
Business Process Execution Language for Web Services (BPEL)

Mark Lewis-Prazen
Web Services
Fall, 2006

Outline

- Brief History of BPEL
 - Context
 - Definition
 - Composition vs. Choreography
 - Examples – Templates
 - Examples – Business
 - Implementations
 - Limitations/Strengths/the Future
-

BPEL – A Brief History

- BPELWS (Business Process Execution Language for Web Services) is a process modeling language.
 - Developed by IBM, Microsoft, and BEA
 - Version 1.1, 5 May 2003
 - It supersedes XLANG (Microsoft) and WSFL(IBM).
 - It is built on top of WSDL.
 - For descriptions of what services do and how they work, BPELWS references port types contained in WSDL documents
 - Today BPEL represents the widely accepted standard in web service composition.
 - BPEL supports the specification of both composition schemas and coordination protocols, taking advantage of their similarity.
-

BPEL Development

- WSDL 1.1, XML Schema 1.0: data model
 - XPath 1.0: data manipulation (default)
 - Microsoft, IBM, Siebel Systems, BEA Systems, SAP
 - Version 1.1: 5/03 (supercedes v1.0, 8/02)
 - Standardization: OASIS Web Services Business Process Execution Language - v2.0 – Draft - *WSBPEL* – OASIS
-

Brief Definition & Purpose

- A language, grammar, or notation for specifying business process protocol and behavior based on Web Services
 - An abstract XML description that exports/imports functionality via Web Service interfaces
 - Executable language to define control logic for coordinating Web Services
 - Strong roots in traditional flow models
 - To model business interactions as a workflow
 - An aggregation/orchestration of Web services
 - Cross-enterprise automated business
-

BPEL - Overview

- BPEL is a **block-structured programming language**, allowing recursive blocks but restricting definitions and declarations to the top level.
 - The language defines **activities** as the basic components of a process definition.
 - Structured activities prescribe the order in which a collection of activities take place.
 - Ordinary sequential control between activities is provided by **sequence**, **switch**, and **while**.
 - Concurrency and synchronization between activities is provided by **flow**.
 - Nondeterministic choice based on external events is provided by **pick**.
-

BPEL Overview

- The implementation logic for a service is defined in XML documents by *executable* and *abstract* process specifications.
 - Such a process consists of:
 - *roles* taking part in the message in message exchanges
 - *port types* supported by the roles and the process itself
 - *orchestration* and other aspects of process definition
 - *correlation information*, how to route messages.
 - BPEL assumes processes and partners to be WSDL messages with specified port types and operations.
-

Types of Processes

- Abstract process (public):
 - Business protocol/process definition: potential execution order of operations from a collection of Web services; data shared between these Web services
 - WSDL-based interface: operations allowed, messages exchanged, partners involved; allow interoperability between the process and other Web services
 - Executable process/application (private):
 - Model actual behavior of participants in a business interaction (e.g., event handling, branching)
 - Private workflows conveying internal details (logic, state)
 - Lifecycle
-

