Composing and Orchestration of Distributed Services, WS-BPEL

Eetu Mäkelä and Tuukka Ruotsalo

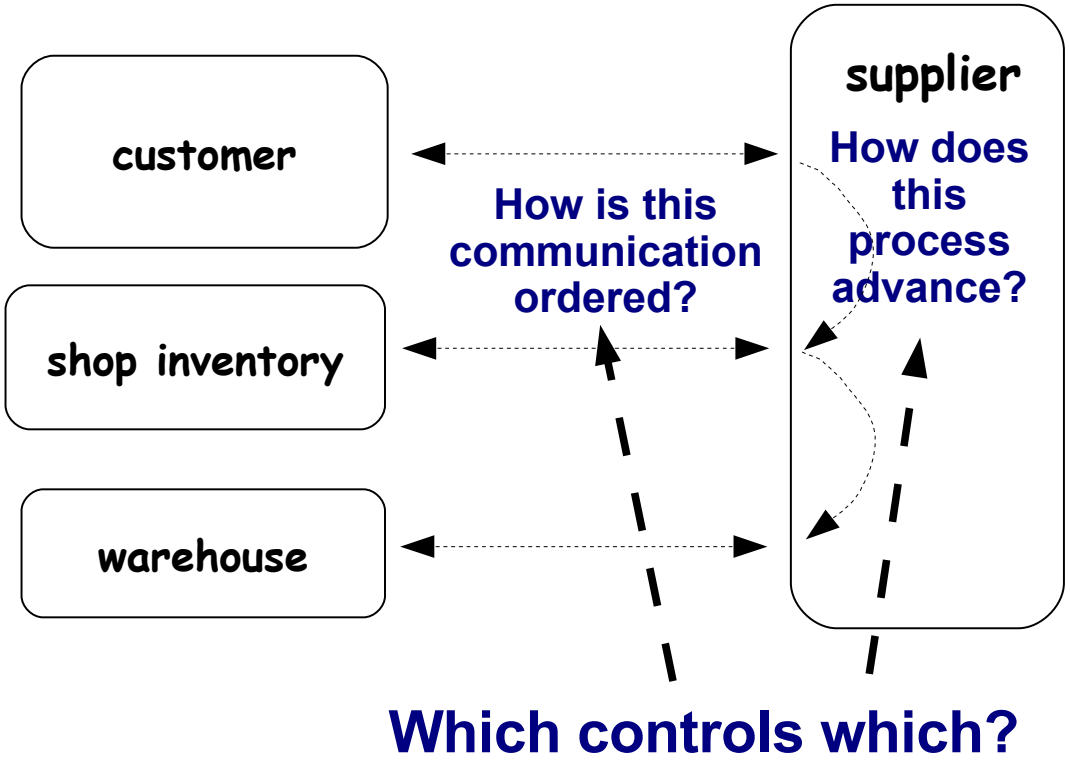
(partly based on Alonso & Pautasso, 2004)

Contents

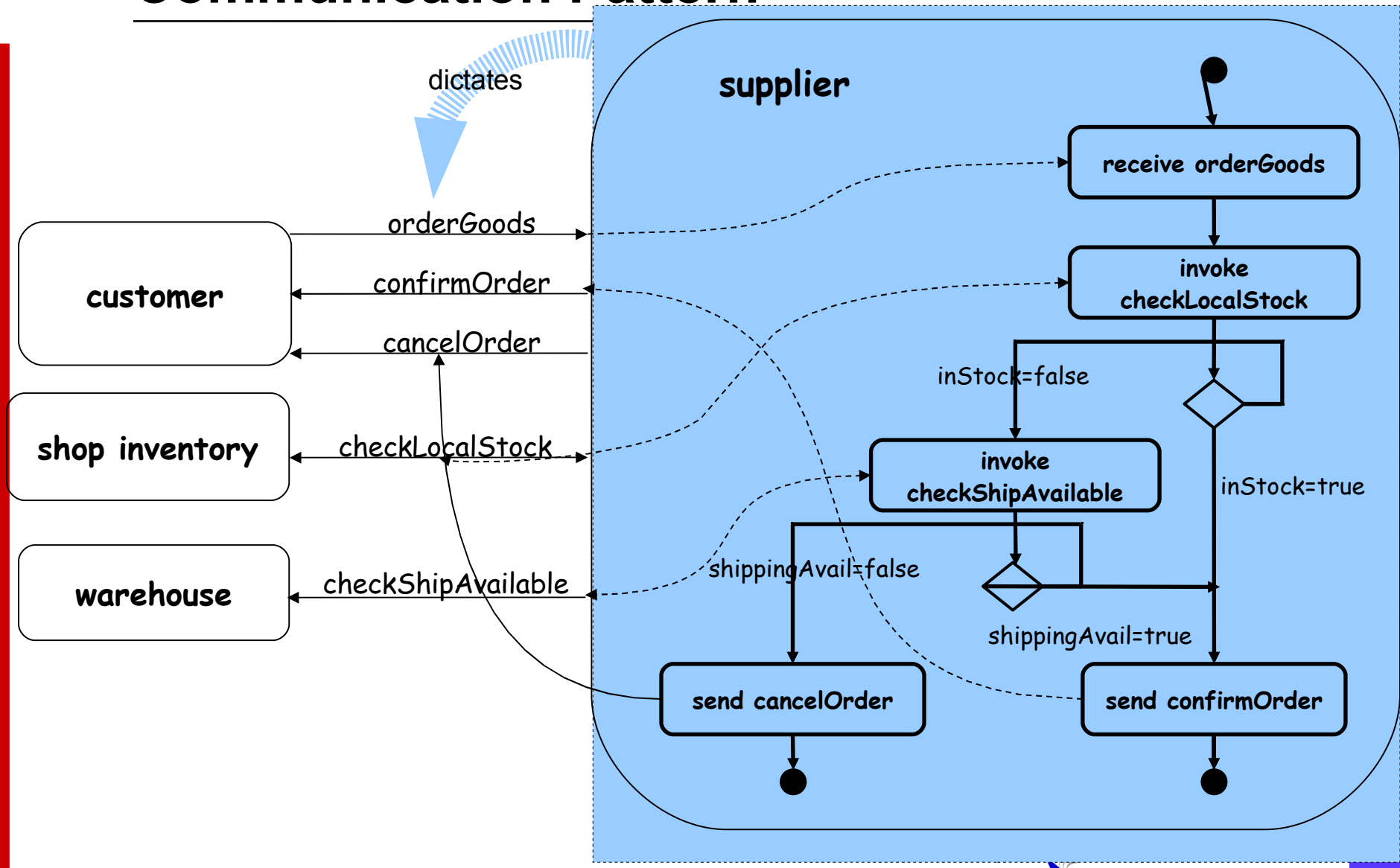
- Introduction and example
- Orchestration vs. choreography
- Different approaches to composition
- Composition and coordination
- Composition middleware
- Modelling composite processes
- WS-BPEL / BPEL4WS

Introduction and Example: Buying skateboots

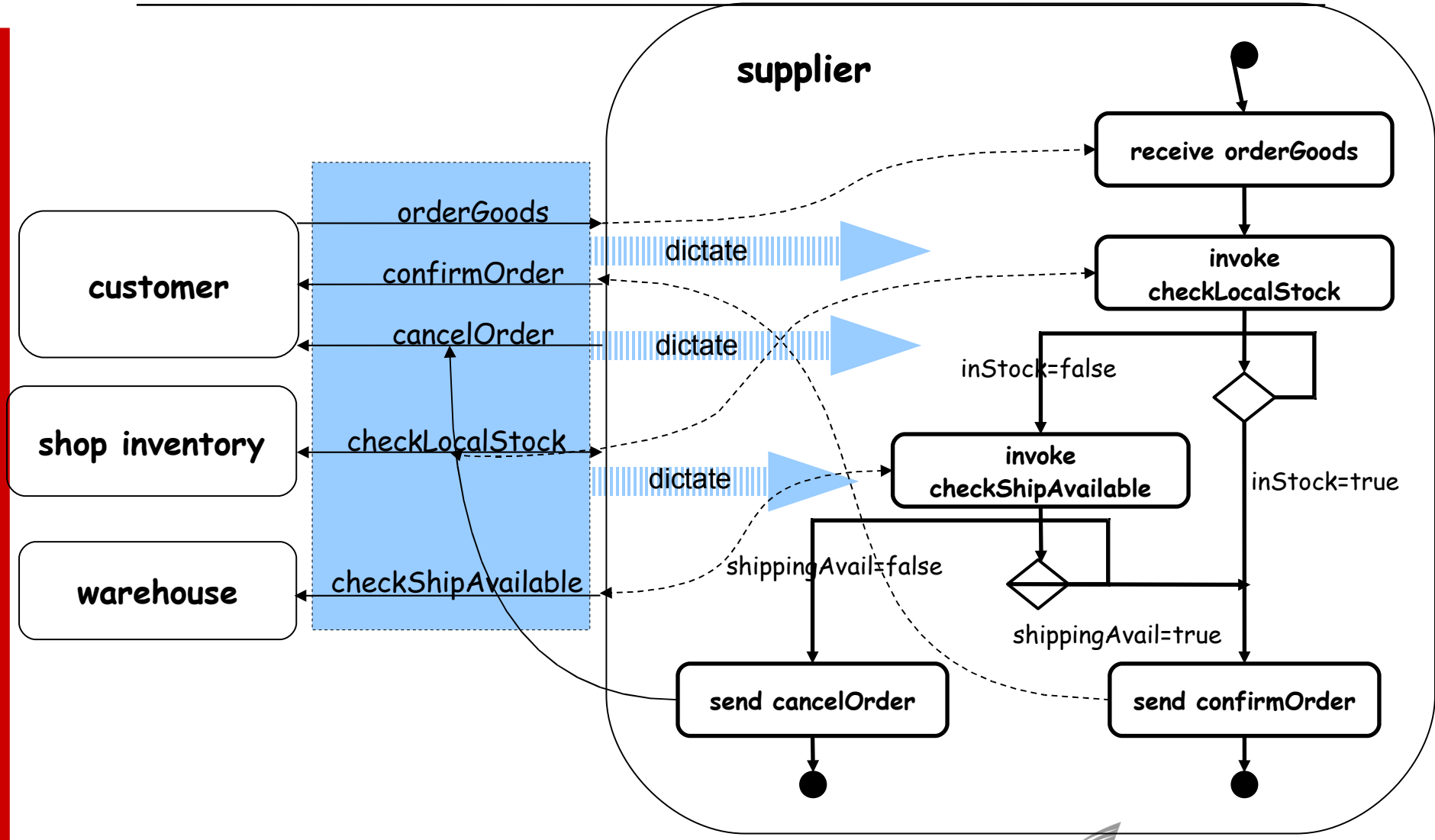
- In a Web Services environment, there is a need for combining the functionality provided by Web Services into a composite service. This is called composition.
- Depending on the messages arriving and sent within the partner network, we should be able to decide what to do next.
- Two viewpoints:
  - Process description dominant: Orchestration
  - Communication pattern dominant: Choreography



