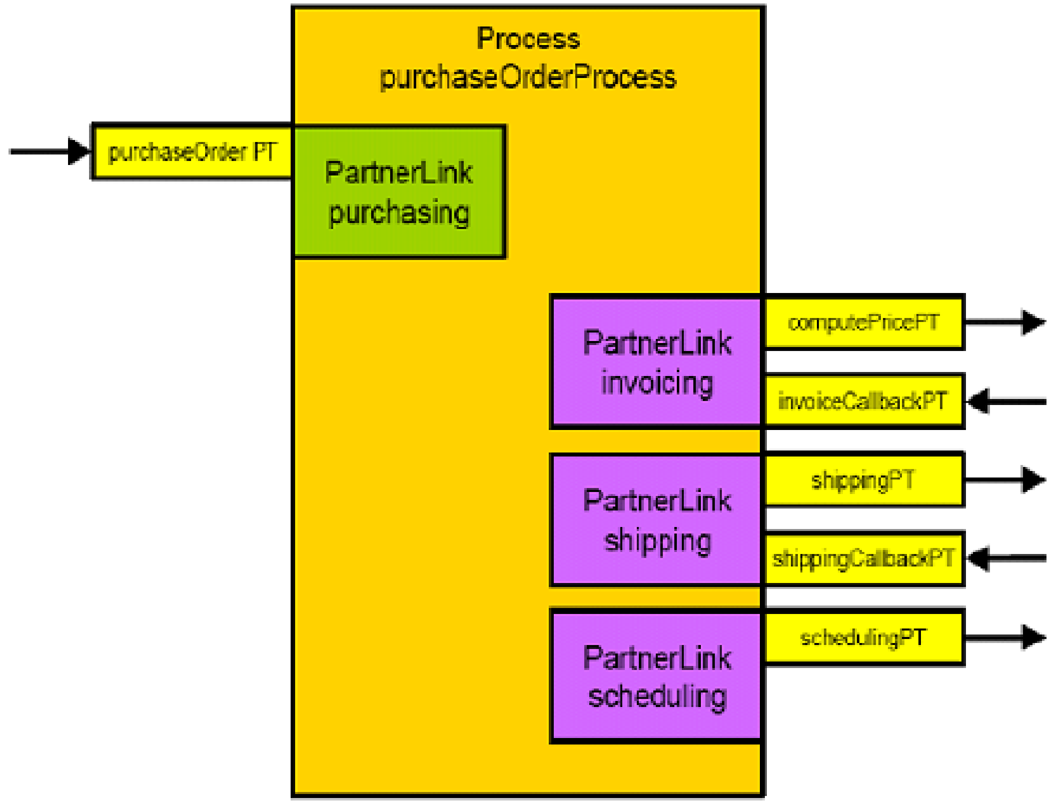
Partner Link Types (2)



Example


Receive Purchase Order
Initiate Price Calculation
Decide On Shipper
Initiate Production Scheduling
Complete Price Calculation
Arrange Logistics
Complete Production Scheduling
Invoice Processing

Example Partner Links



Properties (1)

- Properties give a global and abstract definition of data elements which are intended to be used to correlate messages with process instances. (e.g., order numbers)
- Property aliases map such properties to specific message types. This ensures that the same property can be reused across different messages

Properties (2)

- Correlation sets are a named group of properties used to uniquely identify a stateful conversation that is handled by a process. They define a mapping among data, messages and properties that help a process instance identify the messages that should be handled by itself

Correlation sets (1)

- Correlation is used when receiving asynchronous messages
- Correlation sets are referred from activities which involve the exchange of a message with external partners
- The content of each message is checked against the correlation set to establish the link between the message and the corresponding process instance

Correlation sets (2)

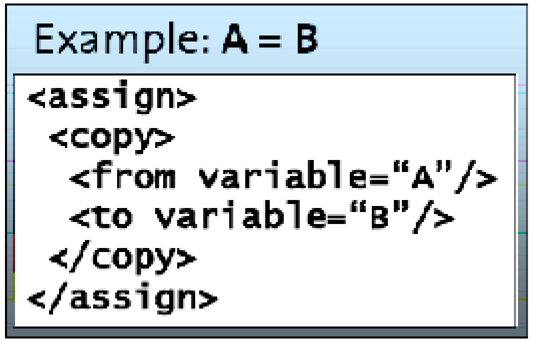
- A correlation set is initialized once the first message of the conversation is exchanged and cannot be changed during the rest of the interaction
- Correlation properties must be set to unique values among all process instances

