COMP9243 Distributed Systems, 13s1

Web Services

Dr. Ingo Weber, NICTA &
School of Computer Science and Engineering
University of New South Wales

9 May 2013

The material in this lecture is based on Dr. Ihor Kuz's material from 12s1, as well as material from COMP9322 lecture slides, prepared by myself, Dr. Helen Hye-young Paik, and Dr. Sherif Sakr. Some material is adapted from one of the textbook *Web Services: Concepts, Architectures and Applications* and is thus under Copyright ©2003 Gustavo Alonso, ETH Zurich and/or Copyright ©2004 Springer-Verlag Berlin Heidelberg. Other references are given throughout the slides.

Outline

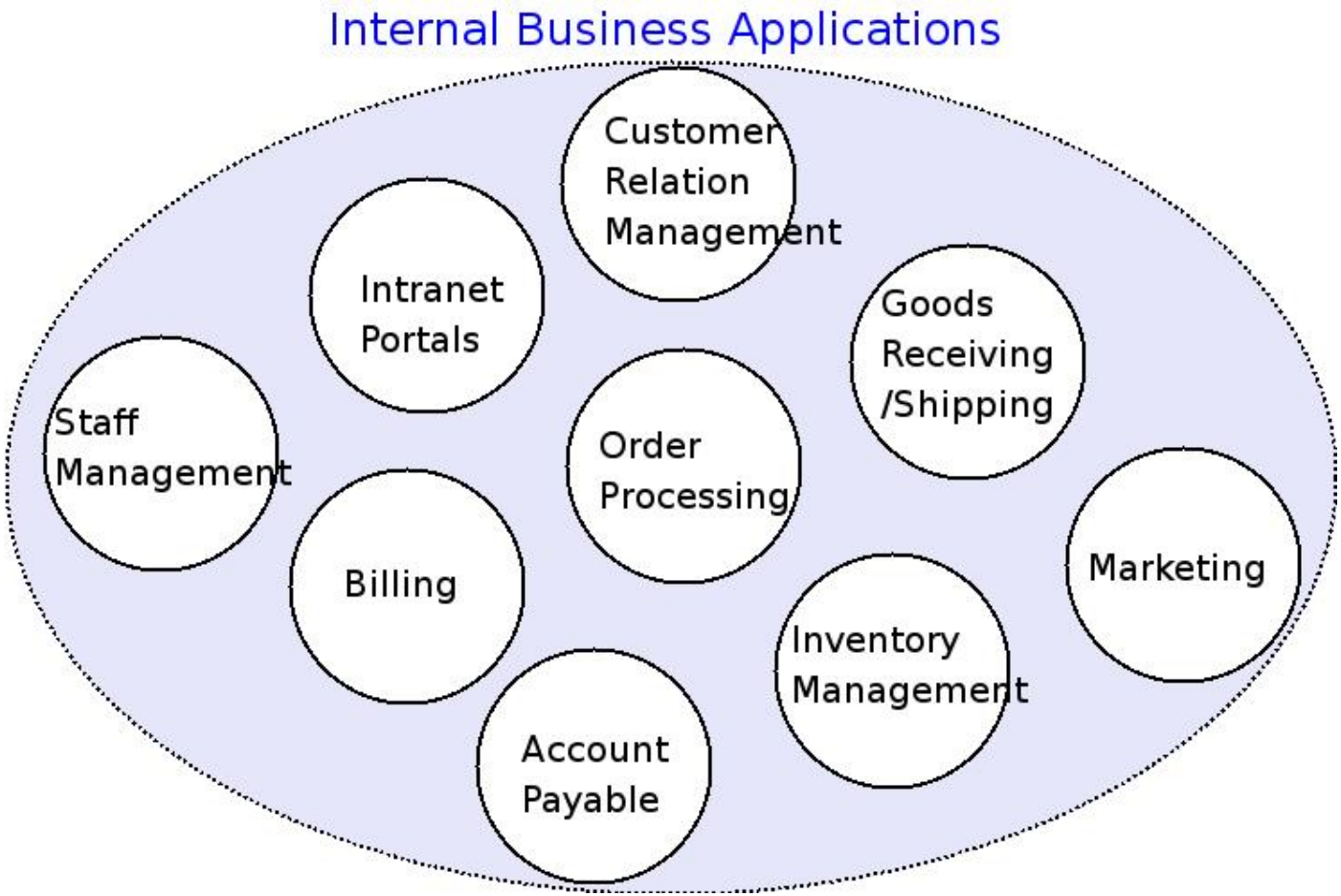
Service-Orientation & SOA
SOAP/WSDL Web Services
RESTful services
Business Process Management & Web Service Composition
WS Composition Technologies: BPMN and BPEL