Orchestration: Supplier Process Dictates Communication Pattern



Choreography: Communication Pattern Requirements Dictate Supplier Workflow



Approaches

- Process description dominant:
  - Primary controller process and pre-defined execution model
  - Code approach
    - Centered traditional programming language based control
  - Web Service Composition Middleware
    - Higher level programming models (Workflows, WS-BPEL)
  - Extensions: semi-automated service composition based on business models (WS-CDL / RosettaNet / ebXML)
    - Any partner capable may participate to controller process requests
- Communication pattern dominant:
  - Automated service composition (AI / Semantic Web Services)
    - Partners are found through their capabilities and consuming is planned “online”

Code Approach

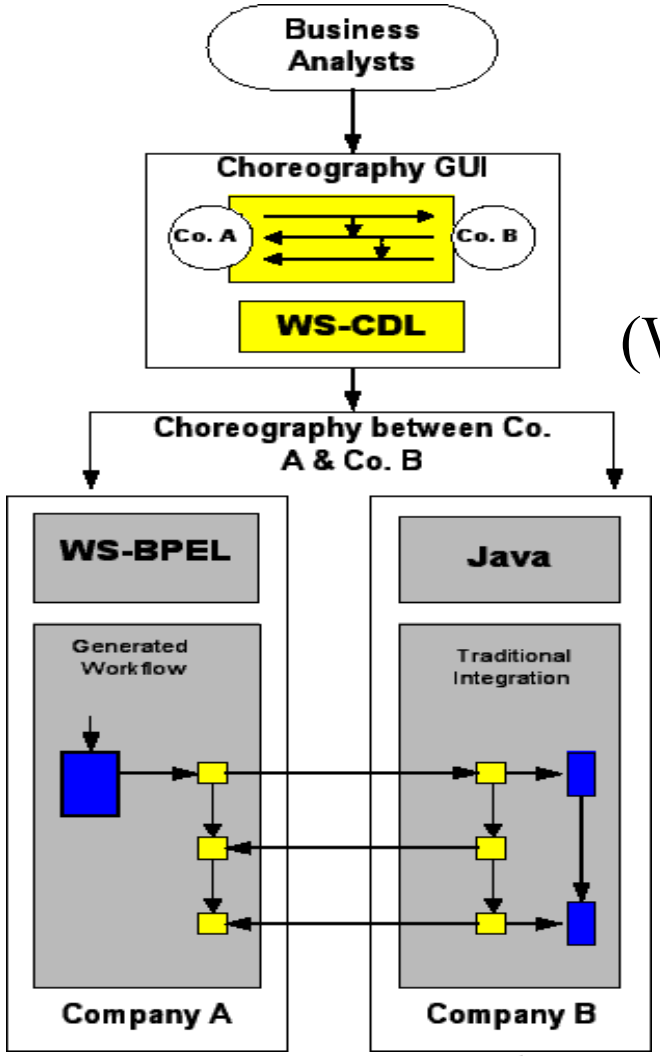
- Use traditional complex programming languages (such as Java) to compose and make decisions of consuming Web Services
- Useful in single enterprise settings to bridge heterogeneous information systems
- Reliable built-in extensions exist (Axis, TP-monitors etc.)
- Complex maintenance, but allows very complex business logic

Orchestration

- Code approach may well be enough
- In a well controlled environment there is a need for a simple process control language that only knows how to consume services and recover from error states (supplier controls the process, partners control any subprocesses).
- Orchestration provides a separated process control for pre-defined services
- Easier maintenance because we just have to reconfigure the process description to change the application logic
- Simple language but enough expression power to handle the workflow execution

Choreography

- In an open environment there is a need for a description language that describes the services and waits for someone to negotiate and consume (e.g. Each subprocess is a software agent that may participate).
- Choreography defines the composition of interoperable collaborations between any type of party regardless of the supporting platform or programming model used by the implementation of the hosting environment
- Extends the orchestration by defining the abstract communication model



(W3C.org)

Composition and Coordination

- Service interfaces are separated from their implementations
  - A consumer of a service sees just the service interface they are interested in
  - **A service may be composed of other services or direct state tracking systems**
- A coordinator is responsible of controlling the external consistency of multiple services (external implementation) – tied closely to choreography
- A composition engine is responsible for internal consistency of a service (= composition of services that have single accesspoint) – tied closely to orchestration
- Conversation control refers to external communication

Web Service Composition Middleware

- Use higher level languages to specify the workflow of the data and processes
- Takes care of the execution and control of the process model and data transformations
- Easy maintenance, because of the configurability
- May include semantics
  - Services describe themselves in a knowledge base. Any service may act as an agent and seek (and negotiate about consuming) suitable services
    - Automated service matchmaking
    - Automated planning of workflows
    - Automated service composition and execution

What Makes Web Services Succesful as Composition Middleware?

- Standardization of Interfaces and Data
  - XML, XSD, WSDL, UDDI
    - Any traditional data structure syntax may be transformed to XML based languages
- Accessibility (WEB)
- Supporting specifications
  - WS-Transactions, WS-Addressing etc.
- Standardization of Process level
  - WS-BPEL?
    - High Level Languages defined just to operate at the process level
- Extendable Semantics and Agent-Based Services
  - RDF, OWL, OWL-S?, WSMO?
    - Ontology based communications and intelligence

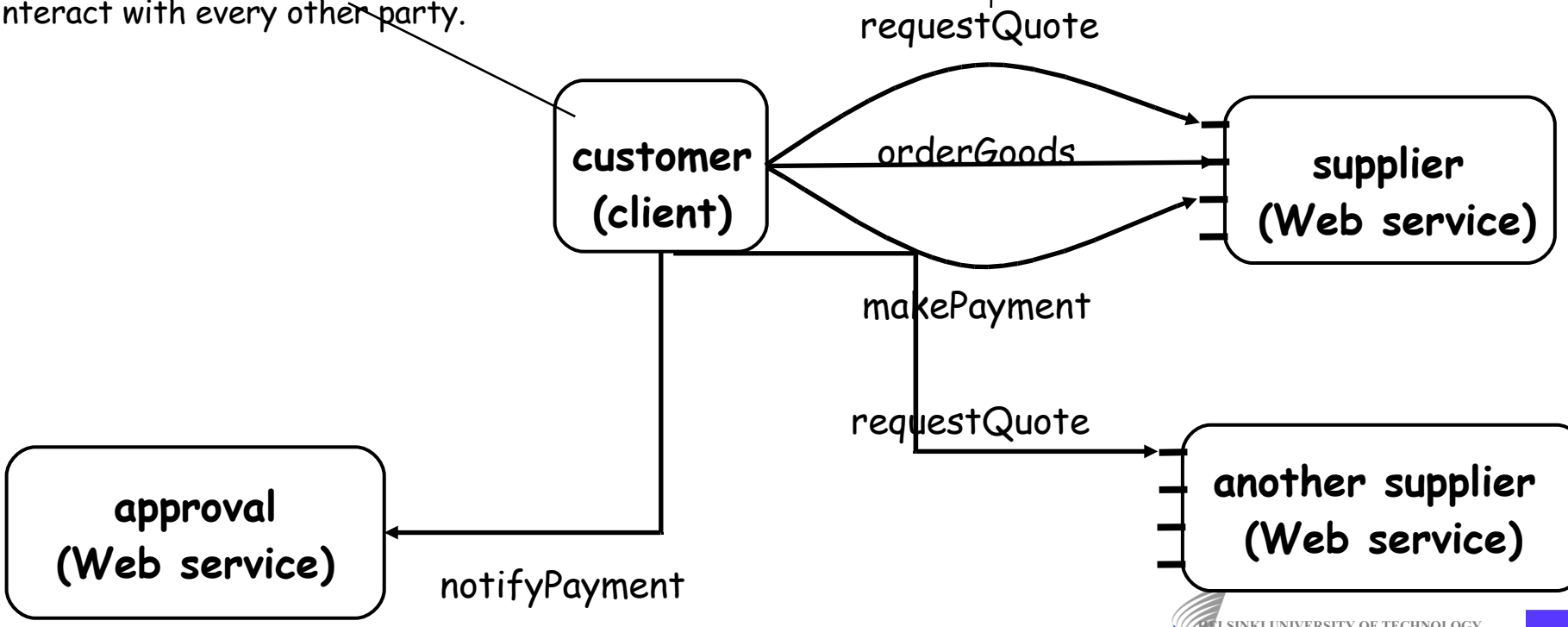
Dimensions of a Web Service Composition Model (Alonso et. al., 2004)

