



Information and  
Communication Systems  
Research Group



**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

# 5 Web Service Composition and BPEL

Gustavo Alonso  
Computer Science Department  
Swiss Federal Institute of Technology (ETHZ)  
alonso@inf.ethz.ch  
<http://www.iks.inf.ethz.ch/>



*Information and  
Communication Systems  
Research Group*



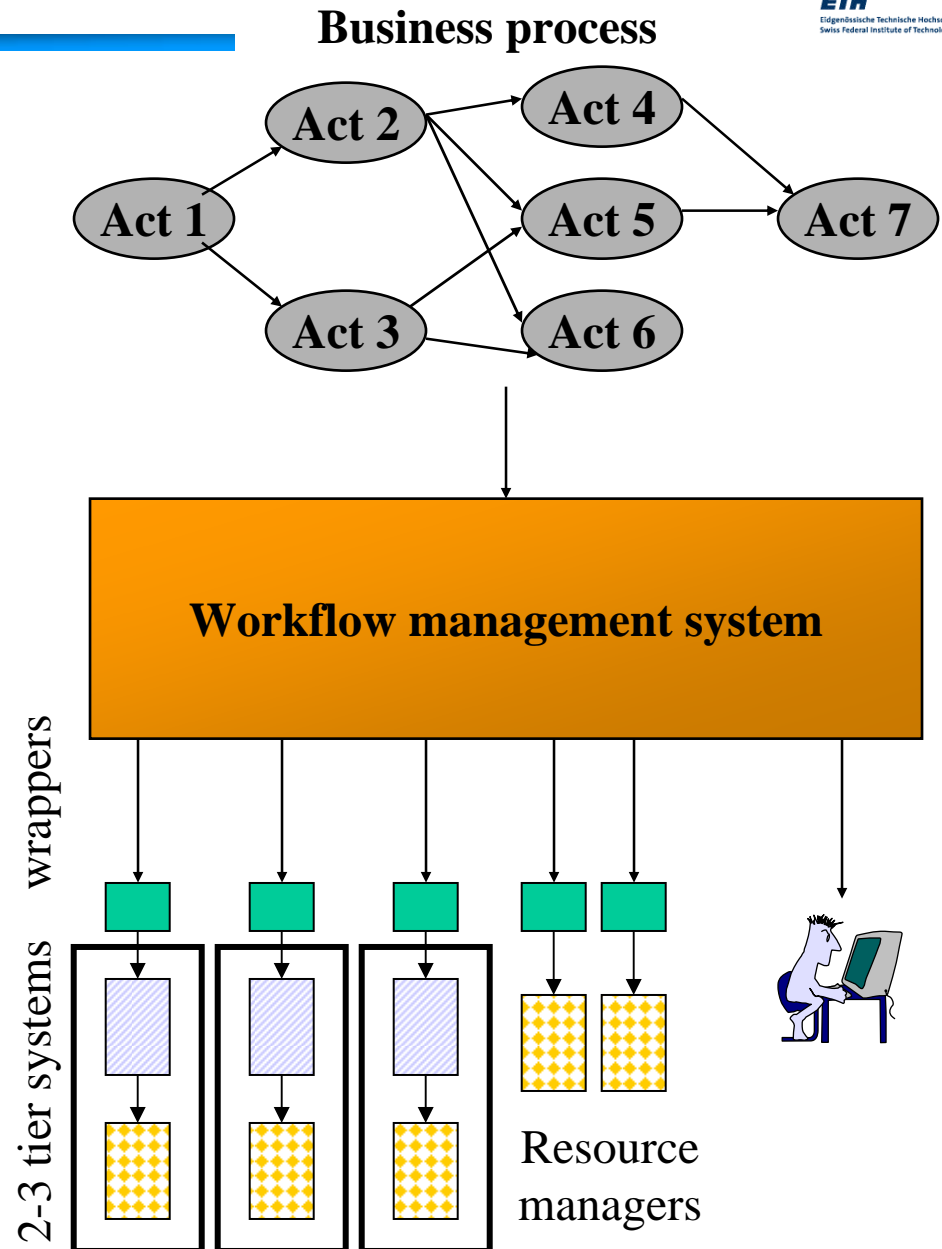
**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

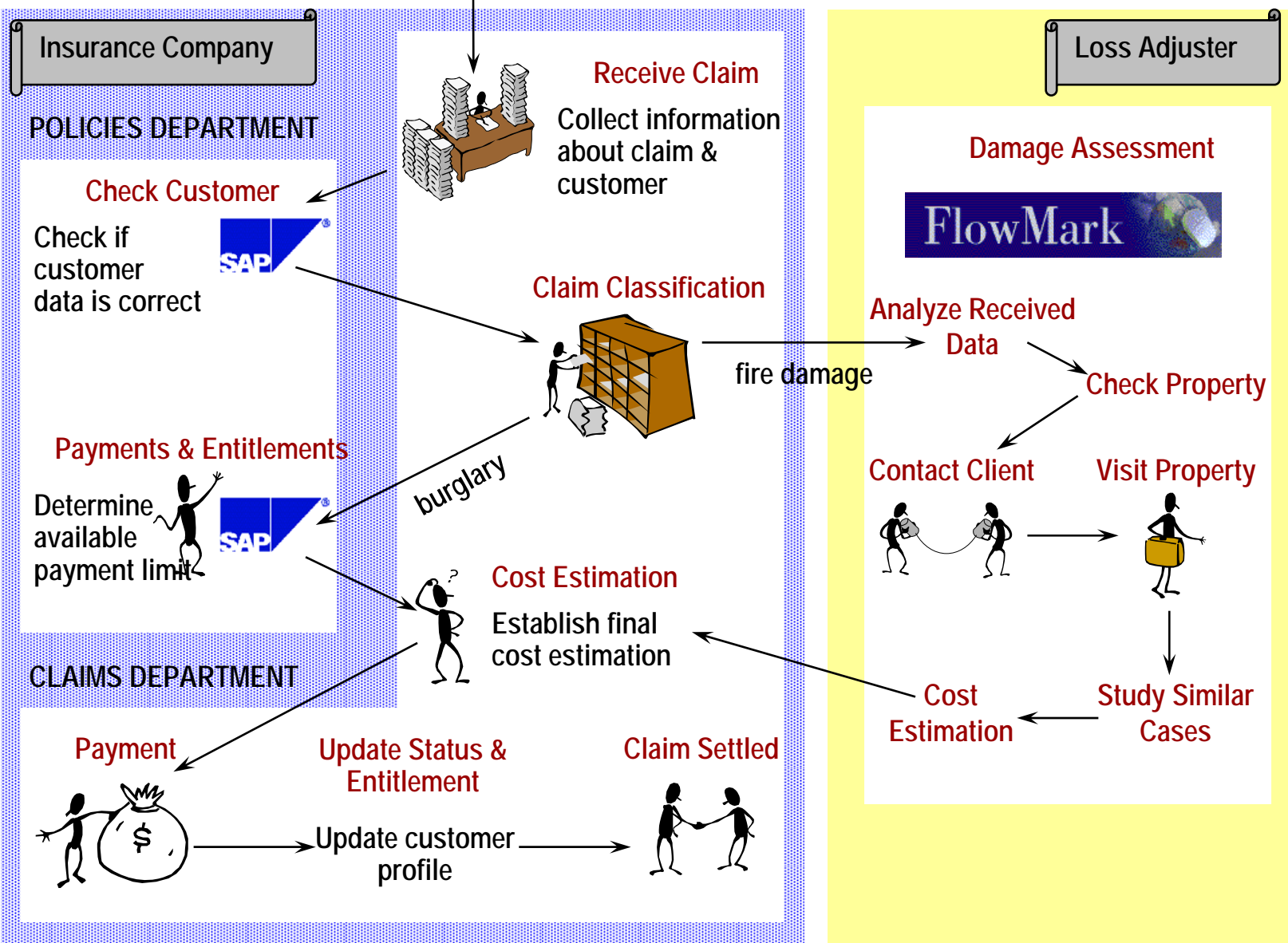
# Background and historical perspective

# Business Processes

- A business process describes key procedures within an organization. They involve:
  - multiple steps
  - numerous persons
  - large amounts of resources
- In large corporations there are many factors that increase the complexity of the business processes:
  - processes are not well documented
  - conformance to rules not guaranteed
  - people lack information
  - company lacks monitoring tools
  - steps, persons and resources are not properly coordinated
- Workflow Management Systems try to address these problems by automating the coordination aspects of a business process, that is, who has to do what, when, and with which tools.



