



ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ  
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

# Εισαγωγή στα Δίκτυα Υπηρεσιών

**Assisting Lecture 10 – WS-BPEL**

Μύρων Παπαδάκης  
Τμήμα Επιστήμης Υπολογιστών

# **Introduction to Service Networks**

## **CS-592 – Spring 2014**

Assisting Lecture : WS-BPEL

Myron Papadakis (myrpap@gmail.com)

# Introduction (1/3)

- Within companies, business applications have to interoperate and integrate
- Integrating different applications has always been a difficult task for various functional and technology related reasons.
- The most recent answer to the integration challenge is the **Service Oriented Architecture (SOA)** and the web services technologies.
- The bottom-up view of the SOA sees different **business applications exposing their functionalities through web services.**

## Introduction (2/3)

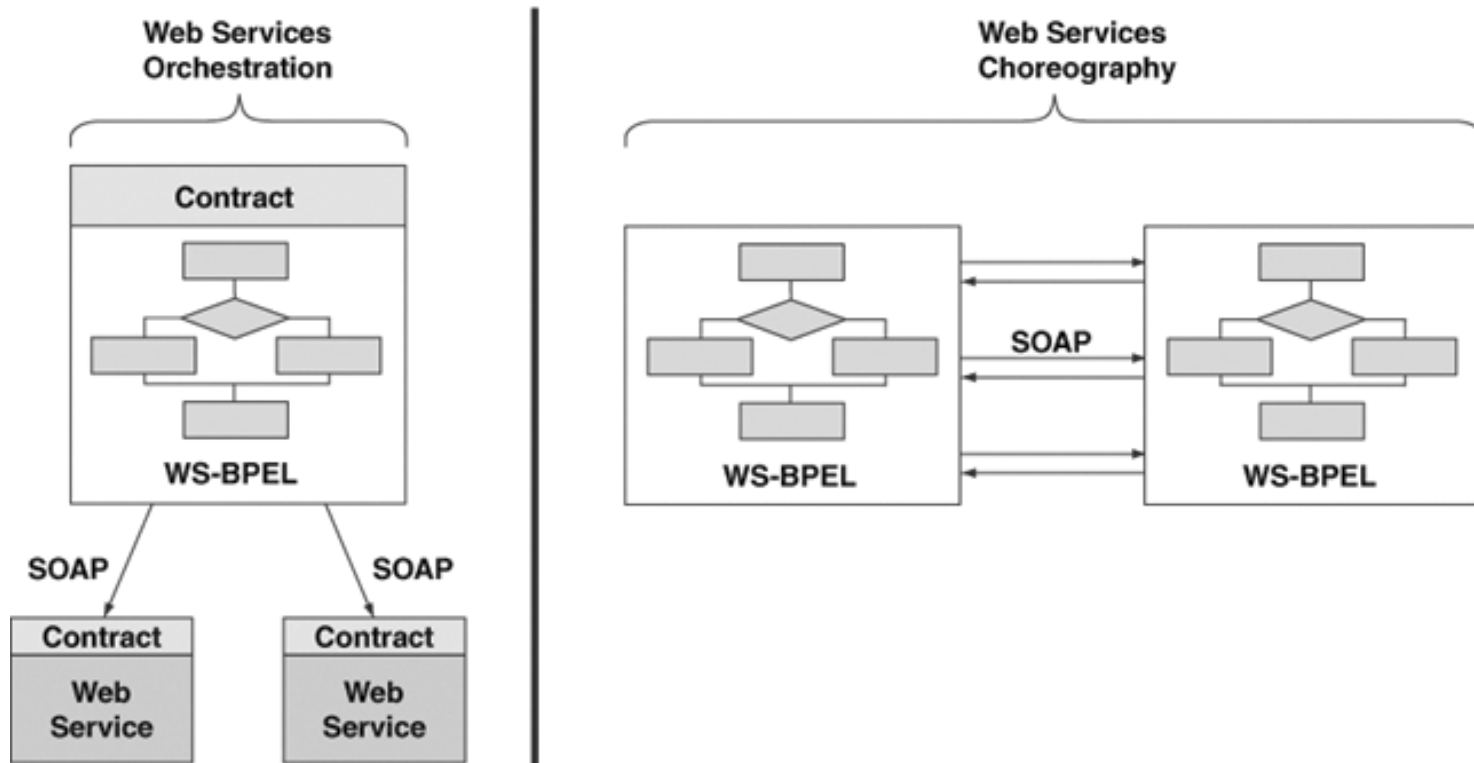
- Thus we can now access different functionalities of different legacy and new developed applications in a standard way (through web services)
- Developing the web services and exposing the functionalities is not sufficient.
- We also need a way to **compose these functionalities in the right order**
  - a way to define business processes which will make use of the exposed functionalities.
- This is where the **BPEL** (Business Process Execution Language for Web Services) becomes important.

# Introduction (3/3)

- The process-oriented approach to SOA requires a language for relatively simple description **of how web services should be composed into business processes**
- BPEL is such a language: it **allows composition of web services and is the top-down approach to SOA — the process-oriented approach to SOA**
- Web services can be composed in two ways: **orchestration** and **choreography**

# Orchestration and Choreography

- **Orchestration: standards: BPEL (OASIS)**
- **Choreography: standards: WS-CDL (W3C), WSCI (W3C)**



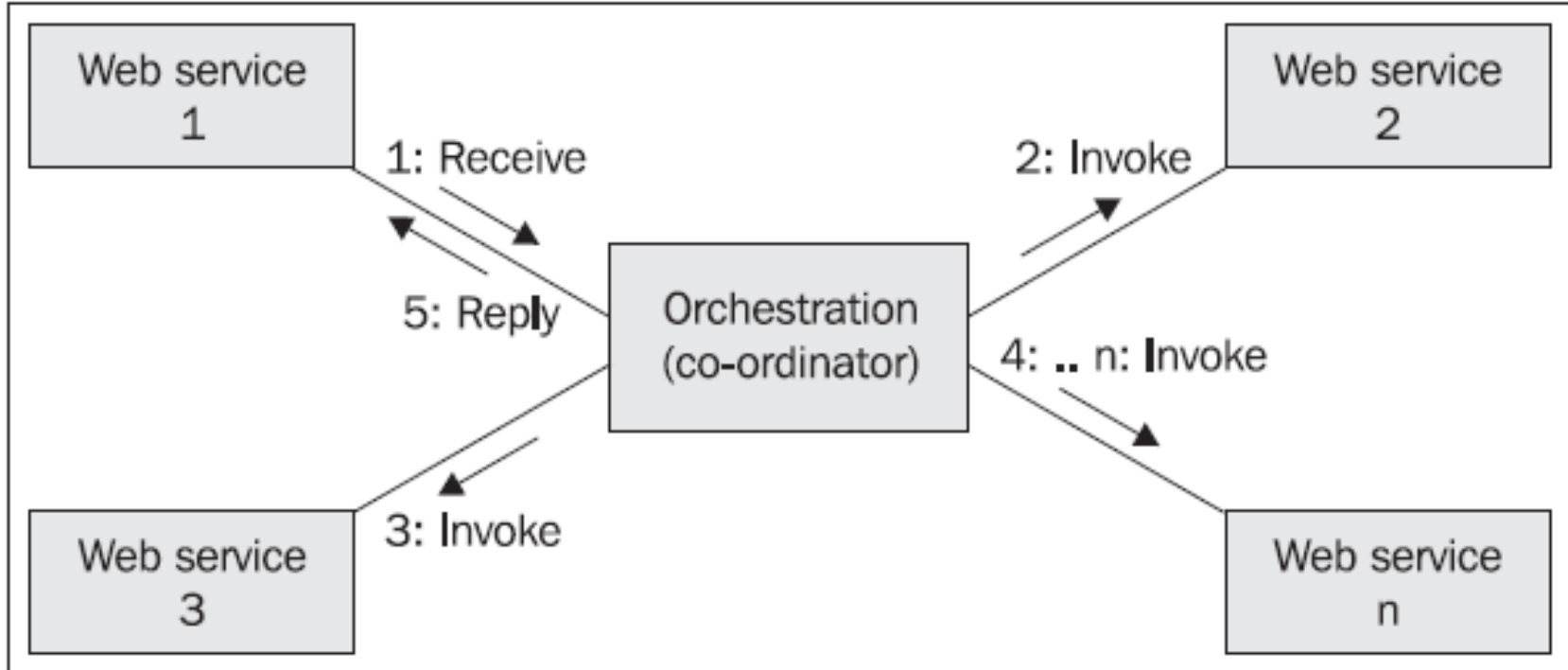
# Orchestration (BPEL)

- In orchestration, a **central process** (which can be another web service) takes control over the involved web services



- It **coordinates the execution of different operations on the web services involved in the operation.**
- The **involved web services do not know (and do not need to know) that they are involved into a composition** and that they are a part of a higher business process.
- Only the central coordinator of the orchestration knows this
  - so the orchestration is centralized with explicit definitions of operations and the order of invocation of web services.