- Component Model
  - Defines the nature of the elements to be composed
  - Example: WSDL
- Orchestration Model
  - Abstractions and languages to execute the components
  - Example: BPEL
- Data and Data Access Model
  - How data is specified and exchanged
  - Example: XML and SOAP
- Service Selection Model
  - How the binding takes place i.e. How the service is selected to be executed
  - Example: URI, UDDI
- Transactions
  - How transactional semantics can be associated to the composition
  - Example: WS-Transactions
- Exception Handling
  - Transactional recovery

Clients in the Web Service World

A client engages in different conversations with several Web services. In general, these conversations may be regulated by different protocols, and each invoked Web service may not be aware that the client is invoking other Web services.

The internal business logic of clients and Web services is quite sophisticated, as it must support the execution of different conversations so that each party can properly interact with every other party.

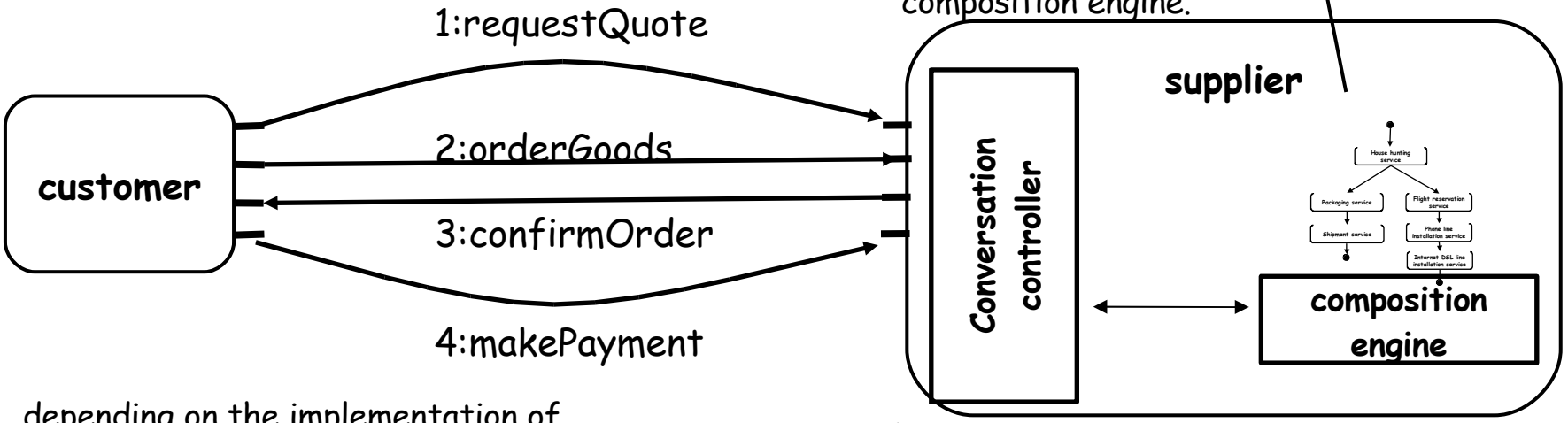


(Alonso & Pautasso, 2004)

Composition Middleware

the *procurement* business protocol executed among Web services

if the supplier is implemented by means of composition technologies, then its business logic is defined by a composition schema and its execution is driven by a composition engine.



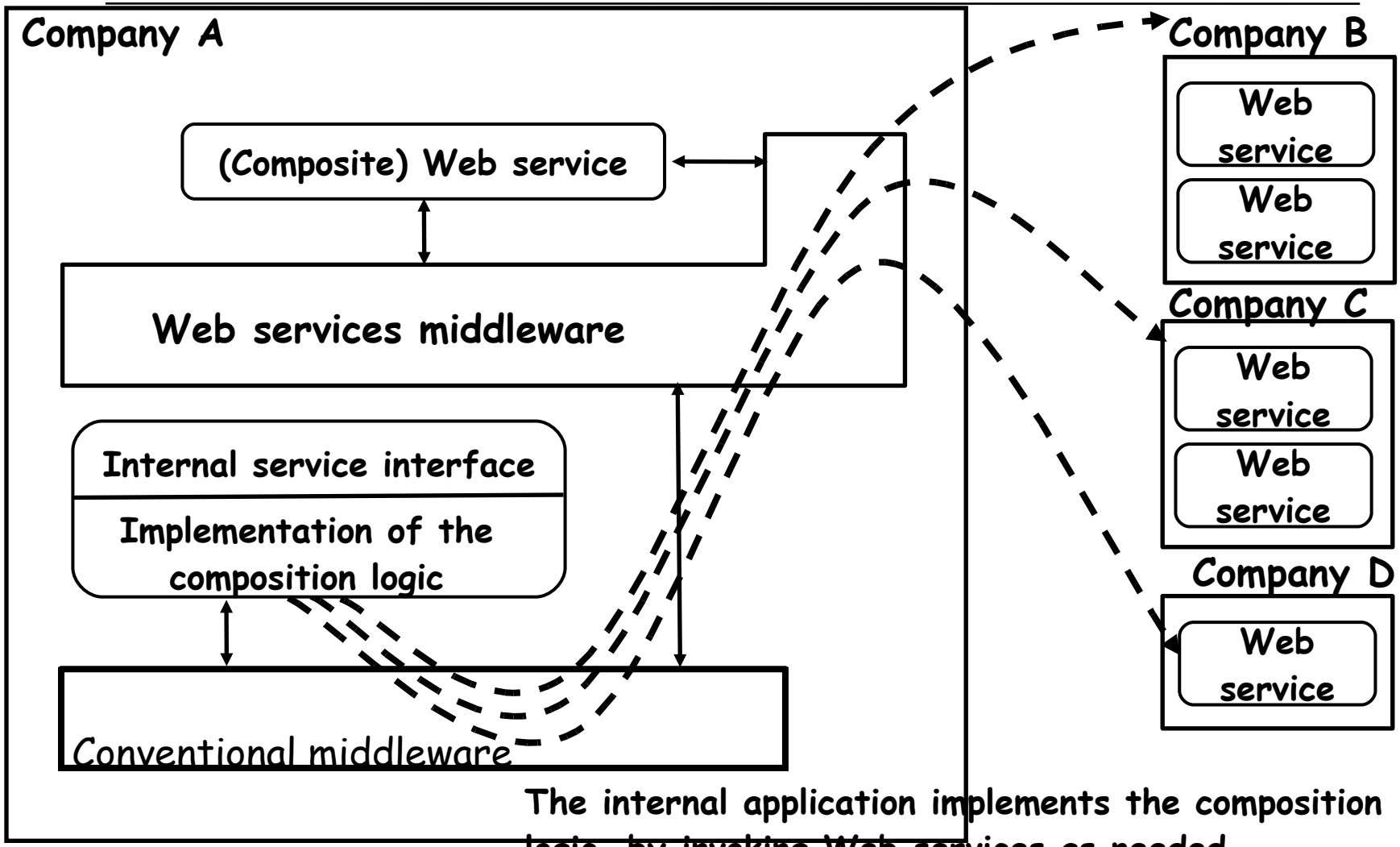
depending on the implementation of the (composite) service, the supplier may contact other Web services. The customer is unaware of these interactions, that may occur according to other protocols.

another Web service, possibly offered by another company

yet another Web service

(Alonso & Pautasso, 2004)