# Business Process (Example)



# Goals of Workflow

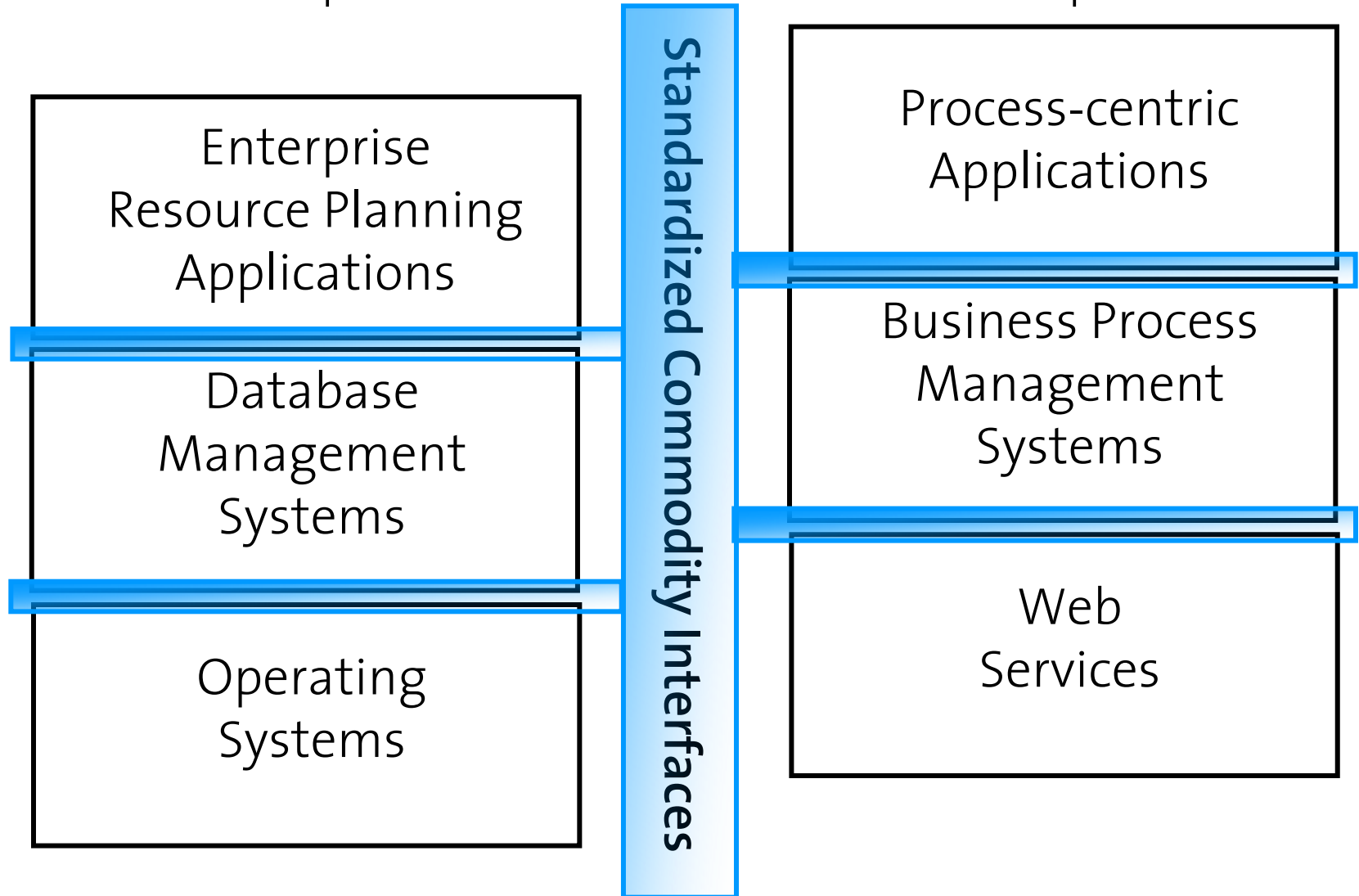
- The complexity of many business processes makes them a key element in a company's ability to evolve. Business Process Re-engineering is a way to define, better understand, and optimize these processes:
  - enhancing processing capabilities (1.5 million/invoices year),
  - reducing overhead,
  - decreasing processing time (7 days to 4 hours) and cutting processing cost.
- Processes that are already implemented with traditional middleware tools are very difficult to understand, let alone modify. A process, can be seen as a program: it needs to be well documented, in a suitable format, and have a consistent life cycle.
- For processes that are not implemented in software, the re-engineering effort often leads to attempts to implement them. This is where workflow technology plays an important role.
- Today, many of the individual activities within a business process are performed directly or indirectly with computers.
- As we rely more and more on computers, every PC and workstation in which individual activities take place becomes an isolated information repository where part of the business process resides. Each of those computers is an island of information.
- But these computers cannot easily talk to each other and it is not straightforward to gather all the information stored in each island.
- Under these circumstances, it is very difficult to get a global view of what is taking place: monitoring, auditing, data gathering...
- All the information islands must be connected together to build a single system in which business processes exist as physical concepts and not as abstract entities.

# Goals of B2B

- The goals of B2b or e-commerce are similar to those of workflow:
  - Reduce the cost of transactions
  - Provide a high level view of the IT operations
  - Map IT to business processes
  - Reduce overhead and time in performing operations
  
- The biggest obstacle for workflow was heterogeneity in the applications
  
- These obstacle goes away to a great extent through the use of web services
  
- Workflows, or processes, seem to be the best language for specifying at a high level how a collection of services should interact
  
- Workflows are often seen as the language of Service Oriented Architectures

# Why Standardize Composition?

- Portability and Interoperability of business process models defining executable compositions and abstract coordination protocols.





*Information and  
Communication Systems  
Research Group*



**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

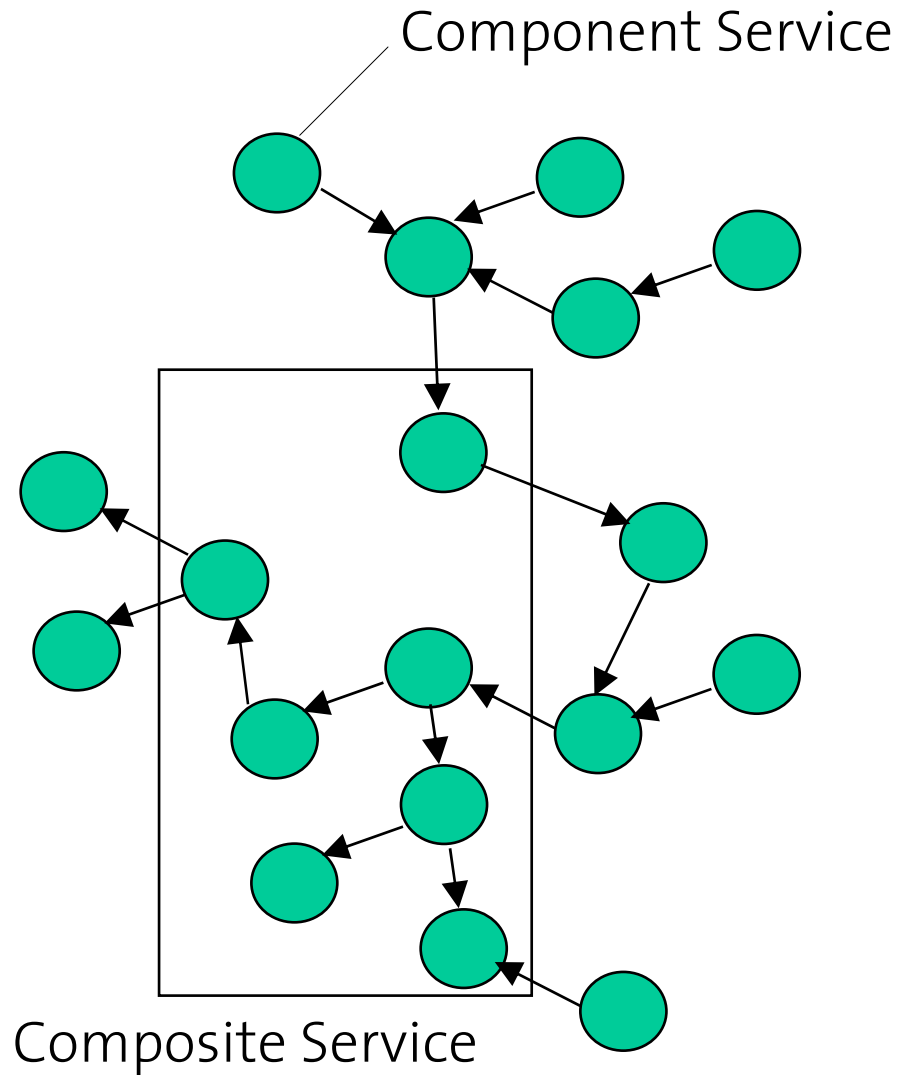
# Composition in the context of Web Services

# Value Added Services

- Once basic services begin to be published on the Web, the natural thing to do is to find and combine them into more complex, value added services.
- Assuming that the necessary basic services are available, it is possible to define the business logic of an application out of their composition. Different applications can be built by composing the same services in different ways.
- Note: composite services are still services, i.e., services that can be composed: **composition is recursive**
- Web service composition middleware provides abstractions for high level modeling of compositions and the infrastructure for executing them efficiently
- Examples:
  - Look for the cheapest price for a book/movie/song on all Internet e-stores and buy it, ensuring that it will be shipped before your birthday
  - Search from different Web search engines, merge all results by removing duplicates
  - Plan for a vacation: order plane tickets, reserve hotels and cars for a set of destinations
  - Convert stock quote prices to a different currency
  - Get weather information and snow & avalanches reports for the nearest ski resorts
  - Gather bids from suppliers, select the winner after a deadline and notify all participants of the outcome

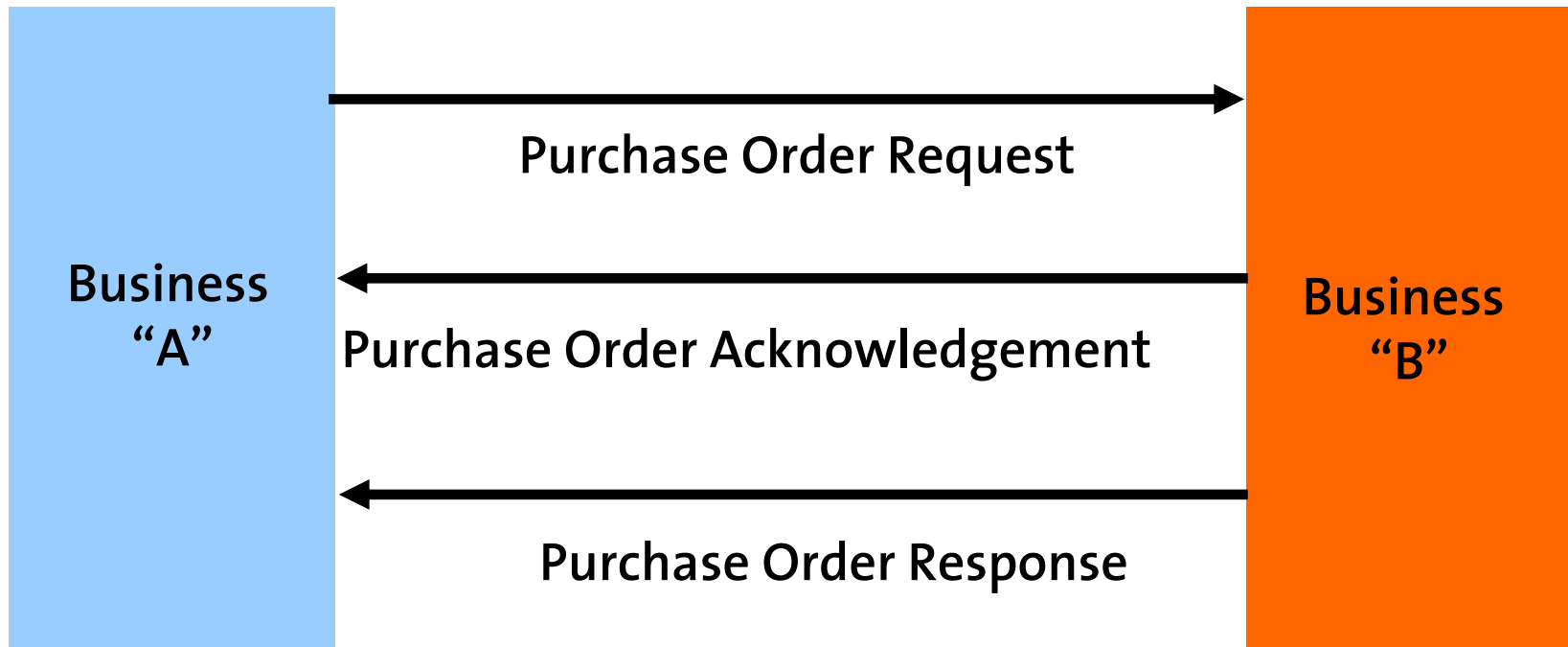
# Composition and Coordination

- Composition and coordination can be seen as two faces of the same problem.
- Composition models the internal structure and implementation of a service
- Coordination protocols focus on the external interactions of a set of services
- The composition must follow the coordination protocol of all of its component services
- The coordination protocol for the composite service can be inferred by looking at its internal structure.