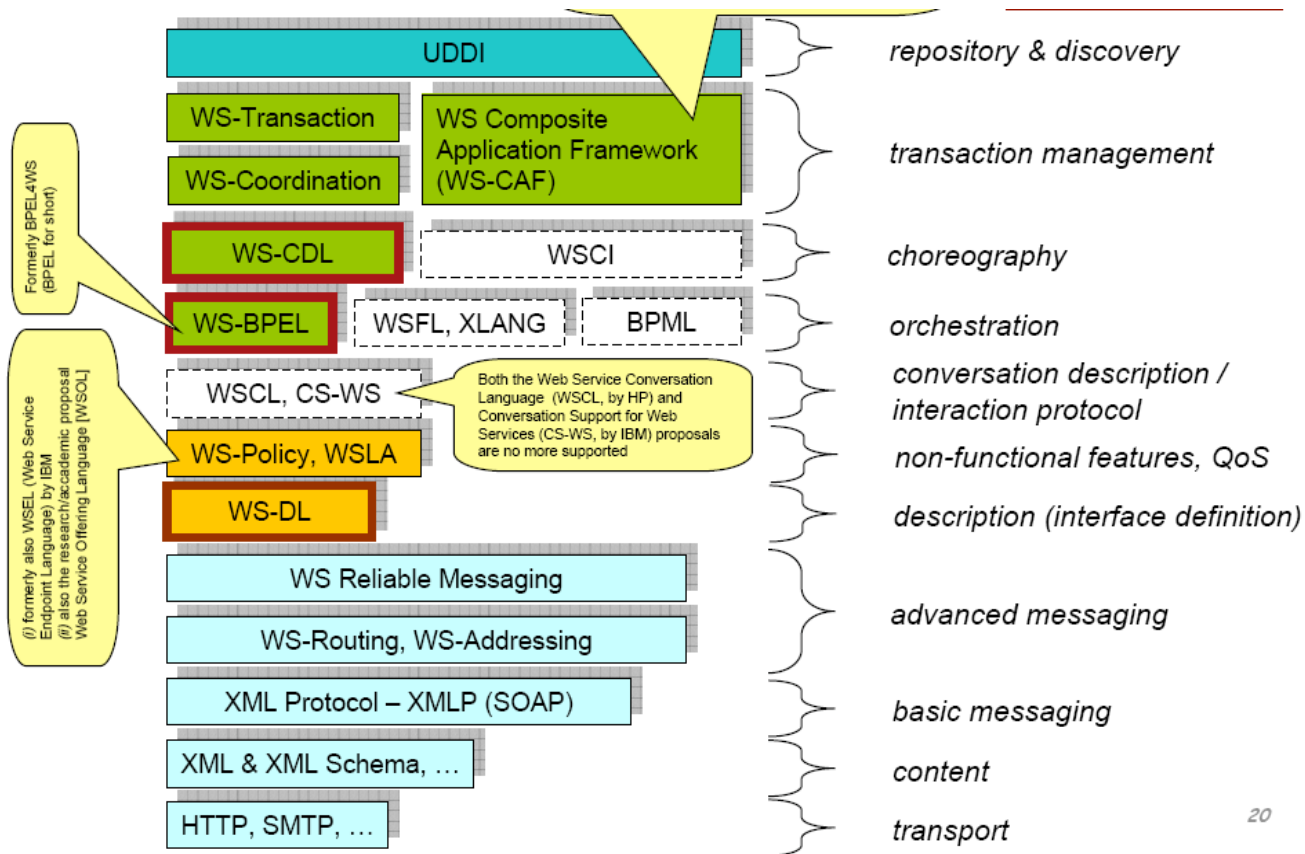
WSDL and BPEL

- WSDL interface defines:
 - Public entry and exit points
 - Specific operations allowed
 - Data types and messages to describe the information that passes between process requests
 - BPEL describes how to:
 - Sequence operations
 - Control logic and state
 - Coordinate interactions between the process and its partners
-

WSDL-Based Web Services Stack

Discovery	UDDI	
Bus process/workflow	BPEL	BPML
Transactions	WS-Transaction	BTP
Choreography	WS-Coordination	WSCI
Conversations	CS-WS	
Nonfunctional description	WSEL	
Service description	WSDL	
XML-based messaging	SOAP	
Network/transport	HTTP, FTP, SMTP, etc.	

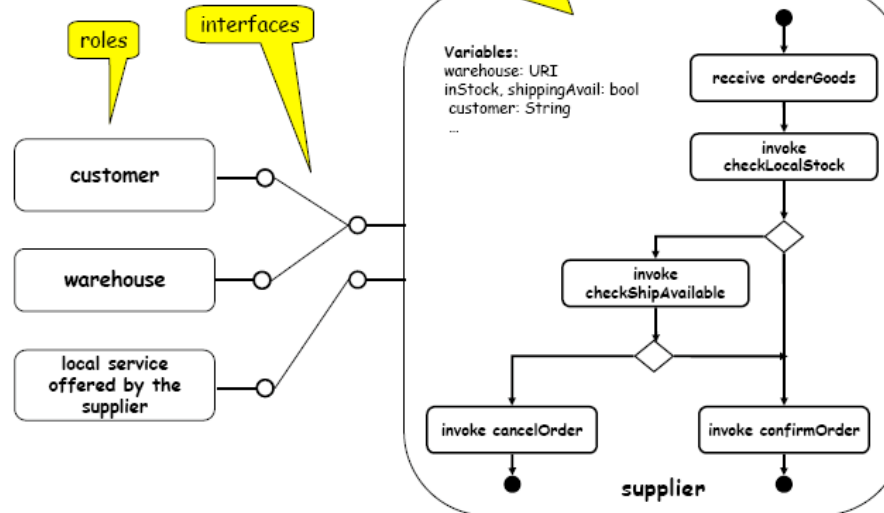
The WSDL Stack - Revised



BPEL Specification

An XML document specifying

- Roles exchanging messages with the composite service/process
- The (WSDL) interfaces supported by such roles
- The orchestration of the process
 - Variables and data transfer
 - Exception handling
 - Correlation information



BPEL Composition/Component Model

- Deals with the implementation of an application (then offered as a service) whose application logic then involves the invocation of operations offered by other services
 - New service is the composite service
 - Invoked services are component services
-

BPEL Composition Model

- Specifying the order of service invocations, depending on conditions
 - Need for abstraction models and languages for such descriptions:
 - activity diagrams
 - statecharts
 - petri-nets
 - pi-calculus
 - activity hierarchies
 - rule-based orchestration approaches
-