Part I

Service-Orientation & SOA

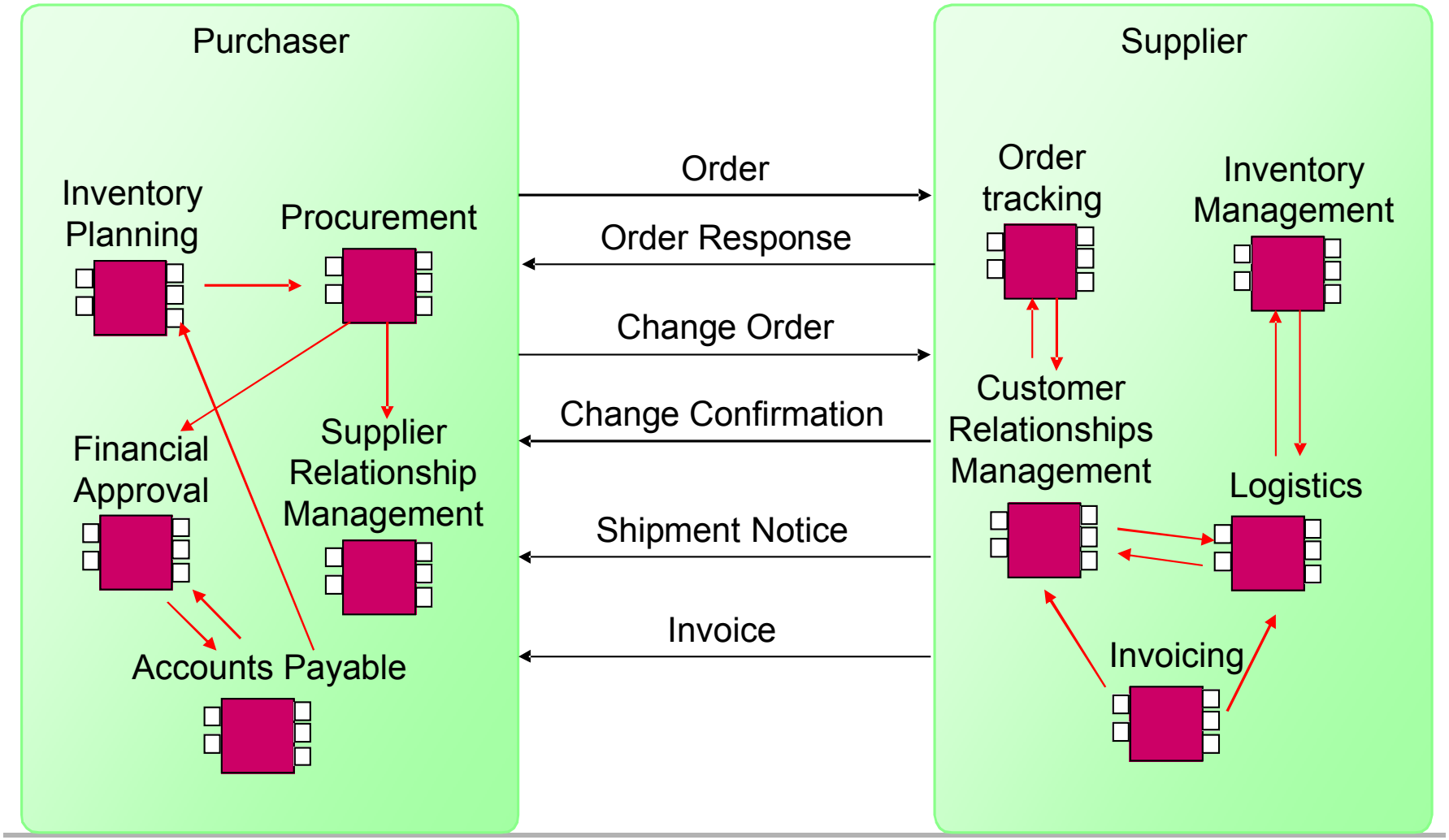
Background

- Distributed information systems in enterprises.
- Pick any sizeable business organisation: it has many departments performing different functionality:



The Problem at a Glance

How to build such a system, integrate existing software, business partners?



Service-Orientation (SO)

How to tackle a complex problem?

- Break things down into smaller bits
- ... abstract, i.e., we need a model
- ... and methodology for model design
- ... and implementation/execution platform for realising the model

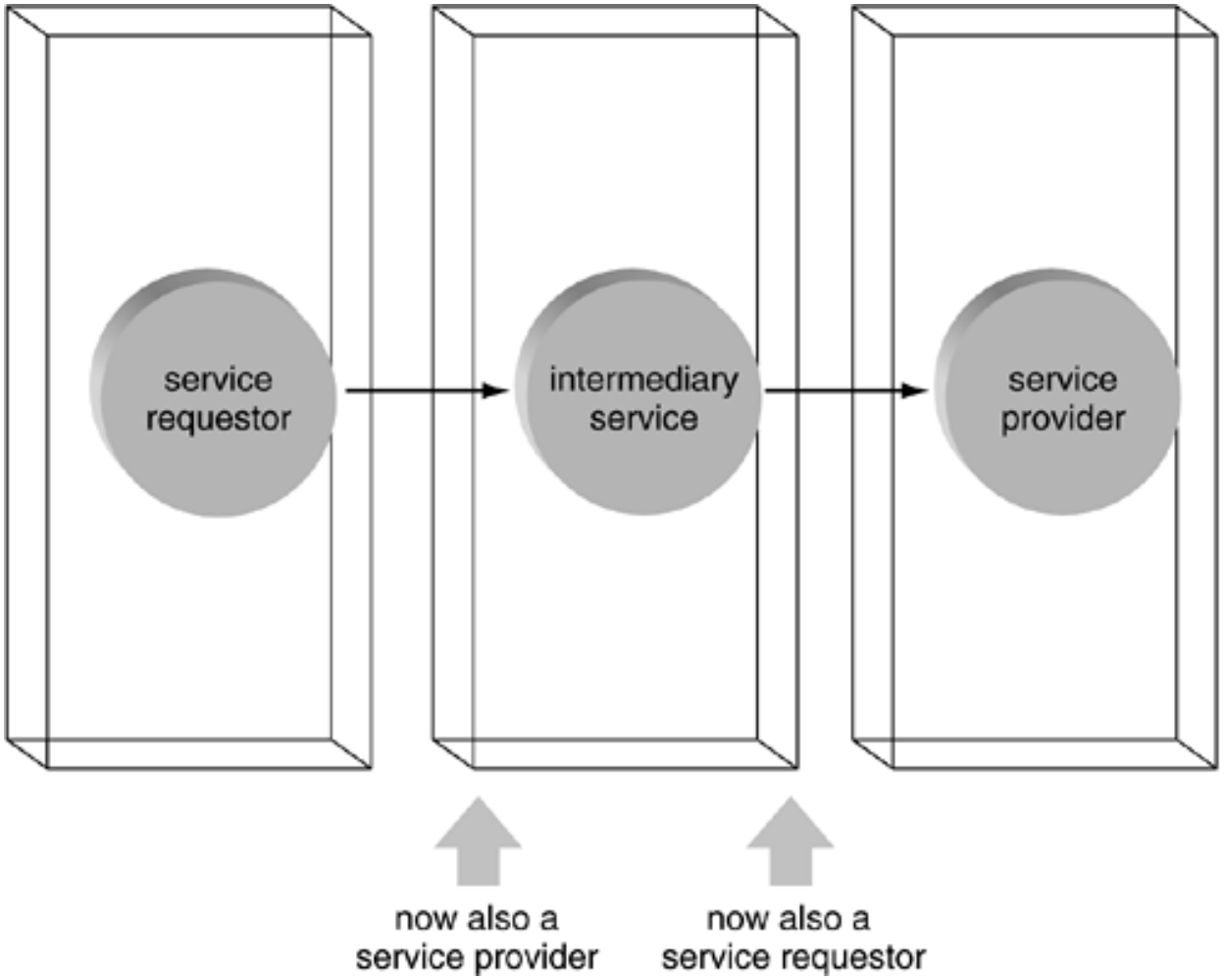
Service-orientation, the principle behind SO Architecture and SO Computing is such an approach.