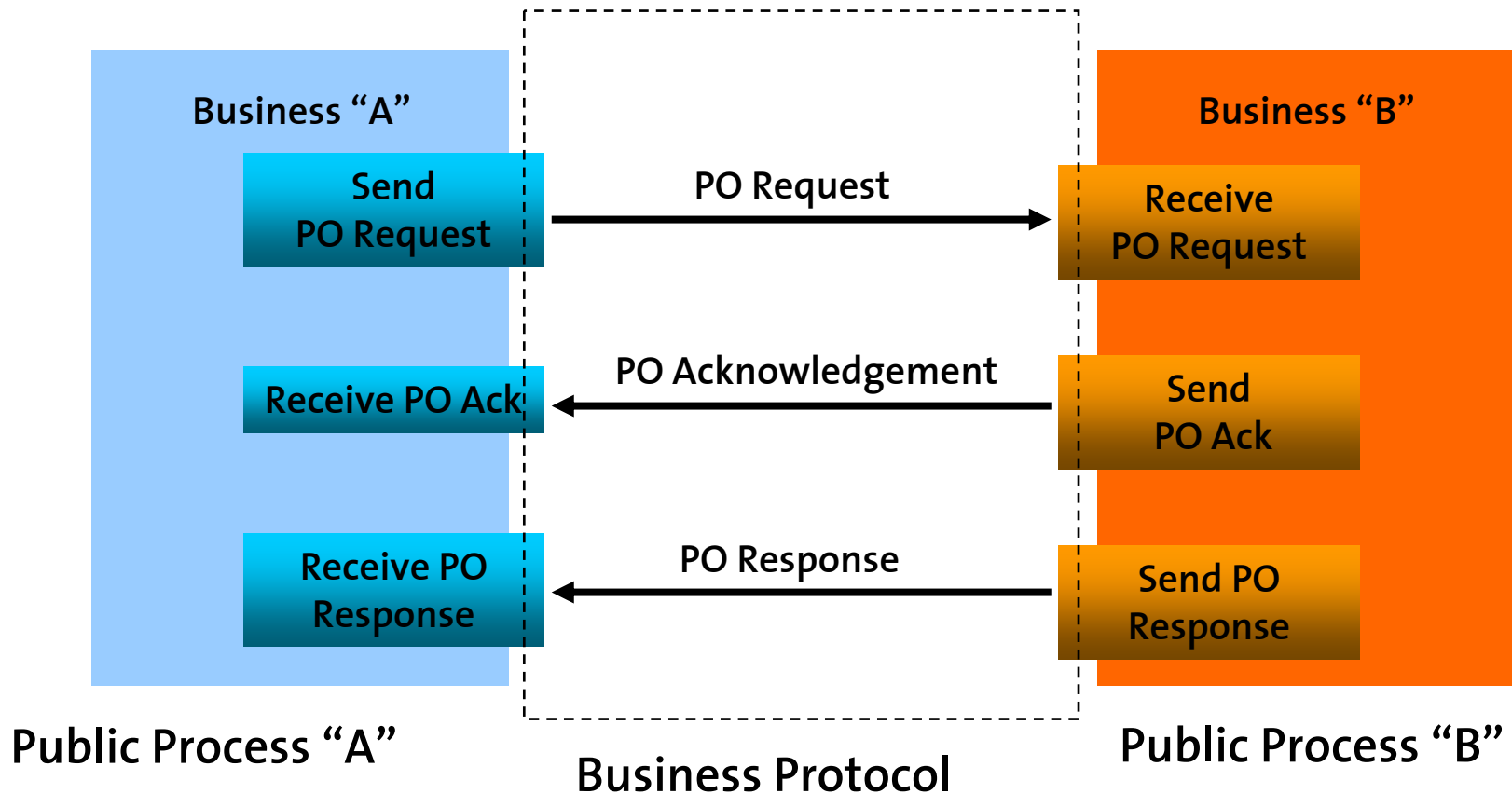
# Composition and Coordination Example

- This simple example defines a protocol to exchange a purchase order between two Web services of two different companies.
- A Purchase Order request is sent, followed by an immediate acknowledgement and a confirmation response later on.



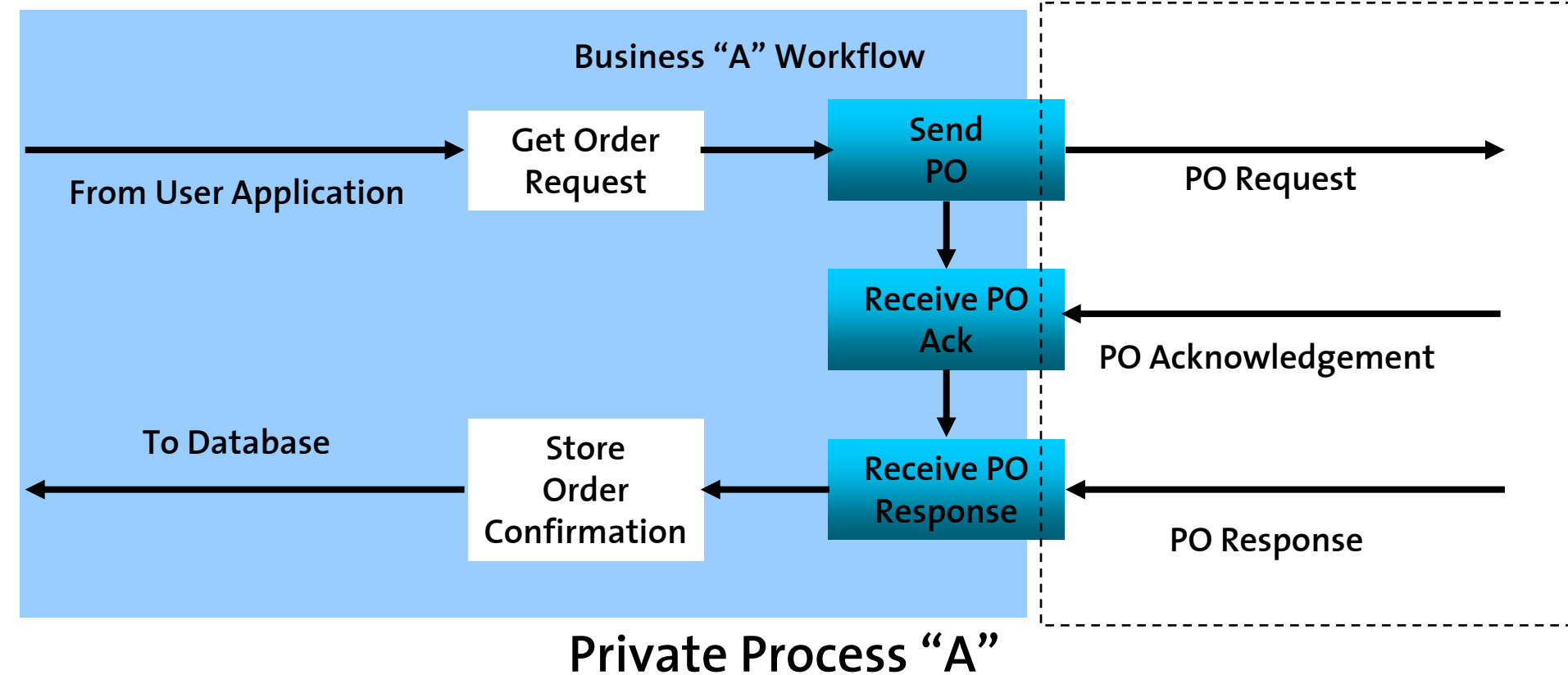
# Example: Coordination View

- The coordination view over this interaction defines the observable exchange of messages between the Web services.
- This is described by the business protocol (or public process) linking the two parties

