Web Services and Business Processes: An Overview

Lars-Åke Fredlund

Web Communication: Evolution

- Classical World Wide Web provided:
  Computer (web page) – **Human** communciation

Web Communication: Evolution

- Classical World Wide Web provided:
  Computer (web page) – **Human** communciation
- But soon people started wanting to use the very successful infrastructure (XML, HTTP) for program–to–program communication and so the Web Services idea was born:
  Web Service – Client Program communication

Web Communication: Evolution

- Classical World Wide Web provided:
  Computer (web page) – **Human** communciation

- But soon people started wanting to use the very successful infrastructure (XML, HTTP) for program–to–program communication and so the Web Services idea was born:
  Web Service – Client Program communication

- Nowadays focus is on:
  Web Service – Web Service communication
  (*business processes*)

The Web Services Vision

- Support distributed applications composed of independent processes which communicate by message passing
- Targets loosely coupled systems
- Much like the Erlang vision
- Except independent of source language:
  mappable to Java, C#, C++, . . .
- Enables communication between web services implemented in different languages, and by different companies, and on different platforms

Web Service Framework Development

- Development of the Web Services framework has been layer-by-layer and rather ad-hoc
- Web Servers $\rightarrow$ Standard Data Format (XML) $\rightarrow$ Message Passing $\rightarrow$ Web Services $\rightarrow$ Business Processes $\rightarrow \ldots$
- As a result there is a huge pile of stacked “standards”: XML, XML Schema, SOAP, WSDL, UDDI, ...
- As the web of services is built bottom-up, lacking a single architect or design team, and evolves rapidly, we get complex solutions
- Interested parties are many – there is a lot of money in web services, and lots of hype!
- Strong players are Microsoft, Google, IBM, Oracle, SAP, Sun, BEA, and open source enterprises such as Apache

This Lecture

- My goal with this lecture is to understand what these standard provide in terms of a component infrastructure and middleware platform for distributed applications (keep in mind comparison with Erlang)
- To investigate Business Process Modelling (BPM) techniques which promises a more ambitious Web Service Framework
- Intriguingly people have been talking formal methods (the π-calculus, petri nets) in connection with Business Processes – we shall look for such a link

A Service Oriented Architecture

The ultimate aim of the web services programme is to build a **service-oriented architecture** (**SOA**) with the properties:

- Access to services is standardised (interfaces defined)
- Network nodes make (reusable) services available to other nodes, independent of physical location (location transparency)
- The publishing of information about available services is standardised (a service directory)
- SOA should be independent of implementation technology – e.g. services can interoperate regardless of implementation environment or language (Java, C#, . . .)

Web Services: A Concrete Architecture

A typical web service is implemented and deployed using a representative stack of protocols and standards which we shall examine in turn:

Language embeddings: **Java**, **C#**, . . .

Web Service Composition: **WS-BPEL**, **WS-CDL**

Distributed Middleware: **WS-Transaction, WS-Security, . . .**

Description: **WSDL** | Advertisement: **UDDI**

Messaging: **SOAP**

Transport: **HTTP** | Data Format: **XML XML Schema**

Web Server: **Apache, . . .**

Transport: HTTP

- Asymmetric protocol: has a client and server side
- A synchronous protocol: one request $\longrightarrow$ one reply
- A stateless protocol: no history of communication between client and server available to server, every request is understandable on its own
- But the statelessness of operations is often too expensive – in practise mechanisms like cookies are used:

Transport: HTTP

- Asymmetric protocol: has a client and server side
- A synchronous protocol: one request $\longrightarrow$ one reply
- A stateless protocol: no history of communication between client and server available to server, every request is understandable on its own
- But the statelessness of operations is often too expensive – in practise mechanisms like cookies are used:

client $\xrightarrow{request1}$ server

Transport: HTTP

- Asymmetric protocol: has a client and server side
- A synchronous protocol: one request $\longrightarrow$ one reply
- A stateless protocol: no history of communication between client and server available to server, every request is understandable on its own
- But the statelessness of operations is often too expensive – in practise mechanisms like cookies are used:

  client $\xrightarrow{request1}$ server

  server $\xrightarrow{response1+cookie(name=value)}$ client

Transport: HTTP

- Asymmetric protocol: has a client and server side
- A synchronous protocol: one request $\longrightarrow$ one reply
- A stateless protocol: no history of communication between client and server available to server, every request is understandable on its own
- But the statelessness of operations is often too expensive – in practise mechanisms like cookies are used:

$$\text{client} \xrightarrow{request1} \text{server}$$

$$\text{server} \xrightarrow{response1+cookie(name=value)} \text{client}$$

$$\text{client} \xrightarrow{request2+cookie(name=value)} \text{server}$$

Transport: HTTP

- Asymmetric protocol: has a client and server side
- A synchronous protocol: one request $\longrightarrow$ one reply
- A stateless protocol: no history of communication between client and server available to server, every request is understandable on its own
- But the statelessness of operations is often too expensive – in practise mechanisms like cookies are used:

client	$\xrightarrow{request1}$	server
server	$\xrightarrow{response1+cookie(name=value)}$	client
client	$\xrightarrow{request2+cookie(name=value)}$	server

- Universal addressing of resources (URIs)
- HTTPS – for encryption using SSL/TLS

REST – Representational State Transfer

A software architecture style for hypermedia – the core of HTTP

- A stateless client/server protocol for resource access
- Defines a few basic operations that involve state transfers:
  - GET (resource) – from server
  - PUT (resource) – to server
  - POST (resource) – submit new resource to a server
  - DELETE (resource)
- GET, PUT should be **idempotent**
  that is multiple sequential requests should yield same answer
- A universal syntax for resource identification
- Allows for **easy caching** (no strange session dependencies)

Data Formats: XML

- XML $\equiv$ Extensible Markup Language: a general purpose markup language
- Human readable as well as machine readable format
- Data is described verbosely, using text, in a tree hierarchy
- Basic elements are: characters, containers (elements), and attributes (name-value pairs) on containers
- Example:

```
<country>
  <name castellano="francia">france</name>
  <population>59.7</population>
</country>
```

XML Advantages

- Text format makes for readability, understanding and easier debugging of services on top of XML
- Easy to define new formats on top of XML
- Makes for extensible documents: a tool doesn't need to know everything about a format to extract information useful to itself

XML Drawbacks

- Data is described hierarchically rather that relationally. For example: what is the hierarchy between actors and movies?
- Can be complex to parse and unparse
- XML is inefficient for many uses: makes for slow applications and communication
- Binary data is stored using Base64 encoding, which increases the size 1.33 times (problems for transmission of movies, audio data, . . .)
- And so attempts to improve exists: JSON (JavaScript Object Notation), YAML, Binary XML, compressed and binary XML (Fast Infoset, BiM – Binary MPEG format for XML)

XML Typing

- XML Schema
  - Also know as XSD (XML Schema Definition)
  - Constrains XML documents – *types for XML*
  - Defines allowable combinations of elements
  - Characterises data types
  - Basic types: decimal, float, string, base64Binary, list, union, restriction. . .
  - Complex types: defines allowed elements and attributes
- Alternative: Relax NG – a more compact format

XML and XML Schema Example

- Schema definition (country.xsd):

```
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="country" type="Country"/>
 <xs:complexType name="Country">
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="population" type="xs:decimal"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

- Schema instance:

```
<country
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
 xsi:noNamespaceSchemaLocation="country.xsd">
  <name>France</name>
  <population>59.7</population>
</country>
```

Messaging: SOAP

- Embeds XML into messages
- Supports remote procedure calls (RPC)
- Messages have a header and a body in an envelope
- Fault information (for RPC:s) can be communicated

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soa
   <soap:Body>
     <getProductDetails
          xmlns="http://warehouse.example.com/ws">
       <productID>827635</productID>
     </getProductDetails>
   </soap:Body>
 </soap:Envelope>
```

- As can be seen, a lot of text overhead. . . – but understandable!
- Normally uses HTTP for transport

WSDL – Web Service Description Language

- An XML format for describing how to communicate with web services
  - Describes the *public abstract interface* to the service, i.e., *which* operations the interface provides
  - Describes a *binding* – *how* to exchange messages with a server implementing the service interface (e.g., using SOAP and HTTP)
  - Describes *where* the service is available
    That is, at which URL address the service is located
- Normally used with SOAP (messages) and XML Schema (defining data structures)

WSDL – Web Service Description Language

- The abstract service interface of a web service description provides operations (interactions between client and service)
- These operations are composed of messages (either input or output), or faults, whose format is typically defined in XML
- An operation can be
  - `request--response`,
  - `input only`,
  - `output only`,
  - `robust-in-only` (in case of an error a reply is delivered to the client),
  - ...

WSDL 2.0 example – room reservation

```
<interface  name = "reservationInterface" >

  <fault name = "invalidDataFault"
         element = "ghns:invalidDataError"/>

  <operation name="opCheckAvailability"
     pattern="http://www.w3.org/2006/01/wsdl/in-out"
     style="http://www.w3.org/2006/01/wsdl/style/iri"
     wsdlx:safe = "true">

     <input messageLabel="In"
            element="ghns:checkAvailability" />
     <output messageLabel="Out"
             element="ghns:checkAvailabilityResponse" />
     <outfault ref="tns:invalidDataFault"
               messageLabel="Out"/>
  </operation>
</interface>
```

WSDL 2.0 example – data definitions

```
<types>
  <xs:schema ..>
      <xs:element name="checkAvailability"
                  type="tCheckAvailability"/>

      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate"
                       type="xs:date"/>
          <xs:element  name="checkOutDate"
                       type="xs:date"/>
          <xs:element  name="roomType"
                       type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>

    ...
</types>
```

Binding to protocols

```
<binding name="reservationSOAPBinding"
         interface="tns:reservationInterface"
         type="http://www.w3.org/2006/01/wsdl/soap"
         wsoap:protocol="http://www.w3.org/2003/05/soap/b

  <fault ref="tns:invalidDataFault"
         wsoap:code="soap:Sender"/>

  <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-
</binding>
```

Address information

```
<service name="reservationService"
         interface="tns:reservationInterface">

 <endpoint name="reservationEndpoint"
    binding="tns:reservationSOAPBinding"
    address="http://greath.example.com/2004/reservation"/
</service>
```

Service Publishing & Discovery

Remember the SOA (services-oriented architecture:

- We need a method to publish information about available services (a service directory)
- And methods to search the directory for “suitable” web services
- For web services the **UDDI** directory service can be used

Web Service Publish & Discovery: UDDI

- UDDI (Universal Description, Discovery and Integration) for publishing&discovery of web services
- UDDI was heavily hyped:
  1. UDDI would provide a universal *registry* for business to provide service listings (web service descriptions, etc)
  2. *web service discovery*: UDDI would spawn a service broker infrastructure where a client looking for a service searchers for a suitable provider in the UDDI registry
- Nowadays the basic aspects of UDDI is emphasised: publish and search for web services, and private (in-business) use
- An UDDI description has three parts:
  - details on the business providing the service
  - functional characterisation of service (WSDL)
  - technical details: access to service, transport mechanism

Status Check – Web Services

What have we got so far?

Status Check – Web Services

What have we got so far?

- A stack of standards for describing operations, data formats, and the locations at which the operations are available

Status Check – Web Services

What have we got so far?

- A stack of standards for describing operations, data formats, and the locations at which the operations are available
- And a set of bindings to concrete programming languages such as Java, C# for invoking and providing web services

Status Check – Web Services

What have we got so far?

- A stack of standards for describing operations, data formats, and the locations at which the operations are available
- And a set of bindings to concrete programming languages such as Java, C# for invoking and providing web services
- But this is not very different from earlier attempts like CORBA??
  - Except we have “cool” Web Servers that implement Web Services instead of “boring” CORBA Servers and applications
  - But the connection of WebServices and language (Java) is indirect leading to ugly, complex and slow solutions

Status Check: Positive

In truth: more interactivity and easier debugging through better data formats like XML

- Seeing your data instead of decoding it is very rewarding
- Possibly a key to the success of scripting languages too – easy construction/deconstruction of data leads to less separation of code and data!

Missing So Far

- Are Web Services (as we have seen them so far) suitable for implementing a (distributed) component framework?

Missing So Far

- Are Web Services (as we have seen them so far) suitable for implementing a (distributed) component framework?

- **No**: several features from other frameworks are missing:
  - To enable communication between Web Services we need to establish various basic rules regarding communications:
    - *communication guarantees*
    - *security mechanism*
    - *trust model*, . . .