Analogy:

- OO allows us to model/implement software components as objects,
- SO allows us to model/implement software systems in terms of services

SO: (nearly) everything is a service

Simplified View of Service Orientation:



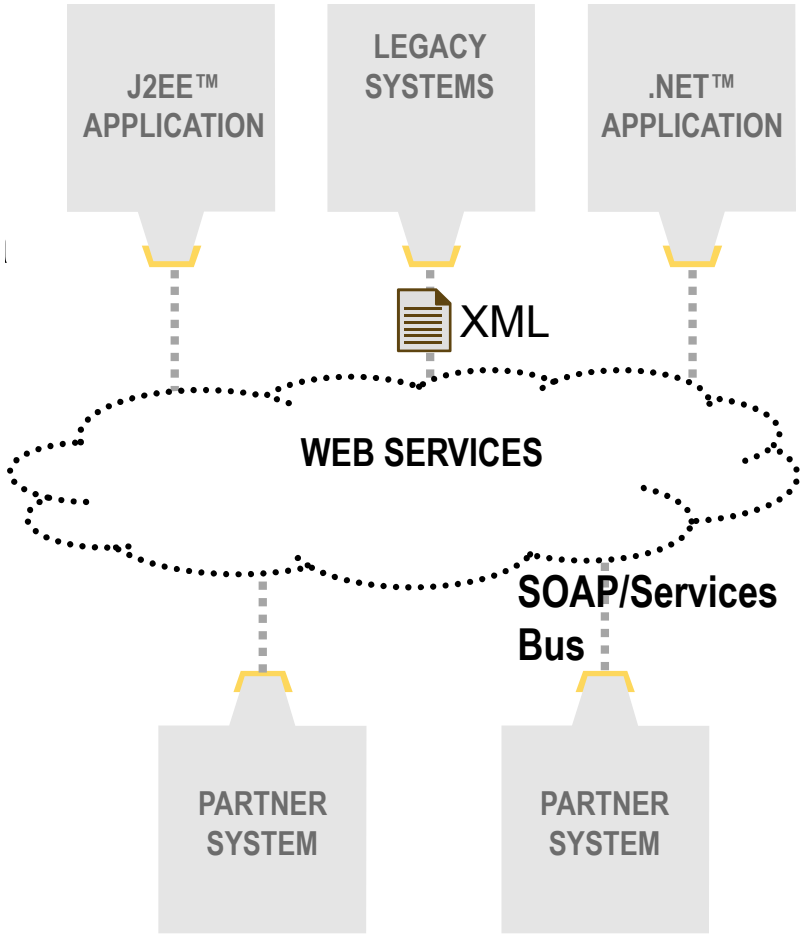
What is SOA?

One important difference with conventional middleware is related to the standardization efforts at the W3C that should guarantee:

- SOA = Services Oriented Architecture.
  - **Services:** another name for large scale components wrapped behind a standard interface (Web services although not only).
  - **Architecture:** SOA is intended as a way to build applications and follows on previous ideas such as software bus, IT backbone, or enterprise bus.
- The part that it is not in the name.
  - **Loosely-coupled:** the services are independent of each other, heterogeneous, distributed
  - **Message based:** interaction can be through message exchanges rather than through direct calls (unlike CORBA, RPC, etc.)

Service Oriented Architectures (SOA)

- Standard protocols and languages (protocols, components, composers).
- Virtualisation (business services, data services, utility services, service registries).
- If the services perform business functions, SOA can help with business-IT alignment.



The novelty behind SOA

- The concept of SOA is not new:
  - Message oriented middleware.
  - Message brokers.
  - Event based architectures.
- The context is different
  - Emergence of standard interfaces (Web services).
  - Emphasis on simplifying development (automatic).
  - Use of complex underlying infrastructure (containers, middleware stacks, etc.)
- Interest in SOA arises from a number of reasons:
  - Basic technology in place.
  - Clearer understanding of distributed applications.
  - The key problem is reuse and integration, not programming.
  - Business-IT alignment.

Web Services

Key concept:

- Allow applications to share data and invoke capabilities from other applications
  - Without regard to how those applications were built, what operating system or platform they run on, and what devices are used to access them.
- Can be called across platforms and operating systems, regardless of programming language.

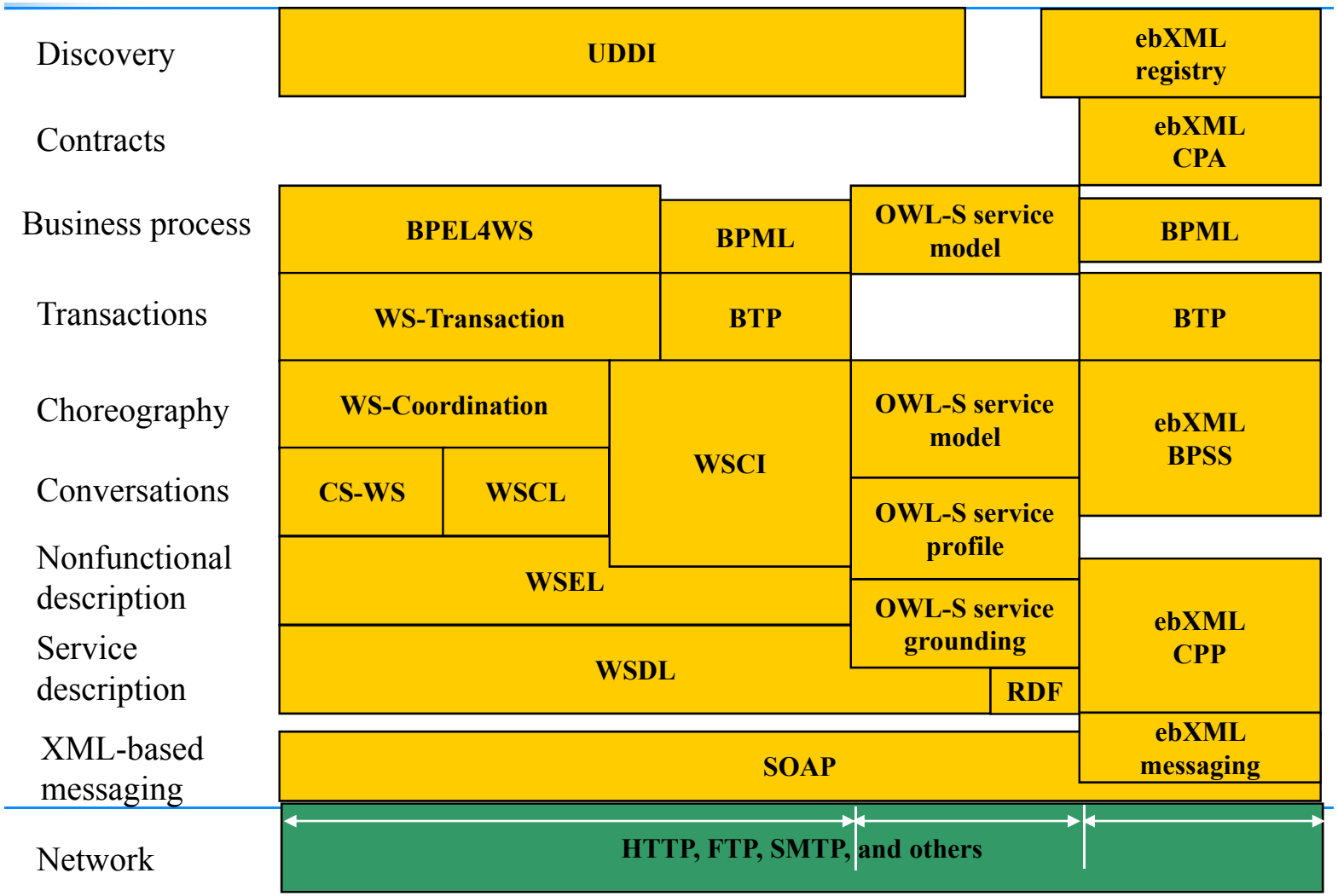
Key facts:

- A standardised way of application-to-application communication based on open XML standards (i.e., SOAP, WSDL and UDDI) and over a standard Internet protocol / backbone.
- Unlike traditional client/server models, Web services do not require GUI, HTML or browser.