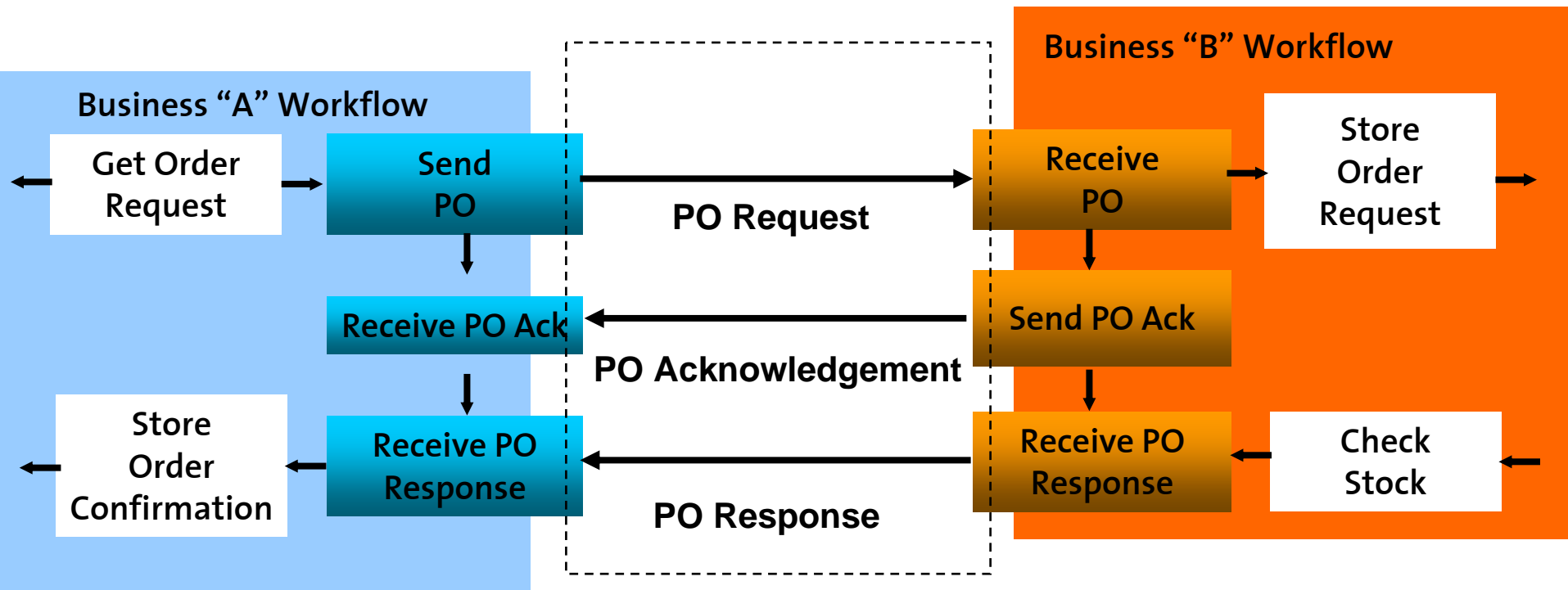


# Example: Composition View

- The composition view over this interaction defines the internal behavior of each of the Web services involved.
- This is described by the orchestration (or private process) of each of the parties involved



# Example: Complete View



- ❑ The public and private processes of each of the Web services must be consistent in order for the interaction to work.
- ❑ The public process (business protocol) that coordinates how the Web services interact must be agreed upon.
- ❑ The private process that orchestrates the message exchange with the internal systems is usually kept confidential.



*Information and  
Communication Systems  
Research Group*



**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

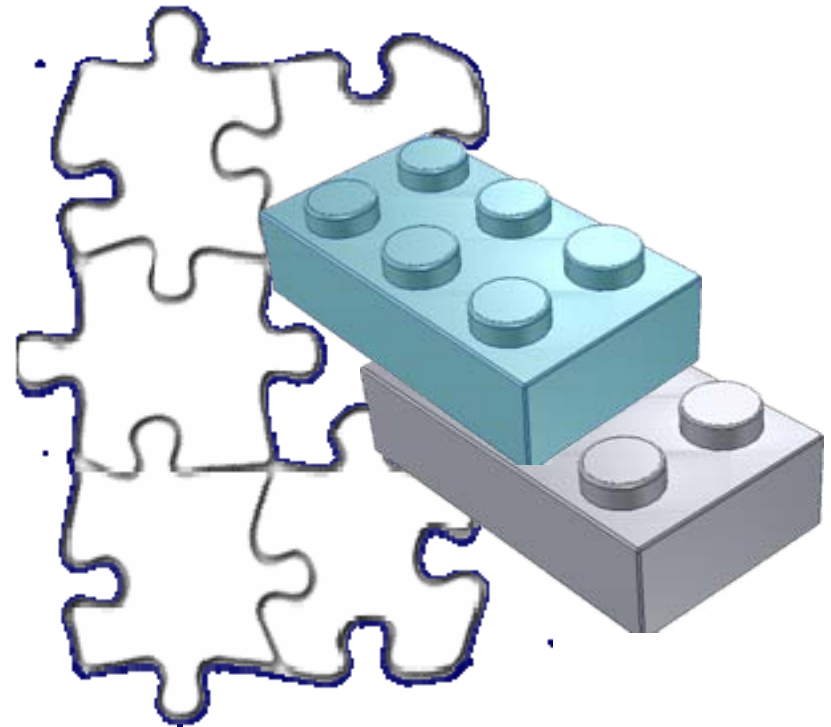
# Modeling Executable Compositions

# Aspects of Composition

- Component Model
  - Web services (Synchronous and Asynchronous)
- Composition Model
  - Formal: Statecharts, Petri Nets,  $\pi$ -Calculus
  - XML-Oriented: Activity Hierarchies
  - Workflow Based: WS-BPEL (XML), JOpera (Visual)
  - Others: Rule Based, UML Activity Diagrams, Data Driven
- Service Selection Model
- Data Transfer Model
- Exception Handling
- Transactions

# Component Model

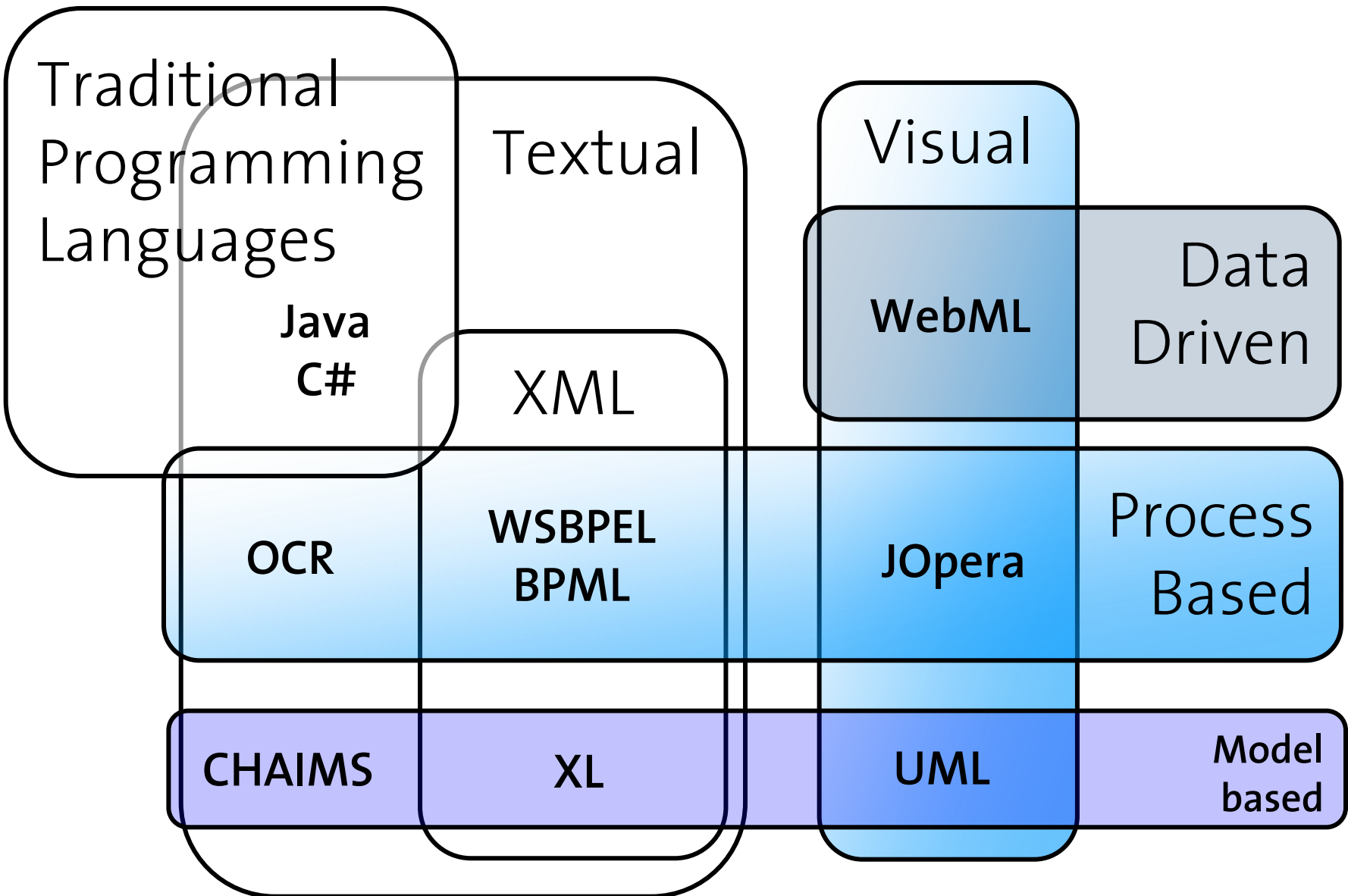
- A component model defines the assumptions about the components that should be composed.
- In general, composing homogeneous components is easier than composing heterogeneous ones as the corresponding infrastructure is less complex.
- In one case, all components could be Web services (accessed with the SOAP/HTTP protocols and described by a WSDL document)
- On the other extreme, the components are just “services” that can be accessed using a variety of invocation mechanisms.