# Orchestration



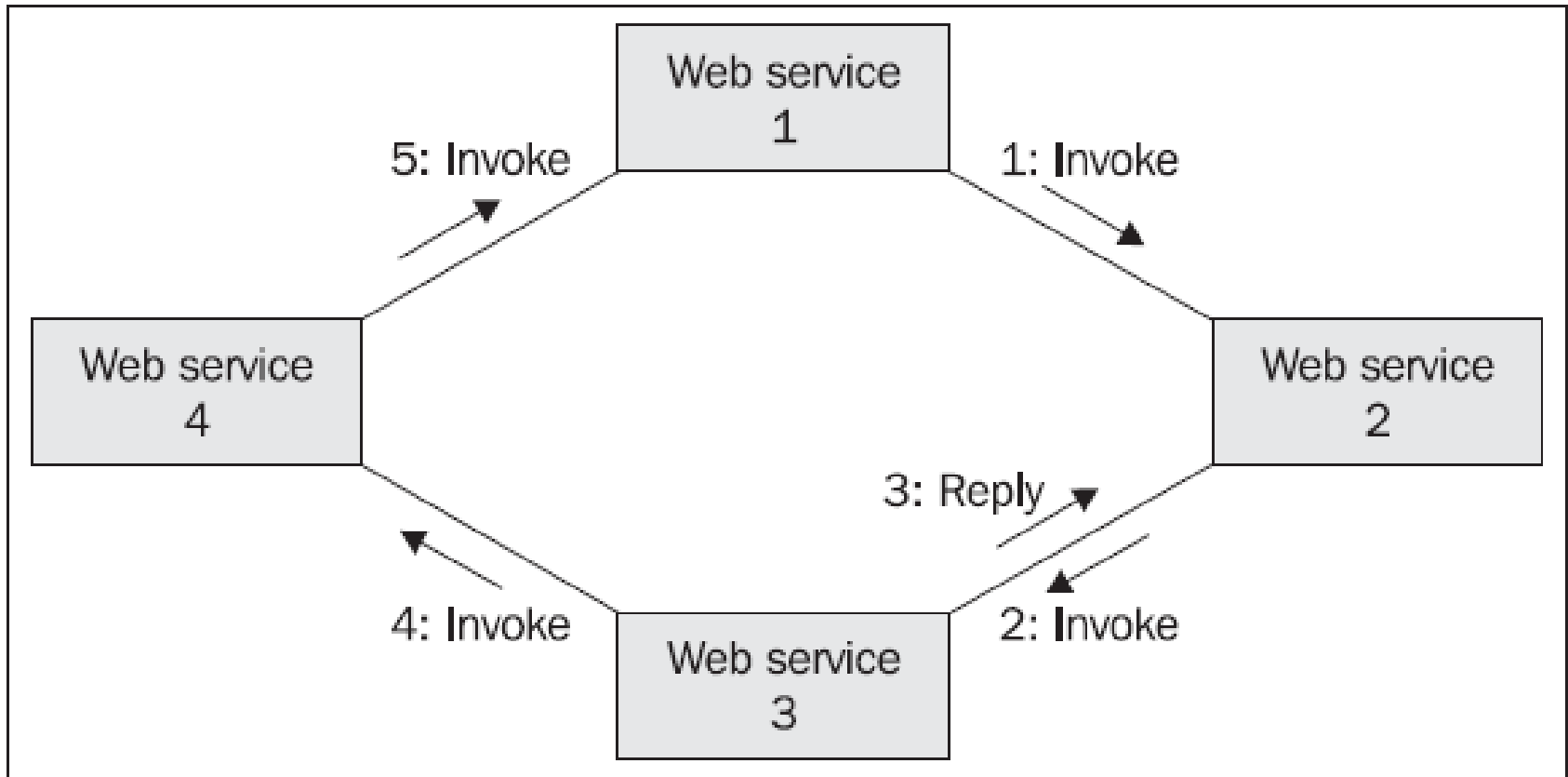


# Choreography



- **Choreography does not rely on a central coordinator.**
- Each web service involved in the choreography knows exactly when to execute its operations and whom to interact with.
- **All participants of the choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges.**
- **A choreography is not directly executable**
- Choreographies can be used to better understand message exchange patterns and to monitor message exchanges

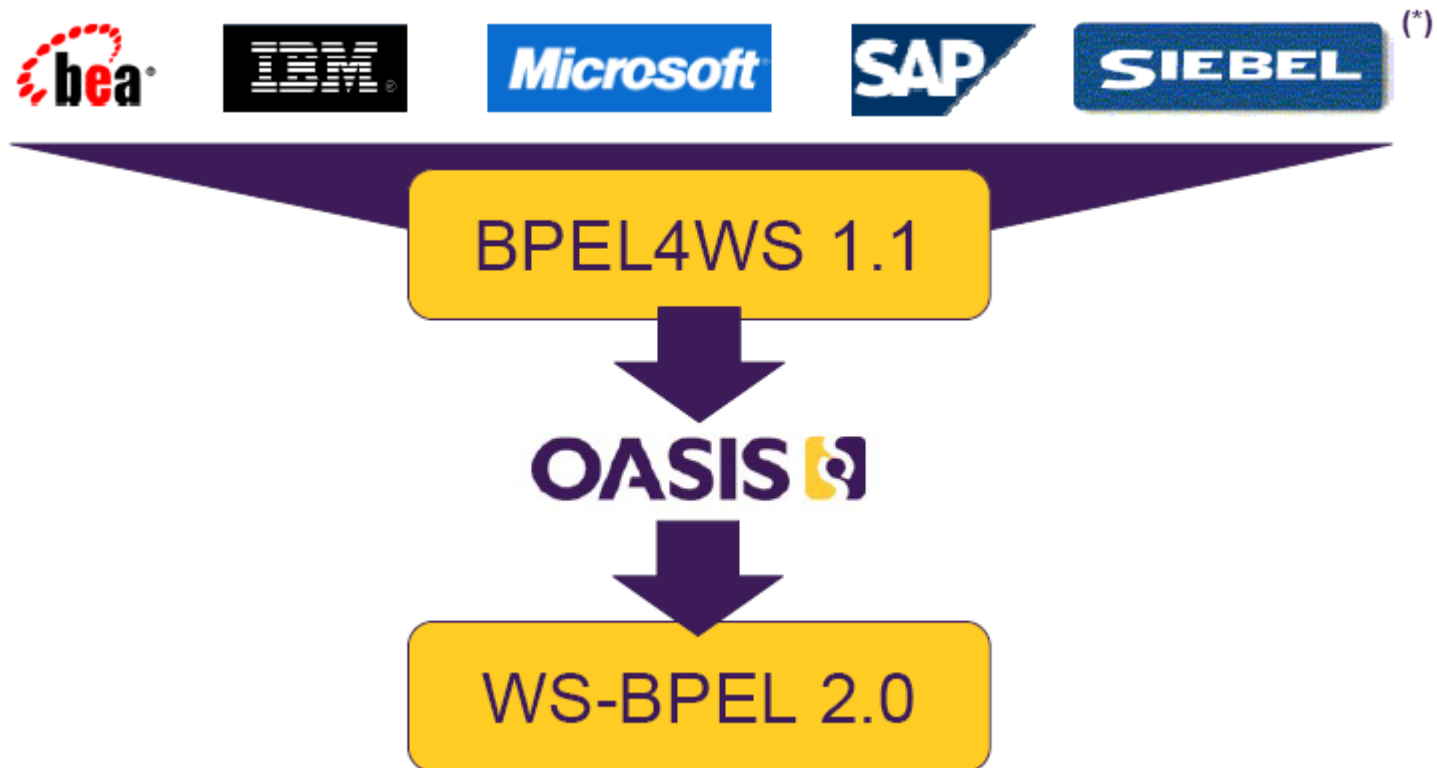
# Choreography



# Web Service Composition Languages

- BPML: Business Process Modeling Language
- XLANG: Extension of Web Services Definition Language
- WSFL: Web Services Flow Language
- **WS-BPEL: Web Services Business Process Execution Language**
- WS-CDL: Web Services Choreography Description Language
- WSCI: Web Services Choreography Interface
- WS-CAF: Web Services Composite Application Framework
- ...

# BPEL Standard Sponsorship



(\*) BPEL4WS 1.1 authors, May 2003

# WS-BPEL Design Goals

- Business processes defined using an **XML-based language**
- **Web services** are the model for process decomposition and assembly
- The same orchestration concepts are used for both the external (abstract) and internal (executable) views of a business process
- Both hierarchical and graph-like control regimes are used, reducing the fragmentation of the process modelling space
- An identification mechanism for process instances is provided at the application message level
- The basic lifecycle mechanism is in implicit creation and termination of process instances.
- A long-running transaction model is defined to support failure recovery for parts of long running business processes
- Language built on compatible Web services standards in a composable and modular manner

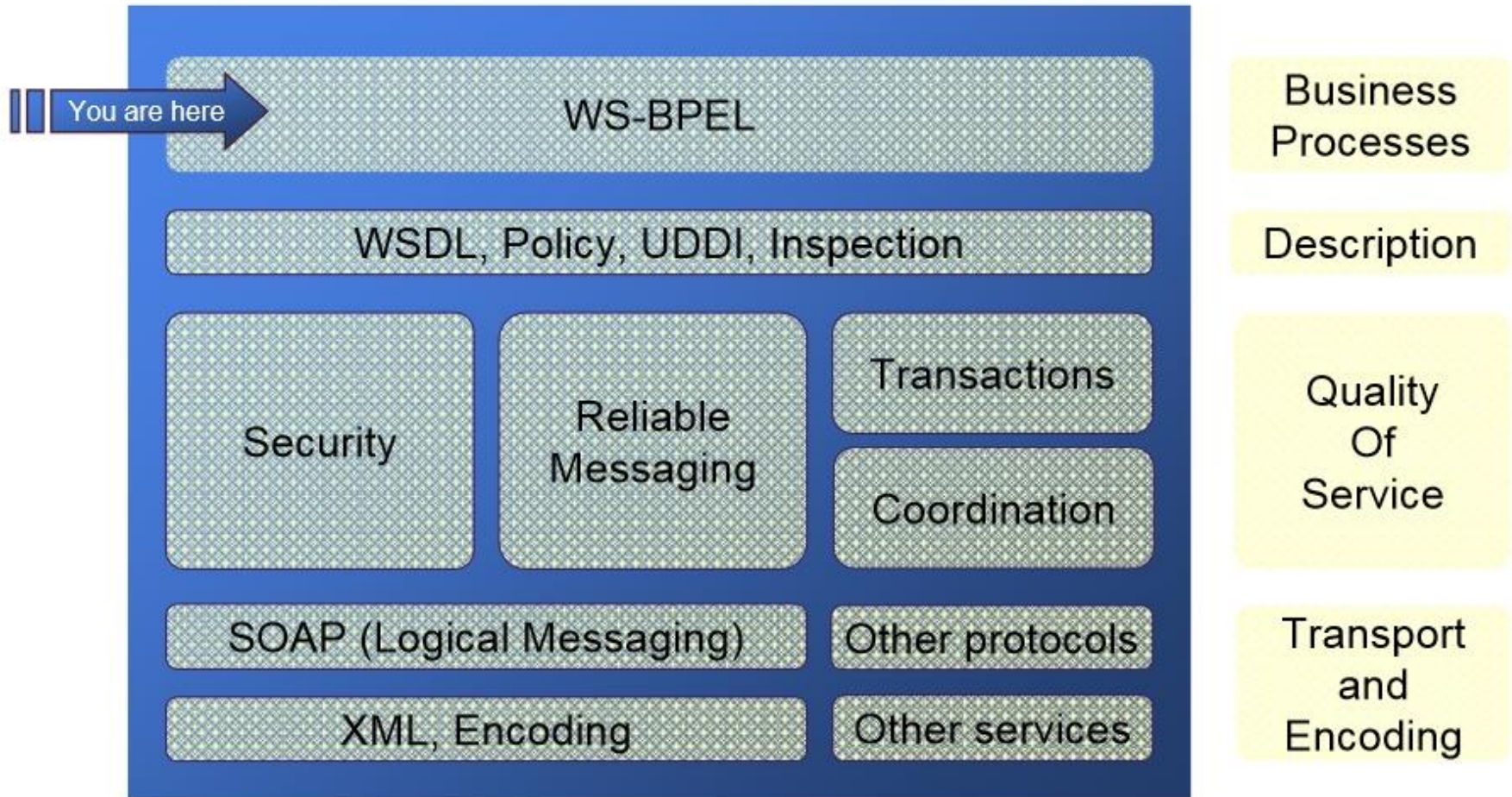
# BPEL

- BPEL is a language **used for composition, orchestration, and coordination of web services.**
- **BPEL is an orchestration language, not a choreography language**
- BPEL represents a convergence of two early workflow languages, WSFL (Web Services Flow Language) and XLANG.
  - **WSFL** was designed by IBM and is based on the concept of directed graphs.
  - **XLANG** was designed by Microsoft and is a block-structured language.
- BPEL combines both approaches and provides a rich vocabulary for the description of business processes.