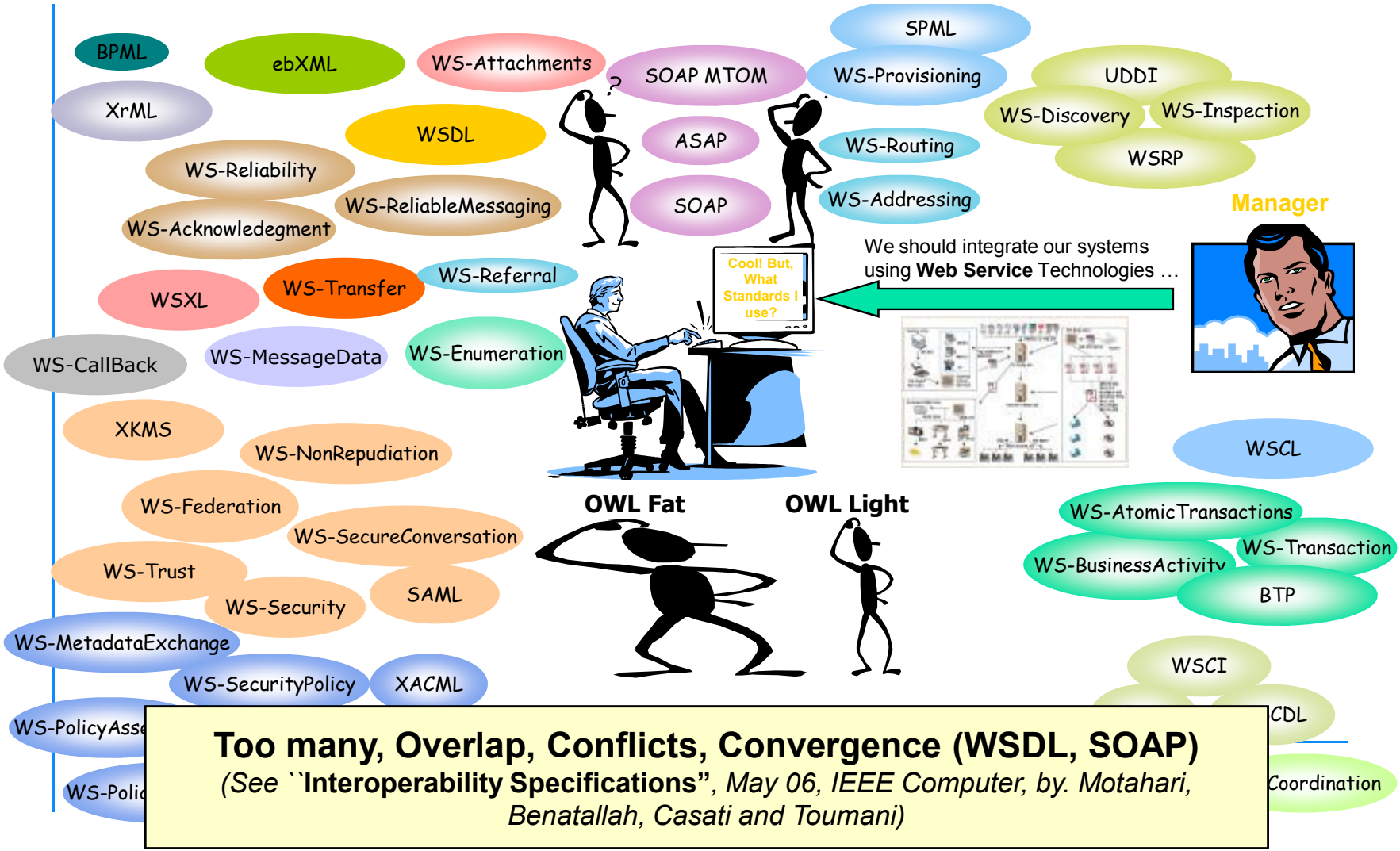
SOA vs. Web services

- Web services are about:
  - Interoperability.
  - Standardization.
  - Integration across heterogeneous, distributed systems.
- Service Oriented Architectures are about:
  - Large scale software design.
  - Software Engineering.
  - Architecture of distributed systems.
- SOA introduces some radical changes to software:
  - Language independence (what matters is the interface).
  - Event-based interaction (asynchronous; synchronous still supported).
  - Message-based exchanges (RPC-like calls are an option, not must).
  - Composition of services.
- SOA is possible but more difficult without Web services.

SOA Layers


Discovery
Contracts
Business process
Transactions
Choreography
Conversations
Nonfunctional description
Service description
XML-based messaging
Network
UDDI
ebXML registry
ebXML CPA
BPEL4WS
BPML
OWL-S service model
BPML
WS-Transaction
BTP
BTP
WS-Coordination
WSCI
OWL-S service model
ebXML BPSS
CS-WS
WSCL
OWL-S service profile
WSEL
OWL-S service grounding
ebXML CPP
WSDL
RDF
ebXML messaging
SOAP
HTTP, FTP, SMTP, and others

WS-* Standard Maze



Too many, Overlap, Conflicts, Convergence (WSDL, SOAP)

*(See ``**Interoperability Specifications**", May 06, IEEE Computer, by. Motahari, Benatallah, Casati and Toumani)*

Service-Orientation Design Principles[1] 1

- **Standardized Service Contract**: the public interfaces of a services make use of contract design standards. (Contract: WSDL in WS*.)
- **Service Loose Coupling**: to impose low burdens on service consumers. (Coupling $\sim$ degree of dependency.)
- **Service Abstraction**: "to hide as much of the underlying details of a service as possible".
- **Service Reusability**: services contain agnostic logic and "can be positioned as reusable enterprise resources".
- **Service Autonomy**: to provide reliable and consistent results, a service has to have strong control over its underlying environment.