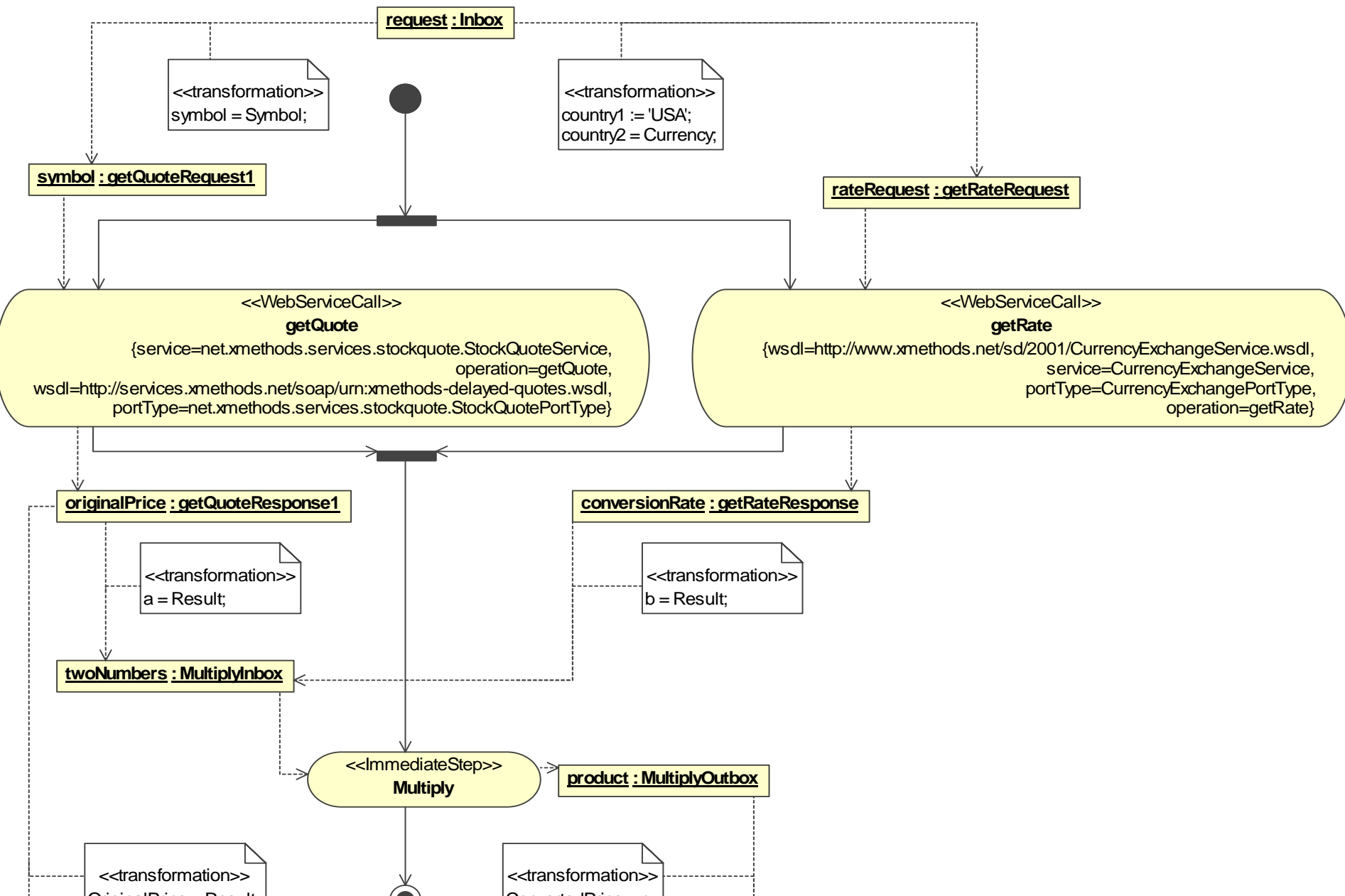
# Composition Model

- Composition (or orchestration) models define how to build a coherent system out of a collection of services.
- They define:
  - what is the order in which the services are invoked and what are the conditions under which a certain service may or may not be invoked (**control flow**)
  - how the services exchange data with one another (**data flow**) and how they interact.
  - Some form of **exception handling**, i.e., what happens if a service is not available
- Many different languages for modeling compositions exist.
- Composition can be defined along the spatial or temporal dimension.
- Architectural Description Languages (ADL) describe the spatial architecture of a system in terms of components and connectors.
- Workflow definition languages use the notion of process to define how components interact in time.
- Note: traditional programming languages do also fulfill the requirements for composition languages.

# Languages for Composition



# Extended UML Activity Diagrams



- XML based
- textual
- mirrors workflow languages

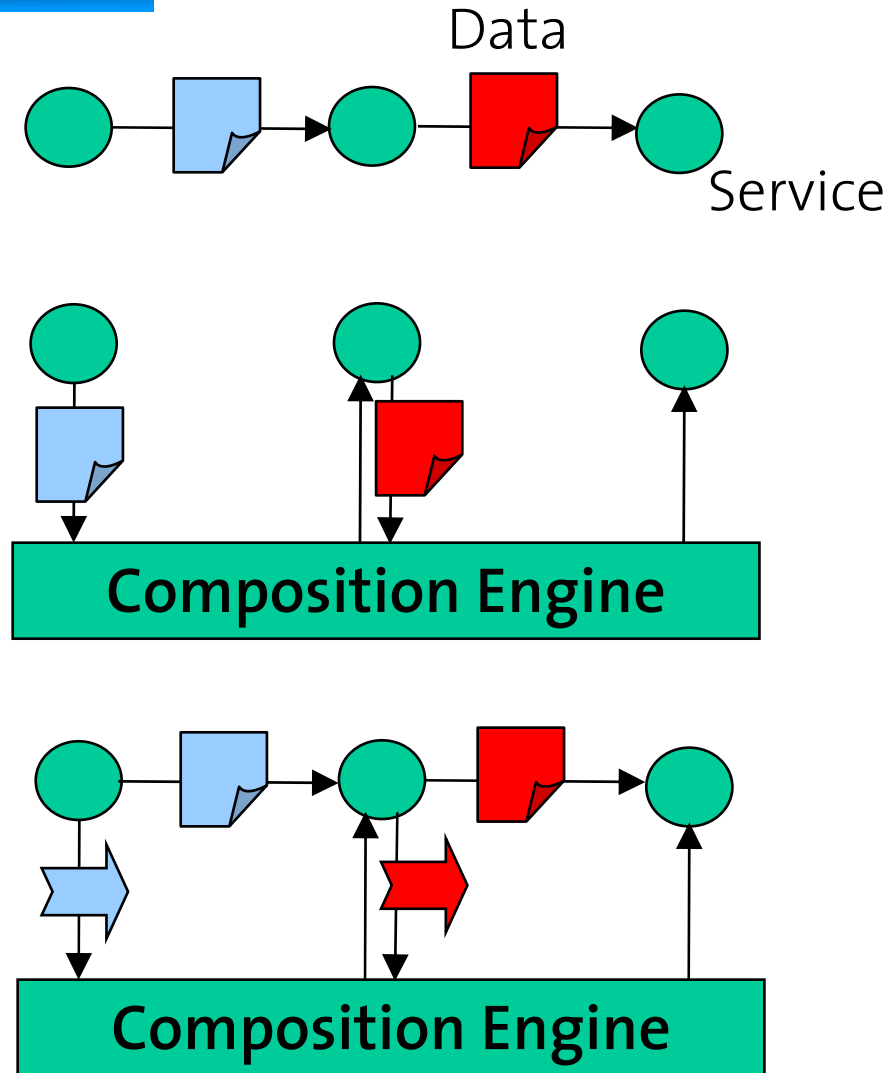
```
<process name="shippingService"
  targetNamespace="http://acme.com/shipping"
  xmlns="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
  xmlns:sns="http://ship.org/wsd/shipping"
  xmlns:props="http://example.com/shipProps/"
  abstractProcess="yes">
  <partnerLinks>
    <partnerLink name="customer"
      partnerLinkType="sns:shippingLT"
      partnerRole="shippingServiceCustomer"
      myRole="shippingService"/>
  </partnerLinks>
  <variables>
    <variable name="shipRequest"
      messageType="sns:shippingRequestMsg"/>
    <variable name="shipNotice"
      messageType="sns:shippingNoticeMsg"/>
    <variable name="itemsShipped"
      type="props:itemCountType"/>
  </variables>
  <correlationSets>
    <correlationSet name="shipOrder"
      properties="props:shipOrderID"/>
  </correlationSets>
  <sequence>
    <receive partnerLink="customer"
      portType="sns:shippingServicePT"
      operation="shippingRequest"
      variable="shipRequest">
      <correlations>
        <correlation set="shipOrder" initiate="yes"/>
      </correlations>
    </receive>
```

# Service Selection Model

- In addition to specifying what are the messages to be exchanged, a composition model should also define how to select the services that should receive or send such messages.
- In order to enhance the reusability of a composition, such services are usually not hard-coded into the composition but bound into it at different times:
  - Composition time
  - Compilation and deployment time
  - Startup time
  - Invocation time
- The service selection model defines how services are bound into a composition:
  - Static binding (URL of service endpoint is hard-coded)
  - Dynamic binding by reference: (Service URL is computed and stored into a variable)
  - Dynamic binding by lookup (before each service invocation a query is sent to a registry to locate a suitable implementation)
  - Dynamic operation selection (no assumptions are made about the signature of the arbitrary service to be invoked)

# Data Model

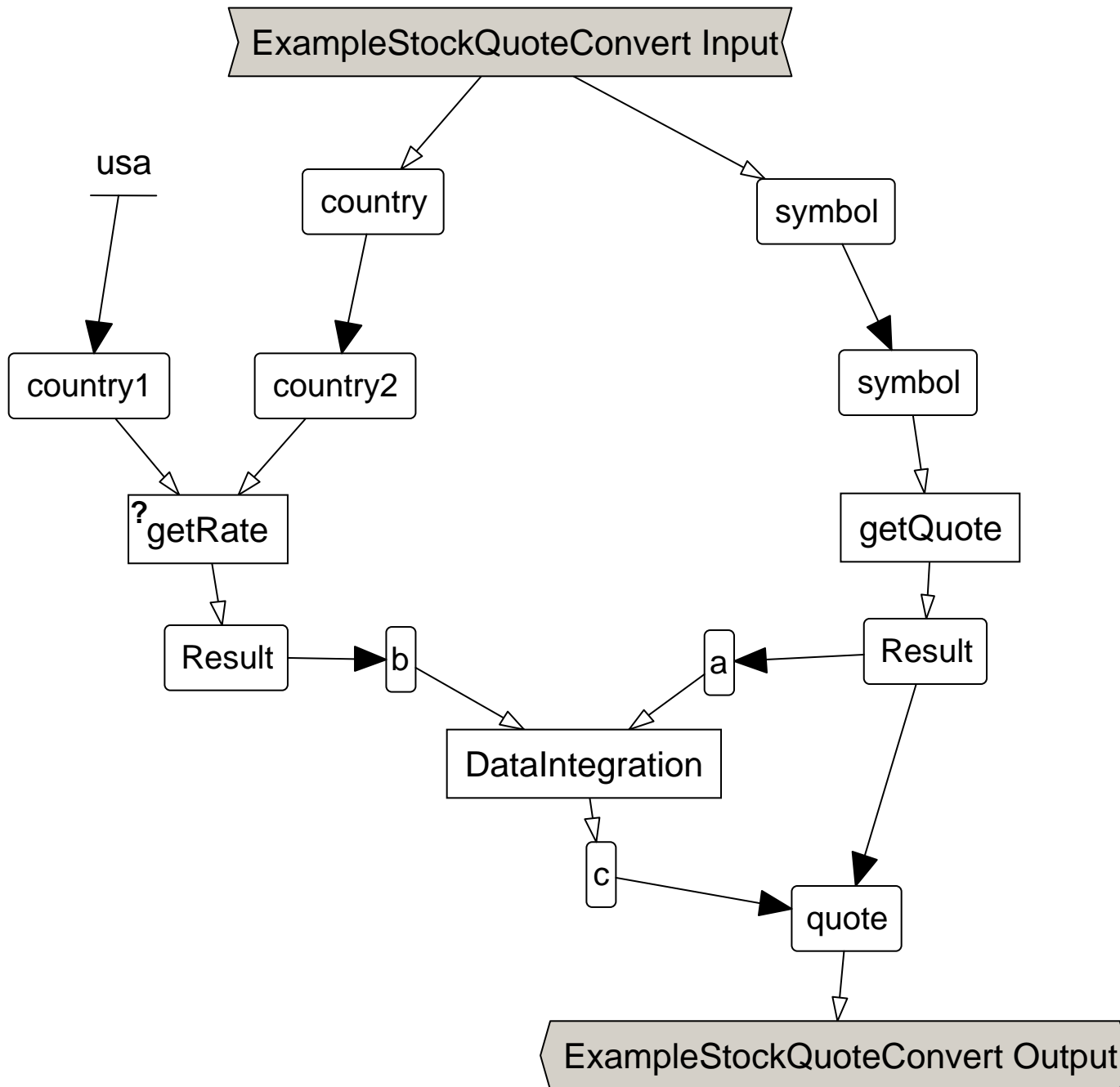
- Services typically interact by exchanging some data.
- Not all composition languages include an explicit model of the data flow.
- Data is not always treated in the same way:
  - Composition-level data is modeled at a finer level as it is used to control the composition (control flow branches) and has data types associated with it.
  - Application-specific data is seen as an opaque pointer (URL) which is forwarded between services