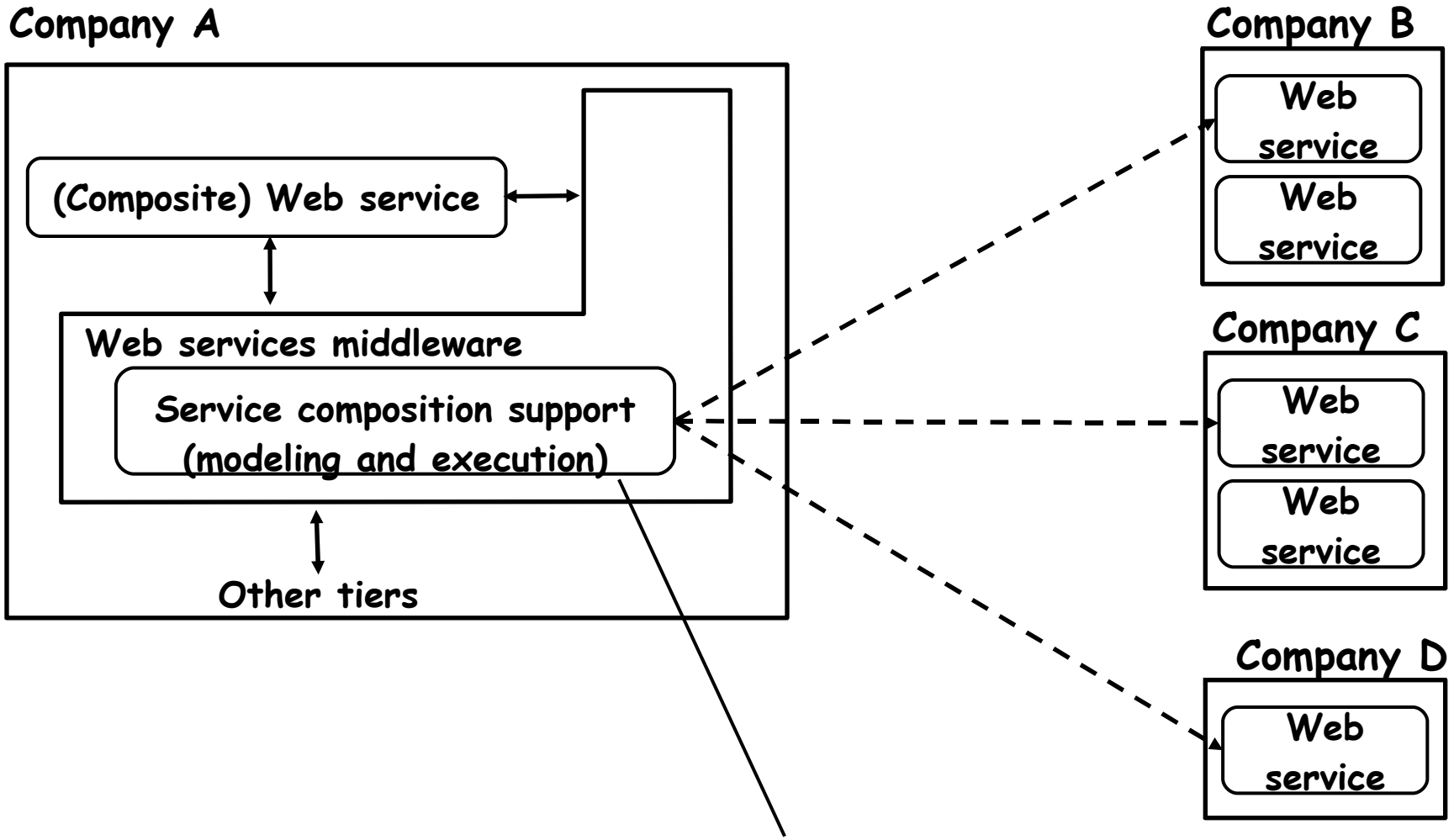
Code Approach



The internal application implements the composition logic, by invoking Web services as needed.
No support is provided by the Web services middleware in this case

(Alonso & Pautasso, 2004)

Composition Middleware



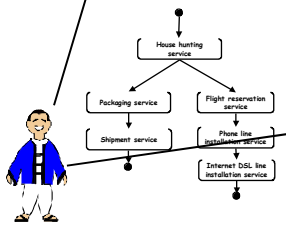
The composite service is directly implemented at the middleware level, by the composition engine.

(Alonso & Pautasso, 2004)

Composition Middleware

service composition model and language (usually characterized by a graphical and a textual representation)

the run-time environment executes the Web service business logic by invoking other services (through SOAP and HTTP modules)



schema designer

Web service composition middleware

development environment

run-time environment (composition engine)

schema definitions

composite service execution data

other Web services middleware (e.g., SOAP engine and conversation controller)

a service provider

services offered by other providers

supplier

warehouse

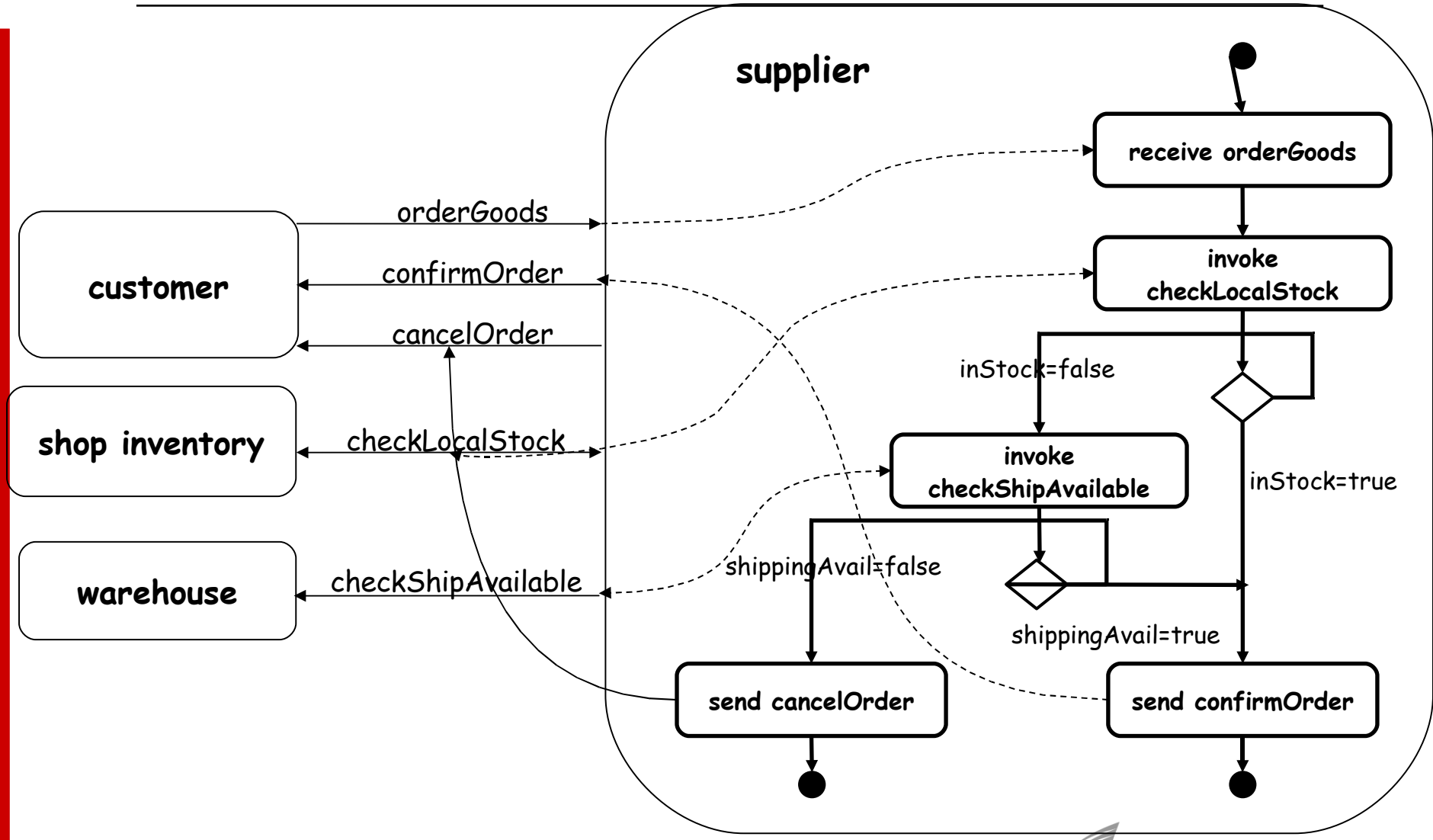
accounting

(Alonso & Pautasso, 2004)