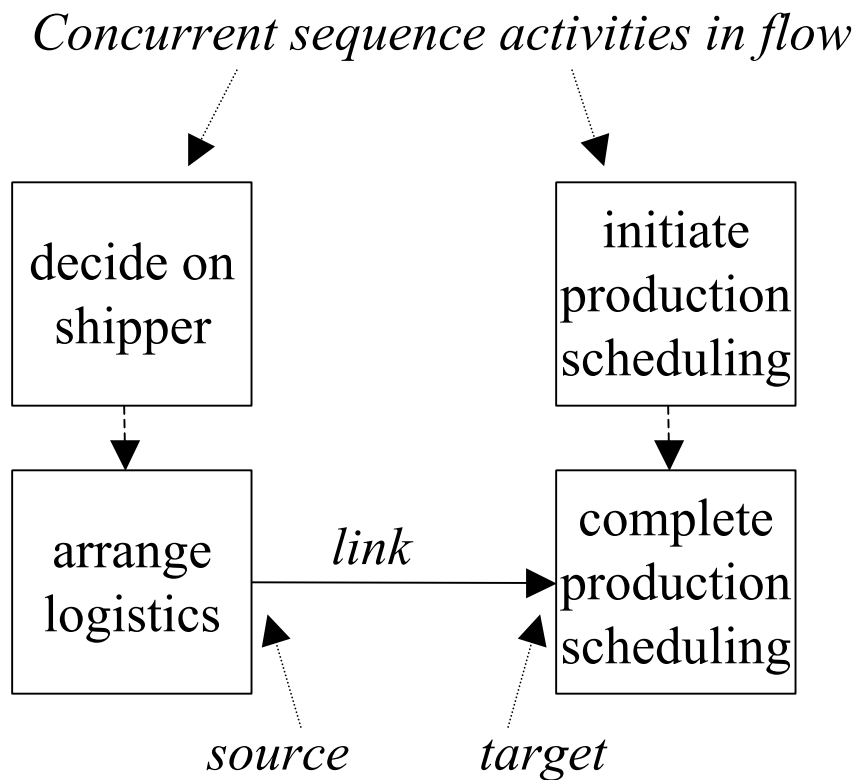
Composition Schema

- Specifies the “process” of the composite service – the workflow of the service
 - Different clients interact with the service, thereby satisfying their specific needs
 - Specific execution of the composite schema for a given client is an orchestration instance
-

BPEL Orchestration Model

- ... combines the activity diagram and the activity hierarchy approaches.
 - Ordering constraints among activities are defined by structured activities (grouping other basic or structured activities):
 - *Sequence*: definition of an arranged sequence
 - *Switch*: ... like in C, Java; branching according to a conditions/case
 - *Pick*: execution of an activity upon a specific associated event
 - *While*: repeated execution of an activity while a condition is “true”
 - *Flow*: parallel execution of activities; completes after each activity has terminated. (from WSFL)
 - Flows can be combined with *links* to connect source and target activities (supporting multiple incoming and outgoing links, as well a associated conditions and *join*-constructs).
-

BPEL Orchestration Model

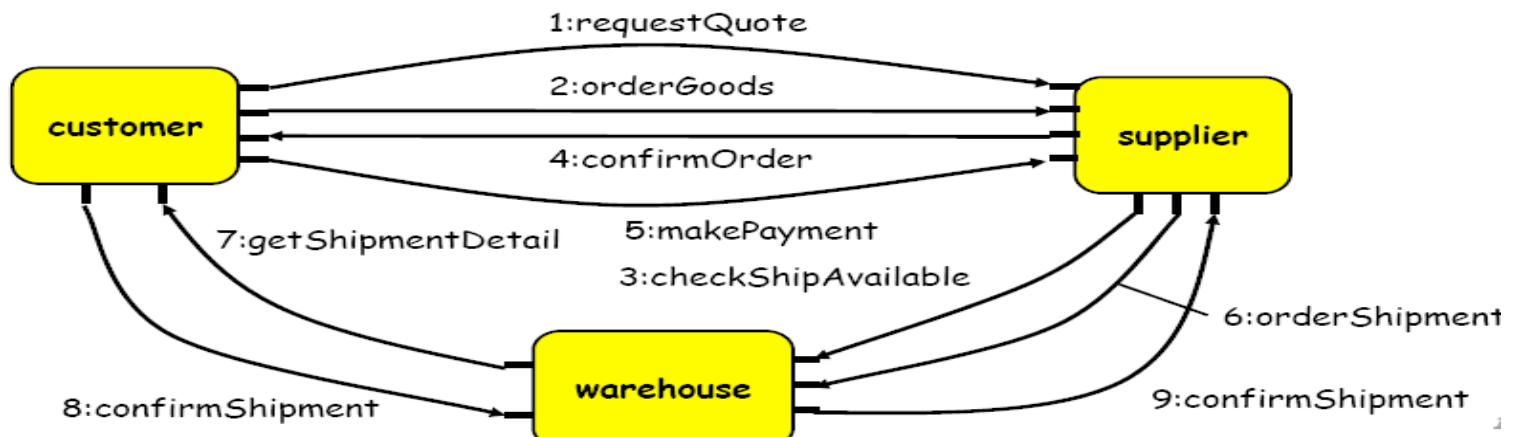


BPEL Orchestration Model



Orchestration (Choreography)