# Data Transfer Model

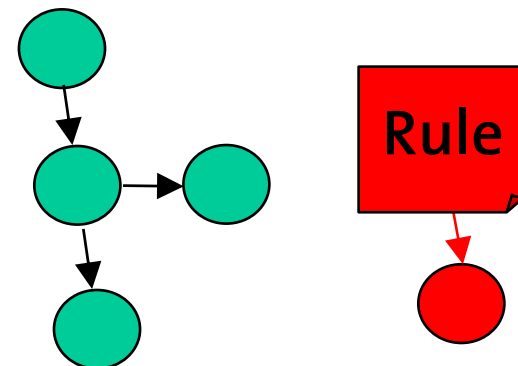
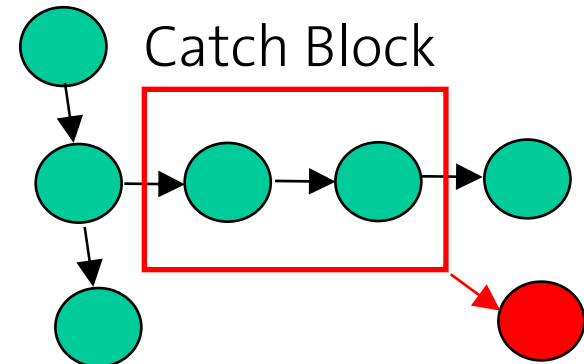
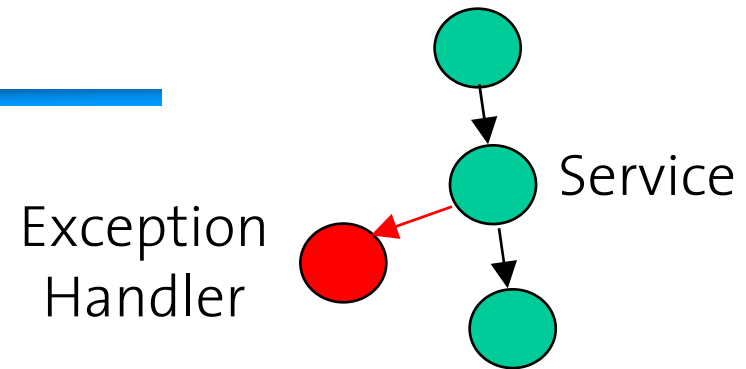
```
<containers>
<container name="input" messageType="...">
<container name="request" messageType="...">
<container name="response" messageType="...">
<container name="output" messageType="...">
</containers>
<sequence>
  <receive container="input"/>
  <assign>
    <copy><from container="input"/>
      <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>
```

- 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 between 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.

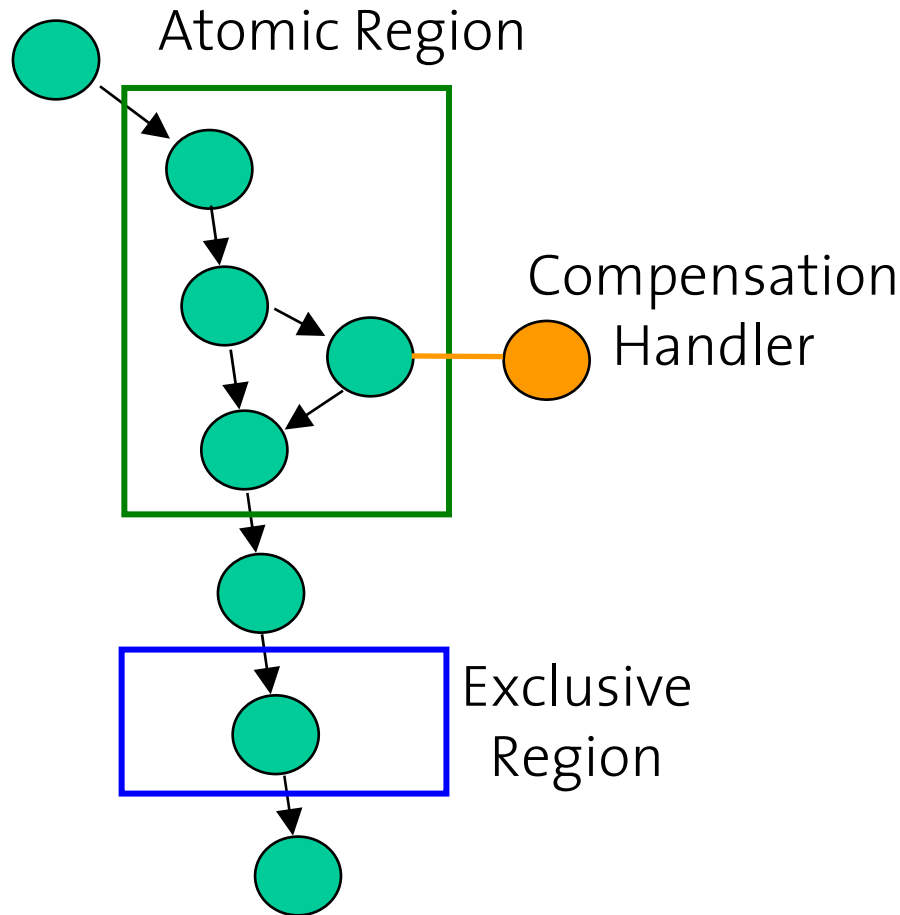


# Exception Handling

- Exception Handling is very 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.
- Rules may also be associated to a composition in order to detect global exceptional conditions



# Transactions and Exception Handling



- 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.



*Information and  
Communication Systems  
Research Group*



**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

# Web Services Business Process Execution Language (WS-BPEL)

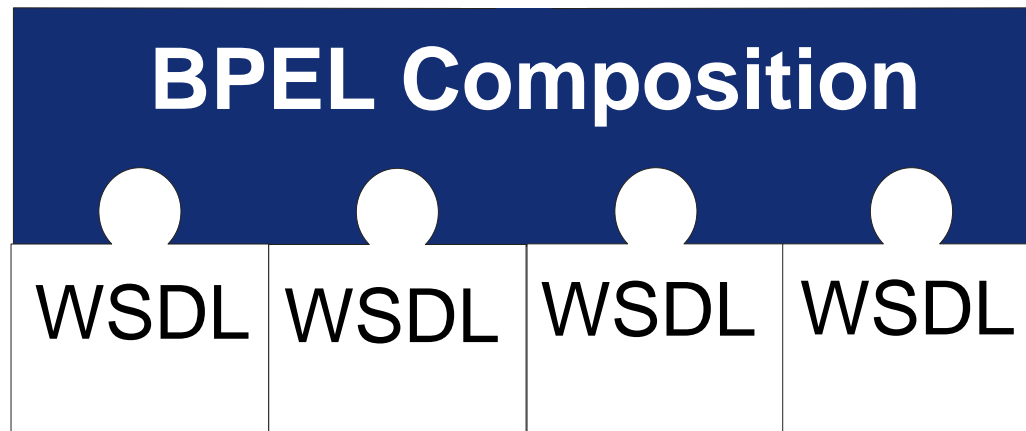
# What is WS-BPEL?

- 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
- 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 between different services. These process 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.