Missing So Far

- Are Web Services (as we have seen them so far) suitable for implementing a (distributed) component framework?

- **No**: several features from other frameworks are missing:
  - To enable communication between Web Services we need to establish various basic rules regarding communications:
    - *communication guarantees*
    - *security mechanism*
    - *trust model*, . . .
  - Design-by-contract: where are the abstract contract descriptions of web services?
    (post– and pre–conditions, invariants)

Distributed Middleware Concerns

There are a large number of “add-on” specifications that provide support for web service design and interaction:

Distributed Middleware Concerns

There are a large number of “add-on” specifications that provide support for web service design and interaction:

- *WS-Addressing*: embedding address information in XML (including reply information)

Distributed Middleware Concerns

There are a large number of “add-on” specifications that provide support for web service design and interaction:

- *WS-Addressing*: embedding address information in XML (including reply information)
- *WS-ReliableMessaging*: provides reliable SOAP message delivery

Distributed Middleware Concerns

There are a large number of “add-on” specifications that provide support for web service design and interaction:

- *WS-Addressing*: embedding address information in XML (including reply information)
- *WS-ReliableMessaging*: provides reliable SOAP message delivery
- *WS-Transactions*: transaction support for web services

Distributed Middleware Concerns

There are a large number of “add-on” specifications that provide support for web service design and interaction:

- *WS-Addressing*: embedding address information in XML (including reply information)
- *WS-ReliableMessaging*: provides reliable SOAP message delivery
- *WS-Transactions*: transaction support for web services
- *WS-Security*: provides end-to-end security (message confidentiality, integrity)

Distributed Middleware Concerns

There are a large number of “add-on” specifications that provide support for web service design and interaction:

- *WS-Addressing*: embedding address information in XML (including reply information)
- *WS-ReliableMessaging*: provides reliable SOAP message delivery
- *WS-Transactions*: transaction support for web services
- *WS-Security*: provides end-to-end security (message confidentiality, integrity)
- *WS-Trust*: manages trust (credentials, who has the permission to do an operation)

Distributed Middleware Concerns

There are a large number of “add-on” specifications that provide support for web service design and interaction:

- *WS-Addressing*: embedding address information in XML (including reply information)
- *WS-ReliableMessaging*: provides reliable SOAP message delivery
- *WS-Transactions*: transaction support for web services
- *WS-Security*: provides end-to-end security (message confidentiality, integrity)
- *WS-Trust*: manages trust (credentials, who has the permission to do an operation)
- *WS-Policy*: permits web services to announce policies (Quality–of–service, security), and users to state requirements upon web services

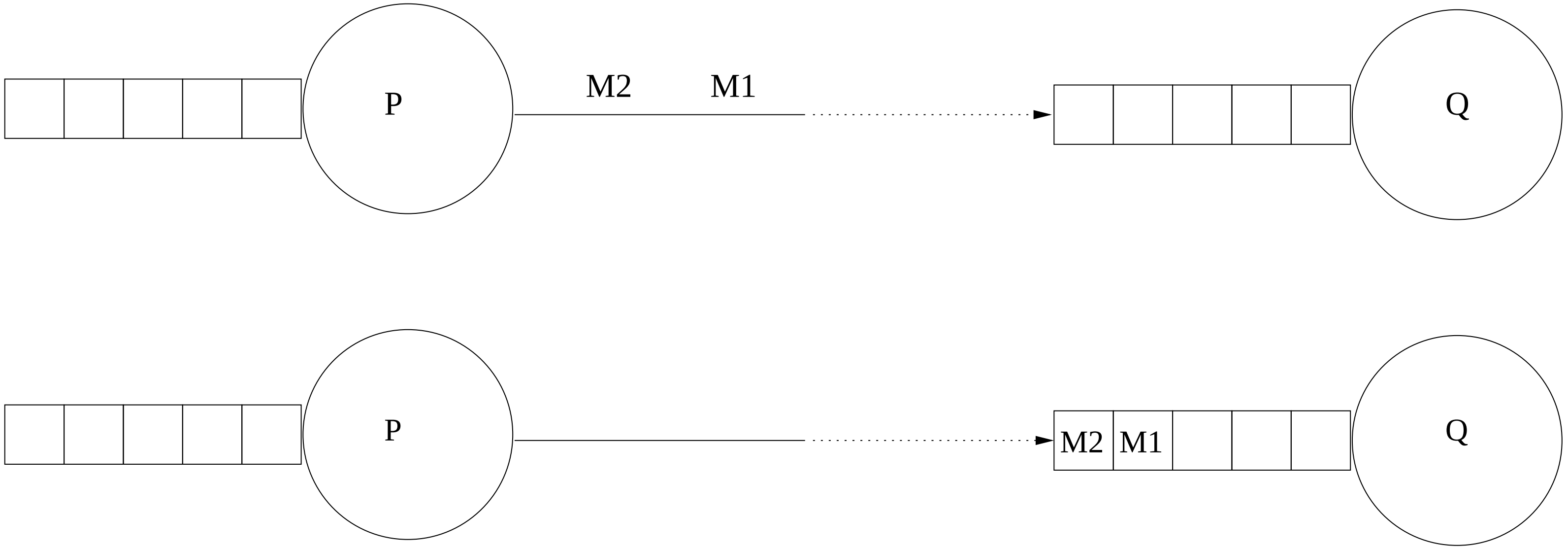
Guarantees for Message Passing

- Delivery assurances for message passing (promises to clients and servers)

Guarantees for Message Passing

- Delivery assurances for message passing (promises to clients and servers)
- As an example recall the Erlang message delivery assurances:

  Messages sent from any process P to any process Q is delivered in order (or P or Q crashes)



- Implies no duplication of messages, and no loss of messages in the middle of a communication

WS-ReliableMessaging

Provides delivery assurances for messages:

- **AtLeastOnce**: each message will be delivered at least once to the receiver (or an error returned to the sender)
- **AtMostOnce**: each message delivered at most once
- **ExactlyOnce**: each message delivered exactly once
- **InOrder**: messages delivered in order (combines with other assurances)

WS-ReliableMessaging

Provides delivery assurances for messages:

- **AtLeastOnce**: each message will be delivered at least once to the receiver (or an error returned to the sender)
- **AtMostOnce**: each message delivered at most once
- **ExactlyOnce**: each message delivered exactly once
- **InOrder**: messages delivered in order (combines with other assurances)

Erlang communication guarantees?

WS-ReliableMessaging

Provides delivery assurances for messages:

- **AtLeastOnce**: each message will be delivered at least once to the receiver (or an error returned to the sender)
- **AtMostOnce**: each message delivered at most once
- **ExactlyOnce**: each message delivered exactly once
- **InOrder**: messages delivered in order (combines with other assurances)

Erlang communication guarantees? **AtMostOnce**, **InOrder**

Transactions

Classical transaction guarantees: ACID (atomicity, consistency, isolation, durability)

- **atomicity**: a sequence of operations $op_1, op_2, op_3, \ldots$ either *all* occur or *none* occurs

Transactions

Classical transaction guarantees: ACID (atomicity, consistency, isolation, durability)

- **atomicity**: a sequence of operations $op_1, op_2, op_3, \ldots$ either *all* occur or *none* occurs
- **consistency**: the database is kept consistent by a transaction (if a transaction invalidates an integrity constraint it is *aborted*)
- **isolation**: a transaction runs in isolation from other transactions. That is, conceptually there are never two concurrent transactions executing
- **durability**: if the user is informed of a successful transaction then the transaction is persistent (survives system failures)

WS-Transaction

Web based transaction standards:

- **WS-AtomicTransaction** implements a classical two-phase atomic commit protocol for short transactions (intended for classical applications like managing access to data bases)
- **WS-BusinessActivity** permits transactions with a longer lifetime

WS-BussinessActivity

- A transaction may last a long time (in contrast to WS-AtomicTransaction one cannot exclusively reserve resources for the whole duration of the transaction)

WS-BussinessActivity

- A transaction may last a long time (in contrast to WS-AtomicTransaction one cannot exclusively reserve resources for the whole duration of the transaction)
- A business transaction may involve human actors

WS-BussinessActivity

- A transaction may last a long time (in contrast to WS-AtomicTransaction one cannot exclusively reserve resources for the whole duration of the transaction)
- A business transaction may involve human actors
- Participating in a “business transaction” may have a cost

WS-BussinessActivity

- A transaction may last a long time (in contrast to WS-AtomicTransaction one cannot exclusively reserve resources for the whole duration of the transaction)
- A business transaction may involve human actors
- Participating in a “business transaction” may have a cost
- If some partner (process) decides to back-out it may be nontrivial for other processes to also back-out of the transaction

WS-BussinessActivity

- A transaction may last a long time (in contrast to WS-AtomicTransaction one cannot exclusively reserve resources for the whole duration of the transaction)
- A business transaction may involve human actors
- Participating in a “business transaction” may have a cost
- If some partner (process) decides to back-out it may be nontrivial for other processes to also back-out of the transaction
- Instead of **aborting a transaction** (undoing only the actions of the transaction) the aim is to **compensating for the failed transaction** (reaching an acceptable state for all processes involved in the failed transaction)

WS-BusinessActivity Example

- A business transaction:

Operation 1:	Company A pays for delivery of goods via DHL to a company B
Operation 2:	DHL delivers goods to company B
Operation 3:	and receives payment afterwards from company B

WS-BusinessActivity Example

- A business transaction:

  Operation 1: Company A pays for delivery of goods via DHL to a company B

  Operation 2: DHL delivers goods to company B

  Operation 3: and receives payment afterwards from company B

- What happens if company B goes bankrupt during the transaction?

WS-BusinessActivity Example

- A business transaction:

  Operation 1: Company A pays for delivery of goods via DHL to a company B

  Operation 2: DHL delivers goods to company B

  Operation 3: and receives payment afterwards from company B

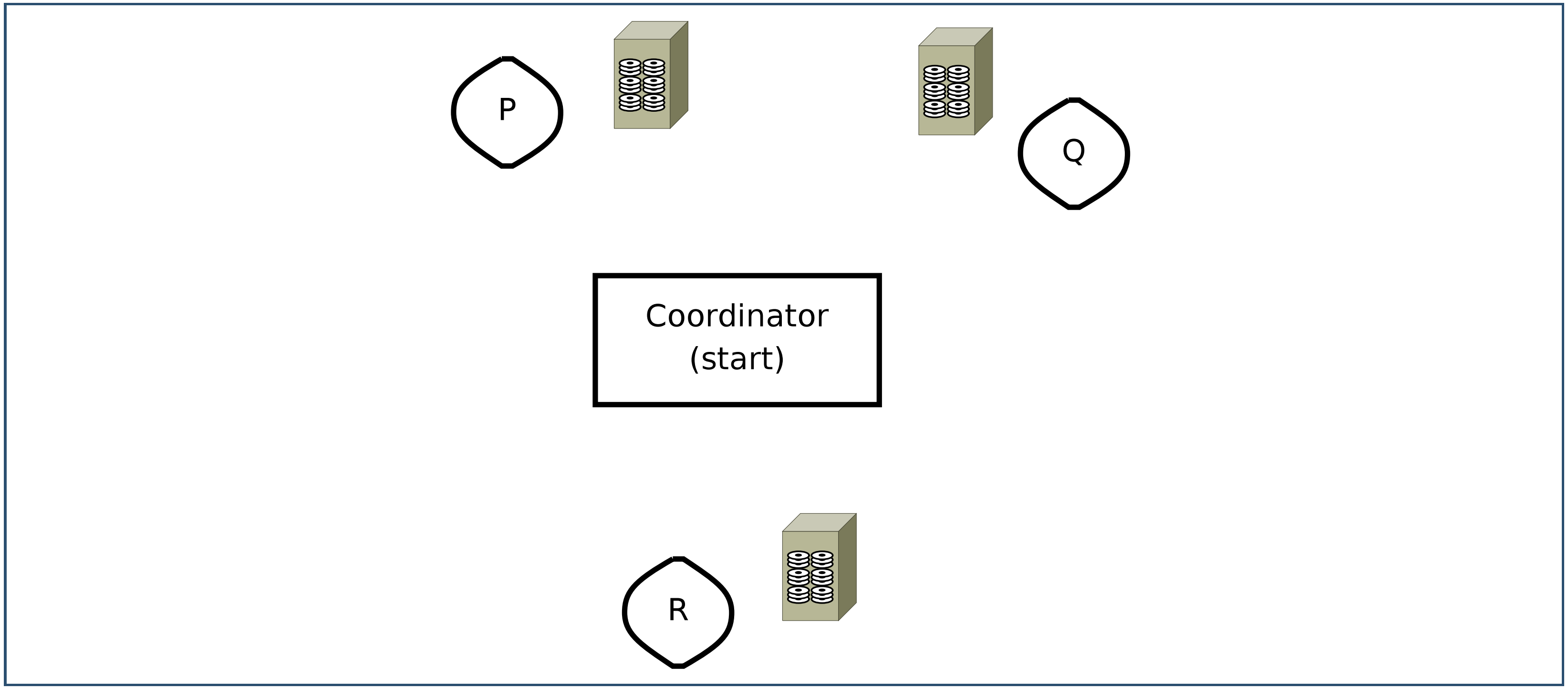
- What happens if company B goes bankrupt during the transaction?
  - Where should the goods be delivered by DHL?
  - Who should pay the price of shipping via DHL?

WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: init (success example)

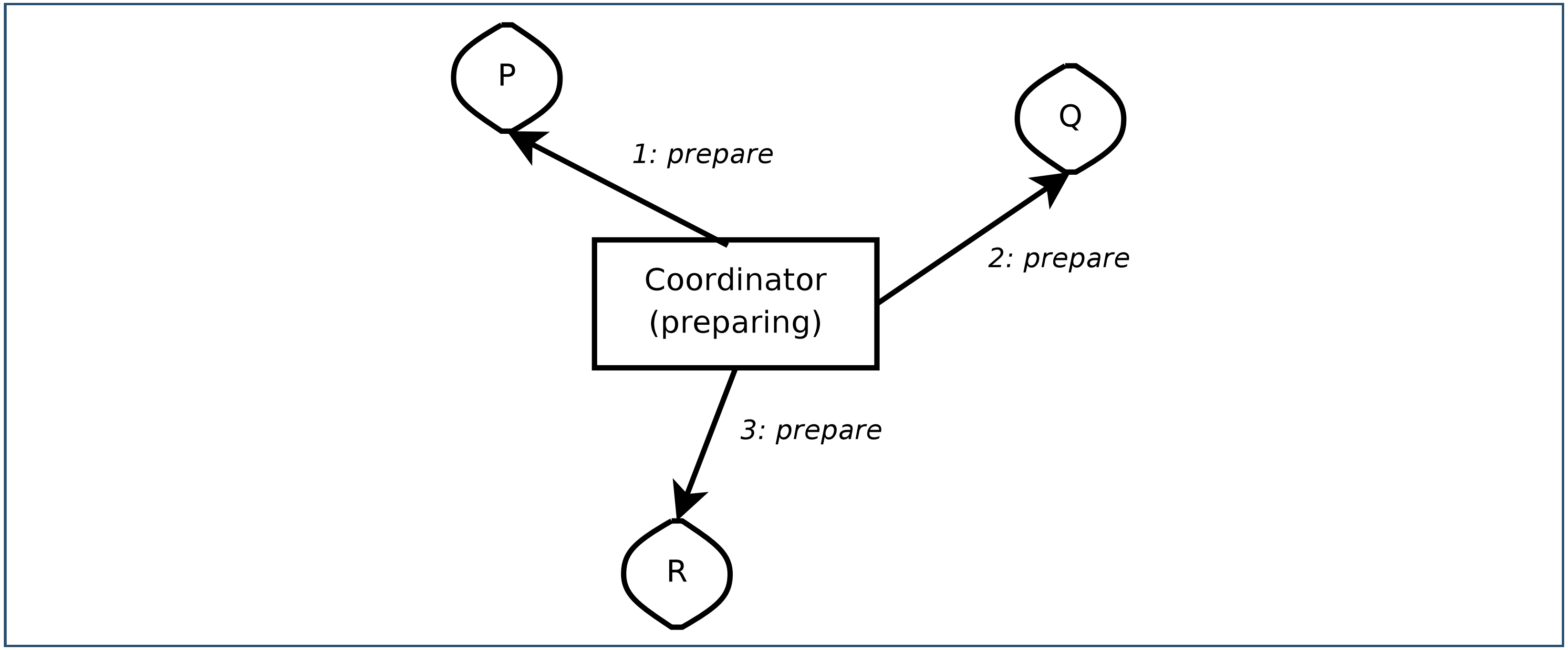


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: asking participants to prepare

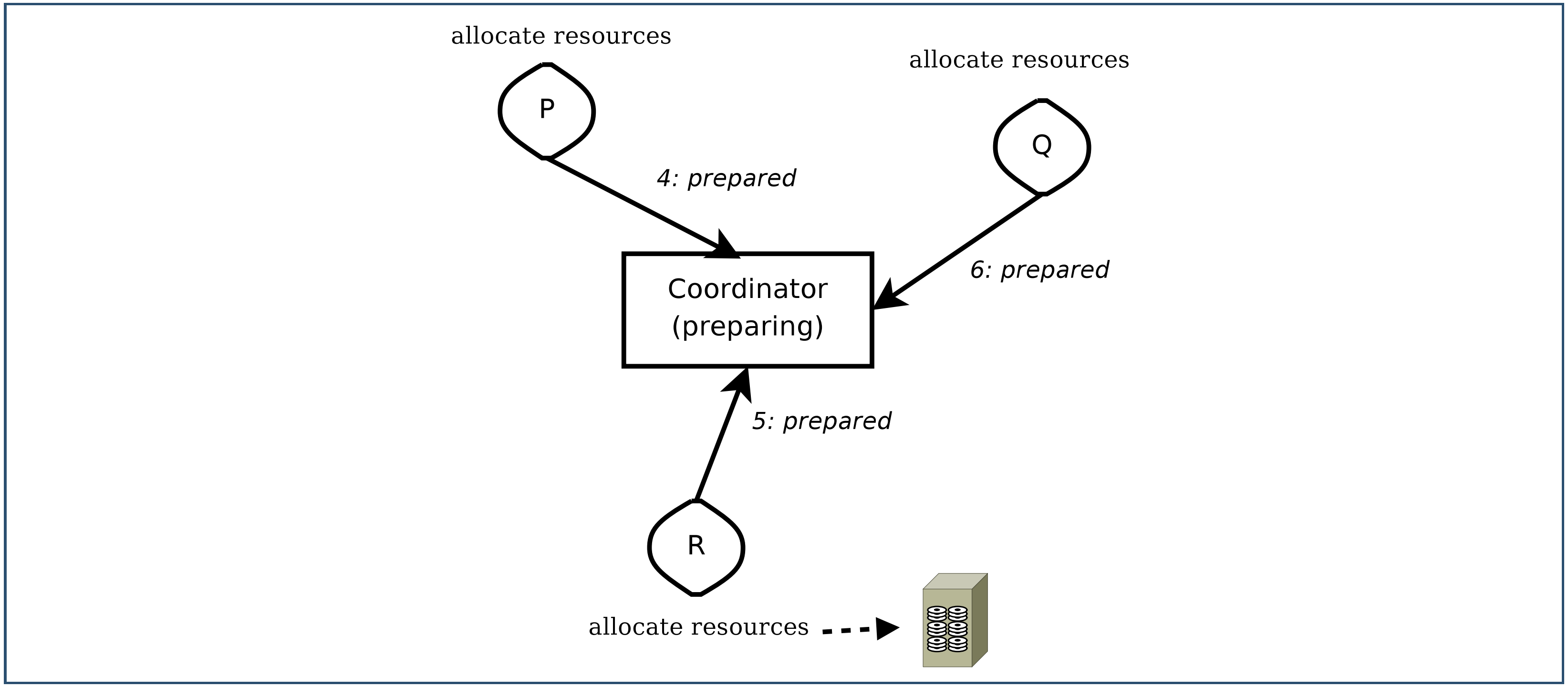


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: participants have prepared

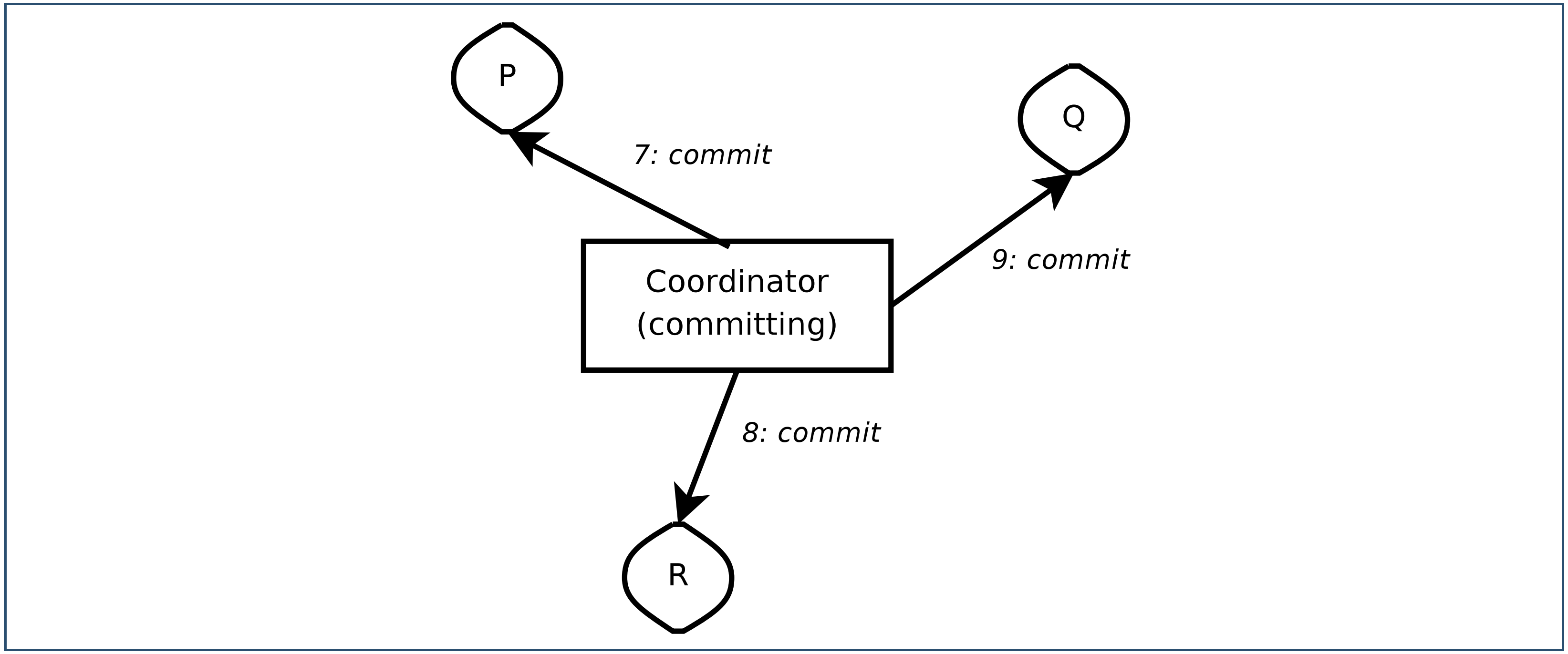


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: coordinator tells everyone to commit