- Coordination of conversations of N peer conversations in terms of
 - Roles
 - Message exchanges
 - Constraints on order



Composition vs. Choreography

- Composition is about implementing new services
 - Choreography is about global modelling of N peers to prove correctness and run-time discovery of possible partners
 - Composition schema of A dictates coordination protocols that it can support (i.e. the choreographies it can participate in) ; Without the information provided by the coordination protocol the order in which operations have to be invoked while executing a composite web-service would be difficult to determine.
 - While specifications of composite services are usually kept private, coordination protocols on the other hand are available to the public and often also advertised by means of **Web service registries**.
-

Coordination Protocols and Composition Schemas

- The definition of a **protocol** imposes constraints on the **composition** schema of the Web service implementing it.
 - Protocol definitions are often used to guide the design of composition schemas, but the inverse route, building the role – specific protocol from the internal process is also possible.
 - Process oriented languages take different routes in how they address **composition, protocols** and the **relationships** between them
 - **Workflow languages**
 - focus on internal implementation and provide minimal or no support for abstract processes.
-

Features

- Partner links and roles (model the business relationship)
 - Sequence vs. parallel flow
 - Synchronization of concurrent flows (source and target specification)
 - Event handlers: asynchronous msgs, errors/faults
 - Data handling: variables, property definition, data extraction and assignment
 - Scope (group a set of activities)
-

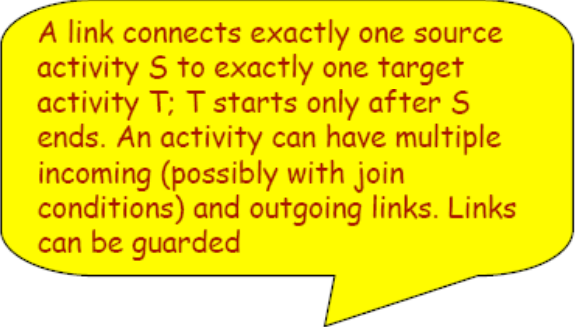
Basic Structure

```
<process>  
  <partners>  
    <partnerLink name='ncname' myRole='ncname'  
      partnerRole='ncname' />  
  <variables>  
  <correlationSets>  
  <faultHandlers>  
  <compensationHandler>  
  <eventHandlers>  
    <onMessage> | <onAlarm>
```

ACTIVITIES

Process Model - Activities

- Primitive
 - **invoke**: to invoke a Web Service (in-out) operation
 - **receive**: to wait for a message from an external source
 - **reply**: to reply to an external source message
 - **wait**: to remain idle for a given time period
 - **assign**: to copy data from one variable to another
 - **throw**: to raise exception errors
 - **empty**: to do nothing
- Structured
 - **sequence**: sequential order
 - **switch**: conditional routing
 - **while**: loop iteration
 - **pick**: choices based on events
 - **flow**: concurrent execution (synchronized by **links**)
 - **scope**: to group activities to be treated "transactionally" (managed by the same fault handler, within the same transactional context)



A link connects exactly one source activity S to exactly one target activity T; T starts only after S ends. An activity can have multiple incoming (possibly with join conditions) and outgoing links. Links can be guarded

Basic Activities

- Basic activities—instructions that interact with an external entity:
 - receive (incoming message)
 - reply (outgoing response or notification message)
 - invoke (call a web service)
-

Structured Activities

- Structured activities—manage overall process flow and define underlying programming logic:
 - ordering: sequence, flow
 - conditional looping: while
 - dynamic branching: switch
-

BPEL Handlers and Scopes

- A **scope** is a set of (basic or structured) activities.
 - Each scope can have two types of **handlers** associated:
 - **Fault handlers.** Many can be attached, for different fault types.
 - **Compensation handlers.** A single compensation handler per scope.
-

How Handlers Work

- A compensation handler can reverse the work performed by an already completed scope
 - A compensation handler **can only** be invoked by the fault handler or compensation handler of its immediate enclosing scope
 - A fault handler defines alternate execution paths when a fault occurs within the scope.
 - Typical scenario:
 1. Fault is thrown (returned by invocation or explicitly by process)
 2. Execution of scope is terminated
 3. Appropriate fault handler located
 4. Main execution is compensated to “undo” business effects of unfinished work.
-