Variables and assignment (1)

- Variables can be used to store:
  - the content of the SOAP messages that are exchanged with the partners (with message Typesdefined as part of the WSDL interface description)
  - intermediate, temporary data used in the business logic of the process to generate messages
  - private data that holds the internal state of the process (e.g., counters)
- Variables are referenced by activities which exchange messages in order to define from where to read the content of a message and where the received message should be stored

Variables and assignment (2)

- The entire content of variables (or only parts) can also be copied from a different variable with an <assign> activity
- The <assign> activity can also be used to assign constant values and apply XPathqueries to messages in order to extract relevant information


Example: **A = B**

```
<assign>
 <copy>
  <from variable="A"/>
  <to variable="B"/>
 </copy>
</assign>
```

Fault handling

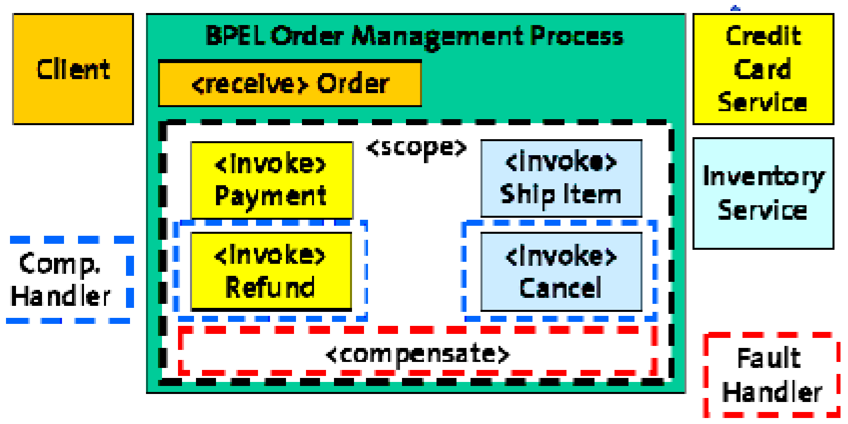
- A form of block-based exception handling that uses <catch> blocks to handle the receipt of fault messages or explicit exceptions (<throw>). Once an exception is caught, the necessary logic for correcting failures can be programmed into the business process
- Fault handlers are a basic component for fault tolerance and fulfil the same role as exception handlers in normal programming languages. Every process needs to have this type of handlers

Compensation

- Additionally, if a failure occurs, already (and successfully) completed activities can be undone by using BPEL’s advanced rollback capabilities
- Compensation handlers are a legacy of advanced transaction model sand map very well to the notion of transactional coordination espoused by WS-Coordination and WS-Transactions
- Compensation handlers assume the process contains the entire logic of what is to be done, something that it is rarely true in complex processes

Compensation handling example

- An order management process charges the customer using a credit card service and removes the ordered item from the inventory service
- If either of the two services fail, the other should be compensated