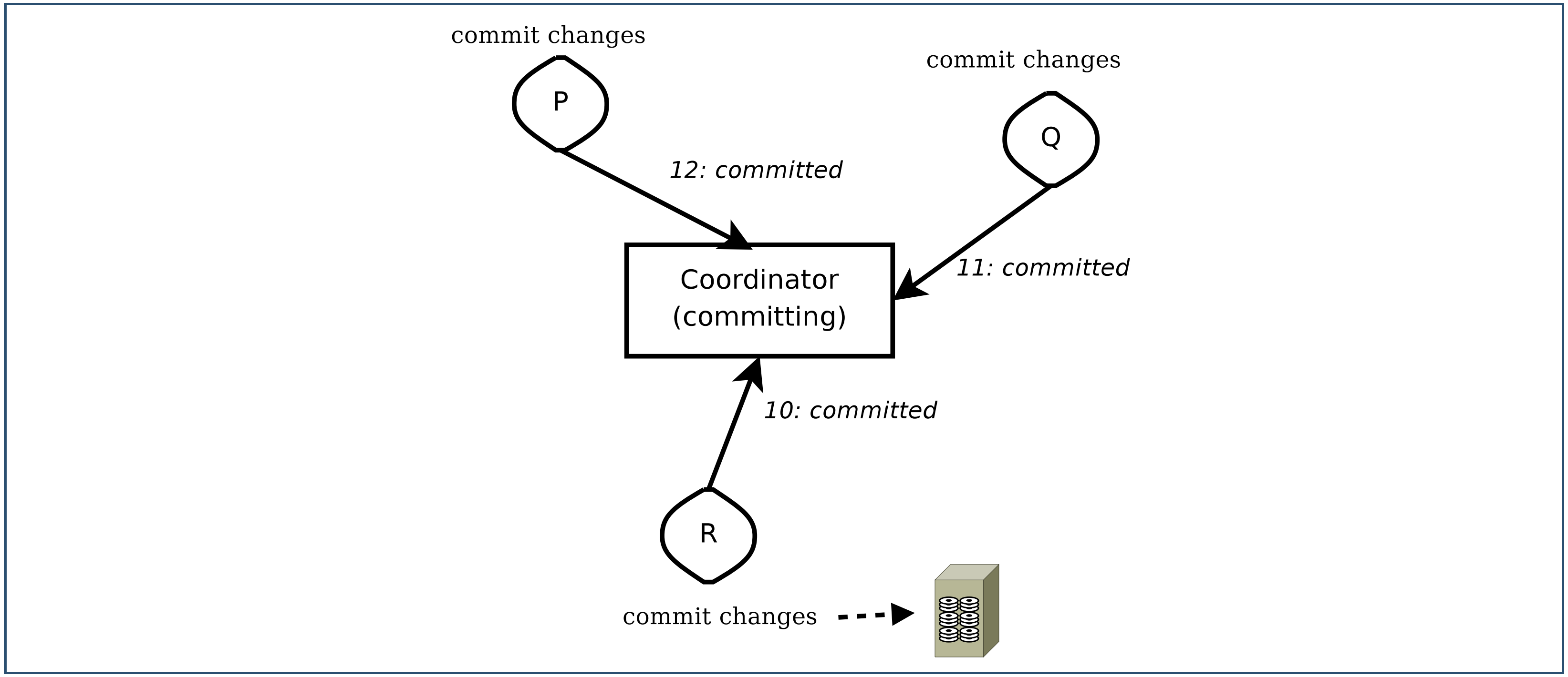


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: participants commits locally

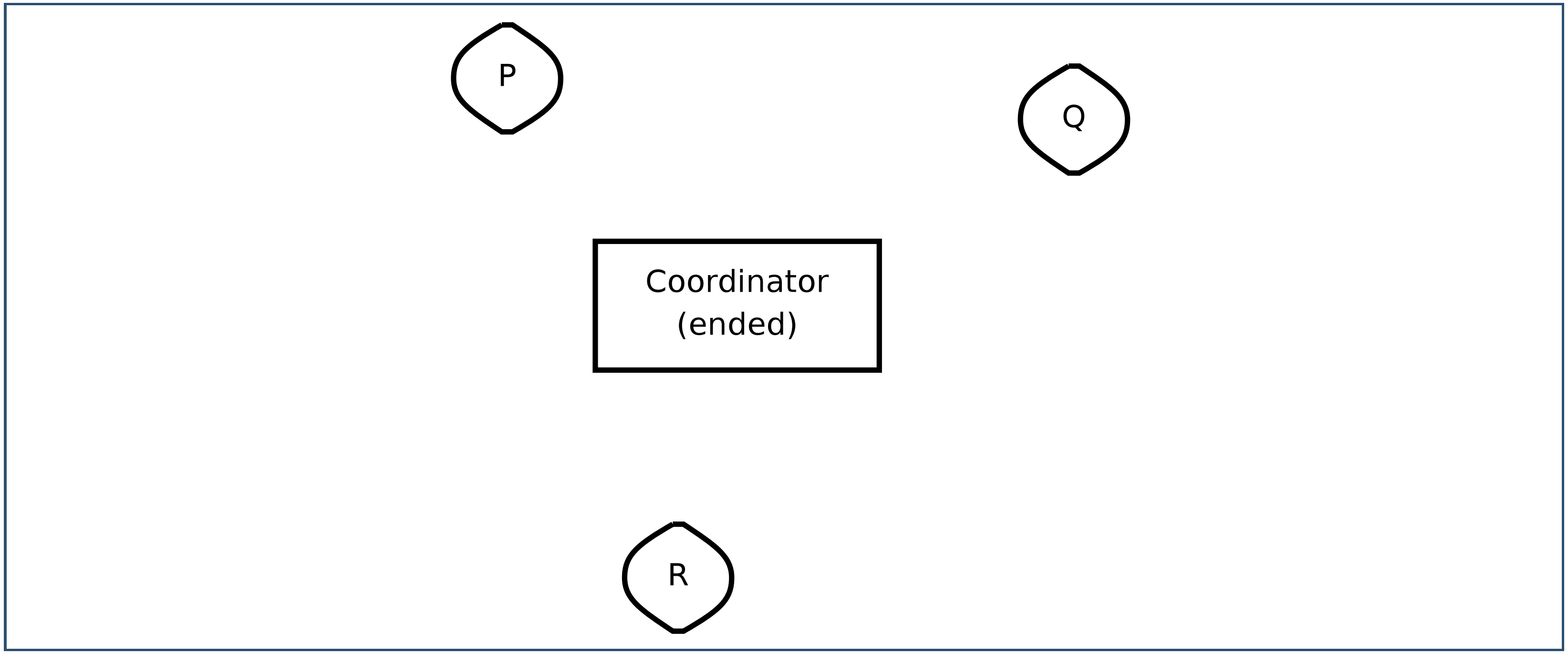


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: transaction completed successfully

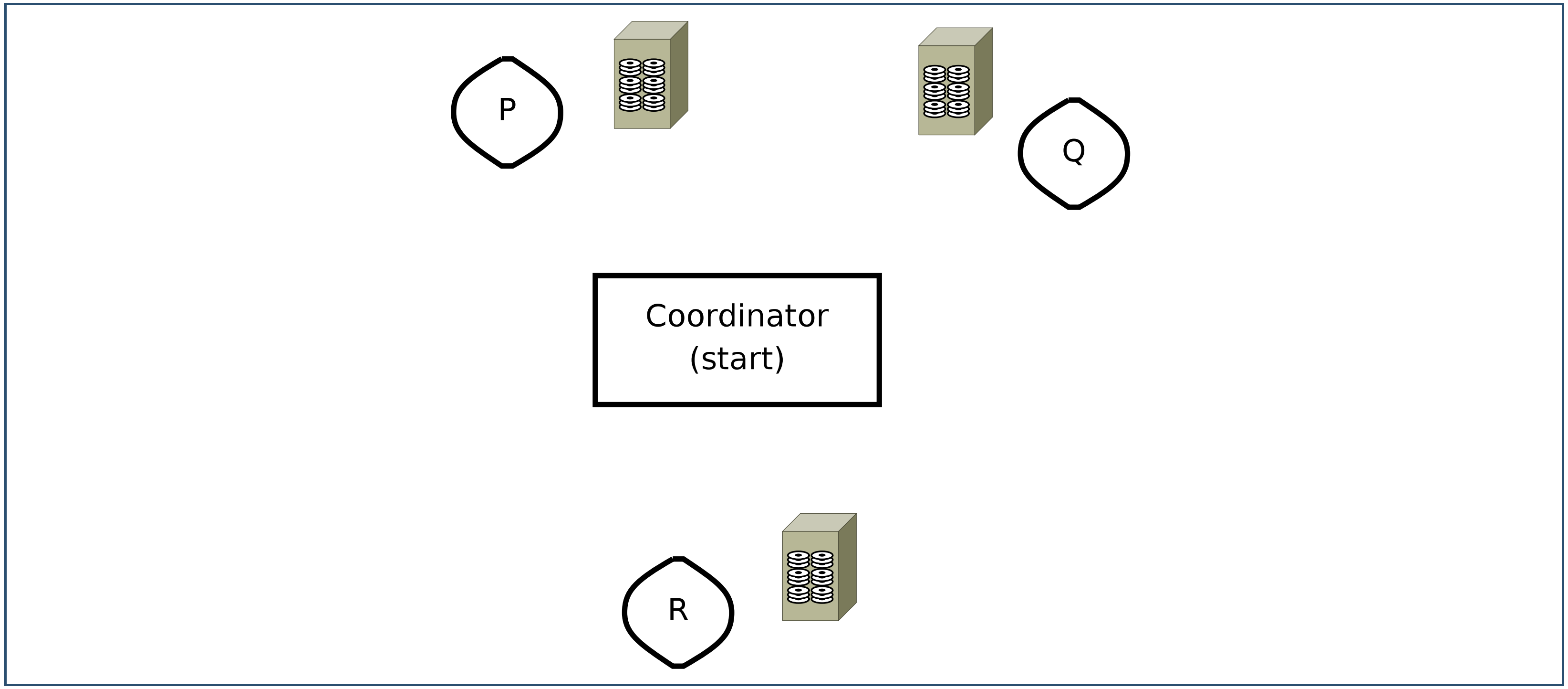


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: init (failure example)

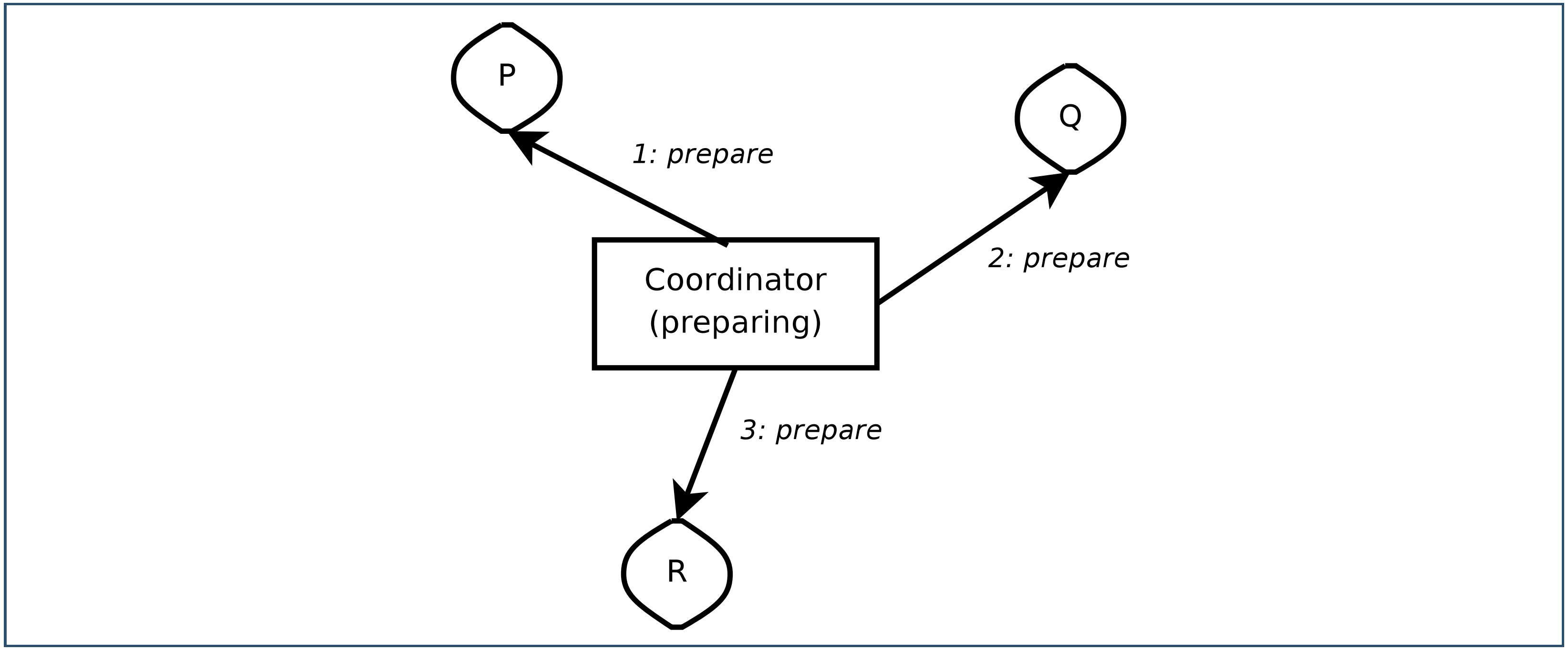


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: asking participants to prepare