Formal Models for Processes

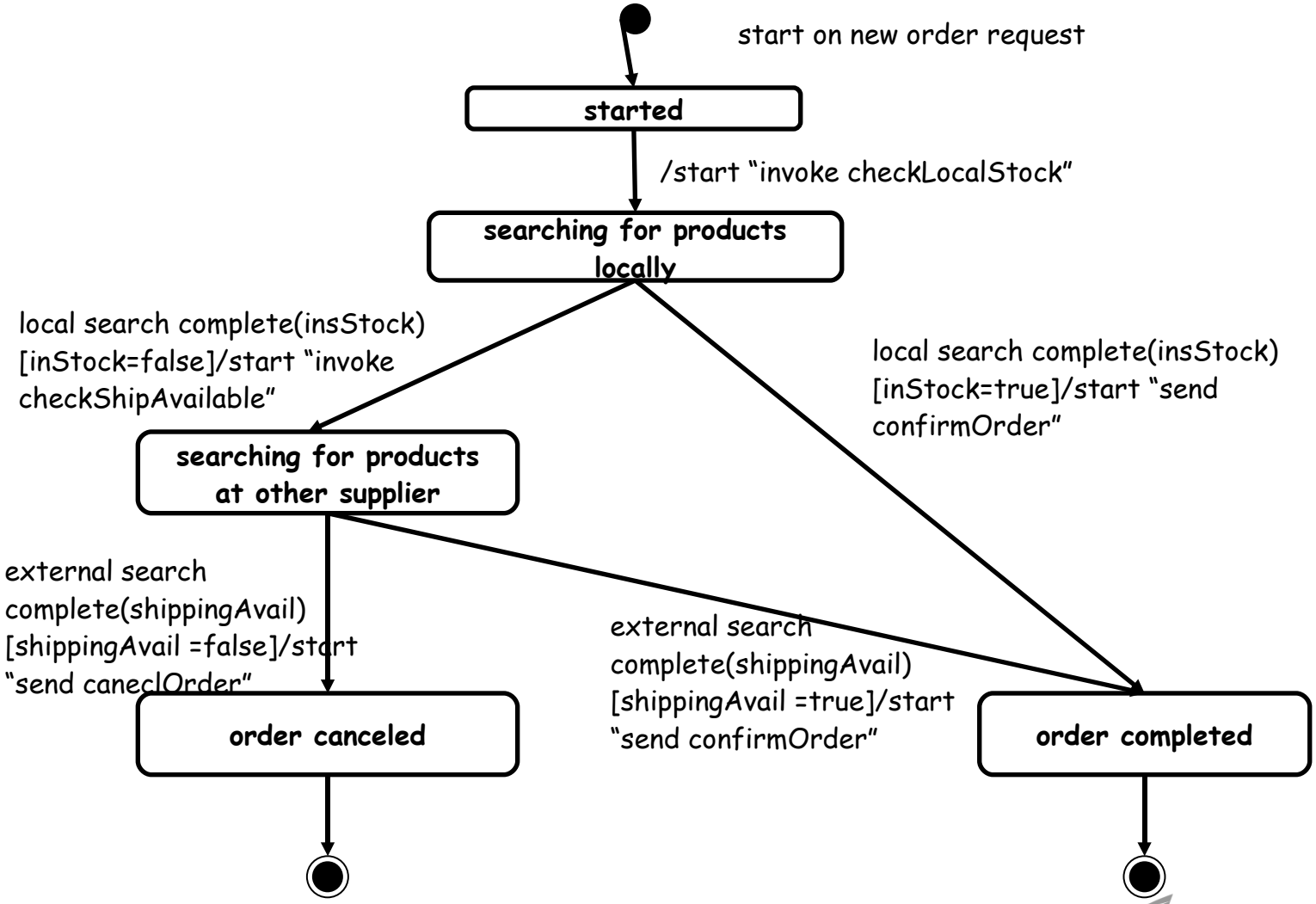
- Activity Diagrams, State Charts, Petri nets, Activity Hierarchies
- Rule based orchestration
  - Logical rules + event action pairs
- Situation Calculus
  - IOPEs (Semantic Web Services are often based on this paradigm)

Activity Diagram



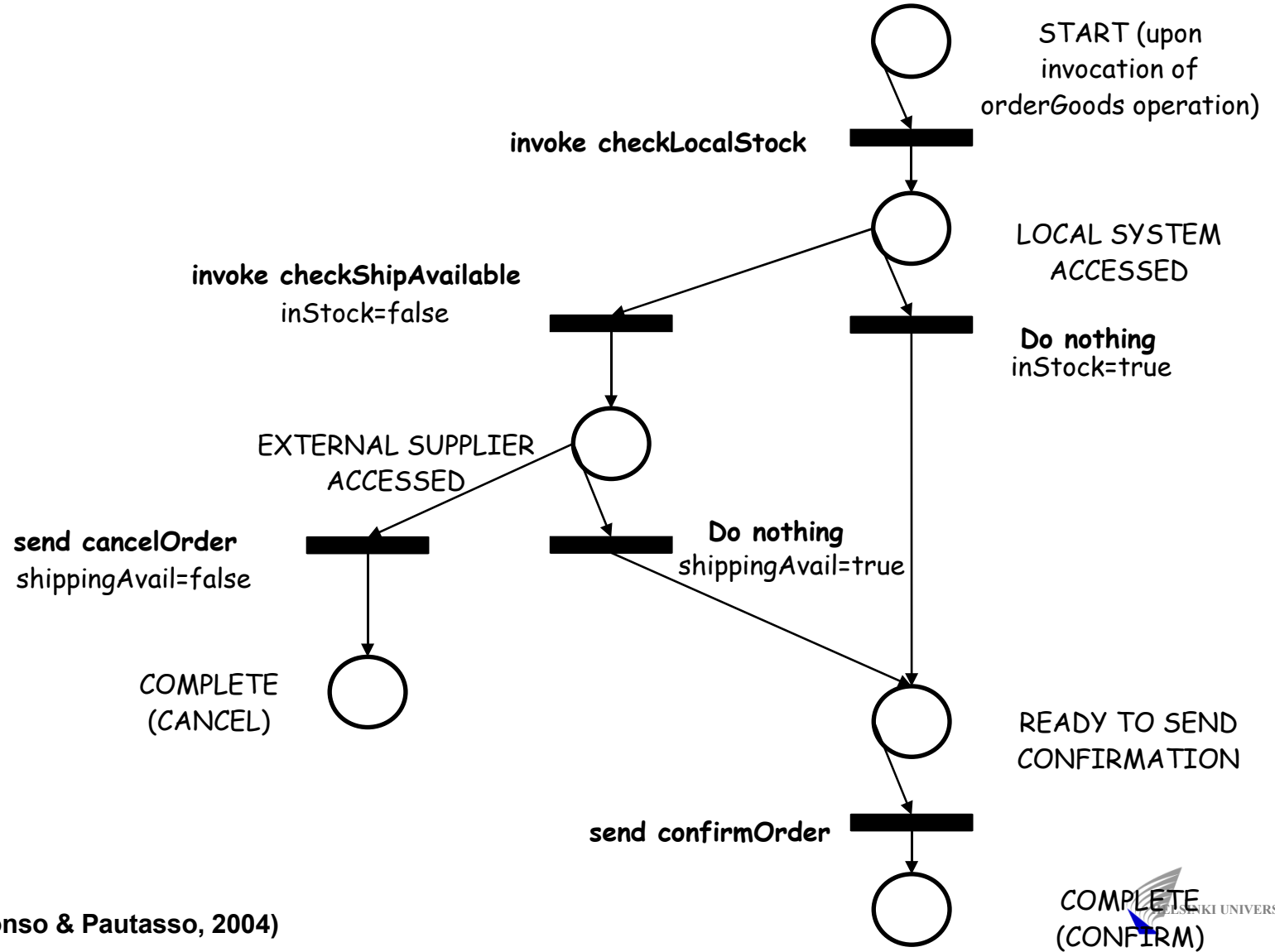
(Alonso & Pautasso, 2004)

Statechart



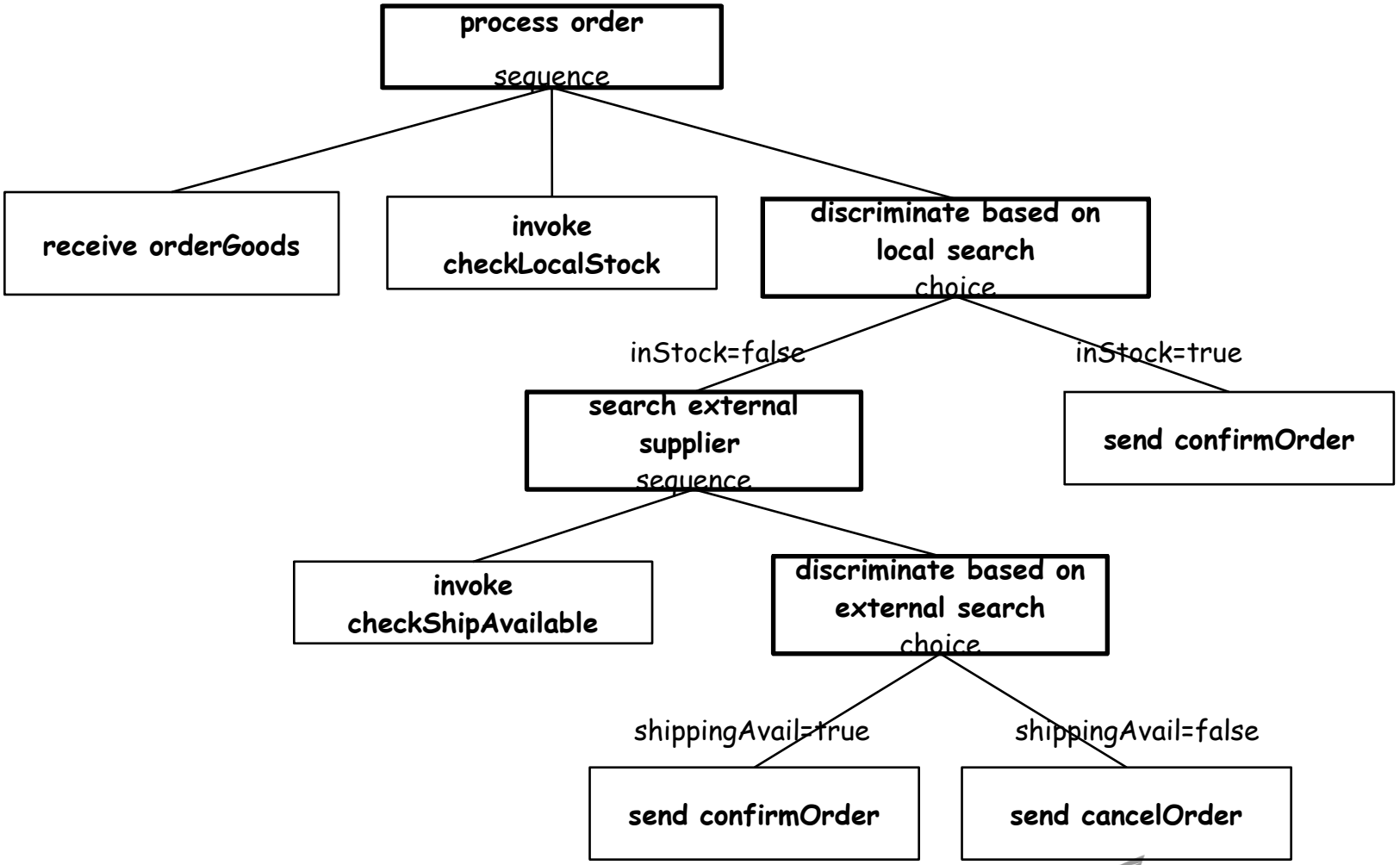
(Alonso & Pautasso, 2004)