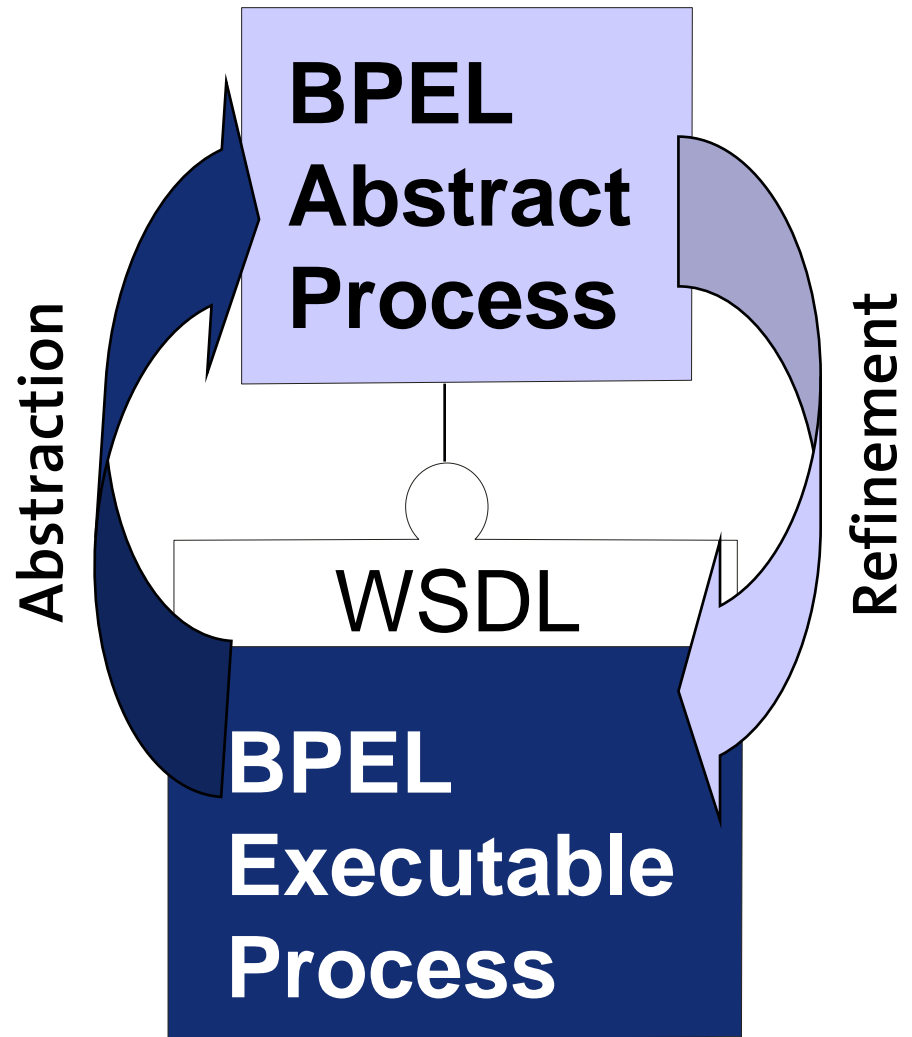
## WSDL



## Web Service Interfaces

# Using Abstract Processes

- Abstract Processes define constraints on the WSDL interface of a Web service so that it can be used correctly.
- Application Example: RosettaNet PIPs (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.
- The abstract process definition can also drive the implementation of the private executable process behind a Web service.



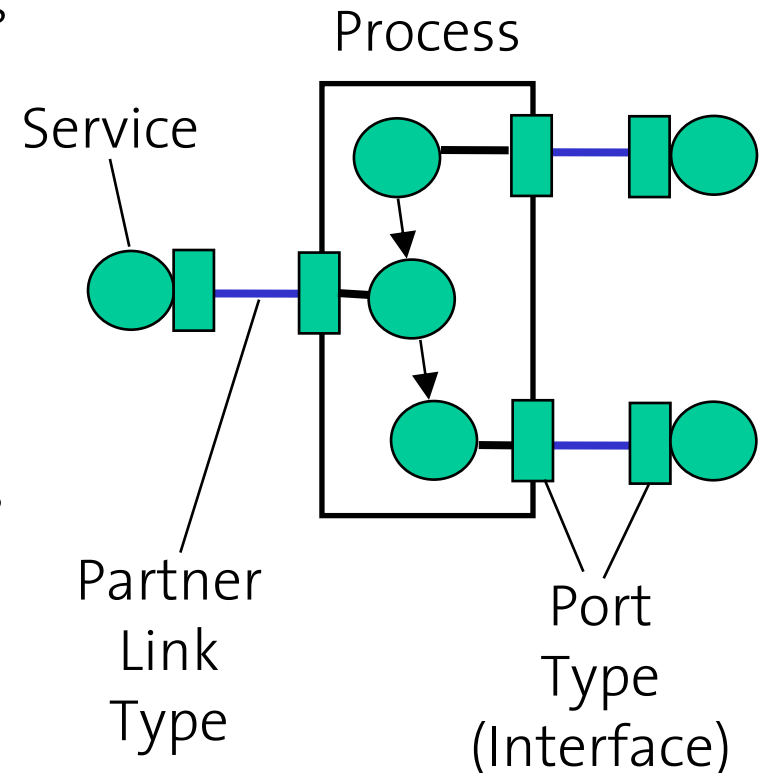
# Elements of WS-BPEL

- Partners
- Properties
- Correlation sets
- Variables
- Scopes
  
- Fault Handlers
- Compensation Handlers
- Event Handlers
- Activities
  - Structured activities
  - Simple activities
  
- Equivalent to declarations in a normal programming language. They defines 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

# Partners Link Types (Service Selection)

- 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 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, 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.

- The notion of Partner Link Type reflects a peer-to-peer relation between the process and each one of the services the process calls



# Example

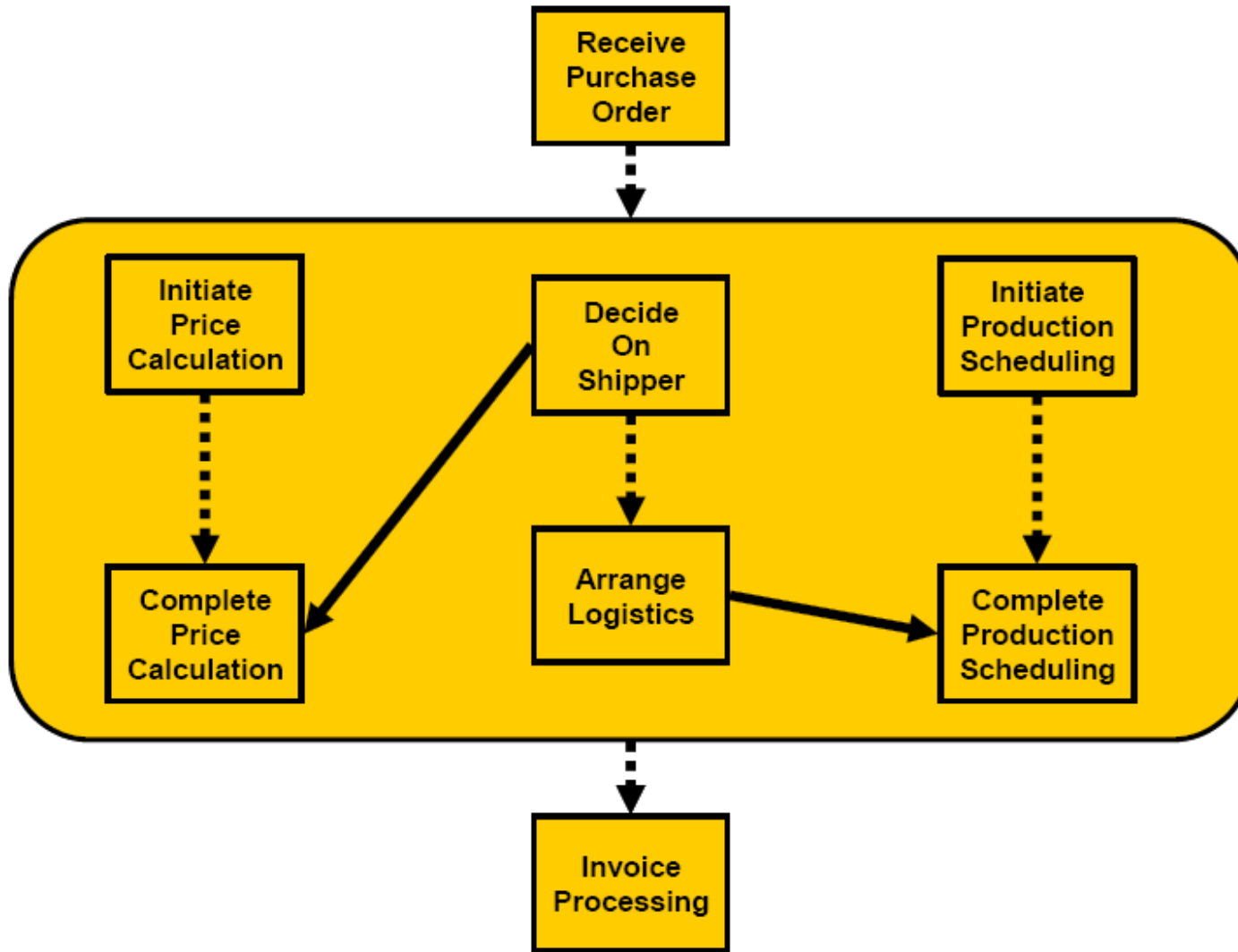
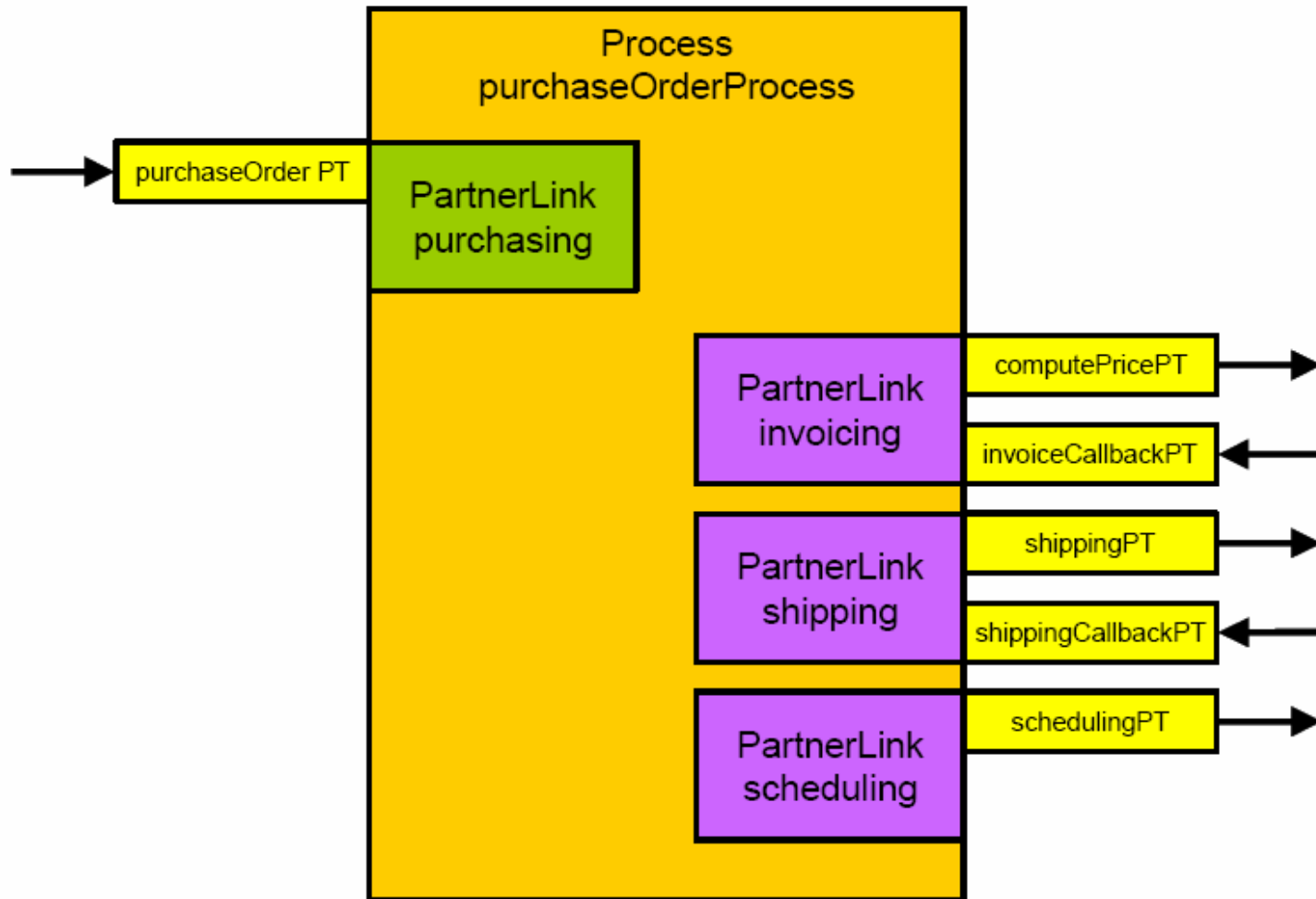