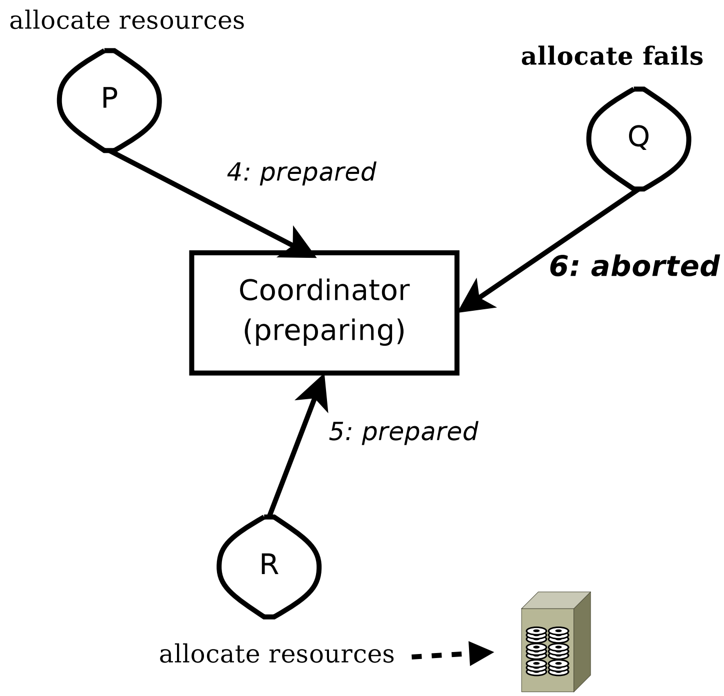


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: preparation fails

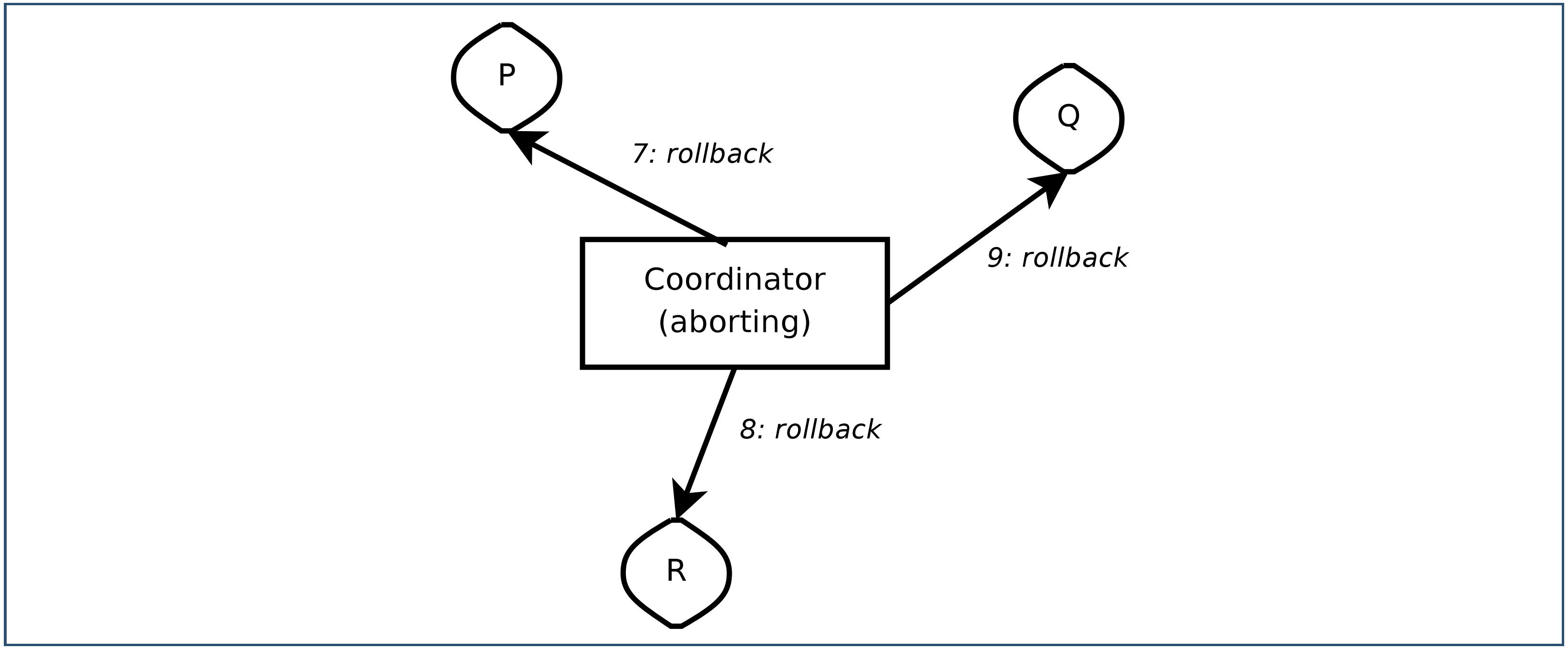


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: asking participants to **rollback**

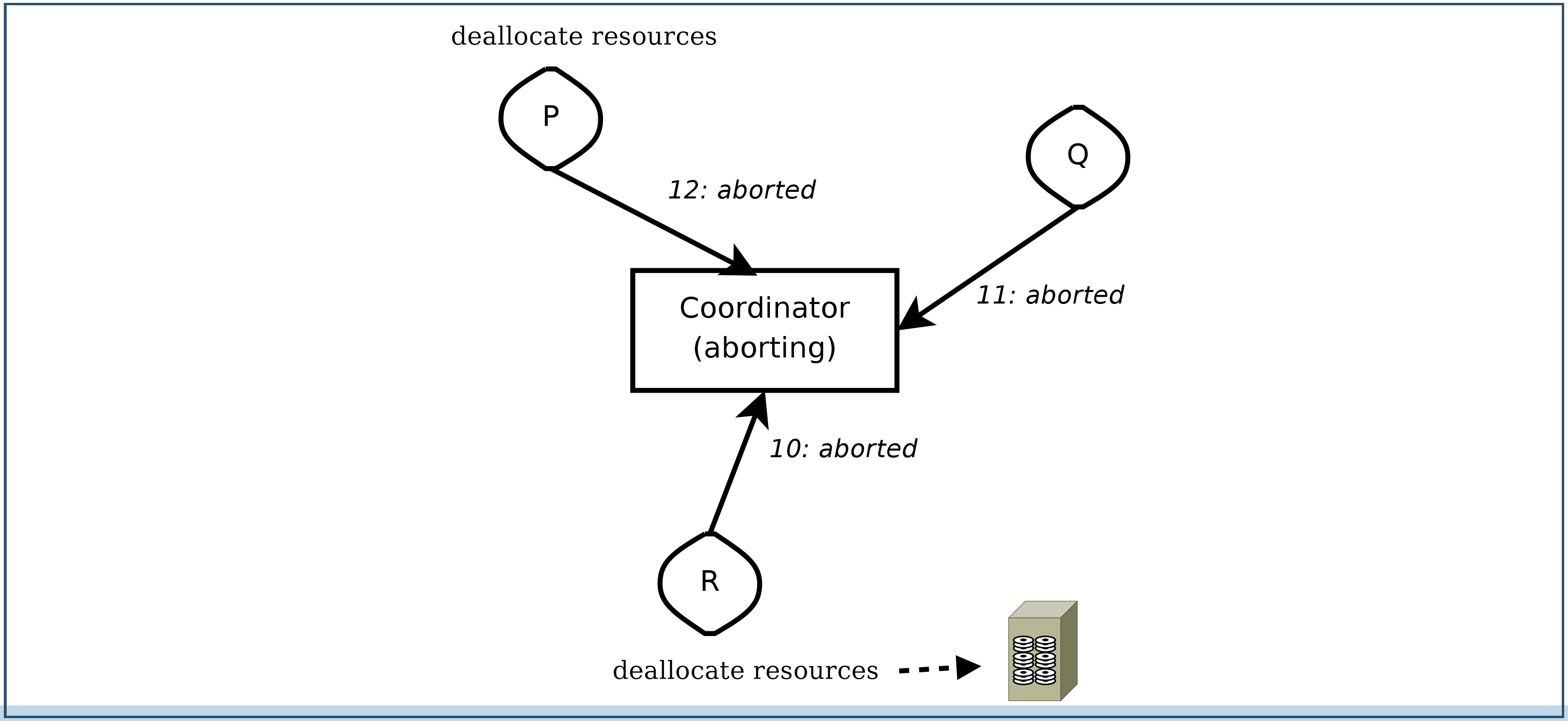


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: participants rollback locally

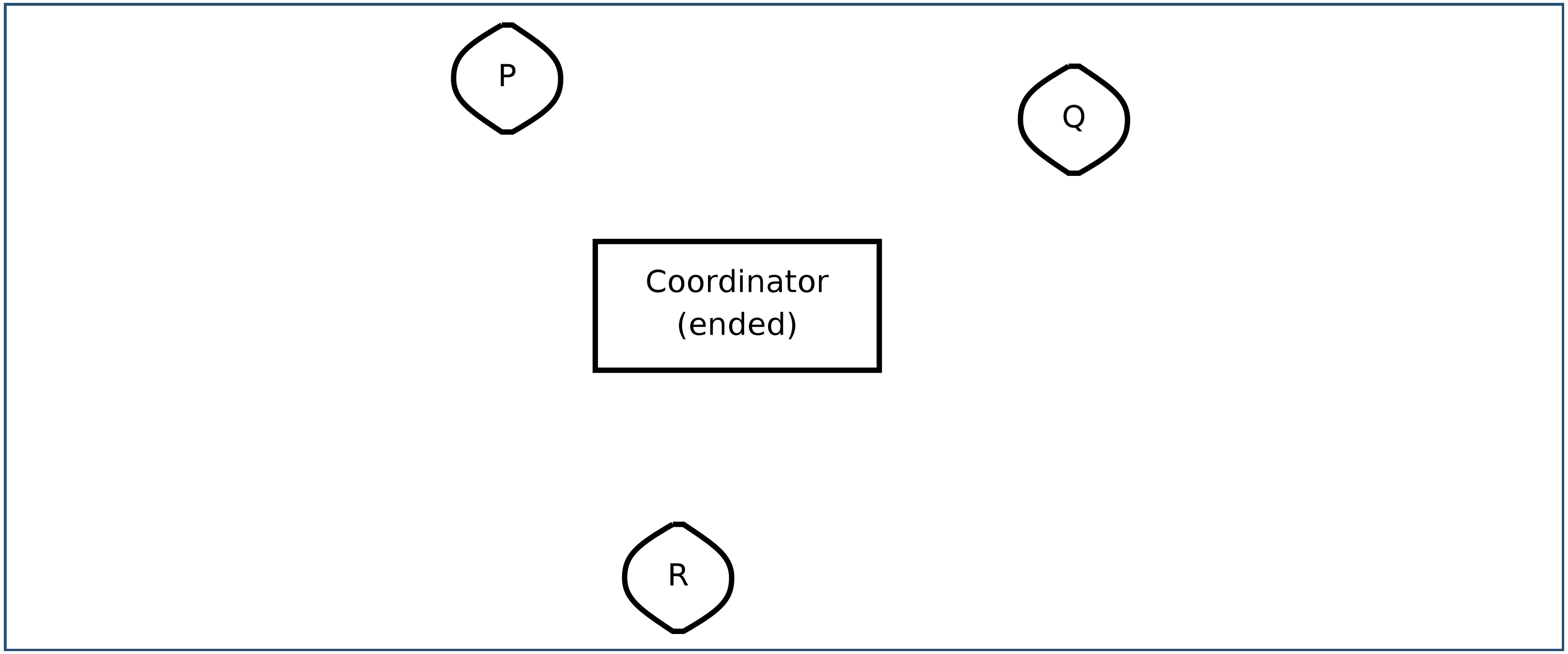


WS-AtomicTransaction

Provides a two-phase commit protocol:

- A coordinator process (Coordinator)
- A set of participants (P,Q,R)

Transaction state: transaction failed



Web Services: continued. . .

- Ok, so we have middleware support as well (WS-Addressing, WS-Transactions, WS-ReliableMessaging, WS-Security, . . .)
- Compare propaganda for "Enterprise Service Buses"
- But we still have no way of describing (web service/process) **behaviour**!

Processes and Interactivity

- One solution for more directly defining the concrete behaviour of processes is using **business process work flow diagrams**

- In defining business processes it is common to define
  - needed tasks
  - how tasks are ordered, and
  - how one task can invoke other task to solve a task using such work flows

Processes and Interactivity

- One solution for more directly defining the concrete behaviour of processes is using **business process work flow diagrams**
- In defining business processes it is common to define
  - needed tasks
  - how tasks are ordered, and
  - how one task can invoke other task to solve a task using such work flows
- What if you (roughly) combine work flows with web services?
- You get: *business processes* and *business process web standards*!

Why are Business Processes interesting to us?