Correlation Defined

- BPEL can model many types of interactions:
 - simple stateless interactions
 - Stateful, asynchronous interactions.
 - Correlation sets provide support for the stateful types:
 - CSs represent the data that is used to maintain the state of the interaction
 - At the process end of the interaction, CSs allow incoming messages to reach the right process instance.
 - What is a correlation set?
 - A set of business data fields that capture the state of the interaction (“correlating business data”). For example: a “purchase order number”, a “customer id”, a sales invoice, etc.
 - Each set is initialized once
 - Sets values do not change in the course of the interaction.
-

Summary of Constructs

<receive>

<reply>

<invoke>

<variable>

<assign>, <copy>

<catch>, <throw>

<terminate>

<wait>

<empty>

<sequence>, <flow>

<links>, <link>, <source>, <target>

<switch>, <case>, <otherwise>

<while>

<pick>

<onMessage>

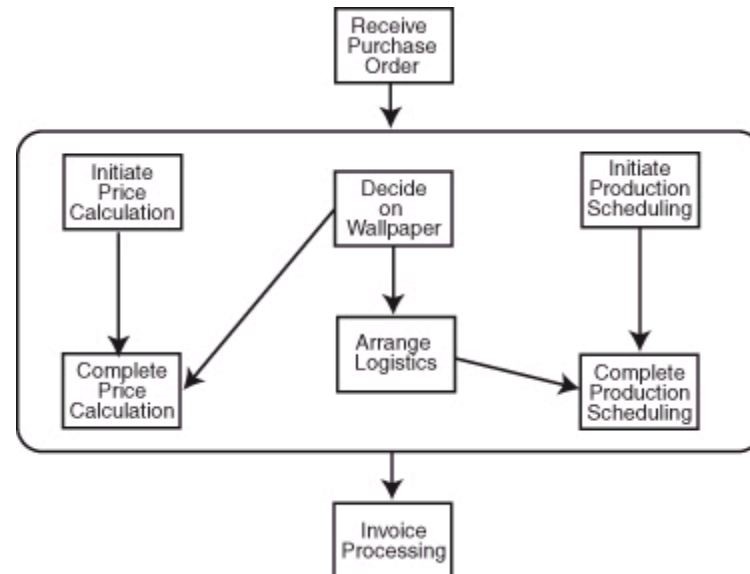
<onAlarm>

<scope>

<compensate>

<correlations>

Example



Example

```
<definitions targetNamespace="http://manufacturing.org/wsd/purchase"
  xmlns:sns="http://manufacturing.org/xsd/purchase"
  ...
  <message name="POMessage">
    <part name="customerInfo" type="sns:customerInfo"/>
    <part name="purchaseOrder" type="sns:purchaseOrder"/>
  </message>
  ...
  <message name="scheduleMessage">
    <part name="schedule" type="sns:scheduleInfo"/>
  </message>
  <portType name="purchaseOrderPT">
    <operation name="sendPurchaseOrder">
      <input message="pos:POMessage"/>
      <output message="pos:InvMessage"/>
      <fault name="cannotCompleteOrder"
        message="pos:orderFaultType"/>
    </operation>
  </portType>
  ...
  <slnk:serviceLinkType name="purchaseLT">
    <slnk:role name="purchaseService">
      <slnk:portType name="pos:purchaseOrderPT"/>
    </slnk:role>
  </slnk:serviceLinkType> ...
</definitions>
```

Messages

The WSDL portType offered by
the service to its customer

Roles

Example

```
<process name="purchaseOrderProcess"
  targetNamespace="http://acme.com/ws-bp/purchase"
...
<partners>
  <partner name="customer"
    serviceLinkType="Ins:purchaseLT"
    myRole="purchaseService"/>
...
</partners>

<containers>
  <container name="PO" messageType="Ins:POMessage"/>
  <container name="Invoice"
    messageType="Ins:InvMessage"/>
...
</containers>

<faultHandlers>
  <catch faultName="Ins:cannotCompleteOrder"
    faultContainer="POFault">
    <reply partner="customer"
      portType="Ins:purchaseOrderPT"
      operation="sendPurchaseOrder"
      container="POFault"
      faultName="cannotCompleteOrder"/>
  </catch>
</faultHandlers>
```

This section defines the different parties that interact with the business process in the course of processing the order.

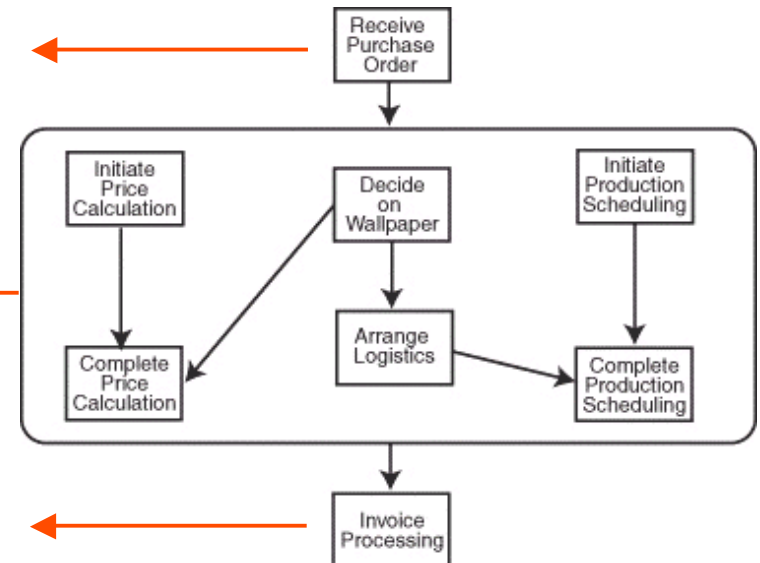
This section defines the data containers used by the process, providing their definitions in terms of WSDL message types.