[1] SOA Principles of Service Design, Thomas Erl, Prentice Hall, 2007, `http://serviceorientation.com/serviceorientation`. Summary: I. Weber, Semantic Methods for Execution-level Business Process Modeling. Springer LNBIP Vol. 40, 2009.

Service-Orientation Design Principles[1] 2

- **Service Statelessness**: services should be "designed to remain stateful only when required."
- **Service Discoverability**: "services are supplemented with communicative meta data by which they can be effectively discovered and interpreted."
- **Service Composability**: "services are effective composition participants, regardless of the size and complexity of the composition."
- **Fundamental requirement – interoperability of services**: "...stating that services must be interoperable is just about as evident as stating that services must exist."

Part II

SOAP/WSDL Web Services

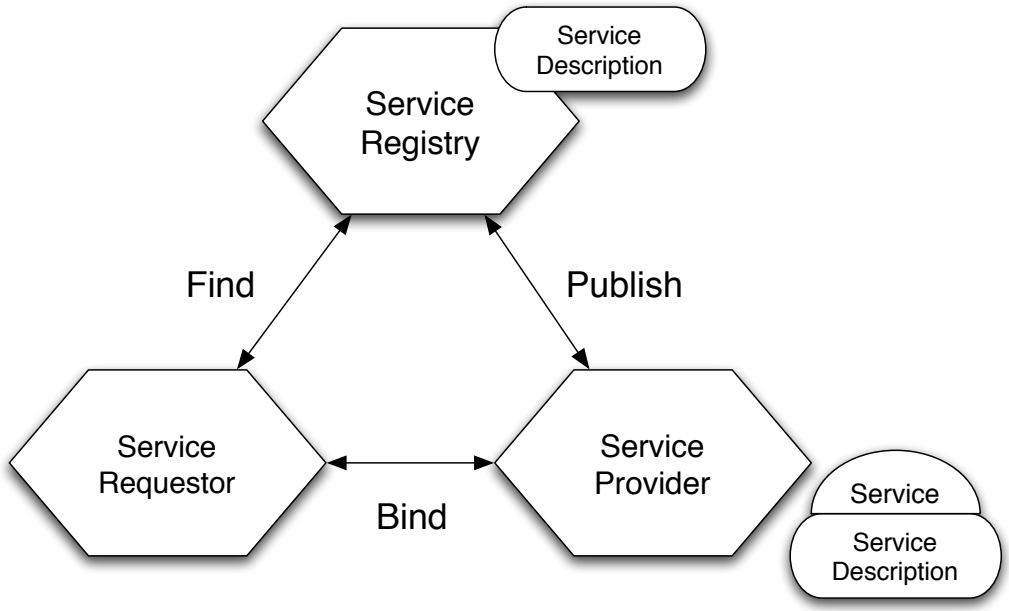
What's a Web Service?

- The Web is for humans!
- How to build a Web-scale distributed system?
- Problem: traditional RPC often makes assumptions on code language and version (cf. Erlang)
- Web service: stand-alone piece of software with standardized interface.

  A Web service (WS) is a self-describing, self-contained software module available via a network, such as the Internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application.
  [M. Papazoglou. Web Services: Principles and Technology. Prentice Hall, 2007.]

Web Services Architecture

- A popular interpretation of Web services is based on three elements:
  - **Service requester:** The potential user of a service (the client).
  - **Service provider:** The entity that implements the service and offers to carry it out on behalf of the requester (the server).
  - **Service registry:** A place where available services are listed and that allows providers to advertise their services and requesters to lookup and query for services.



Web Services Conceptual Architecture[2]

Three roles:

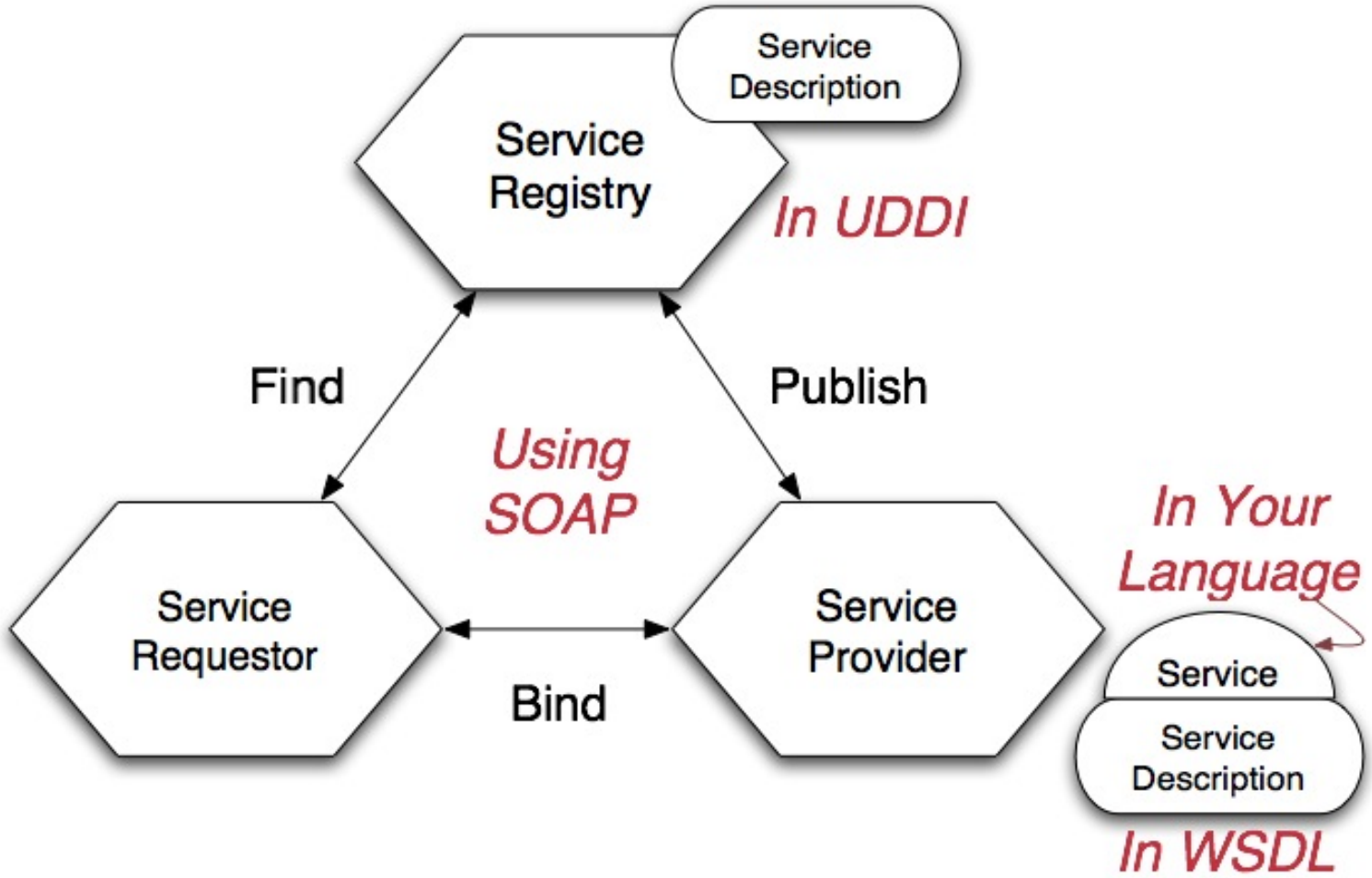
- service provider: develops an electronic service and registers its description at a publicly accessible service registry.
- service registry: stores web service descriptions
- service requestor: queries the registry to find an electronic service that meets his or her requirements. A binding occurs between the service provider and the service requestor.