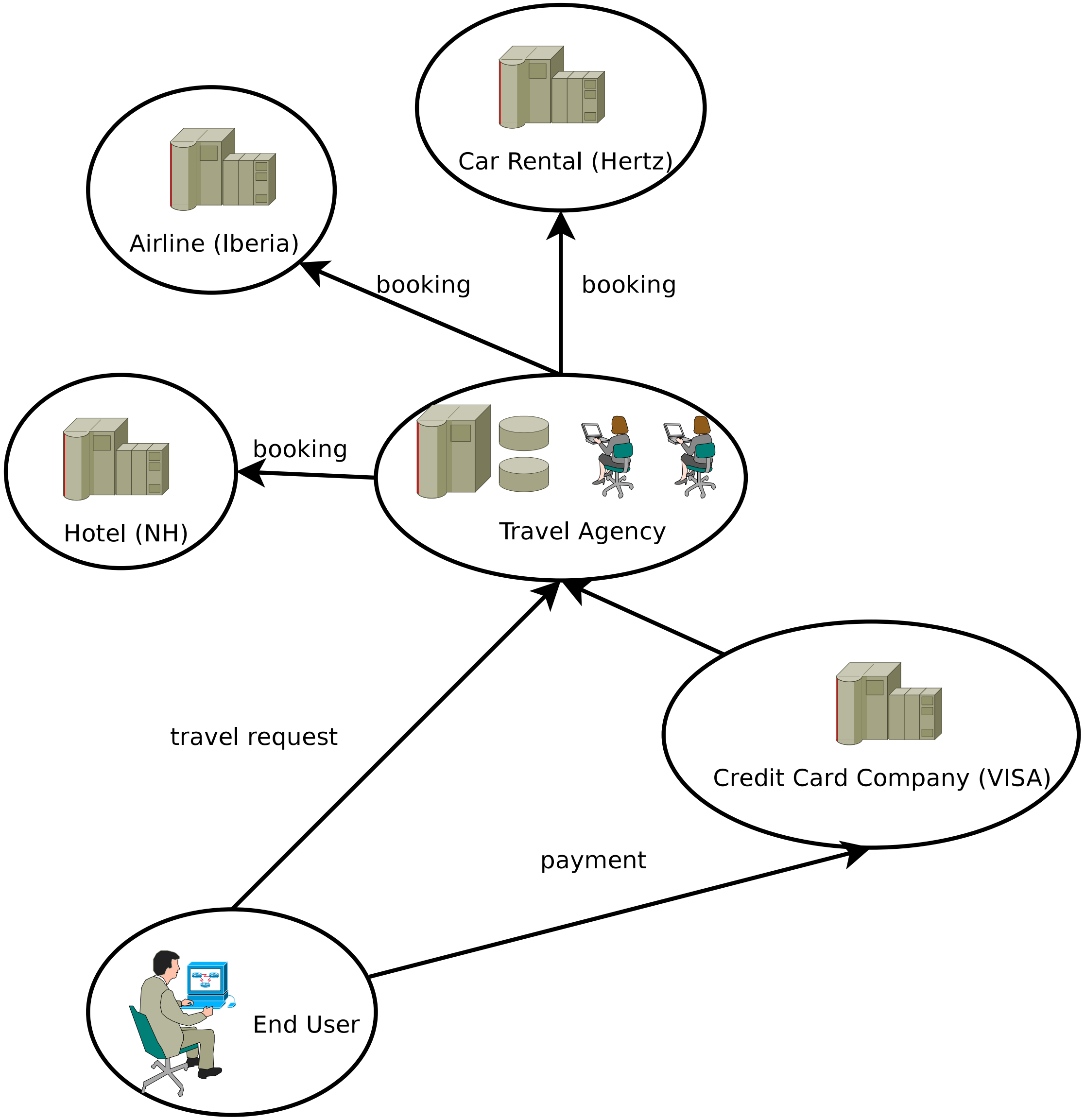
- They are more ambitious than Web Service definition languages like WSDL which only talk about static service properties
- Business process languages directly address behaviour aspects; i.e., not only which operations are made *available* but what is their *effect*
- They address concurrency and distribution directly
- An often stated claim is that they are based on formal standards (petri nets, π-calculus) $\longrightarrow$ could lead to verifiable web services!

What is a Business Process

One definition:

- has a state, upon which tasks operate
- is long-running (i.e., the process spans hours, days, months or more)
- the state should be persistent (i.e., stored in a database)
- bursty, sleeps most of the time (responds to triggering events)
- the system is responsible for orchestration of system or human communication
  (the system manages the communication of human and system agents)

Travel Agency Example



A Typical Business Process

A travel agent business process can contain the following tasks:

1. Get a customers itinerary (travel and time plans)
2. For each item in the itinerary, attempt to book it (flights with airlines, rooms with hotels, cars with rental agency)
3. If all bookings succeed, get payment from customer and send confirmation to customer
4. If at least one booking fail, report problem to customer
5. If customer wants to continue return to task (1) otherwise stop

A Typical Business Process

A travel agent business process can contain the following tasks:

1. Get a customers itinerary (travel and time plans)
2. For each item in the itinerary, attempt to book it (flights with airlines, rooms with hotels, cars with rental agency)
3. If all bookings succeed, get payment from customer and send confirmation to customer
4. If at least one booking fail, report problem to customer
5. If customer wants to continue return to task (1) otherwise stop

- Note that task 1 must be done before 2, whereas all tasks in 2 can proceed in parallel, but all tasks must complete before task 3 begins, and so on
- Languages for Business Processes typically talk about such task flow relations (**in sequence**, **in parallel**, . . .)

Combining manual and automatic tasks

- Business processes typically combine automated tasks and manual (involving people)
- For instance, getting a customers itinerary and attempting to book them could be mainly automatic tasks
- If a booking fail, a human should contact the customer with this information and offer to provide assistance and resolve the problem
- There is a *management component*: typically human travel agents will see a screen of ongoing failed reservation attempts and can choose to focus on a particular one to resolve it

Business Process Advantages

For a company, using computer supported tools for tracking and steering business process offers clear advantages:

- Quick status: what is the status of a certain business order
- Mechanising simple steps (eliminating expensive humans)
- Letting human experts (experienced travel agents) focus on difficult problems
- Optimising other resources

Process Composition

- A set of standards have been developed for integrating business process modelling with web services
- These address the question of how one web service can utilise other web services to achieve its goals (web service - web service communication)

Roughly these standards can be separated into two different styles, depending on how such web service cooperations are defined:

Process Composition

- A set of standards have been developed for integrating business process modelling with web services
- These address the question of how one web service can utilise other web services to achieve its goals (web service - web service communication)

Roughly these standards can be separated into two different styles, depending on how such web service cooperations are defined:

- *Orchestration*: the system behaviour is defined only from the point of view of a single process, and how it interacts with the outside world

Process Composition

- A set of standards have been developed for integrating business process modelling with web services
- These address the question of how one web service can utilise other web services to achieve its goals (web service - web service communication)

Roughly these standards can be separated into two different styles, depending on how such web service cooperations are defined:

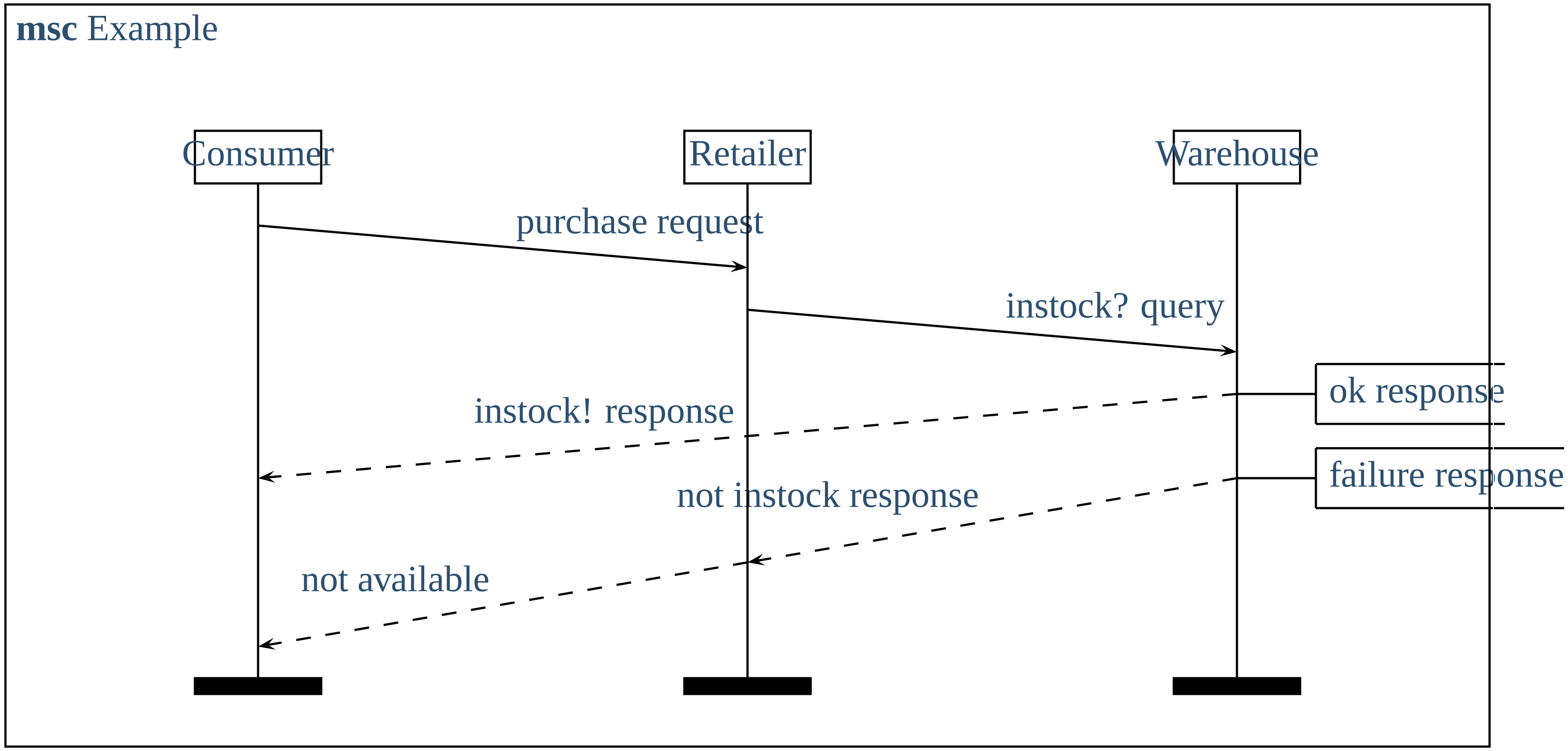
- *Orchestration*: the system behaviour is defined only from the point of view of a single process, and how it interacts with the outside world
- *Choreography*: the behaviour is defined as an multi-party collaboration (a “dance”) between several processes

Orchestration versus Choreography

- In other words:
  - orchestration is about defining one web service,
  - choreography is about describing collaboration among web services
- Choreography is (supposedly) important for business-to-business communication: defining contracts between peers
- Note similarities between web service choreography and (distributed) protocol definitions