Petri net



(Alonso & Pautasso, 2004)

Activity Hierarchy



(Alonso & Pautasso, 2004)

Rules

```
ON receive orderGoods
IF true
THEN invoke checkLocalStock;

ON complete(checkLocalStock)
IF (inStock==true)
THEN send confirmOrder;

ON complete(checkLocalStock)
IF (inStock==false)
THEN invoke checkShipAvailable;

ON complete(checkShipAvailable)
IF (shippingAvail ==true)
THEN send confirmOrder;

ON complete(checkShipAvailable)
IF (shippingAvail ==true)
THEN send cancelOrder;
```

(Alonso & Pautasso, 2004)

IOPEs

- Based on on the analyses of in which state the system is in
  - **I**nput
  - **O**utput
  - **P**recondition
  - **E**ffect
- More on IOPEs and situation calculus on Semantics in Web Services lecture

Back to the real world: WS-BPEL / BPEL4WS

- Business Process Execution Language
- WS-BPEL is the new 2.0 standard (minor changes to the current de-facto-standard)
- OASIS standard
- Originally developed by IBM and Microsoft
- Multiple implementations available from major vendors such as Oracle, IBM, BEA, Microsoft etc...

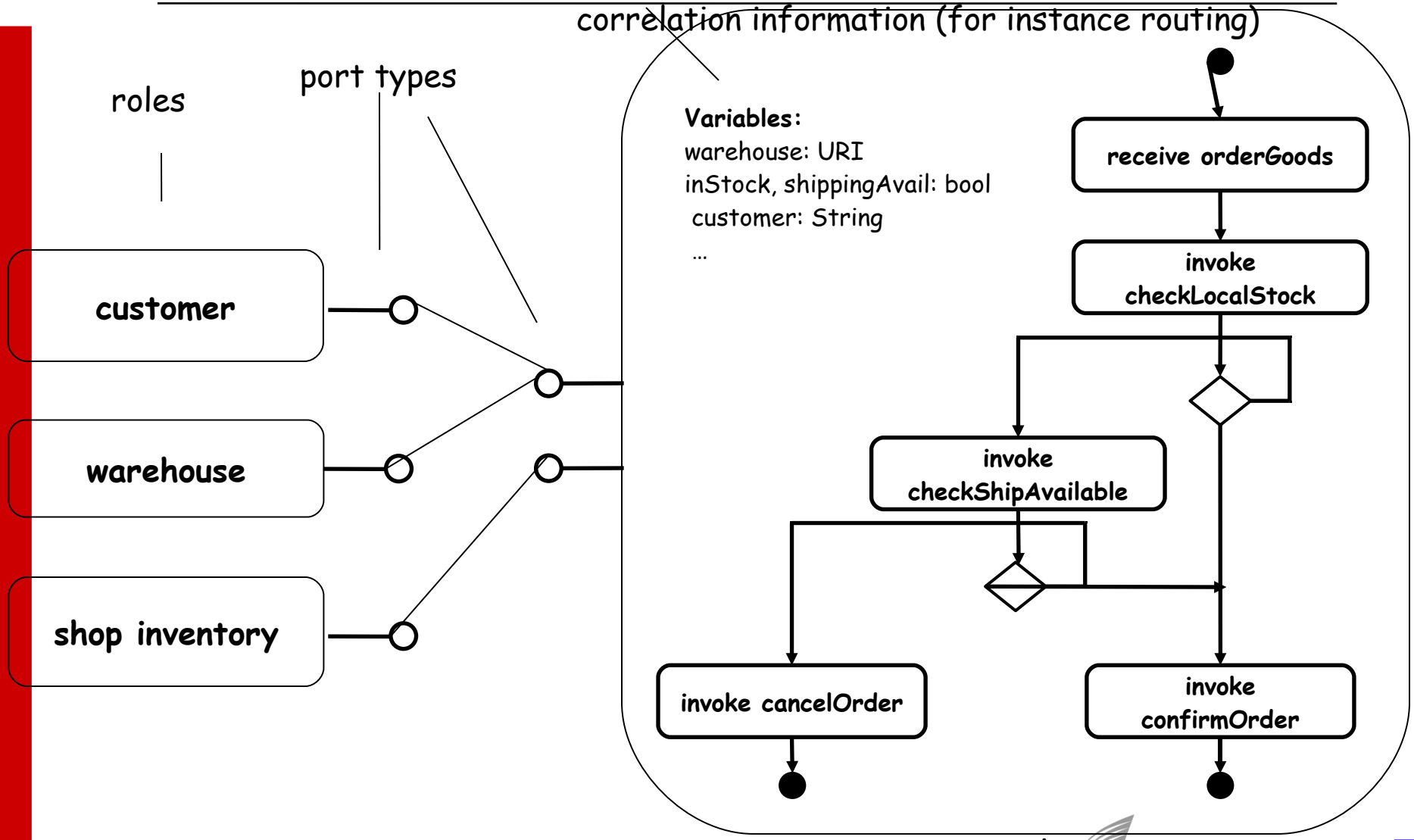
BPEL

- Enabling users to describe business process activities as Web services and define how they can be connected to accomplish specific tasks
- It does not directly deal with implementation of the language but only with the semantics of the primitives it provides. The emphasis is on interoperability between systems rather than portability of specifications
- Used to define:
  - Abstract processes: conversations and protocols for how to use a given service or between different services
  - Executable processes: essentially workflows extended with Web service capabilities

BPEL

- Roles
  - That take part in the message exchange
- Port Types
  - Operations that must be supported
- The Orchestration
  - And other ascpects in process execution
- Correlation Information
  - How messages are routed to the correct composition instances

Abstract and/or executable process
orchestration,
variables and data transfers,
exception handling,
correlation information (for instance routing)



(Alonso & Pautasso, 2004)

Basic Elements of BPEL (Alonso & Pautasso, 2004)

PROCESS

- **PARTNERS: Web services taking part in the process**
- **VARIABLES: the data used by the process**
- **CORRELATION SETS: constructs used to deal with conversations**
- **FAULT HANDLERS: what to do in case of errors (exceptions)**
- **COMPENSATION HANDLERS: what needs to be done to undo an activity**
- **EVENT HANDLERS: what to do when an event arrives**
- **ACTIVITIES: what the process does**

Equivalent to declarations in a normal programming language. It defines the way services are to be called, which data is to be used and which data is to be treated as stateful

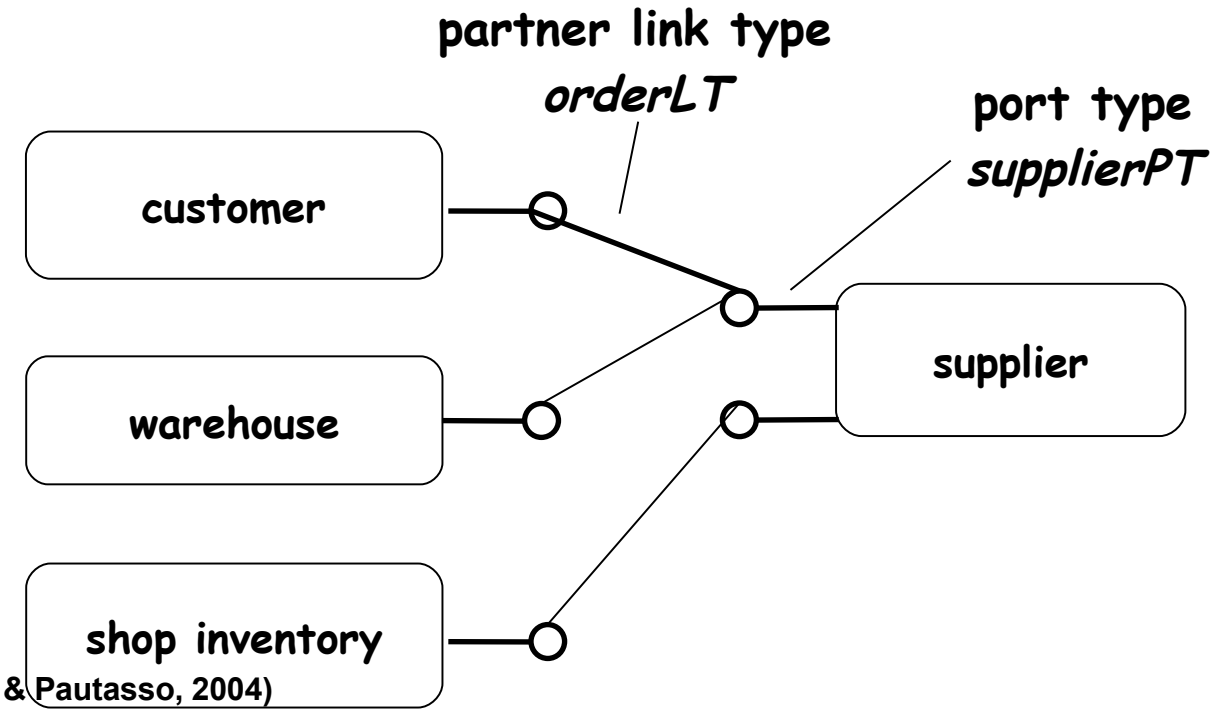
These elements establish what the process does, how it reacts under different circumstances (errors, message arrivals, events, etc.), and how data moves from one step to the next