# BPEL

- **BPEL is built on top of a number of XML-related specifications**
  - XML is used as the syntax for BPEL
  - WSDL is used as the interface description of Web Services
  - XML Schema is used to describe the types used by BPEL processes
  - XPath is used to extract parts of data in a BPEL process
- **It is an XML-based language which supports the web services technology stack, including:**
  - SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination and WS-Transaction.
- **WS-BPEL Specification is administered by OASIS**

# WS-BPEL in the WS-\* Stack





# BPEL

- BPEL is a language for creating executable code
- **BPEL processes can be executed and thus are programs**
- BPEL is a specialized and dedicated programming language
- **BPEL combines two tasks when working with Web Services**
  - it creates a new Web Service which is described by a WSDL interface
  - it implements the Web Service by orchestrating a number of partners

# Business Processes in BPEL

- With BPEL, we can describe business processes in two distinct ways (BPEL supports two types of business processes):
- We can specify the exact details of business processes.
  - Such processes are called **executable business processes** and follow the **orchestration** paradigm.
  - They can be **executed by an orchestration engine**.
  - In most cases BPEL is used for executable processes.
- We can specify the public message exchange between parties only.
  - Such processes are called **abstract business processes**.
  - They do not include the internal details of process flows and are not executable.
  - They are rarely used

# Service Composition with BPEL

- **A BPEL process specifies the exact order in which participating web services should be invoked.**
- **Described in an XML file with extension .bpel.**
- This can be done **sequentially** or in **parallel**.
- With BPEL, we can express conditional behavior, for example, a web service invocation can depend on the value of a previous invocation.
- We can also construct loops, declare variables, copy and assign values, define fault handlers, and so on.
- By combining all these constructs, we can define complex business processes in an algorithmic manner.

# Service Composition with BPEL

- Most BPEL applications are executable processes
  - describing the interfaces to external data sources
  - describing the control flow for orchestrating these data sources
- BPEL is used for defining a new Web Service
  - the process is invoked through one of its partner links
  - it starts executing by following the process description
  - it may contact other partners through other partner links
  - as the final result, it may send back a response to the initial caller

# Developing Business Processes with BPEL

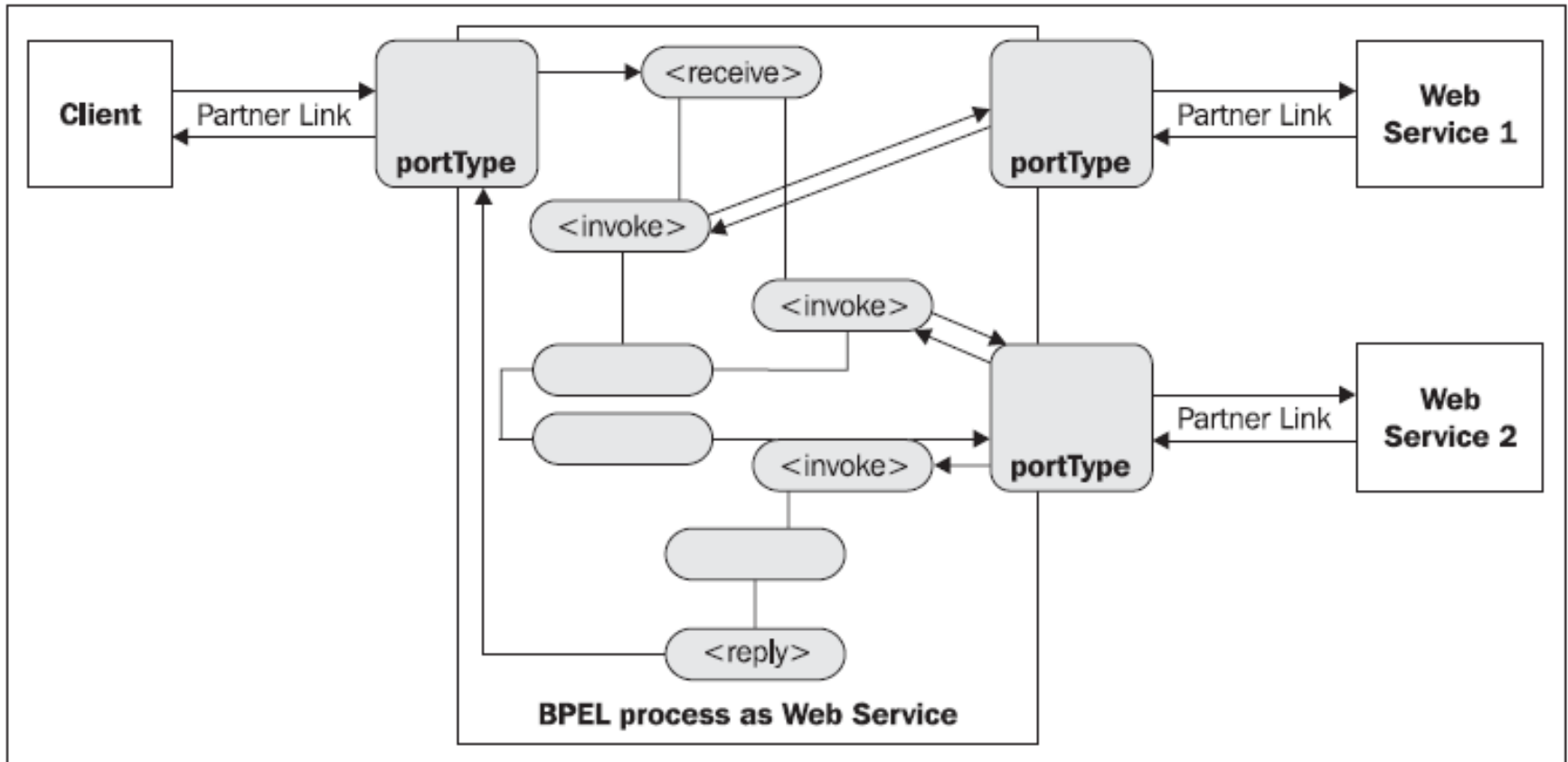
- With BPEL we can define simple and complex business processes.
- **To a certain extent**, BPEL is similar to traditional programming languages.
  - It offers constructs, such as loops, branches, variables, assignments, etc. that allow us to define business processes in an algorithmic way.
- On the other hand, it is less complex than traditional programming languages, which simplifies learning.
- The most important BPEL constructs are related to the **invocation of web services**.
  - BPEL allows to invoke operations of web services either **synchronously** or **asynchronously**.

# Developing Business Processes with BPEL

- Anyone developing BPEL processes requires a good understanding of **WSDL** and other related technologies.
- BPEL introduces WSDL extensions, which enable us to accurately specify relations between several web services in the business process.
  - These relations are called **partner links**.
- The following figure shows a BPEL process and its relation to web services (partner links)

# Developing Business Processes with BPEL

*A typical BPEL process and its relation to Web services (partner links)*



# Developing Business Processes with BPEL

- Executable business processes are processes that compose a set of existing services.
- When we describe a business process in BPEL, we actually define a **new web service that is a composition of existing services.**
- **For its clients a BPEL process looks like any other web service**
- **The interface of the new BPEL composite web service uses a set of port types, through which it provides operations like any other web service.**
- To invoke a business process described in BPEL, **we must invoke the resulting composite web service.**



# Developing Business Processes with BPEL

- A typical BPEL process
  - First, the BPEL business process **receives** a request.
  - To fulfill it, the process then **invokes** the involved web services
  - Finally **responds** to the original caller.
- Because the BPEL process communicates with other web services, it **relies heavily on the WSDL description of the web services invoked by the composite web service.**
- A BPEL process consists of steps.
  - Each step is called an **activity**
- BPEL supports **primitive** and **structured** activities

# BPEL Elements Overview

- **The process element** : It is the root element of BPEL process definition. It has a name attribute and it is used to specify the definition related namespaces.
- **Partner Links elements** : These elements in a BPEL process define the interaction of participating services with the process.
- **Variables elements** : A BPEL process allows to declare variables in order to receive, manipulate, and send data.
- **Fault Handlers element** : A fault handler determines the activity which the process has to perform when an error occurs.
- **Correlation Sets element** : Message correlation is the BPEL mechanism which enables several processes to interact in Stateful conversation.
- **Event handling element** : An event handler allows the scope to response to events, or the expiration of timers, at any time during the execution of a scope.