Main Web Services Standards:

- For service registry: UDDI, Universal Description, Discovery and Integration (`http://uddi.xml.org/`)
- For service description: WSDL, Web-services Description Language (`www.w3.org/TR/wsdl`)
- For messages: SOAP (`www.w3.org/TR/soap`)

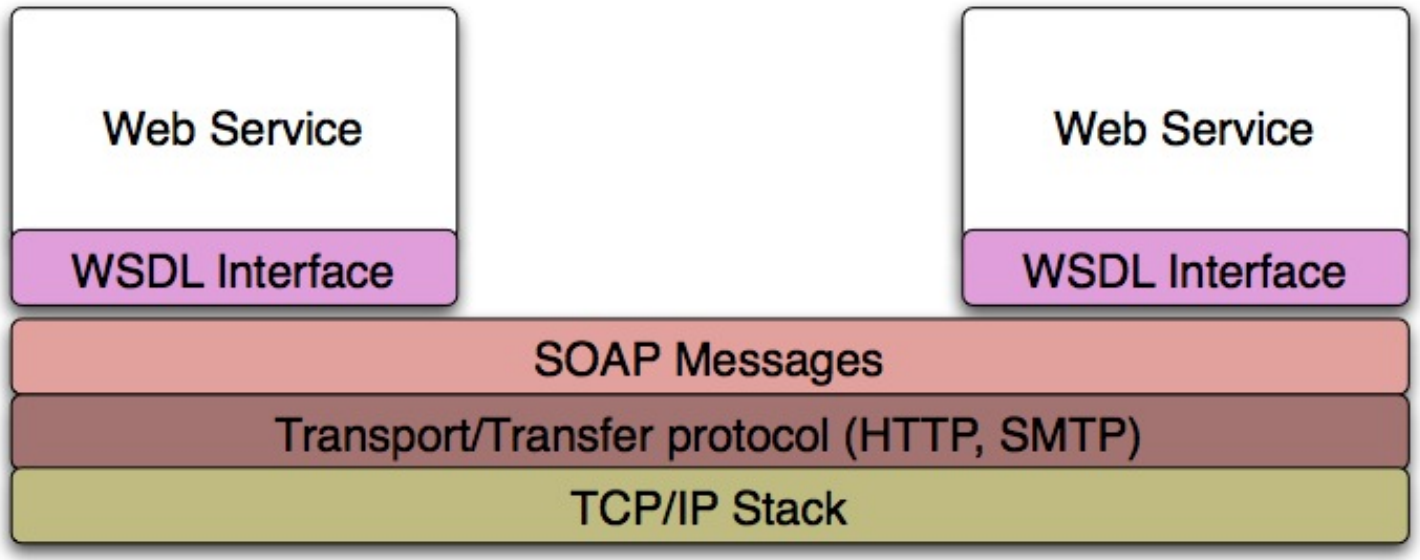
[2] `http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf`

Web Services Conceptual Architecture



SOAP and Web Services

- SOAP is a protocol for transferring data/message across the Internet.
- Web services are remote entities. They use SOAP to transmit data to and from clients (client can also be a Web service)
- So, SOAP is at the heart of Web services architecture in terms of enabling message exchanges

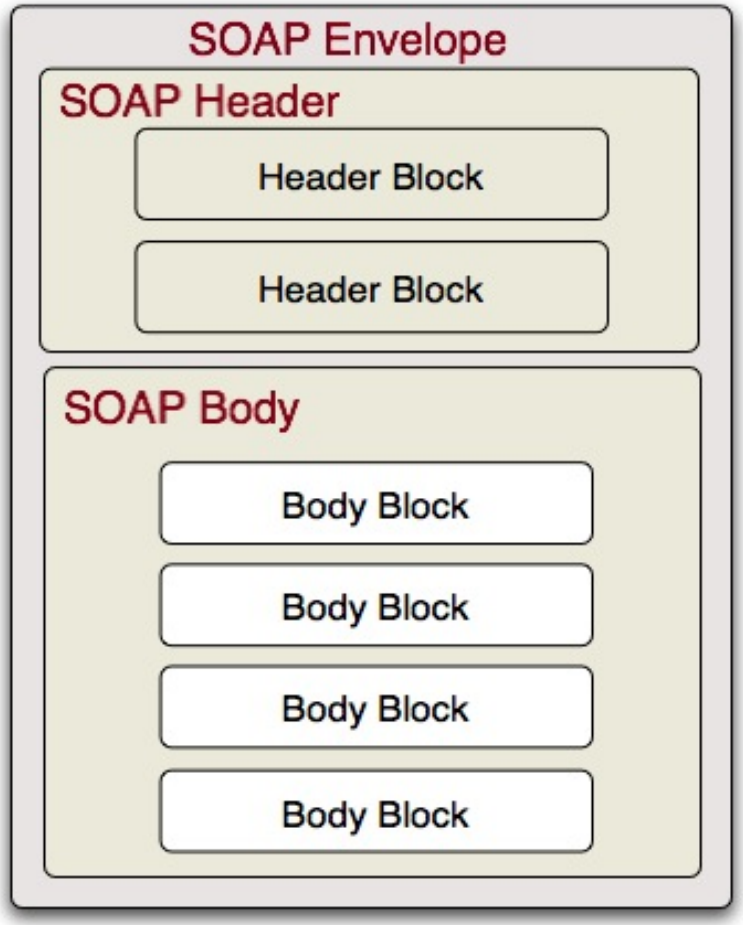


[3]

[3] Developing Enterprise Web Services: An Architect's Guide, S. Chatterjee and J. Webber, Prentice Hall, p72