Partners (Alonso & Pautasso, 2004)

- 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 (see WSDL), one for each of the roles in the partner entry (i.e., one portType is the portType of the process itself, the other one is the portType of the service being invoked).
  - 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

A partner link definition further qualifies the interactions occurring through a partner link type. Its definition refers to a partner link type and specifies the role played by the composite service as well as the one played by the other partner

```
<partnerLink name="customerP"
    partnerLinkType="orderLT"
    myRole="supplier"
    partnerRole="customer">
</partner>
```



(Alonso & Pautasso, 2004)

Variables (Alonso & Pautasso, 2004)

- Variables are used in BPEL to hold data used within the process
- Variables typically contain two basic forms of data:
  - Entire messages (defined in the WSDL description of a service)
  - Process specific data (counters, state variables, etc.)
- Variables are defined in the BPEL description of the process without specifying their type (e.g., what message they correspond to). Like partner link types, the idea is that these definitions are to be found in separate WSDL documents or in the WSDL descriptions of the services to be invoked by the process.

Variables (Alonso & Pautasso, 2004)

- Variables can be:
  - A message (their type is a WSDL message, which can be found in the WSDL description of the service using that message)
  - An XML type (f.i. integer; typically used for internal operations within the process)
  - An XML element (used to refer to complex XML types)

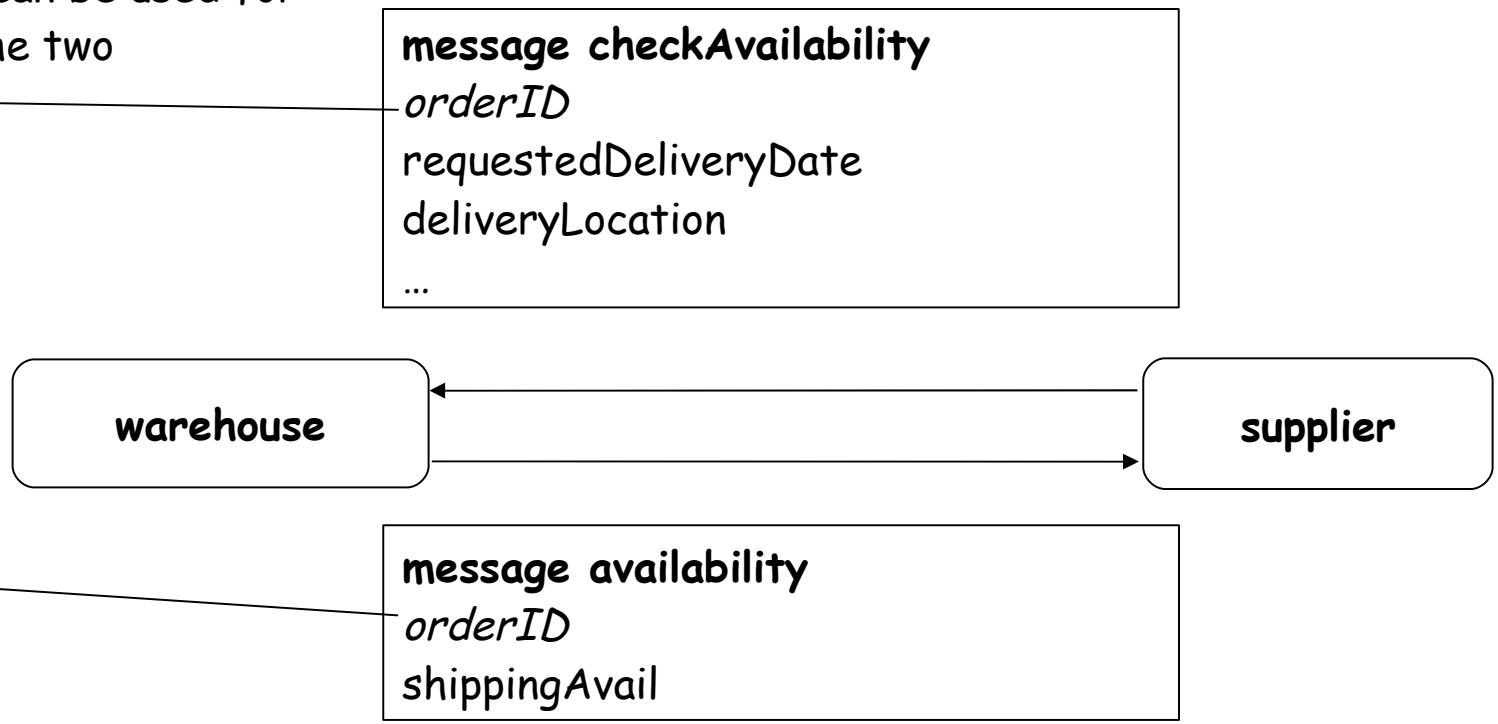
Correlation Sets

- Intended to help in mapping an abstract specification to running instances of that specification
- The problem: an abstract process describes what to do in general, while each running instance of the process must work only on its own data (f.i. on the messages that correspond to a particular purchase order). The correlation problem is how to specify in BPEL the way each running instance can identify the messages it has to process according to that abstract description of the process.
- Example: assign a name to a part of a order message: OrderID. This name is then used to "correlate" the process

Correlation Sets

- BPEL assumes the process is responsible for its own state
- Most middleware solutions (rather than direct compilation to Java, as most current implementations do) would not use correlation sets in the way BPEL describes them
- The problem can be solved in a much easier manner than through correlation sets (f.i. using a case identification number generated at the start of the process)

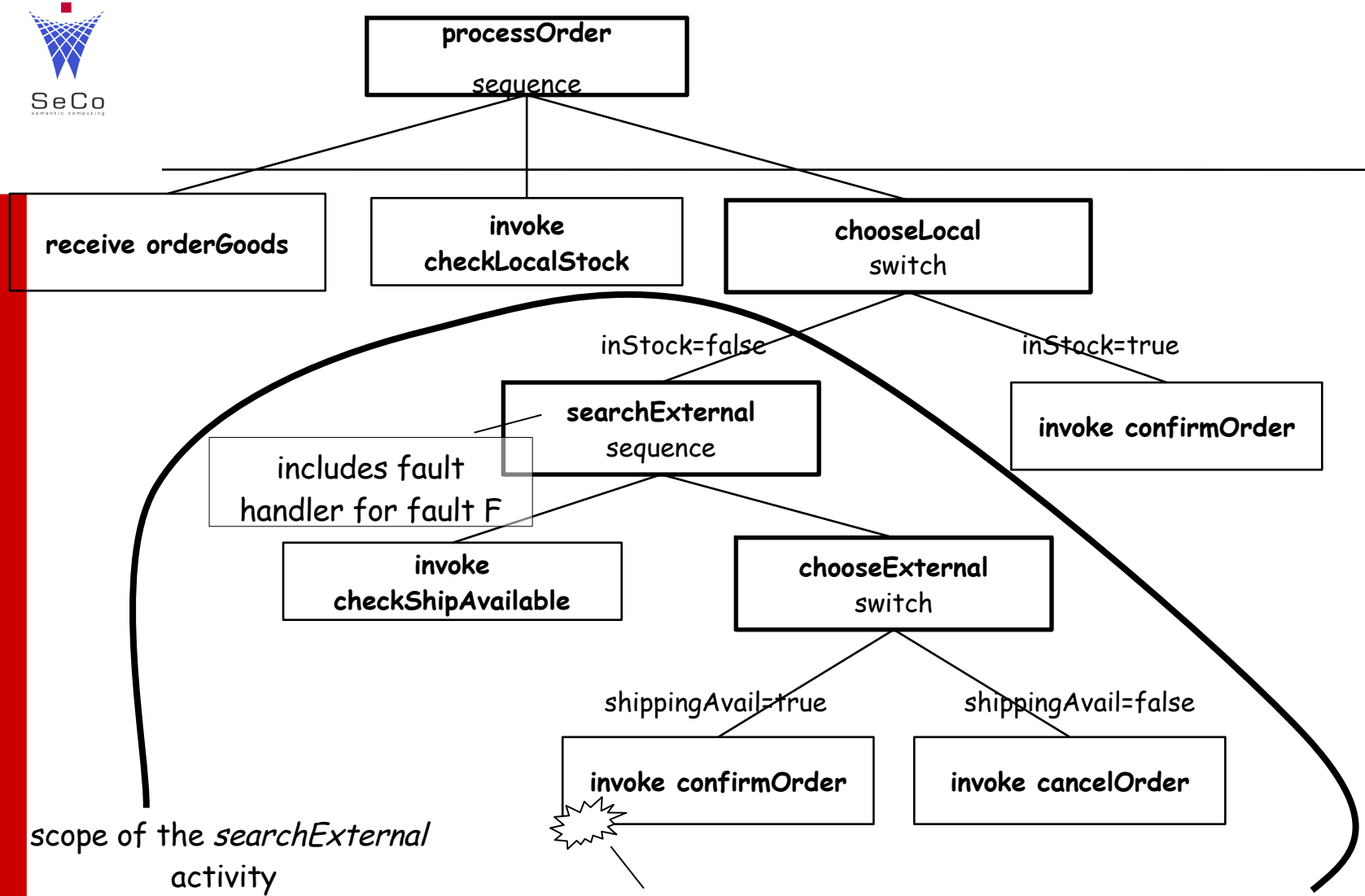
the orderID can be used for correlating the two messages