# BPEL Process Syntax

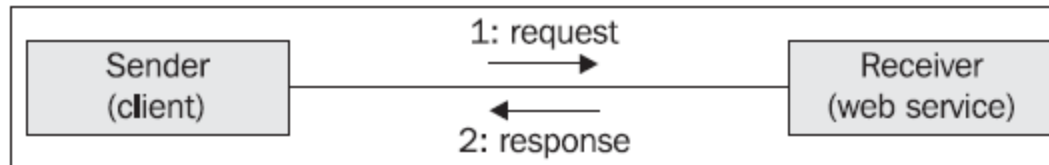
```
<process name="ProcessName">
  <!-- Definition of roles of process participants -->
  <partnerLinks> ... </partnerLinks>
  <!-- Data and state variables used within the process -->
  <variables> ... </variables>
  <!-- Correlation comment -->
  <correlationSets> ... </correlationSets>
  <!-- Exception management -->
  <faultHandlers> ... </faultHandlers>
  <!-- Message and timeout event handler -->
  <eventHandlers> ... </eventHandlers>
  <!-- Processing steps -->
  <sequence>
    </sequenece>
    activities*
  </process>
```

# BPEL Processes

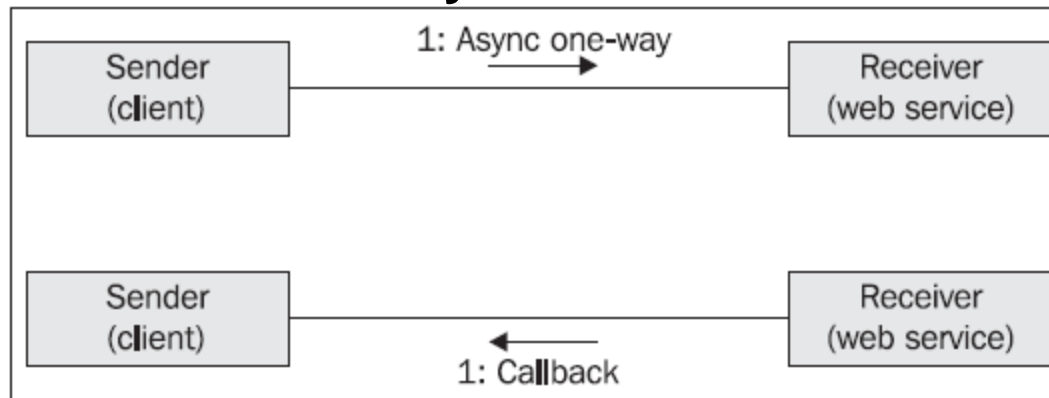
- A BPEL process can be **synchronous or asynchronous**.
- **A synchronous BPEL process blocks the client** (the one which is using the process) until the process finishes and **returns a result to the client**.
  - `<reply>` is used for the response of a synchronous BPEL process
- **An asynchronous process does not block the client**.
  - Rather it uses a **callback** to return the result (if any)
- Usually we use asynchronous processes for longer-lasting processes and synchronous for processes that return a result in a relatively short time

# Synchronous and Asynchronous BPEL Processes

## Synchronous



## Asynchronous



*If operations require that results are sent back to the client, they usually perform callbacks*

# Main Elements of a BPEL Process (Process Definition)

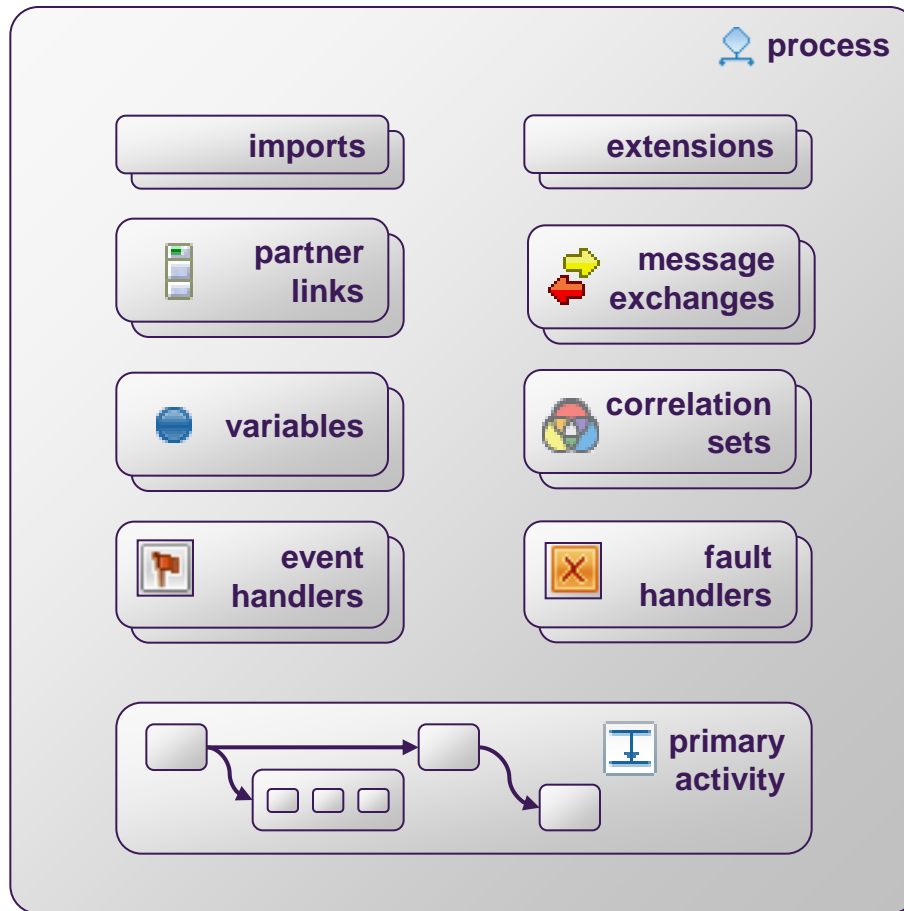
Declare dependencies on external XML Schema or WSDL definitions

Relationships that a WS-BPEL process will employ in its behavior

Data holding state of a business process or exchanged with partners

Concurrently process inbound messages or timer alarms

Perform the process logic – any number of activities may be recursively nested

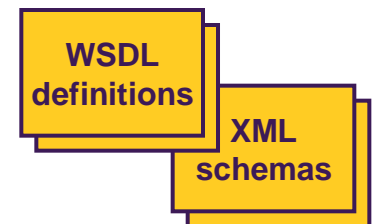


Declare namespaces of WS-BPEL extension attributes and elements

Relationship between inbound and outbound message activities

Application data fields that together identify a conversation

Deal with exceptional situations in a process



# BPEL Process and WSDL

- **As BPEL processes are exposed as web services, we need a WSDL for the BPEL process**
- A client will usually invoke an operation on the BPEL process to start it.
- With the BPEL process WSDL, we specify the interface for this operation.
- We also specify all message types, operations, and port types a BPEL process offers to other partners.

# Partner Link

- The various steps that make up the business process are exposed as services
- Some type representation of that service: known as partner link
- Essentially an **endpoint** representing the service we are going to call..
- Partner link only defined with the interface of that service (no implementation details in the partner link information )
- **Partner links utilize roles**
  - Relationship of partner link with a particular service
- **The endpoint or service is identified by a particular role**



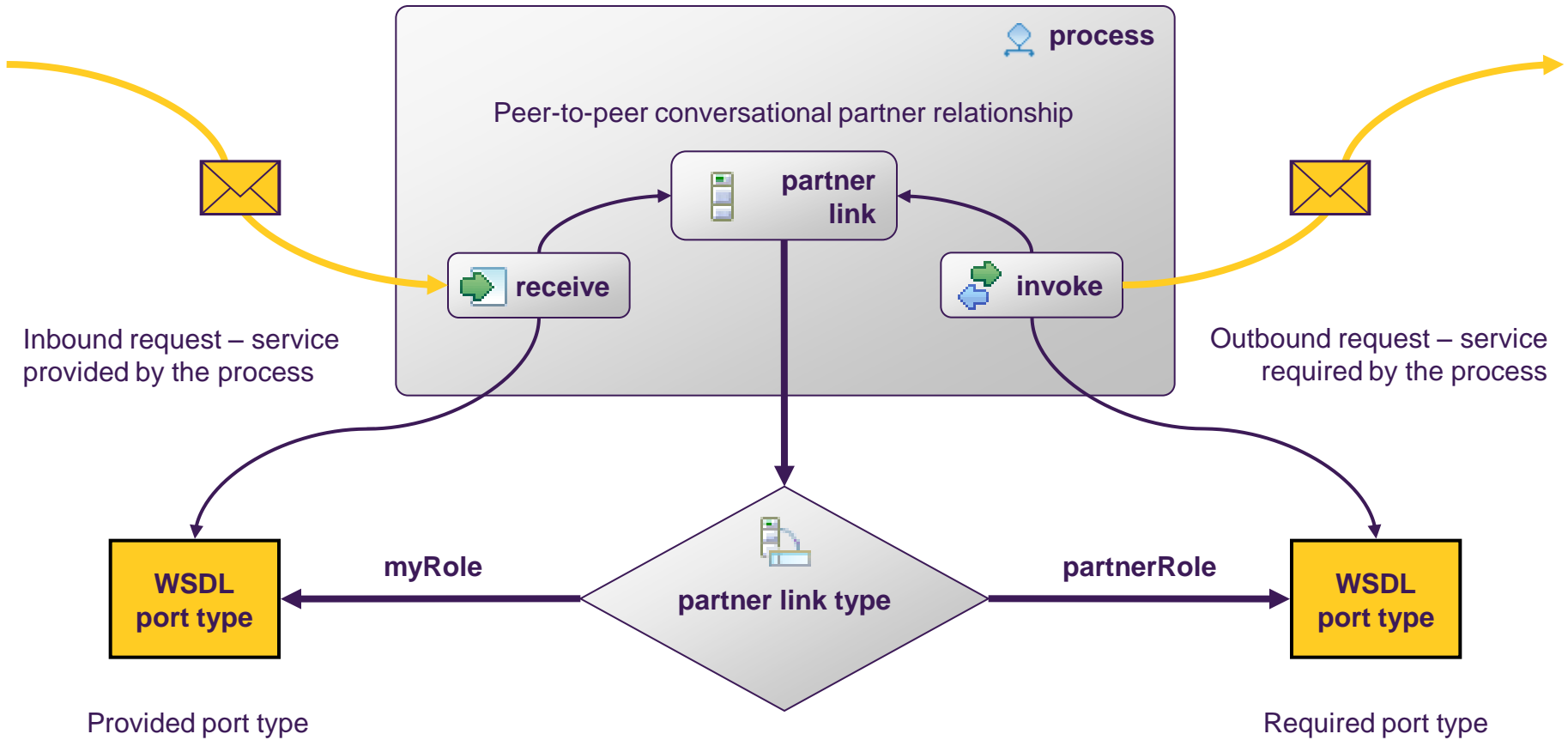
# Partner Links

- **BPEL orchestrates web service interactions.**
  - Each service interaction can be regarded as a communication with a business partner.
  - The interaction is described with the help of partner links.
- **BPEL calls the links to all parties it interacts with as *partner links***
- You can regard one partner link as one particular communication channel.
- Partners might be:
  1. Services that invoke the BPEL process.
  2. Services invoked by the BPEL process.
  3. Services that play both roles - the BPEL process invokes the service and the service invokes a callback on the BPEL process.