SOAP

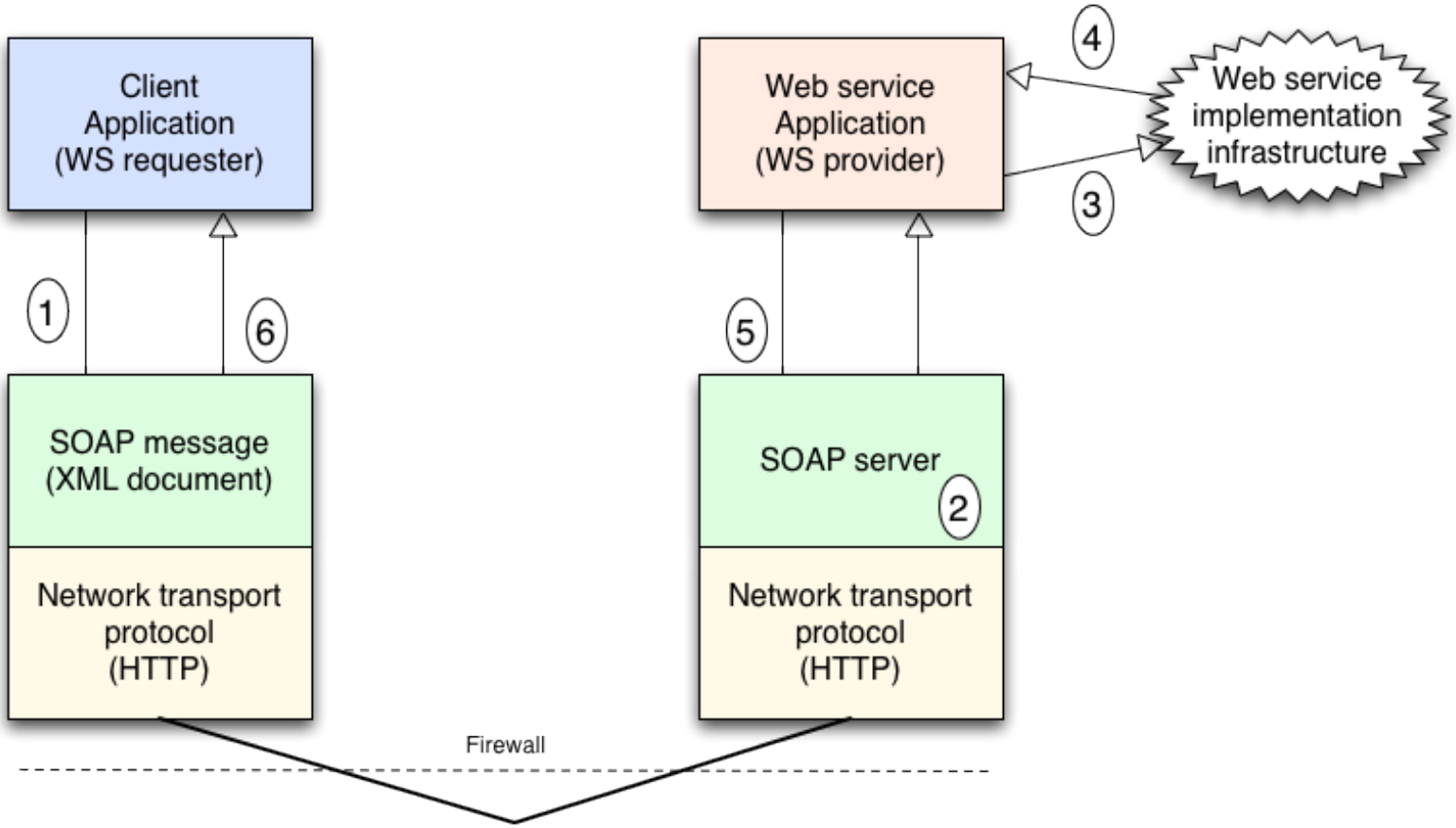
SOAP defines a standard message format for communication, describing how information should be packaged into an XML document. Formerly: abbreviation for Simple Object Access Protocol.




```
<?xml version="1.0"?>
<soap:Envelope>
  <soap:Header>...</soap:Header>
  <soap:Body>...</soap:Body>
</soap:Envelope>
```

- Application payload in the body
- Additional protocol (e.g., security, transaction) messages in the header

Distributed messaging using SOAP[4]



SOAP server: Simply special code that listens for SOAP messages and acts as a distributor and interpreter of SOAP documents

[4] M. Papazoglou. Web Services: Principles and Technology. Prentice Hall, 2007, p.124.

SOAP Encoding

encodingStyle conveys information about how the contents of the particular element are encoded.

- Isn't everything encoded in XML??

XML is expressive and does not constrain the form of document a great deal. The following two are the same by a human reader, but not by a computer program.

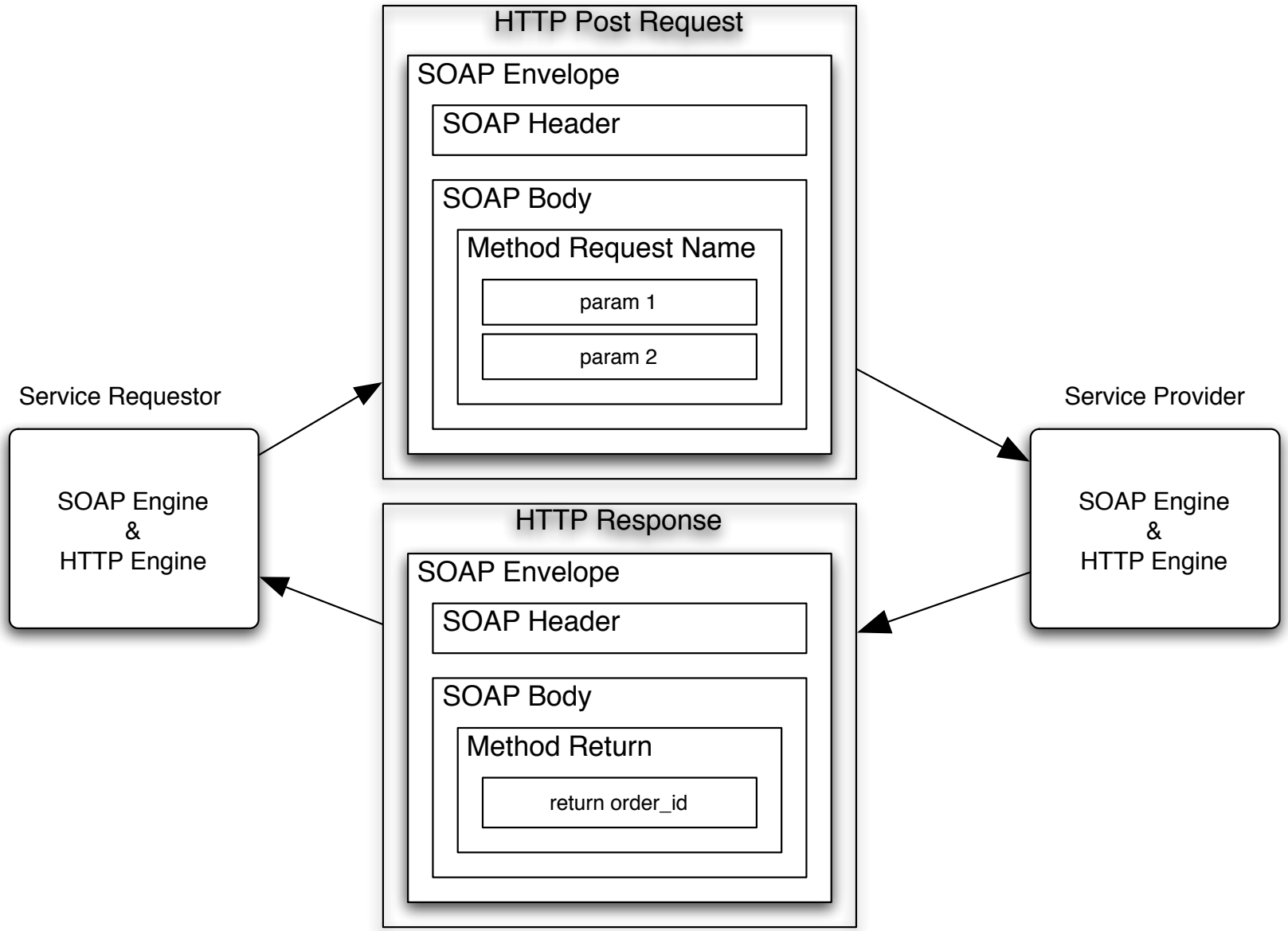
```
<account>
  <balance>
     123.45
  </balance>
</account>
```

```
<account balance="123.45"/>
```