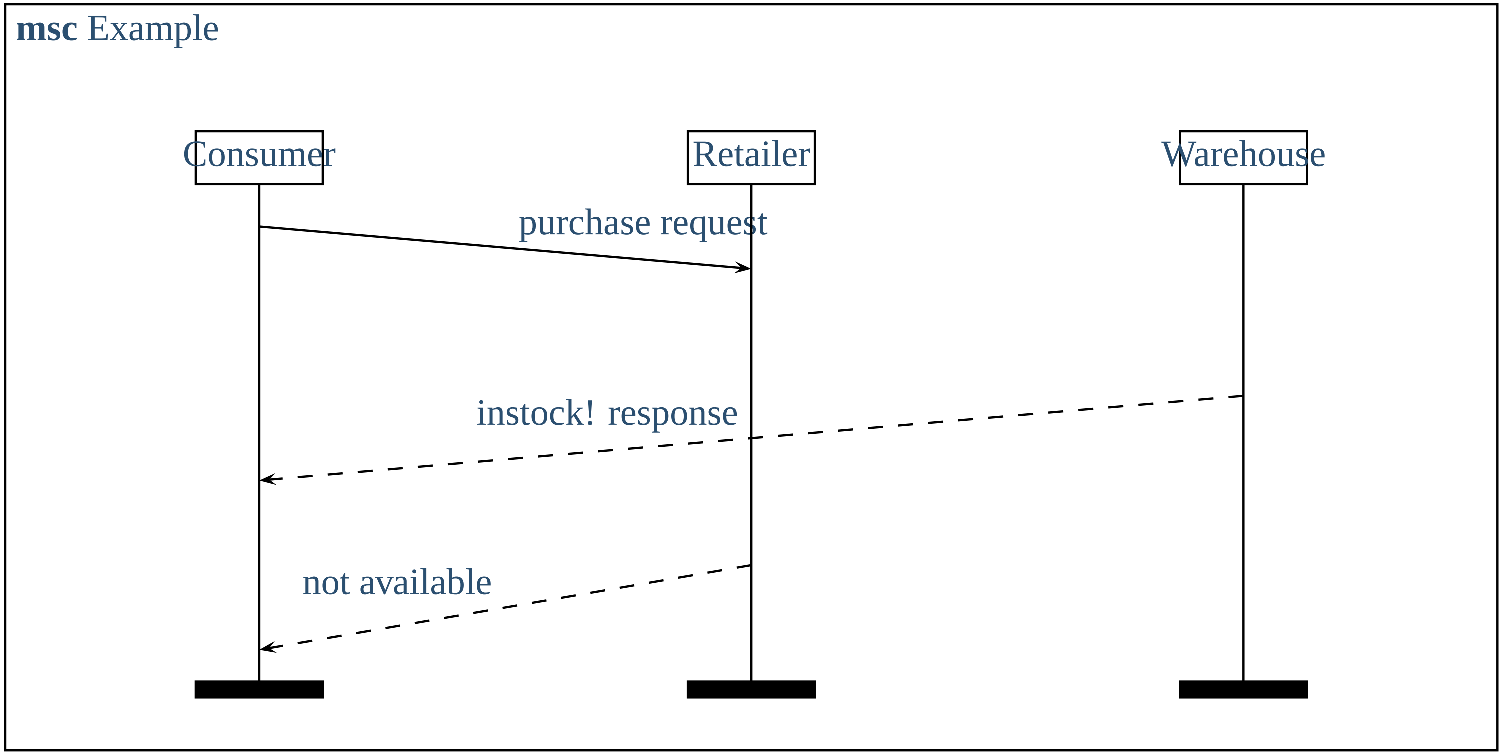
Choreography

Showing protocol negotiation:



Orchestration

Focusing on the Consumer:



Business Process Languages

- For orchestration: BPEL – Business Process Execution Language
- For choreography: WS-CDL – Web Services Choreography Description Language
- Flow-based graphical notation: BPMN – Business Process Modeling Notation (OMG)

Talking with the outside world: BPEL

- BPEL – Business Process Execution Language is a popular standard for defining business processes as web services
- It is used to describe the *orchestration* of Web Services
- Comprises a language of basic behavioural constructs
- A BPEL specification of a business process has two parts:
  - Web Service Definitions (in WSDL) of interfaces implemented by, and called by, the defined process
  - A definition in BPEL which encodes in XML the definition of the behaviour of the web service

BPEL process behaviours

- Basic control structures: sequences, `while`, `if...else` (choice over data)
- Basic data operations (XPath and other sublanguages)
- A BPEL process addresses other processes through bidirectional *partner links* (identifying communication roles)
- Communication: other web services can be called using `invoke`; reception of messages using `receive` or `pick`. A received messages can be replied to using `reply`
- *Flows* describes temporal dependencies between activities
- Fault handlers handle exceptional events
- Long Running Transaction failure recovery is handled through compensation clauses (remember **WS-BusinessActivity**)

BPEL flows

- Suppose there are three activities A, B, C that should run in parallel, and when all three have finished D should execute:

```
<flow>
  A B C
</flow>
D
```

- More flexible process flows can be set up by explicit links between activities (similar to Petri Nets)
- A link has exactly one *source activity* and one *target activity*

BPEL Flow Example

- Suppose activities A and B run in parallel
- When A has terminated activity C can run;
  when either A or B has terminated D can run

```
<flow>
  <links>
     <link name="AC"/> <link name="AD"/> <link name="BD"/
  </links>
</flow>

<invoke partnerLink="A" ...>
  <source linkName="AC">  <source linkName="AD">
</invoke>

<invoke partnerLink="B" ...>
  <source linkName="BD"> </invoke>
<invoke partnerLink="C" ...>
  <target linkName="AC"> </invoke>
<invoke partnerLink="D" ...>
  <target linkName="AD"> <target linkName="BD">
</invoke>
```

BPEL evaluation

- Currently very popular
- It is an implementation language
- Several *execution engines* permit execution of BPEL processes
- BPEL language less useful for describing detailed data dependent behaviour – more intended for describing process flow and communication
- BPELJ is a combination of Java (for data behaviour) and BPEL for flow and communication
- Several alternative languages (BPMN,. . .) exists that provide a graphical notation for XML based business process languages

WS-CDL: choreography of web services

WS-CDL ≡ Web Services Choreography Description Language

- In contrast to web service orchestration language BPEL, WS-CDL is a language for choreography of web services
- orchestration is about defining one web service, choreography describes collaboration among web services
- WS-CDL is used for describing protocols involving several cooperating parties (processes having roles)
- Provides a global service view – so not directly implementable (unlike BPEL)
- However the communication end-points may be extracted and implemented (in BPEL or. . .)
- WS-CDL is a W3C Candidate Recommendation

WS-CDL: the language

Has a static part (providing type definitions) and a dynamic part (defining interactions)

- The static part defines:
  - roles (participants in interactions),
  - relationships (binary relations between roles),
  - message formats (usually XML Schemas),
  - and the channels over which information is passed
- Roles have variables, that are not globally accessible
- Channels play a central role in the language, as information carriers, and have very detailed types

WS-CDL dynamics

The dynamic part defines interactions

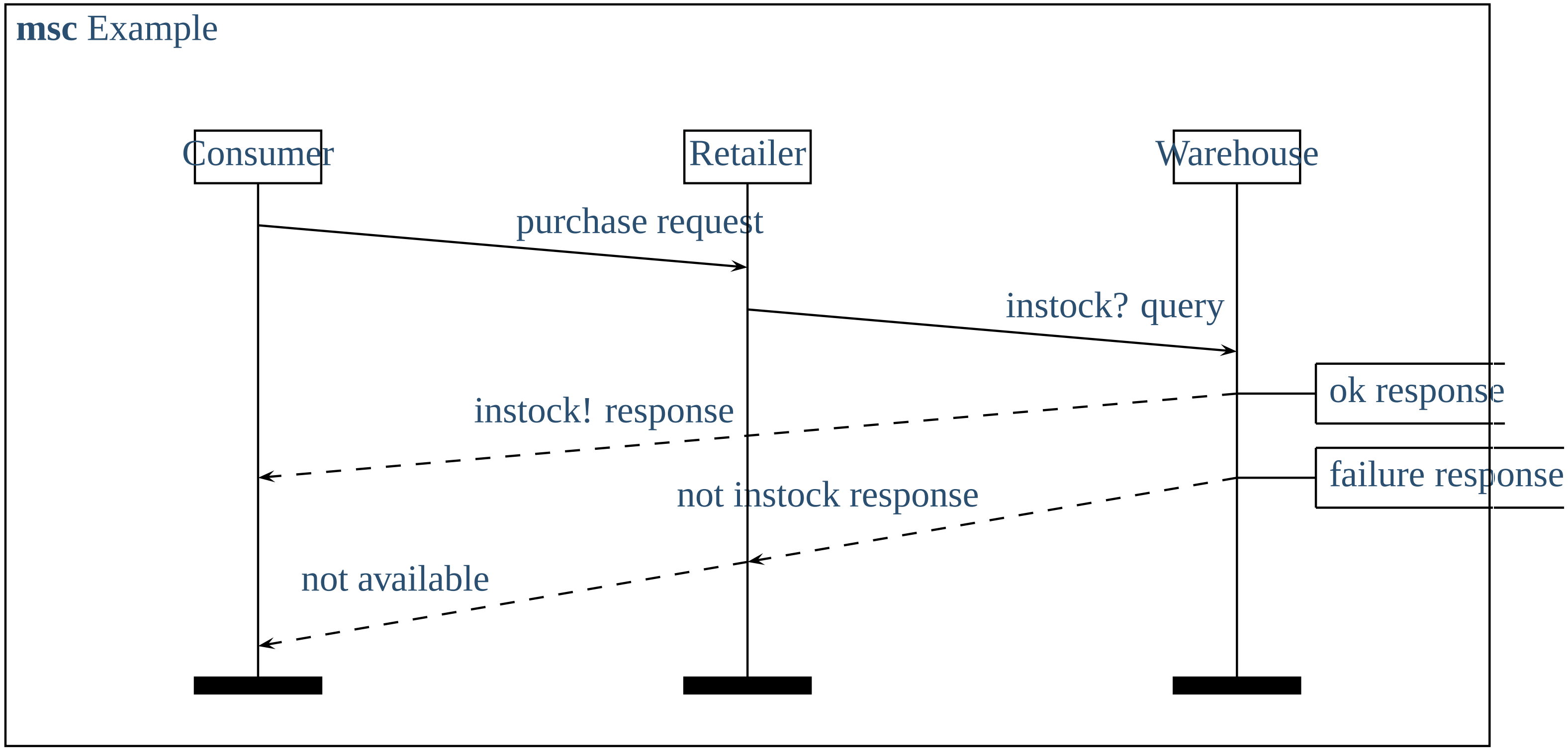
- An interaction takes place between two roles (i.e. *binary* communication is assumed)
- A choreography (or dialogue) between a number of web services is composed of interactions between role types
- Interactions can occur in parallel, in sequence, or can be alternatives (inspired by the π-calculus)
- Interactions typically involve data movement: data moves from the initiator to the responder (and vice versa in a reply)
- Activities can depend on data conditions (guards, looops, . . .)

WS-CDL example

- A highly stylised example:

  1. A consumer issues a buying request to a retailer,
  2. the retailer checks with the warehouse,
  3. if the warehouse contains the item the warehouse responds directly to the consumer,
  4. otherwise the warehous replies to the retailer,
  5. that in turn informs the consumer

WS-CDL example, part II



WS-CDL example, part III

As the language is based on XML specifications quickly grow large – below a small part of the formalised example:

```
<interaction name="createPO"
      channelVariable="tns:RetailerChannel"
      operation="handlePurchaseOrder" >
  <participate relationshipType="tns:ConsumerRetailerRelationship"
     fromRoleTypeRef="tns:Consumer" toRoleTypeRef="tns:Retailer"/>
  <exchange name="request"
     informationType="tns:purchaseOrderType" action="request">
     <send variable="cdl:getVariable('tns:purchaseOrder','','')" />
     <receive variable="cdl:getVariable('tns:purchaseOrder','','')"
            recordReference="record-the-channel-info" />
   </exchange>

   <exchange name="response"
        informationType="purchaseOrderAckType" action="respond">
     <send variable="cdl:getVariable('tns:purchaseOrderAck','','')" />
     <receive variable="cdl:getVariable('tns:purchaseOrderAck','','')" />
   </exchange>
</interaction>
```

WS-CDL dynamics

- The dynamic part is extended with:
  - Exceptions
  - Finalizer blocks
  - Alignment of variables (between invoker and responder)

- To implement a choreography one may need to add new messages as there can be hidden dependencies between processes

- An example is when two processes must agree whether a choreography ends with an exception or not (a *coordinated* choreography)