Figure 1: Purchase Order Process - Outline

# Example partner links



<http://www.oasis-open.org/committees/download.php/22475/wsbpel-v2.0-CS01.pdf>

# Properties and Correlation Sets

- **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.
- 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 between data, messages and properties that help a process instance identify the messages that should be handled by itself.
- 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.
- 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 (Data)

- Variables can be used to store:
  - the content of the SOAP messages that are exchanged with the partners (with messageTypes defined 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
- 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 XPath queries to messages in order to extract relevant information

## Example: A = B

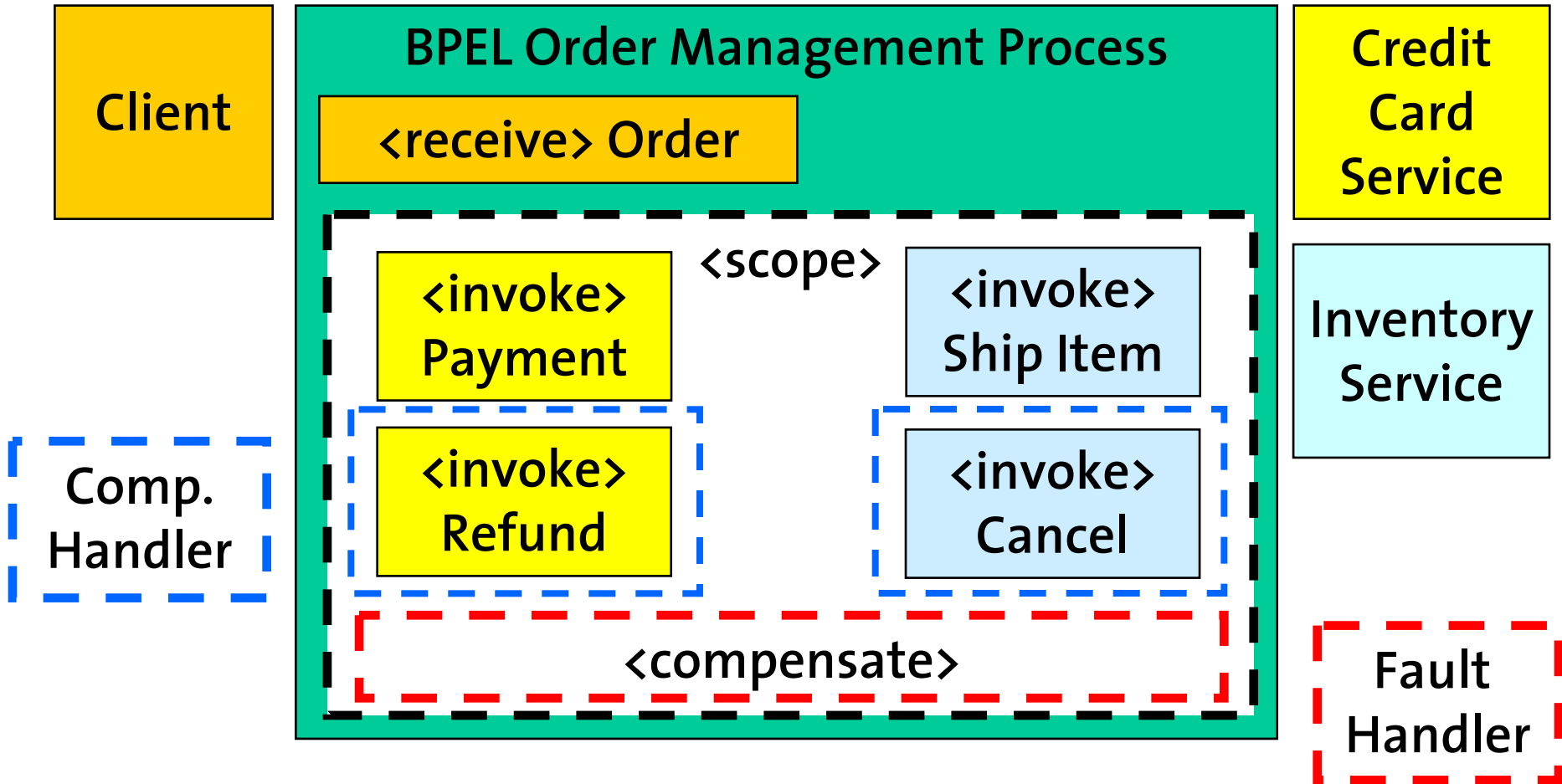
```
<assign>  
  <copy>  
    <from variable="A" />  
    <to variable="B" />  
  </copy>  
</assign>
```

# Recoverability and Compensation

- For long-running business processes it is important to have recoverability built into the composite service.
- BPEL provides different handler constructs that can be associated with a scope.
- **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 models and 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

- Activities are 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 (Control Flow)

- 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

## sequence

### switch

#### sequence

invoke

#### while

invoke

#### flow

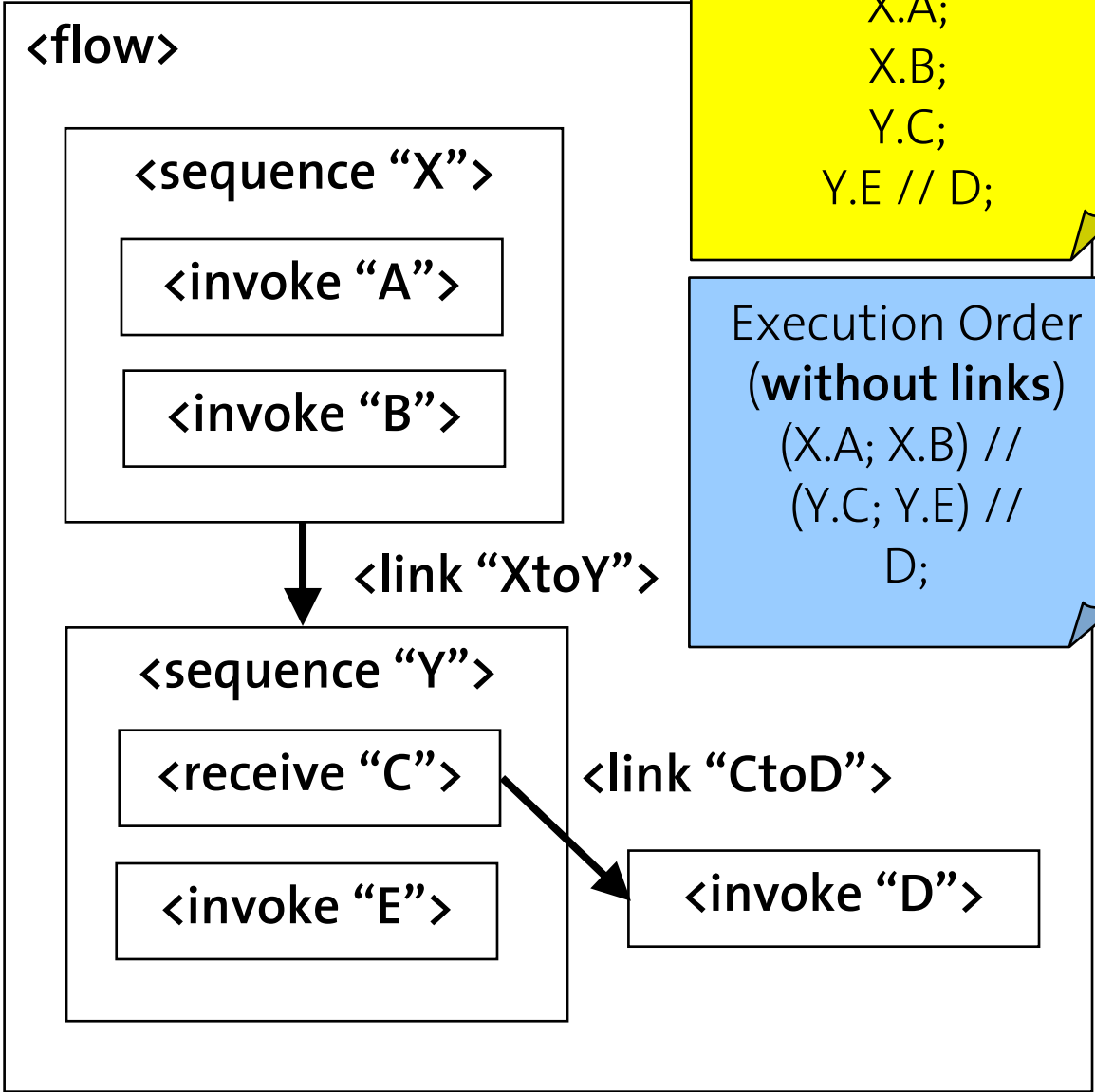
invoke

scope

invoke

# 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>
```



Execution Order  
**(with links)**  
X.A;  
X.B;  
Y.C;  
Y.E // D;

Execution Order  
**(without links)**  
(X.A; X.B) //  
(Y.C; Y.E) //  
D;

# 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.

## 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.