This section contains fault handlers defining the activities that must be executed in response to faults.

...

Example – The Process

```
...  
<sequence>  
  <receive partner="customer"  
    portType="Ins:purchaseOrderPT"  
    operation="sendPurchaseOrder"  
    container="PO">  
  </receive>  
  <flow>  
    ...  
  </flow>  
  <reply partner="customer"  
    portType="Ins:purchaseOrderPT"  
    operation="sendPurchaseOrder"  
    container="Invoice"/>  
</sequence>  
</process>
```



Example – The Process

<flow> **The flow construct provides concurrency and synchronization**

```
<links>
<link name="ship-to-invoice"/>
<link name="ship-to-scheduling"/>
</links>
```

Activities are executed sequentially

```
<sequence>
...
```

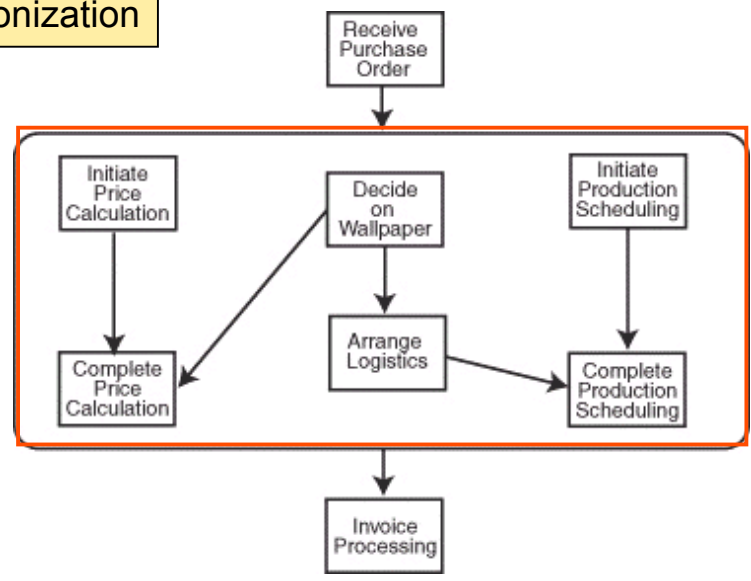
Activity Call

```
<invoke partner="shippingProvider"
portType="Ins:shippingPT"
operation="requestShipping"
inputContainer="shippingRequest"
outputContainer="shippingInfo">
<source linkName="ship-to-invoice"/>
</invoke>
```