(Alonso & Pautasso, 2004)

Scope (Alonso & Pautasso, 2004)

- BPEL has three types of handlers:
  - Fault handlers (executed when an exception is thrown)
  - Compensation handlers (that are executed to undo the effects of operations)
  - Event handlers (executed when a particular message arrives or an alarm is raised)
- All handlers must be associated with a scope (similar to a code block in programming languages):
  - If an exception is raised within a given scope, then the corresponding fault handler for that exception is called (identical to try-catch statements in Java)
  - If the effects of an scope need to be undone, then the corresponding compensation handler is called. If there are nested scopes, then the compensation handlers of the low level scopes are called in reverse order of execution
  - Event handlers are active for as long as the control flow remains within the corresponding scope. Event handlers are executed either when a message arrives or a given alarm condition (f.i. a timer) is raised



(Alonso & Pautasso, 2004)

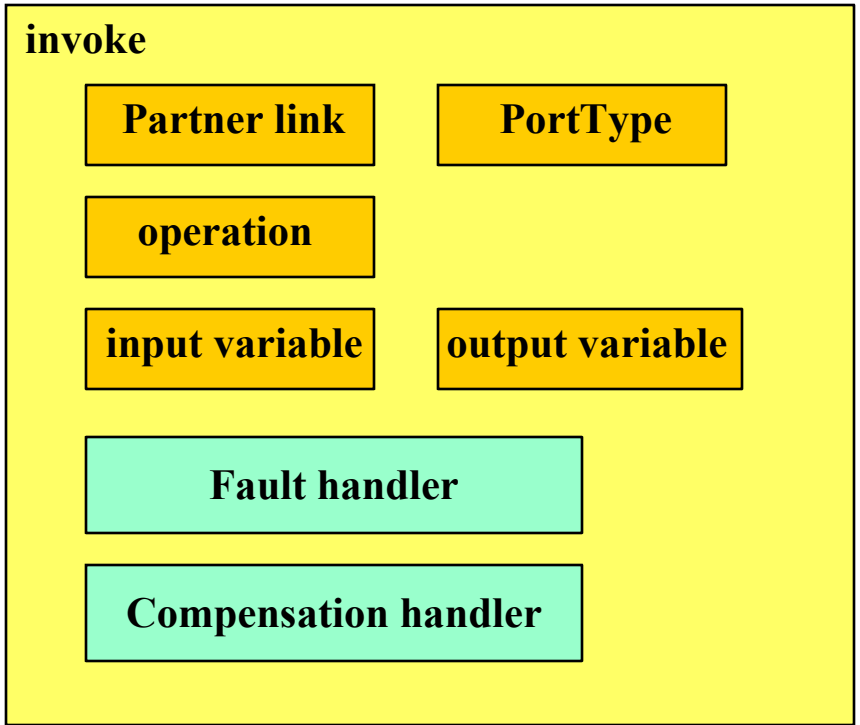
Activities (Alonso & Pautasso, 2004)

- 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 operation on a remote service
• <assign>	updates the value of variables
• <throw>	raises a fault for a fault handler to catch
• <terminate>	finishes the process
• <wait>	suspends execution for a given time period
• <empty>	no-op used for synchronization purposes
• <scope>	defines a block of activities
• <sequence>	executes a set of activities one after another
• <flow>	executes in parallel a set of activities
• <while>	repeats an activity depending on certain conditions
• <switch>	chooses between a set of activities
• <pick>	waits for a message or an alarm
• <compensate>	defines the activities of a compensation block

WSDL in BPEL (Alonso & Pautasso, 2004)

- A call to an operation in WSDL can be mapped to BPEL as follows:
  - Use invoke to call the operation
  - An input variable with the request (the input message of the WSDL operation)
  - An output variable for the response (the output message of the WSDL operation)
  - A WSDL fault can be handled by using a fault handler attached to the invoke activity



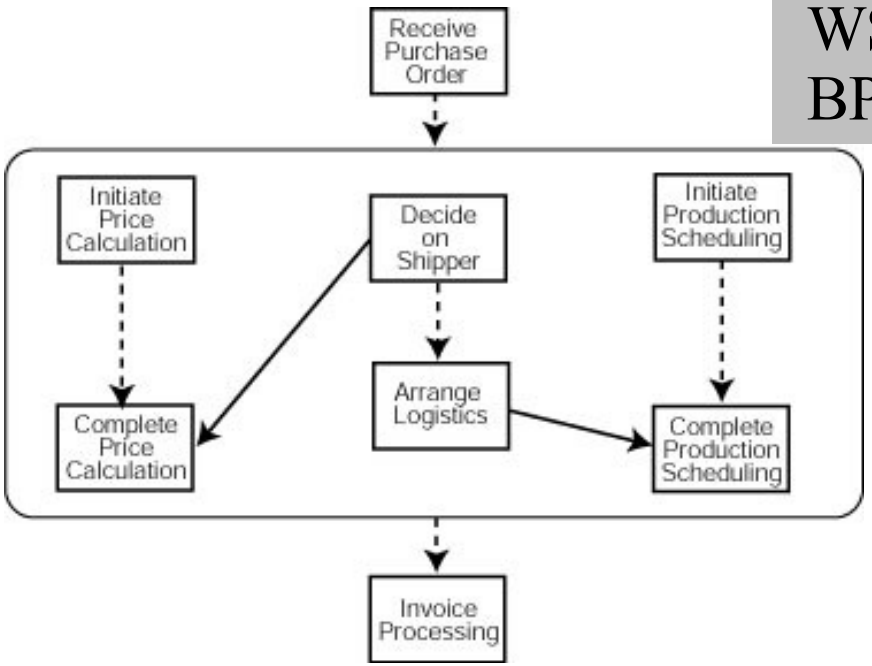
WSDL in BPEL

- A WSDL operation in BPEL
  - Use receive to wait for the input message
  - Use reply to send the output message or a fault message (as applicable)

Example BPEL

[htttp://www-106.ibm.com/developerworks/webservices/library/ws-bpel/]

Let consider the following process.



Two Files are required
- WSDL file
- BPEL file

Example BPEL

```
<definitions targetNamespace="http://manufacturing.org/wsdl/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 BPEL

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

   <containers>
      <container name="PO" messageType="lns:POMessage"/>
      <container name="Invoice"
                 messageType="lns:InvMessage"/>
   …
   </containers>

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

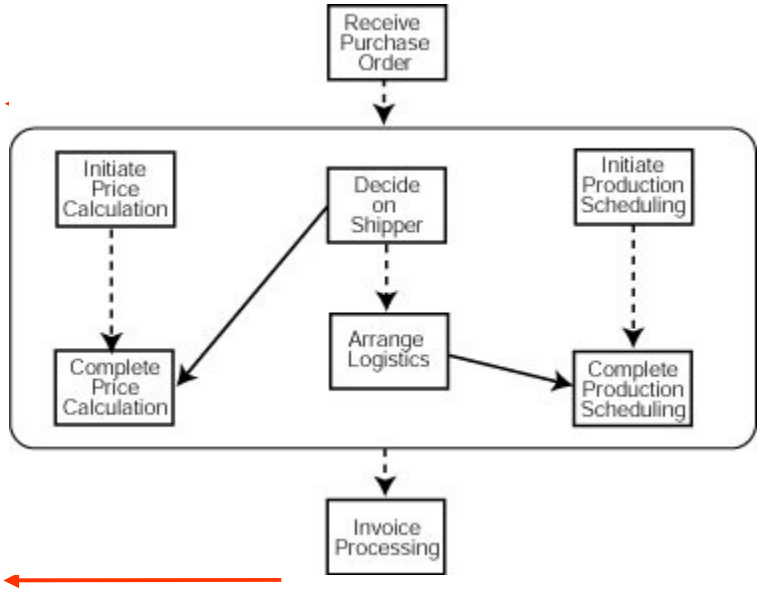
This section defines the different parties that interact with the business process in the course of processing the order.

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 BPEL

```
…
    <sequence>

      <receive partner="customer"

    portType="lns:purchaseOrderPT"

    operation="sendPurchaseOrder"
              container="PO">
      </receive>

      <flow>
       …
      </flow>

      <reply partner="customer"

    portType="lns:purchaseOrderPT"
             operation="sendPurchaseOrder"
             container="Invoice"/>
   </sequence>

</process>
```



SeCo
semantic computing

Questions?

HELSINKI UNIVERSITY OF TECHNOLOGY