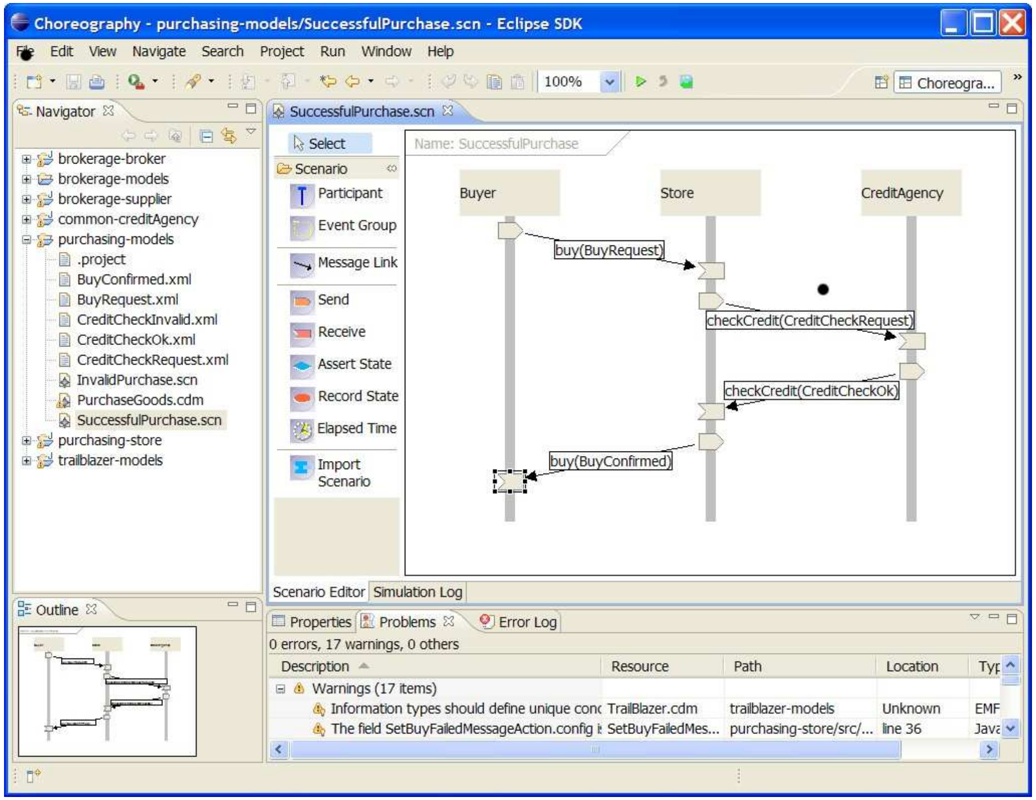
- Channel types are very expressive. One can specify that an instance can only be used by a single process

WS-CDL evaluation

- Descriptions become pretty long; graphical syntax missing
- Who are the users? (not for implementing)
- BPEL can in theory be derived from WS-CDL descriptions

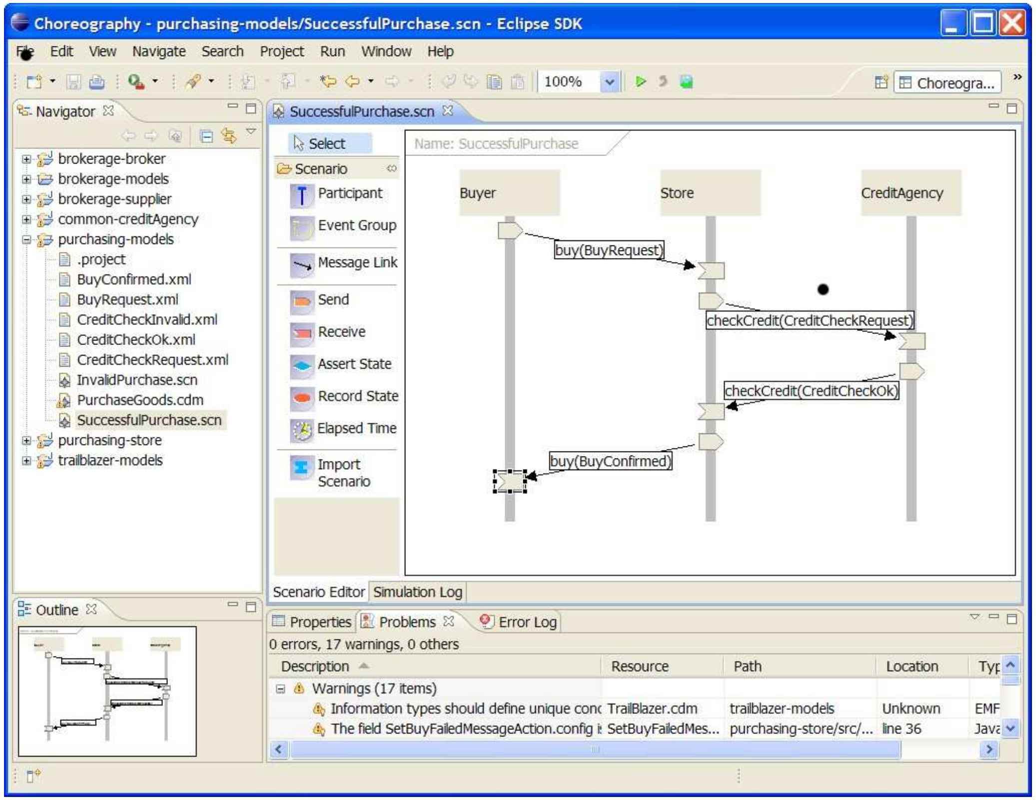
Tool support for WS-CDL

- Pi4soa is an Eclipse-based tool which can be used to experiment with WS-CDL specifications (offering a choreography editor, simulator, generating WS-BPEL. . .)



Tool support for WS-CDL

- Pi4soa is an Eclipse-based tool which can be used to experiment with WS-CDL specifications (offering a choreography editor, simulator, generating WS-BPEL. . .)



- SAVARA (for JBoss)

Popular Web Service Frameworks

- Apache Axis
- Web Services Interoperability Technology (SUN)
- Windows Communication Foundation (Microsoft, .NET-based)
- . . .

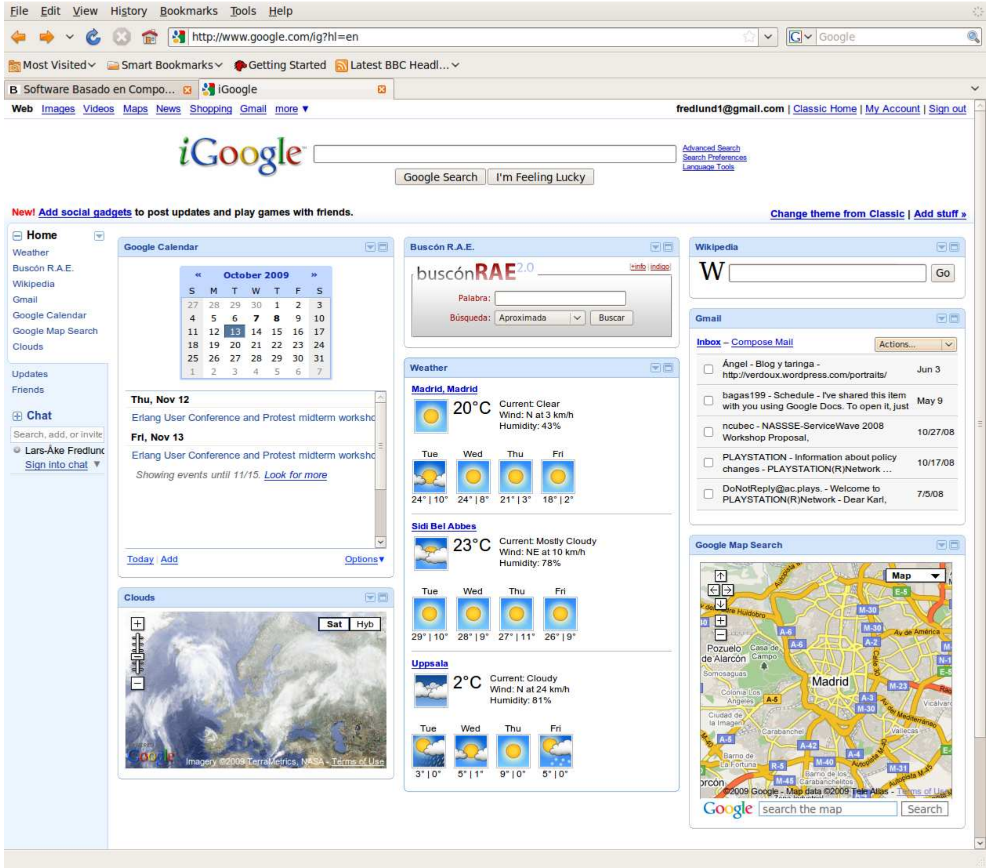
All implement at least WS-Addressing, WS-ReliableMessaging and WS-Security

Coordination Systems: Mashup web application

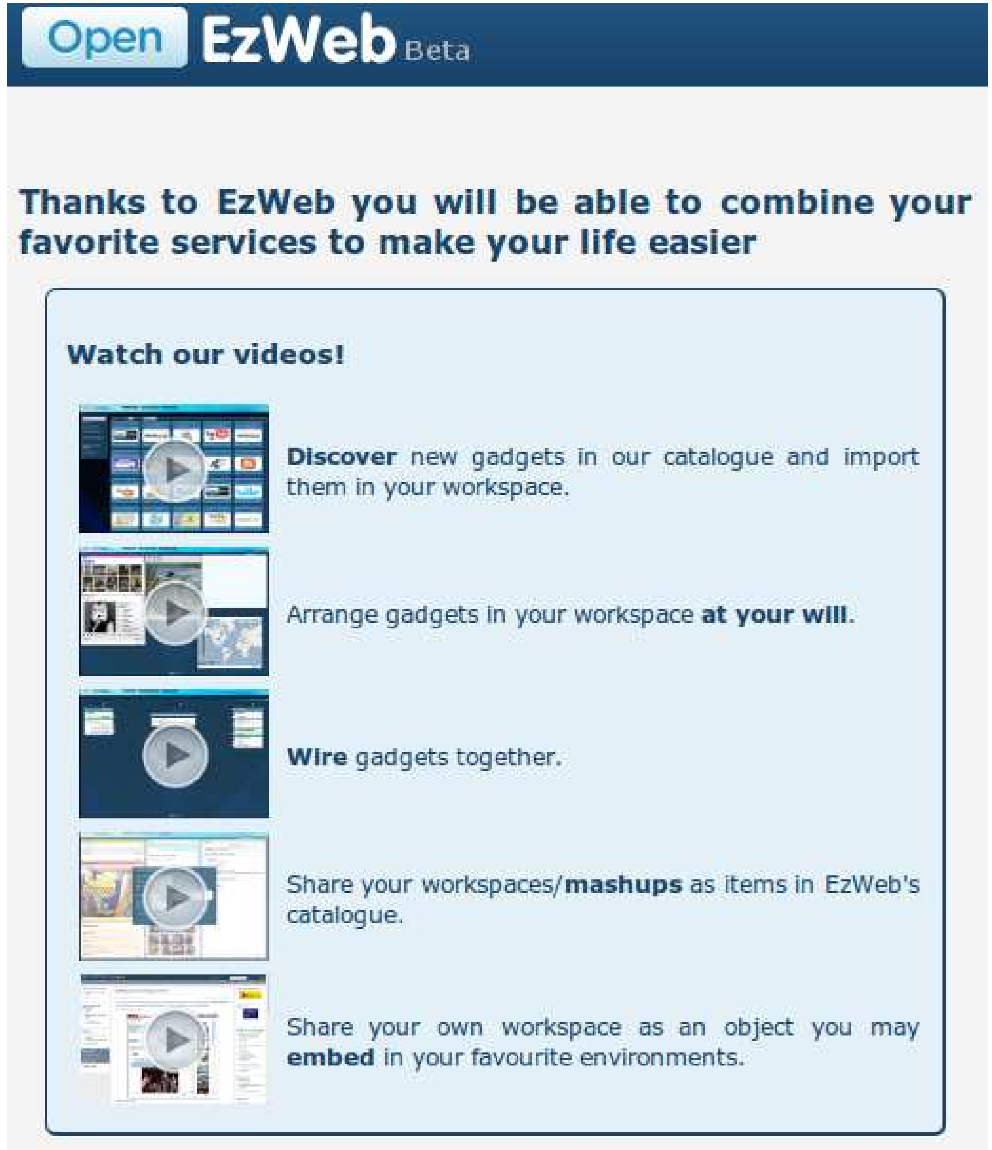
- “A web application that combines data from external sources to create a new service”

Coordination Systems: Mashup web application

- “A web application that combines data from external sources to create a new service”
- Example: a customized google page:



EzWeb: a Spanish Mashup Web Application



EzWeb as a component based platform

- A Telefónica initiative (`http://ezweb.tid.es`)
- Strong connection with component-based thinking
- But the component platform is rather weak (bad concurrency model,. . .)

Conclusions

Web services as a component platform:

- Lots of money in Web Services – as a result a lot of hype driven by companies such as Microsoft, SUN, IBM, Oracle
- Early standards approach yields clumsy solutions
- Layered standards further result in clumsy approaches
- SOA and Enterprise Service Bus are attempts at a more elegant framework – but implemented using the same base standards (XML, SOAP, WSDL)
- Still lacking semantic contract specifications (compare design-by-contract for programming languages)
- Formal methods link for Business Processes Modelling? So far just hype!