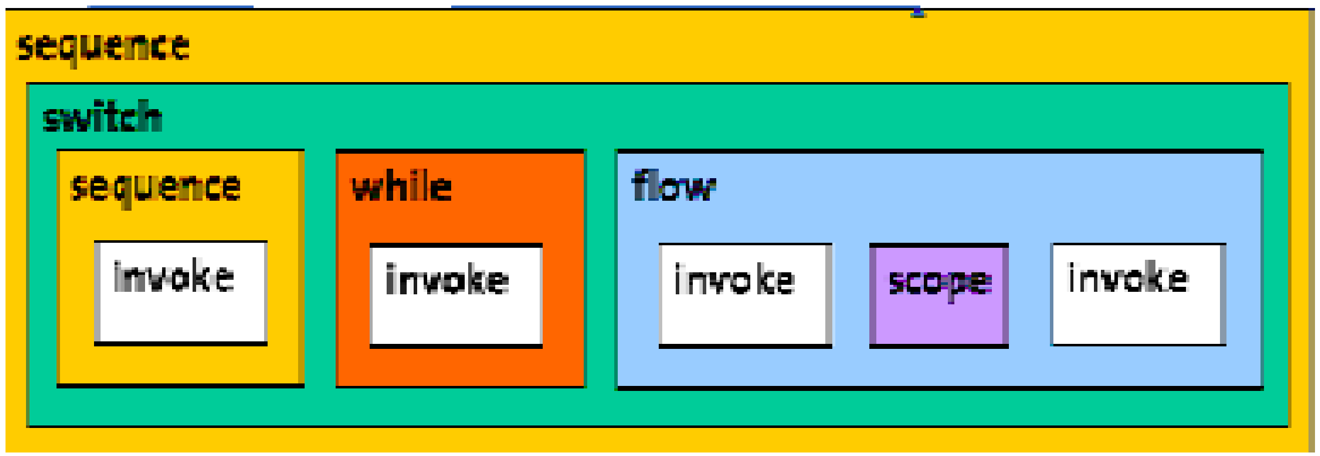
Simple activities

- The actual operations the process will complete:
  - <receive> blocks until a message is received
  - <reply> sends a message in response to a received message
  - <invoke> sends a message to invoke an remote operation
  - <assign> updates the value of variables
  - <wait> suspends execution for a given time period
  - <empty> no-op used for synchronization purposes
  - <terminate> finishes the process
  - <throw>, <rethrow> raises a fault for a fault handler to catch
  - <catch>, <catchAll> catches a fault of a given type
  - <compensate> undo the effects of completed activities

Structured activities

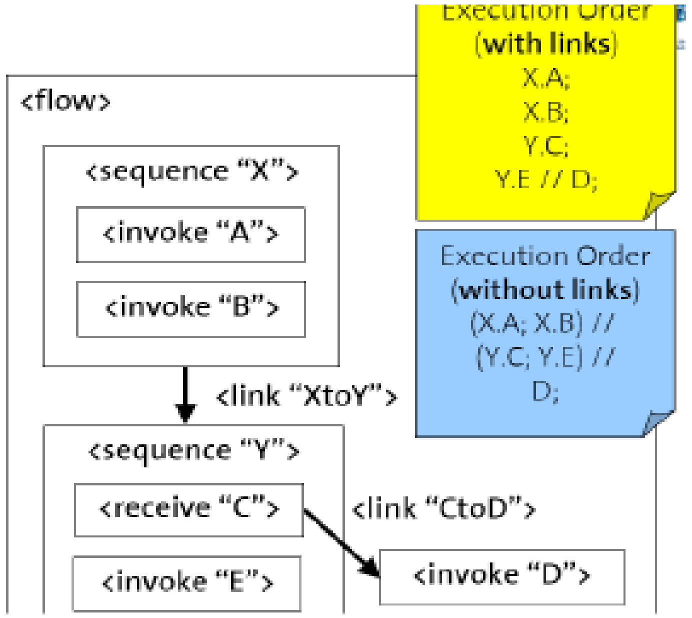
- Define the control flow dependencies in a hierarchical manner by nesting the following elements:
  - <sequence> executes a set of activities one after another
  - <switch> chooses between a set of activities (v1.1)
  - <while> repeats depending on certain conditions
  - <flow> executes a set of parallel activities (with arbitrary control flow dependencies)
  - <pick> waits for alternative messages or an alarm and branches according to the one that arrived first
- <scope> defines a block of activities

Control flow



Control flow links

- <flow>
- <links>
- <link name="XtoY"/>
- <link name="CtoD"/>
- </links>
- <sequence name="X">
- <source linkName="XtoY"/>
- <invoke name="A" .../>
- <invoke name="B" .../>
- </sequence>
- <sequence name"Y">
- <target linkName="XtoY"/>
- <receive name="C">
- <source linkName="CtoD"/>
- </receive>
- <invoke name="E" .../>
- </sequence>
- <invoke name="D">
- <target linkName="CtoD"/>
- </invoke>
- </flow>



Extensions (1)

- BPEL Extensions
  - Several extensions have been proposed to deal with different aspects that are not supported by the current 2.0 specification
- BPELJ
  - Java Code Snippets can be embedded into the BPEL process definition
  - These are used for expressing complex branching (and loop) conditions, variable initializations and message transformation
  - This extension also defines how a BPEL process can directly call J2EE components

Extensions (2)

- BPEL4Sub-Processes
  - Modularize and reuse process definitions. Define how to call a process from another one.
- BPEL4PEOPLE
  - Human tasks and human workflow support
  - The <invoke> activity are tagged with the <staff> element to model the invocation of the services provided by a human resource
  - Similar to traditional workflow management systems, the invocation is qualified by describing the organizational role of the user interacting with the process