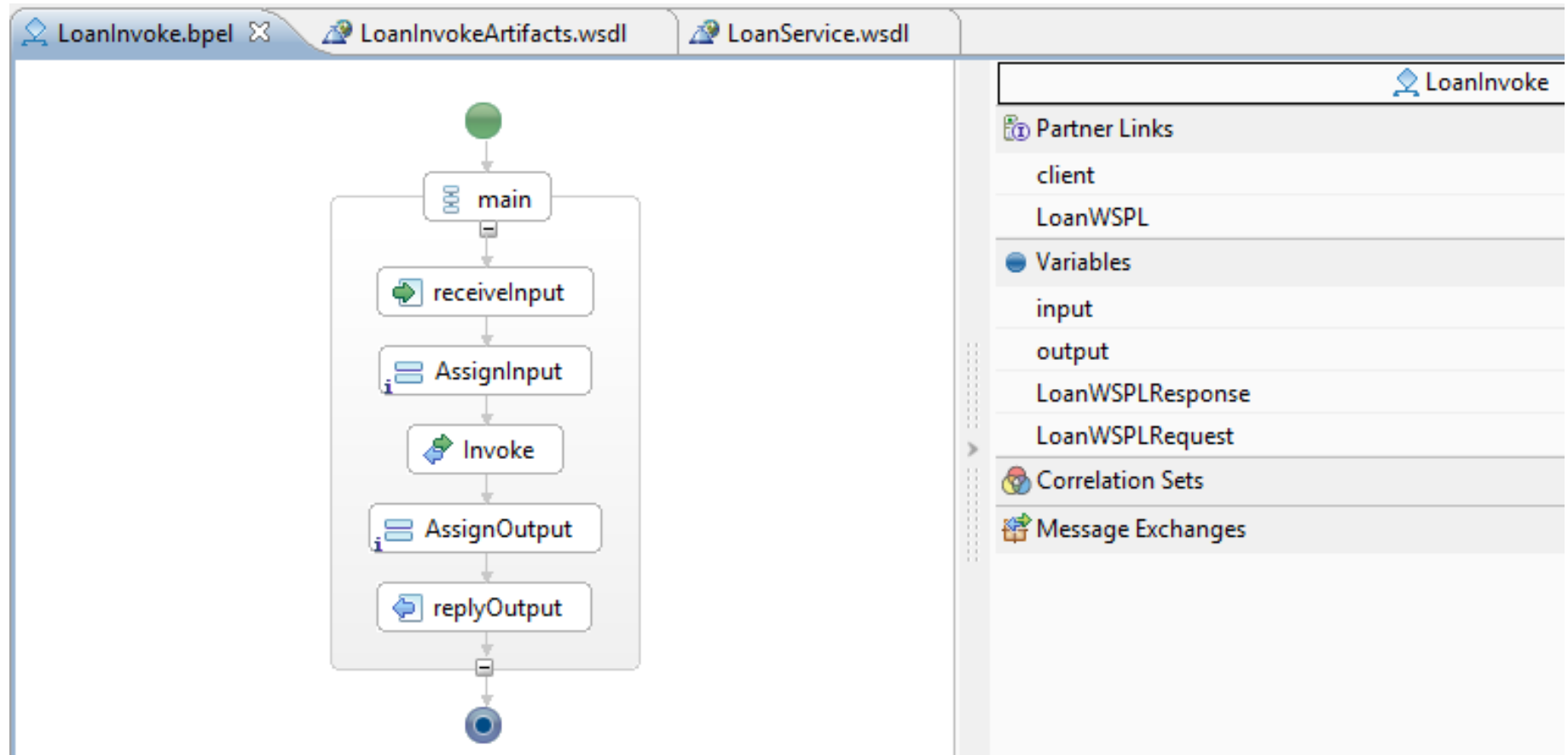
# Partner Links

- The BPEL process uses partner links not only to define services that are invoked by the BPEL process, but also to define the interface of the BPEL process
  - Includes how clients contact/interact with the business process (see WSDL port at the right side of the next slide)
- Each BPEL process has **at least one client partner link**, because there has to be a client that invokes the BPEL process.
- BPEL processes use `<partnerLinks/>` to define partner links
- For each partner link one (synchronous) or two (asynchronous) roles are specified and these roles are associated with portTypes.

# Partner Links



# Partner Links Eclipse BPEL Designer



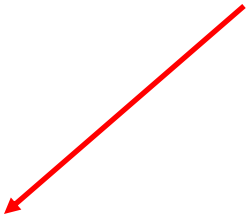
# Partner Links Eclipse BPEL Designer

## LoanInvoke.bpel

```
<!-- ===== -->
<!-- PARTNERLINKS -->
<!-- List of services participating in this BPEL process -->
<!-- ===== -->
<bpel:partnerLinks>
  <!-- The 'client' role represents the requester of this service. -->
  <bpel:partnerLink name="client"
    partnerLinkType="tns:LoanInvoke"
    myRole="LoanInvokeProvider"
  />
  <bpel:partnerLink name="LoanWSPL" partnerLinkType="tns:LoanWSPLT" partnerRole="LoanWSRole"></bpel:part
</bpel:partnerLinks>
```

## LoanInvokeArtifacts.wsdl

```
<plnk:partnerLinkType name="LoanWSPLT">
  <plnk:role name="LoanWSRole" portType="wsdl:LoanService"/>
</plnk:partnerLinkType>
```



*each partnerLink contains a partnerLinkType attribute, which references a partnerLinkType*

# Partner Link Type

- A partner link type declares how two parties **interact** and what each party offers.
- A partner link type must have at least one role and **can have at most two roles (the latter is the usual case)**
  - It contains two PortTypes(WSDL), one for each of the roles in the partner entry (i.e. one portType belongs to the process itself, the other one is the portType of the service being invoked).
- Partner link types are not stored in a process
- **They are be placed in the WSDL document that describes the partner web service** or the BPEL process.

# BPEL Partner Links > WSDL and BPEL Process

- ***In partner Service WSDL***

```
<plnk:partnerLinkType name="FunctionProcessService"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">
  <plnk:role name="FunctionProcessServicePortTypeRole"
    portType="tns:FunctionProcessServicePortType"/>
</plnk:partnerLinkType>
```

- ***In BPEL Process***

```
<partnerLinks>
  <partnerLink name="FunctionProcessPartnerLink"
    xmlns:tns="http://FunctionProcessService.wsdl"
    partnerLinkType="tns:FunctionProcessService"
    myRole="FunctionProcessServicePortTypeRole"/>
</partnerLinks>
```

# Partner Links and Partner Link types

- It is easy to confuse partner links and partner link types, however:
  - **Partner link types** and roles are special WSDL extensions defined by the BPEL specification. As such, they are defined in **WSDL files**, not in the process BPEL file.
  - **Partner Link** is a BPEL 2.0 element (defined in the process **BPEL file**)
- **Partner link types are prerequisites to the Partner Link element definition.**
- Note that multiple partnerLink elements can reference the same partnerLinkType.
  - This is useful for when a process service has the same relationship with multiple partner services. All of the partner services can therefore use the same process service portType elements



# Basic Activities

- Basic activities represent basic constructs and are used for common tasks:
  - Invoking other web services (synchronously or asynchronously), using `<invoke>`
  - `<receive>`: this activity plays an important role in the lifecycle of a business process. It is usually used to initiate the process and its main task is to block and wait for an incoming message.
  - Generating a response for synchronous operations, using `<reply>`
  - Manipulating data variables, using `<assign>`
  - Indicating faults and exceptions, using `<throw>`
  - Waiting for some time (set a duration or deadline), using `<wait>`
  - Terminating the entire process (often used in switches), using `<terminate>` etc
  - ....

# Basic Activities

Do a blocking wait for a matching message to arrive / send a message in reply

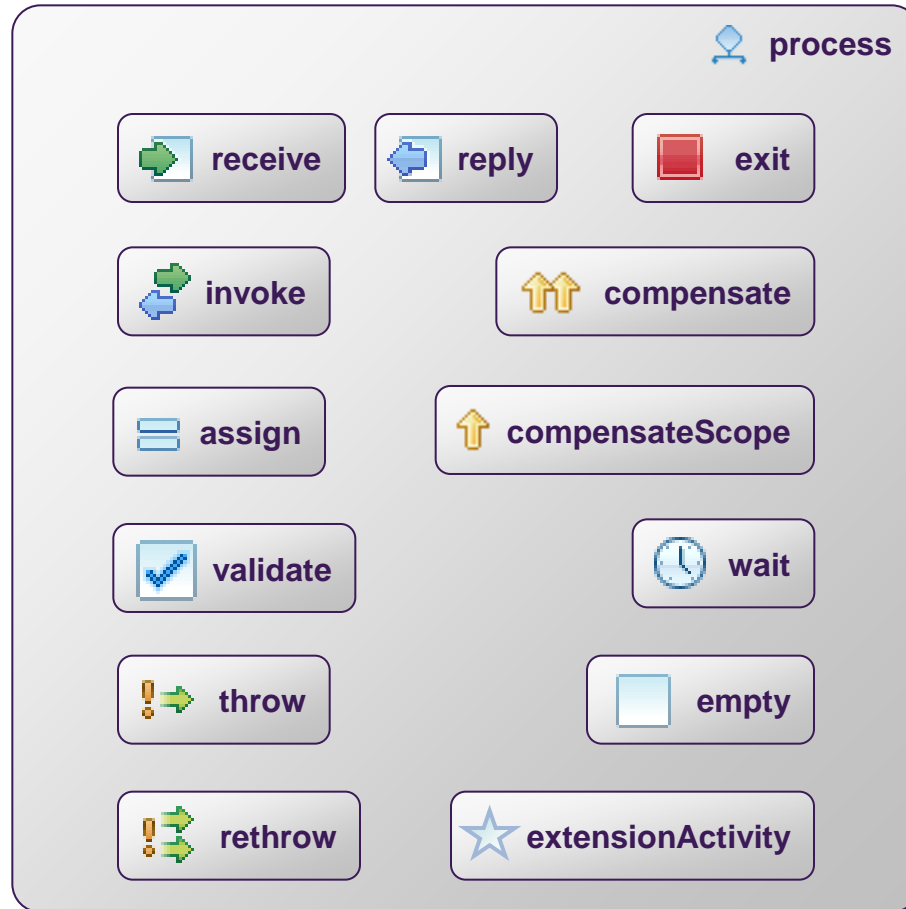
Invoke a one-way or request-response operation