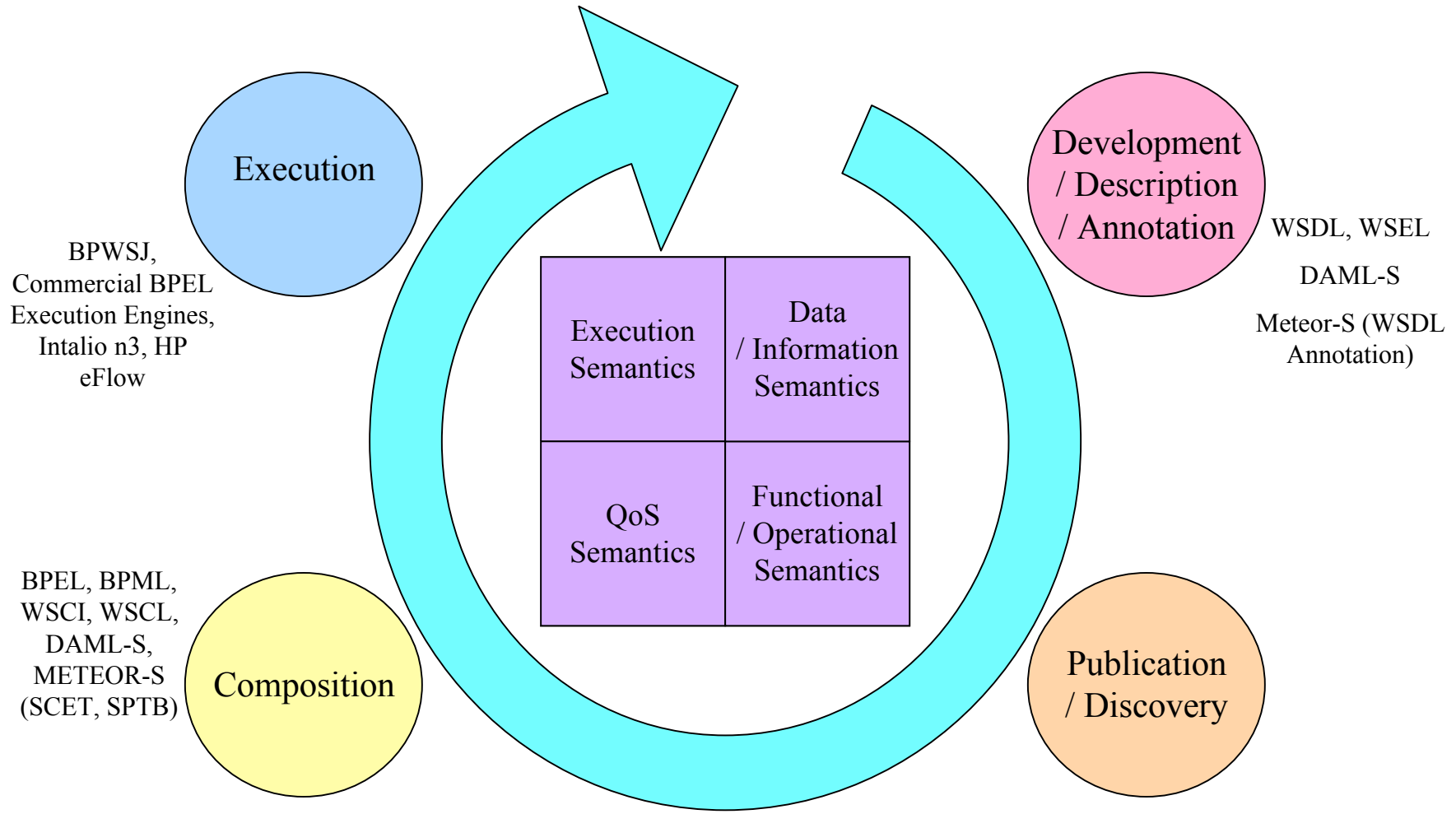
Activity call

```
<receive partner="shippingProvider"
portType="Ins:shippingCallbackPT"
operation="sendSchedule"
container="shippingSchedule">
<source linkName="ship-to-scheduling"/>
</receive>
</sequence>
```