Update the values of variables or partner links with new data

Validate XML data stored in variables

Generate a fault from inside the business process

Forward a fault from inside a fault handler



Immediately terminate execution of a business process instance

Invoke compensation on all completed child scopes in default order

Invoke compensation on one completed child scope

Wait for a given time period or until a certain time has passed

No-op instruction for a business process

Wrapper for language extensions

# Invoking Web Service Operations

- Web Services are called by using **invoke**
  - Web Service invocations can be request/response
  - Web Service invocations can be asynchronous messages
- Invoke need **variables** for input and output
  - output variables are not required for asynchronous invocations
- **Fault handling** can be defined to handle WSDL fault messages
  - resilient BPEL processes should always handle fault messages
- **Compensation handlers** can be defined
  - compensation is a mechanism to handle transactions in BPEL

# Invoking Synchronous Web Services > Sequentially

```
<process ...>
...
  <sequence>
    <!-- Wait for the incoming request to start
      the process -->
    <receive ... />
    <!-- Invoke a set of related web services, one
      by one -->
    <invoke ... />
    <invoke ... />
    <invoke ... />
    ...
  </sequence>
</process>
```

# Invoking Synchronous Web Services Concurrently

```
<process ...>
...
  <sequence>
    <!-- Wait for the incoming request to start the
      process -->
    <receive ... />
    <!-- Invoke a set of related web services,
      concurrently -->
    <flow>
      <invoke ... />
      <invoke ... />
      <invoke ... />
    </flow>
    ...
  </sequence>
</process>
```

# Invoking Synchronous Web Services Concurrently and Sequentially

```
<process ...>
<sequence>
  <!-- Wait for the incoming request to start the process-->
  <receive ... />
  <!-- Invoke two sequences concurrently -->
  <flow>
    <!-- The three invokes below execute sequentially -->
    <sequence>
      <invoke ... />
      <invoke ... />
      <invoke ... />
    </sequence>
    <!-- The two invokes below execute sequentially -->
    <sequence>
      <invoke ... />
      <invoke ... />
    </sequence>
  </flow>
</sequence>
</process>
```

# Invoking Asynchronous Web Services

```
<process ...>
  <sequence>
    <!-- Wait for the incoming request to start
         the process -->
    <receive ... />
    <!-- Invoke an asynchronous operation -->
    <invoke ... />
    <!-- Do something else... -->
    <!-- Wait for the callback -->
    <receive ... />
    ...
  </sequence>
</process>
```

## How do synchronous and asynchronous processes differ in the BPEL specification?

- Both first wait for the initial message, using a <receive>.
- Both also invoke other web services, either synchronously or asynchronously.
- **However, a synchronous BPEL process will return a result after the process has completed.**
  - Therefore, we use a <reply> construct at the end of the process

```
<process ...>
  <sequence>
    <!-- Wait for the incoming request to start the
      process -->
    <receive ... />
    <!-- Invoke a set of related web services -->
    ...
    <!-- Return a synchronous reply to the caller (client)
      -->
    <reply ... /> <!-- reply sends a response to a previous
      receive -->
  </sequence>
..</process>
```



# Asynchronous Processes

- An asynchronous BPEL process does not use the `<reply>` clause.
- If such a process has to send a reply to the client, it uses the `<invoke>` clause to invoke the callback operation on the client's port type
- An asynchronous BPEL process does not need to return anything.

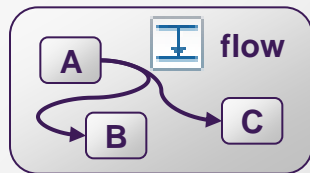
```
<process ...>
  <sequence>
    <!-- Wait for the incoming request to start the
      process -->
    <receive ... />
    <!-- Invoke a set of related web services -->
    ...
    <!-- Invoke a callback on the client (if needed) -->
    <invoke ... />
  </sequence>
</process>
```

# Structured Activities

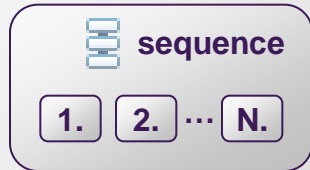
- Sequence ( `<sequence>`), which allows us to define a set of activities that will be invoked in an ordered sequence. The order of execution is determined by their place inside the structure activity.
- Flow ( `<flow>`) for defining a set of activities that will be invoked in parallel
- Case-switch construct ( `<switch>`) for implementing branches
- While ( `<while>`) for defining loops
- `<repeatUntil>`: just like while. It executes the containing activity while a condition is true
- `<pick>`: associates activities with events and waits until an event is triggered. The activities corresponding to the event are executed. The event that occurs first is processed if multiple events are triggered.
  - Pick can be used like the receive activity in order to initialize a process.

# Structured Activities

Contained activities are executed in parallel, partially ordered through control links



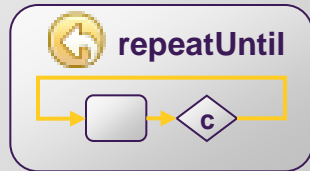
Contained activities are performed sequentially in lexical order



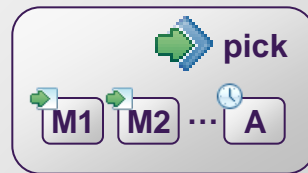
Contained activity is repeated while a predicate holds



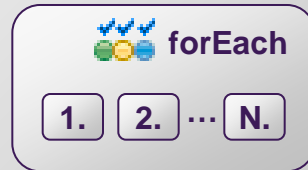
Contained activity is repeated until a predicate holds



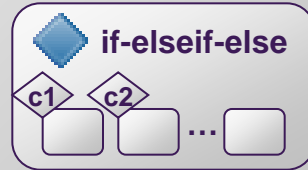
 process



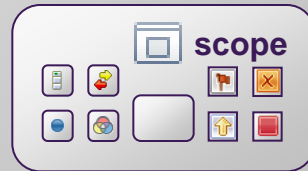
Block and wait for a suitable message to arrive (or time out)



Contained activity is performed sequentially or in parallel, controlled by a specified counter variable



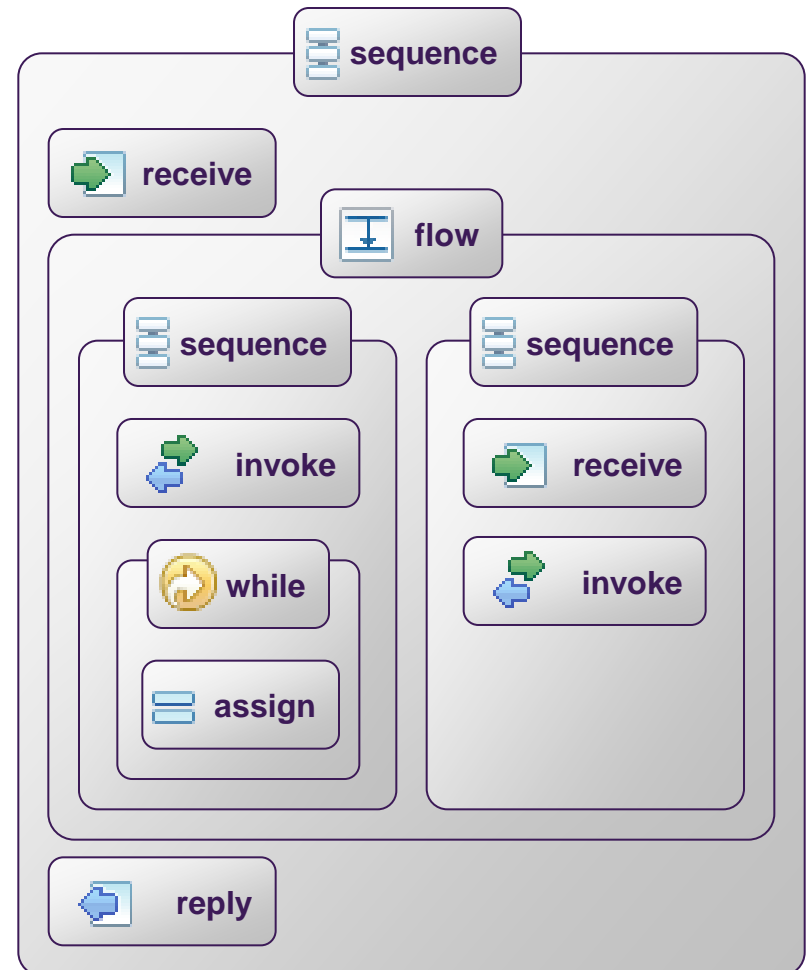
Select exactly one branch of activity from a set of choices



Associate contained activity with its own local variables, partner links, etc., and handlers

# Nesting Structured Activities

```
<sequence>  
  <receive .../>  
  <flow>  
    <sequence>  
      <invoke .../>  
      <while ... >  
        <assign>...</assign>  
      </while>  
    </sequence>  
    <sequence>  
      <receive .../>  
      <invoke ... />  
    </sequence>  
  </flow>  
  <reply .../>  
</sequence>
```



# Variables

- Variables represent the **state** of a business process
- In a BPEL process, variables are used to hold messages that could be:
  - an incoming message from a partner,
  - outgoing message to a partner,
  - data required to hold the state of a process instance (and are never exchanged with partners) etc.
- These are specified in <variable/> elements, inside <variables /> element.
- Each variable has to be declared before it can be used.
- When we declare a variable, we must specify the variable **name** and **type**.

# Variables

- To specify **type** we have to specify one of the following **attributes**:
  - **messageType**: A variable that can hold a WSDL message
  - **element**: A variable that can hold an XML Schema element
  - **type**: A variable that can hold an XML Schema simple type
- Variables can be declared **globally** at the beginning of a BPEL process declaration document or within **scopes**
- Variables are **manipulated using the <assign>** activity

# Variables

```
<variables>  
  <variable name="myVar1"  
    messageType="myNS:myWSDLMessageDataType" />  
  
  <variable name="myVar1" element="myNS:myXMLElement" />  
  
  <variable name="myVar2" type="xsd:string" />  
  
  <variable name="myVar2" type="myNS:myComplexType" />  
  
</variables>
```

# Variables and assigning

- **Copying** the data from one variable to the other is something that will happen very often in a business process.
- Copying data can be achieved with the ***assign*** activity.
- This activity can also be used to copy new data into a variable.
- ```
<assign>  
  <copy>  
    <from variable="ncname" part="ncname"/>  
    <to variable="ncname" part="ncname"/>  
  </copy>  
</assign>
```
- If a variable holds a WSDL message, which is common, we can refine the copy by specifying the part of the message we would like to copy.



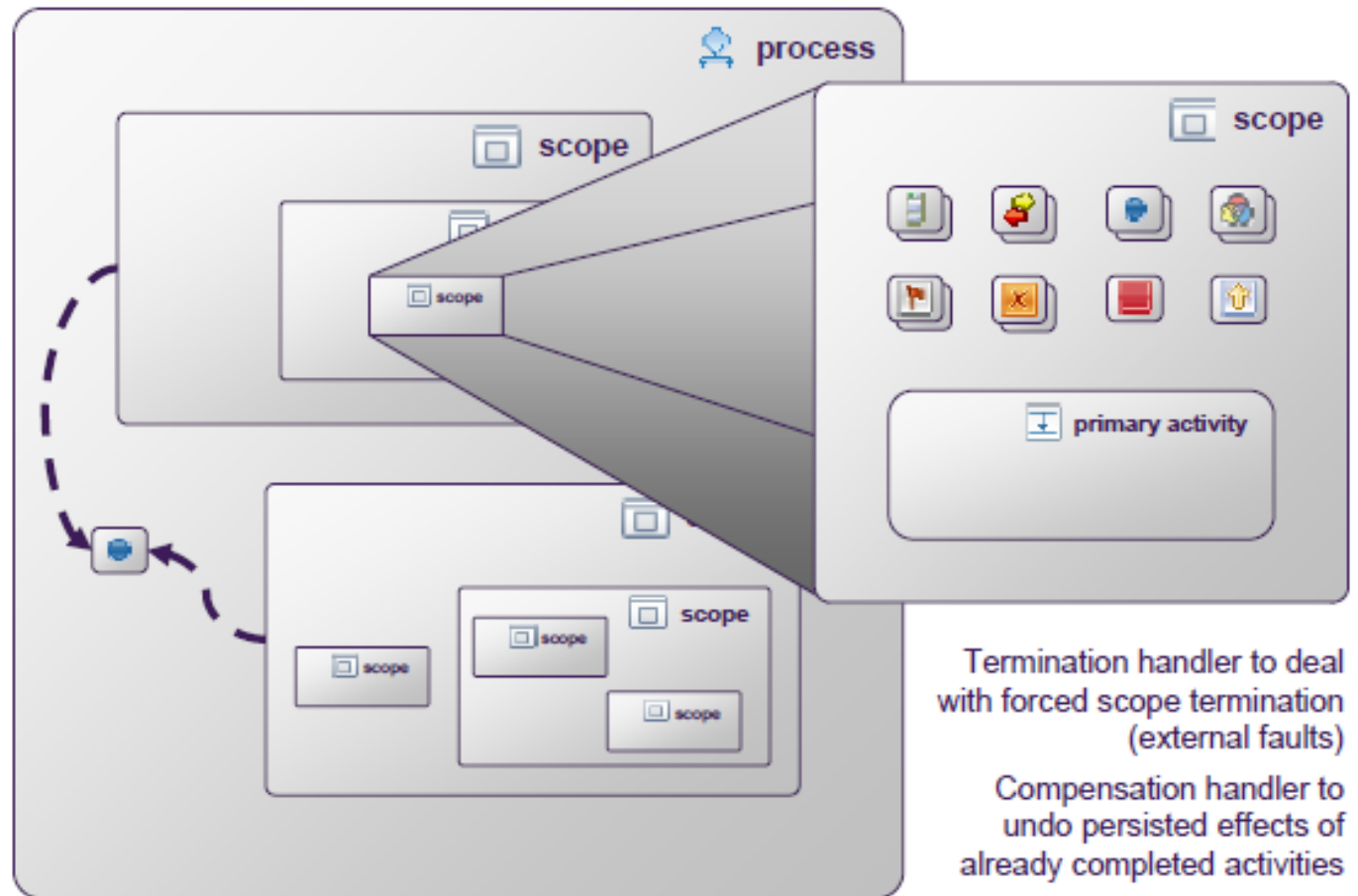
# Scopes

Scopes provide a context which influences the execution behavior of its enclosed activities

Local declarations – partner links, message exchanges, variables, correlation sets

Local handlers – event handlers, fault handlers, a termination handler, and a compensation handler

Isolated scopes provide control of concurrent access to shared resources



# Scopes

- SCOPEs enables you to divide a complex process into several parts
- SCOPEs provide a context for activities:
  - Enables you to define different fault handlers for different activities.
  - You can declare variables that are visible only with the scope.
  - You can also define, correlation sets, compensation handlers, event handlers.
- Each scope **MUST** have a **PRIMARY activity**

# Scope Syntax

```
<scope>  
    <variables>variables local to the  
scope</variables>  
    <correlationSets>...</correlationSets>  
    <faultHandlers>local handlers</faultHandlers>  
    <compensationHandler>...</compensationHandler>  
    <eventHandlers>...</eventHandlers>  
        BASIC OR STRUCTURED ACTIVITIES  
</scope>
```

# Scope Rules

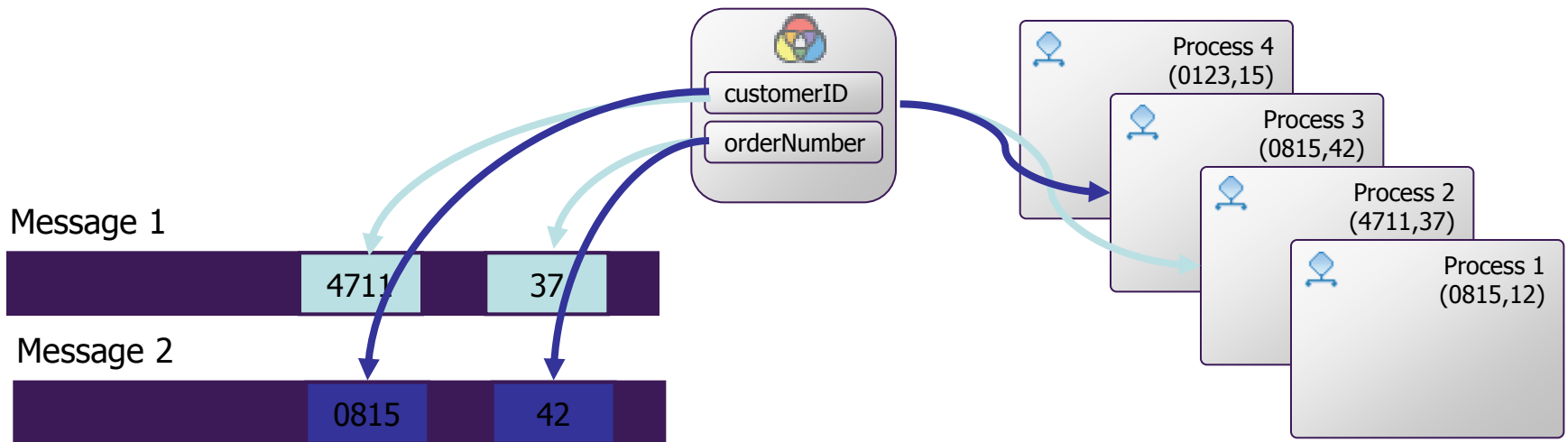
- Each scope has a **primary activity**.
  - This activity may be a basic activity or a structured activity such as sequence or flow.
- If a scope has a structured activity, it can have many nested activities (all in the same scope)
- A scope can also have nested scopes with arbitrary depth.
- **Faults not caught in a scope are re-thrown to the enclosing scope.**
- Scopes in which faults have occurred are considered to have ended abnormally even if a fault handler has caught the fault and not re-thrown it.

# Correlation

- A business process is communicating with multiple services and these services could also be communicating with other services
- It's important to **make sure that you are always talking with the right instance of a service.**
- To realize this in BPEL you could make use of correlation.
- Correlation offers the possibility to make sure you always talking to the same instance of a service by adding **identifying variables.**
- When the process is invoked, these variables always have to be supplied to make sure it's the same process you where talking to earlier on.
- **A set of properties shared by messages and used for correlation is called a correlation set.**
- **Correlation sets are defined and then used in invokes and receives.**