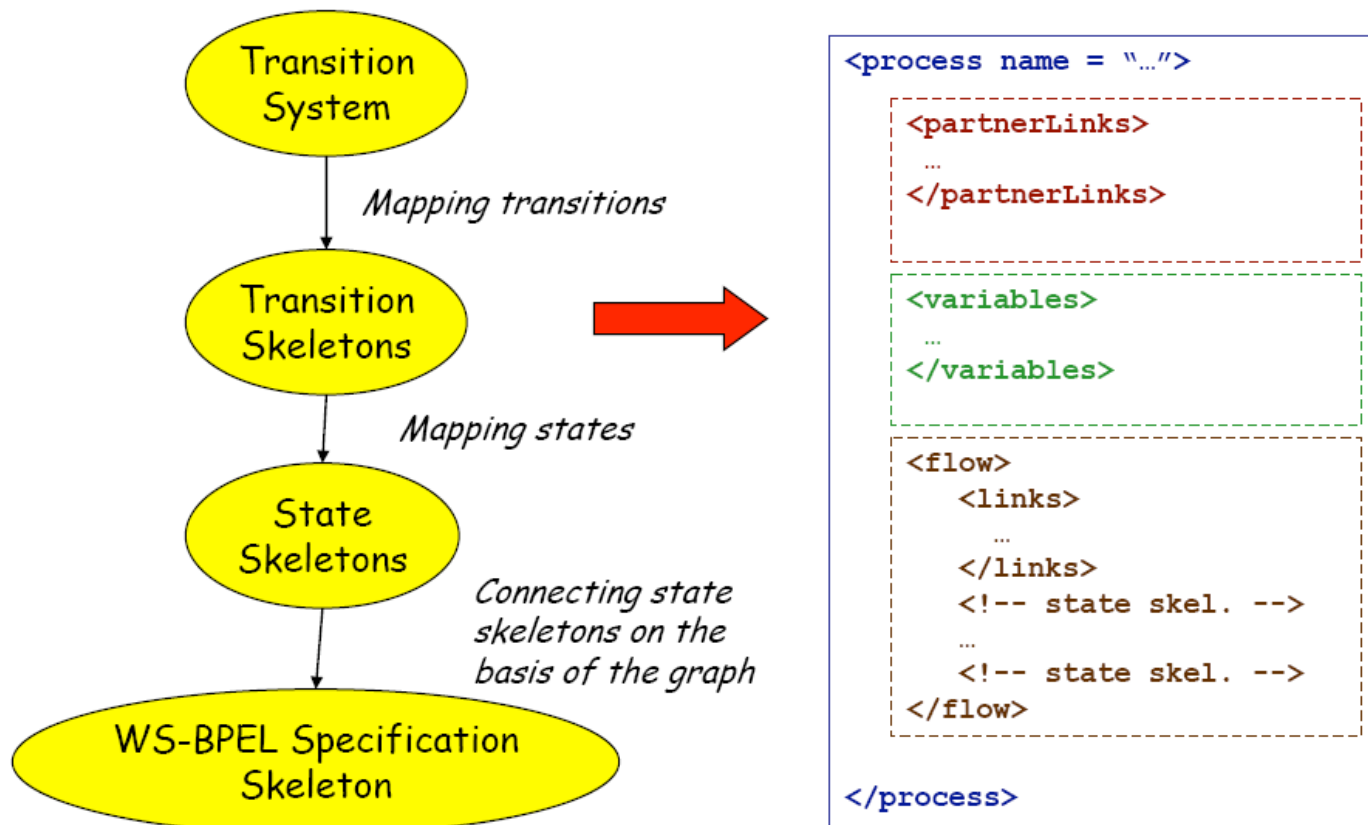
```
...
</flow>
```



Web Services Lifecycle



From a Transition System to WS-BPEL



A New Chance of Success for Composition

- Service composition shares many similarities with workflow technology, with many in the business process being web service operations rather than conventional applications.
 - Considering the limited success of traditional application integration the question of how, if at all, web services will manage to overcome these limitations.
 - Limitations of Conventional Composition Middleware
 - Opportunities for Web Service Composition Middleware
-

Limitations of Conventional Composition Middleware

- Conventional middleware is generally very flexible, thus each time a new component is to be incorporated the programmer faces a considerable development effort.
 - So far, attempts to create a standard composition model had only limited success. Partly because of the difficulties arising during the definitions of such standards but mainly because of the fact that different vendors don't agree on how such standards should look.
 - Established standards, such as **CORBA** are often too low level to be of any real use in EAI or Web services.
-

Opportunities of Web service Composition Middleware

- Web services offer well-defined interfaces and their individual behavior is specified in registries.
 - The already established standards **WDSL**, **XML** and **SOAP** make web services well suited for composition, almost eliminating the need to manually integrate different components often arising in conventional composite applications.
 - When such efforts still arise the developer at least does not have to worry about the different language used in these components, since they all are built using a common one.
-

Opportunities of Web service Composition Middleware

- **BPEL** an already established standard for composing and modeling web services exists, and new ones are likely to appear in the future with the goal to make web service development easier.
 - All these arguments would suggest that web service composition, still in his infancy, will do better than workflow management did.
-

BPEL - Some Critiques

- No human interaction modeling; so needs rollback and restart: BPEL4People ; a need for human intervention when a process stalls (has SAP and IBM support)
 - No support for sub-process interaction: requires new process definition (a code reuse issue)
 - BPEL can only represent multi-party collaborations that have a "center" of control. (Ariba/CommerceOne - some B2B agent in the middle)
 - BPMN waiting on the wings? Lack of tool investment in BPEL will make BPMN more appealing; BPMN hides translation of BPs into XML; so BPEL is redundant?
-

BPELWS Implementations

BPWS4J Engine:

IBM Business Process Execution Language for Web Services Java™ Run Time (BPWS4J)

Platform to execute BPEL4WS processes:

Needs BPELWS documents and the WSDL interfaces for clients and services to be invoked during execution.

Includes an editor (Eclipse plug-in) for BPEL.

Works on Websphere Application Server and Apache Tomcat.

BPELWS Implementations

Oracle BPEL Process Manager:

Former Collaxa BPEL Orchestration Server.

Also comes with an Eclipse plug-in for describing the service.

Offers monitoring of execution of business flows.

Standalone server.

More Information

- <http://blogs.zdnet.com/service-oriented/?p=644>
 - <http://www-128.ibm.com/developerworks/library/ws-bpel>
 - <http://www-106.ibm.com/developerworks/library/ws-bpelwp>
 - <http://www.alphaworks.ibm.com/tech/bpws4j>
 - Peltz, C., “Web Services Orchestration and Choreography,” Computer, IEEE, October 2003, pp 46-52
 - http://www.oasis-open.org/committees/workgroup.php?wg_abbrev=wsbpel
 - <http://www.collaxa.com/>
 - <http://xml.coverpages.org/bpel4ws.html>
 - <http://www.research.ibm.com/journal/sj/452/khalaf.html>
 - http://www.ebizq.net/blogs/it_directions/archives/2006/02/bpmn_will_kill.php
 - <http://xml.coverpages.org/ni2005-10-13-a.html>
 - <http://xml.coverpages.org/ni2005-08-26-a.html>
-