# Properties and Correlation Sets

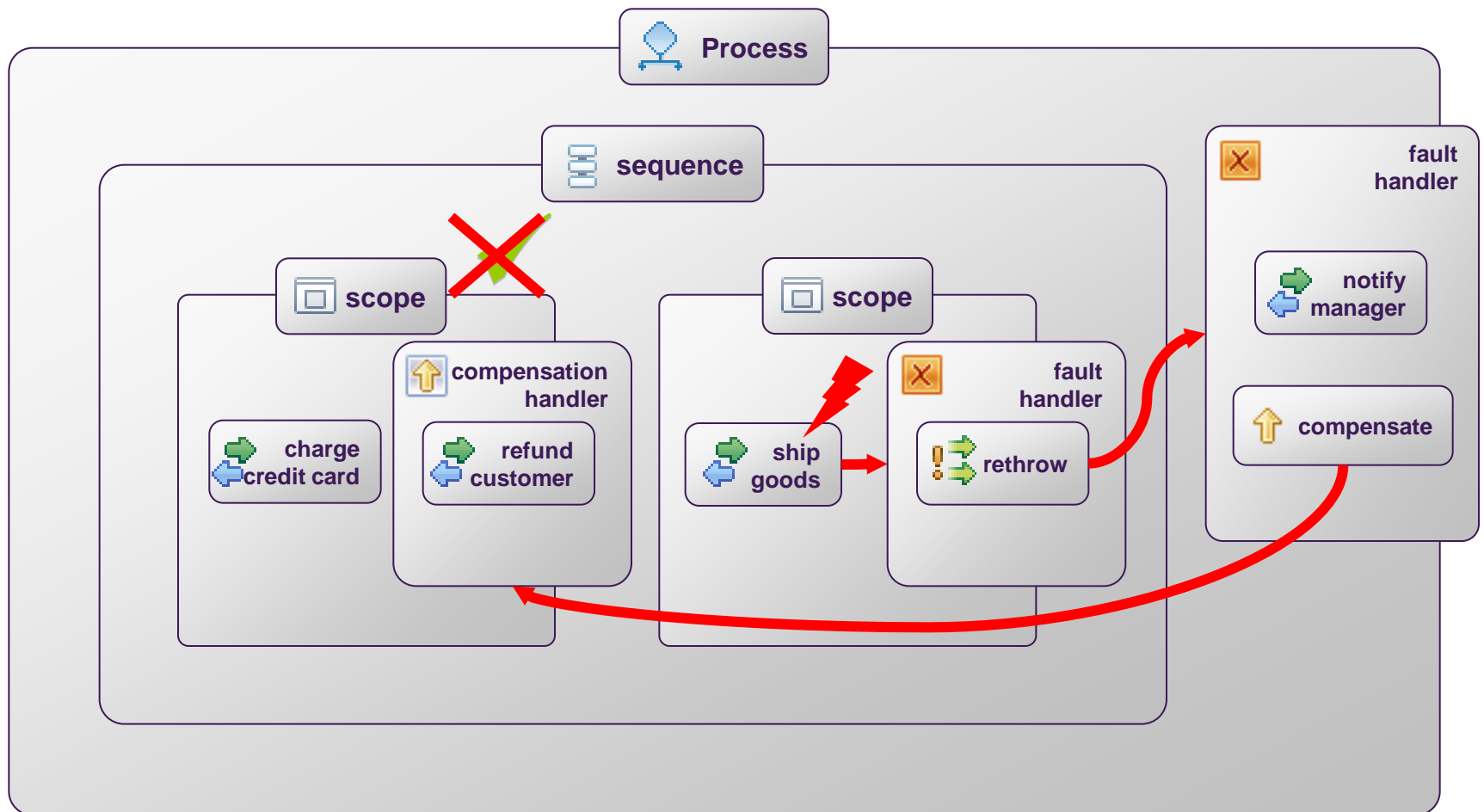
- How to define stateful instances via stateless WS interfaces?
- A process instance is assigned one or more keys
  - Business data is used as key, e.g. customerID
  - A key can be compound, e.g (customerID, orderNo)
  - WS-BPEL calls a key a **correlation set** – it is used to **correlate an incoming message with a process instance**



# Compensation

- The goal of compensation is to **reverse the effects of previous activities** that have been carried out as part of a business process that is being abandoned.
- To define the compensation activities, BPEL provides compensation handlers.
- Compensation handlers gather all activities that have to be carried out to compensate another activity.
- Compensation handlers can be defined:
  - For the whole process
  - For the scope
  - Inline for the <invoke> activity
- **The compensation handler for the whole BPEL process is defined immediately after the fault handlers section and before the main activity of the process**

# Fault Handling & Compensation





# Example

- BPEL process that selects the best insurance offer
  - Two insurance services A and B
- Client invokes the BPEL process
- First we declare the partner links to the BPEL process client (called client) and two insurance web services (called insuranceA and insuranceB)
- Next, we declare variables for the insurance request (InsuranceRequest), insurance A and B responses (InsuranceAResponse, InsuranceBResponse), and for final selection (InsuranceSelectionResponse)
- Finally we specify the process steps

# BPEL Example

```
<?xml version="1.0" encoding="utf-8"?>  
  
<process name="insuranceSelectionProcess"  
  targetNamespace="http://packtpub.com/bpel/example/"  
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"  
  xmlns:ins="http://packtpub.com/bpel/insurance/"  
  xmlns:com="http://packtpub.com/bpel/company/" >  
  
  <partnerLinks>  
    <partnerLink name="client"  
      partnerLinkType="com:selectionLT"  
      myRole="insuranceSelectionService" />  
  
    <partnerLink name="insuranceA"  
      partnerLinkType="ins:insuranceLT"  
      myRole="insuranceRequester"  
      partnerRole="insuranceService" />  
  
    <partnerLink name="insuranceB"  
      partnerLinkType="ins:insuranceLT"  
      myRole="insuranceRequester"  
      partnerRole="insuranceService" />  
  
  </partnerLinks>
```

# Declaring the Variables

Next, we declare variables for the insurance request (InsuranceRequest), insurance A and B responses (InsuranceAResponse, InsuranceBResponse), and for final selection (InsuranceSelectionResponse)

```
...
<variables>
  <!-- input for BPEL process -->
  <variable name="InsuranceRequest"
    messageType="ins:InsuranceRequestMessage" />
  <!-- output from insurance A -->
  <variable name="InsuranceAResponse"
    messageType="ins:InsuranceResponseMessage" />
  <!-- output from insurance B -->
  <variable name="InsuranceBResponse"
    messageType="ins:InsuranceResponseMessage" />
  <!-- output from BPEL process -->
  <variable name="InsuranceSelectionResponse"
    messageType="ins:InsuranceResponseMessage" />
</variables>
...
```

# Specifying the process steps

---

...

```
<sequence>
```

```
<!-- Receive the initial request from client -->
```

```
<receive partnerLink="client"  
  portType="com:InsuranceSelectionPT"  
  operation="SelectInsurance"  
  variable="InsuranceRequest"  
  createInstance="yes" />
```

```
<!-- Make concurrent invocations to Insurance A and B -->
```

```
<flow>
```

```
<!-- Invoke Insurance A web service -->
```

```
<invoke partnerLink="insuranceA"  
  portType="ins:ComputeInsurancePremiumPT"  
  operation="ComputeInsurancePremium"  
  inputVariable="InsuranceRequest"  
  outputVariable="InsuranceAResponse" />
```

```
<!-- Invoke Insurance B web service -->
```

```
<invoke partnerLink="insuranceB"  
  portType="ins:ComputeInsurancePremiumPT"  
  operation="ComputeInsurancePremium"  
  inputVariable="InsuranceRequest"
```

```
outputVariable="InsuranceBResponse" />
```

```
</flow>
```

```
<!-- Select the best offer and construct the response -->
```

```
<switch>
```

```
<case condition="bpws:getVariableData('InsuranceAResponse',  
    'confirmationData','/confirmationData/Amount')  
    <= bpws:getVariableData('InsuranceBResponse',  
    'confirmationData','/confirmationData/Amount')">
```

```
<!-- Select Insurance A -->
```

```
<assign>
```

```
<copy>
```

```
<from variable="InsuranceAResponse" />
```

```
<to variable="InsuranceSelectionResponse" />
```

```
</copy>
```

```
</assign>
```

```
</case>
```

```
<otherwise>
```

```
<!-- Select Insurance B -->
```

```
<assign>
```

```
<copy>
```

```
<from variable="InsuranceBResponse" />
```

```

        outputVariable="InsuranceBResponse" />
</flow>
<!-- Select the best offer and construct the response -->
<switch>
  <case condition=" bpws:getVariableData('InsuranceAResponse',
    'confirmationData','/confirmationData/Amount')
    <= bpws:getVariableData('InsuranceBResponse',
    'confirmationData','/confirmationData/Amount')" >
    <!-- Select Insurance A -->
    <assign>
      <copy>
        <from variable="InsuranceAResponse" />
        <to variable="InsuranceSelectionResponse" />
      </copy>
    </assign>
  </case>
  <otherwise>
    <!-- Select Insurance B -->
    <assign>
      <copy>
        <from variable="InsuranceBResponse" />

```

## Example > Send the response to the client

```
        <to variable="InsuranceSelectionResponse" />
    </copy>
</assign>
</otherwise>
</switch>

<!-- Send a response to the client -->
<reply partnerLink="client"
    portType="com:InsuranceSelectionPT"
    operation="SelectInsurance"
    variable="InsuranceSelectionResponse" />

</sequence>

</process>
```

# References

- <https://www.oasis-open.org/committees/download.php/23964/>
- **Book: Ws-Bpel 2.0 for Soa Composite Applications with Oracle Soa Suite 11G**
- <http://www.oracle.com/technetwork/articles/matjaz-bpel1-090575.html>
- <http://www.csie.ndhu.edu.tw/~showyang/SOC2008/04aBP-EL.pdf>



# Τέλος Ενότητας



Ευρωπαϊκή Ένωση  
Ευρωπαϊκό Κοινωνικό Ταμείο

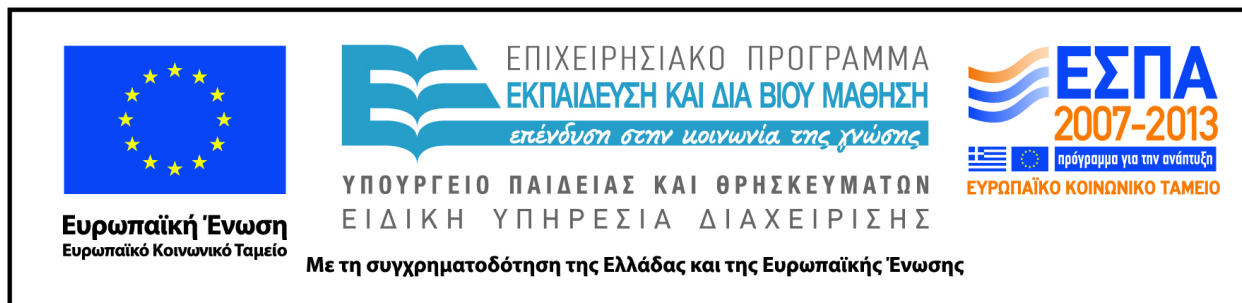


Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης



# Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «**Εκπαίδευση και Δια Βίου Μάθηση**» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



**Σημειώματα**

# Σημείωμα αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-nd/4.0/>

## •Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

•Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

# Σημείωμα Αναφοράς

Copyright Πανεπιστήμιο Κρήτης, Μύρων Παπαδάκης. «**Εισαγωγή στα Δίκτυα Υπηρεσιών. Assisting Lecture 10 – WS-BPEL**». Έκδοση: 1.0.  
Ηράκλειο/Ρέθυμνο 2015. Διαθέσιμο από τη δικτυακή διεύθυνση:  
<https://elearn.uoc.gr/course/view.php?id=416/>