encodingStyle attribute allows the form of the content to be constrained according to some schema shared between the sender and recipient.

What if the sender and recipient want some simple communications, but do not share any schema ...? $\rightarrow$ SOAP Encoding

Binding SOAP with a Transfer Protocol



An example of SOAP Binding over HTTP

Here is a request to a web service for the current price of stock DIS:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

An example of SOAP Binding over HTTP

And here is the response from the service:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
         <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Another example: SOAP Request and Response Message for Google's Web Service Interface (illustration purpose only):
`http://www.w3.org/2004/06/03-google-soap-wsdl.html`

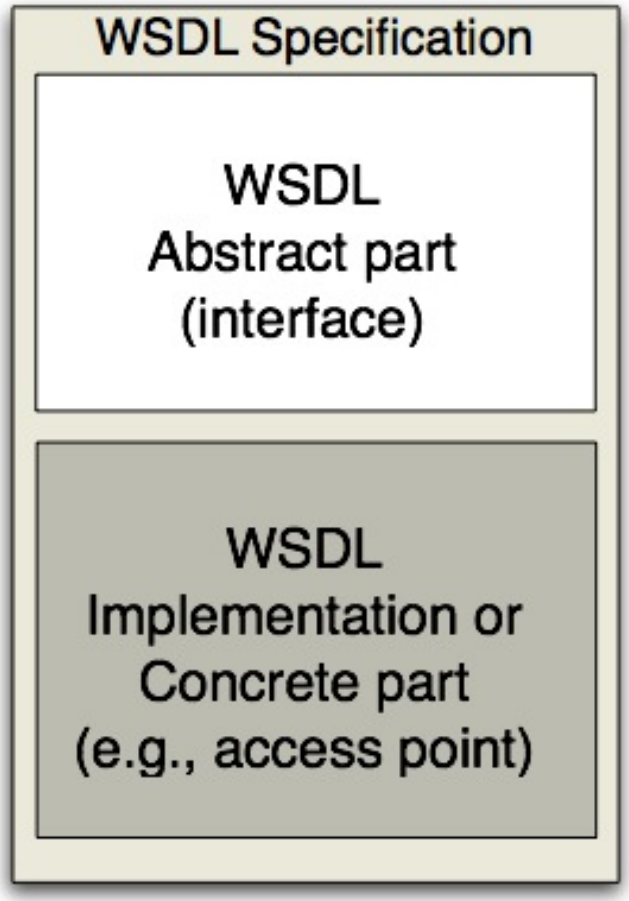
Web Services Description Language

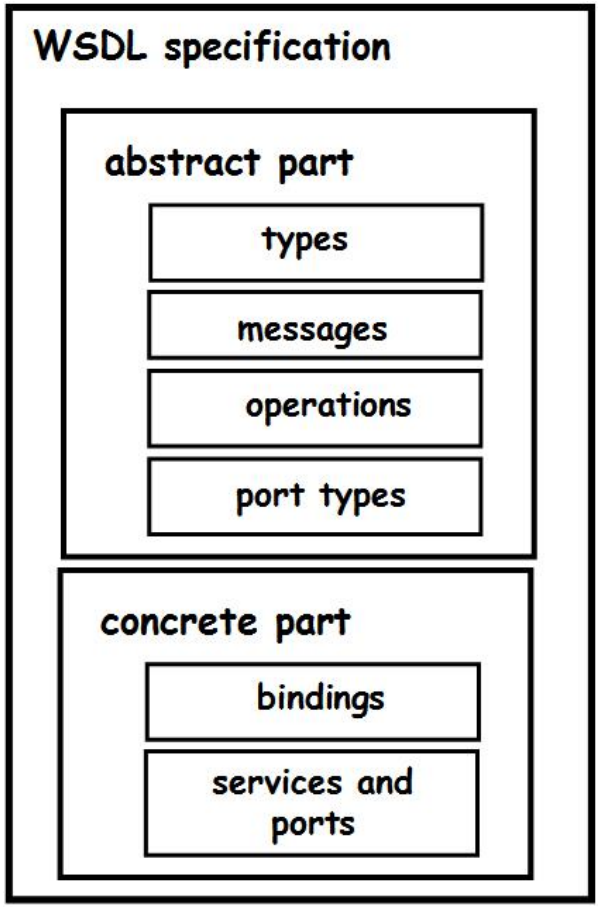
WSDL – pronounced "Whiz Dell"

- Machine-processable specification of the Web service's interface, written in Web Service Description Language (WSDL).
- Specification: `http://www.w3.org/TR/wsdl`
- Describes in an XML syntax a service in terms of the **operations** that make up the service, the **messages** that each operation uses, and the parts from which each message is composed.
- Used by the client to generate a proxy (client stub) to the Web service using some development tool. The proxy then acts as a go-between between the WS and the client.

WSDL tells you which (SOAP) messages you can send where, and which answers to expect.

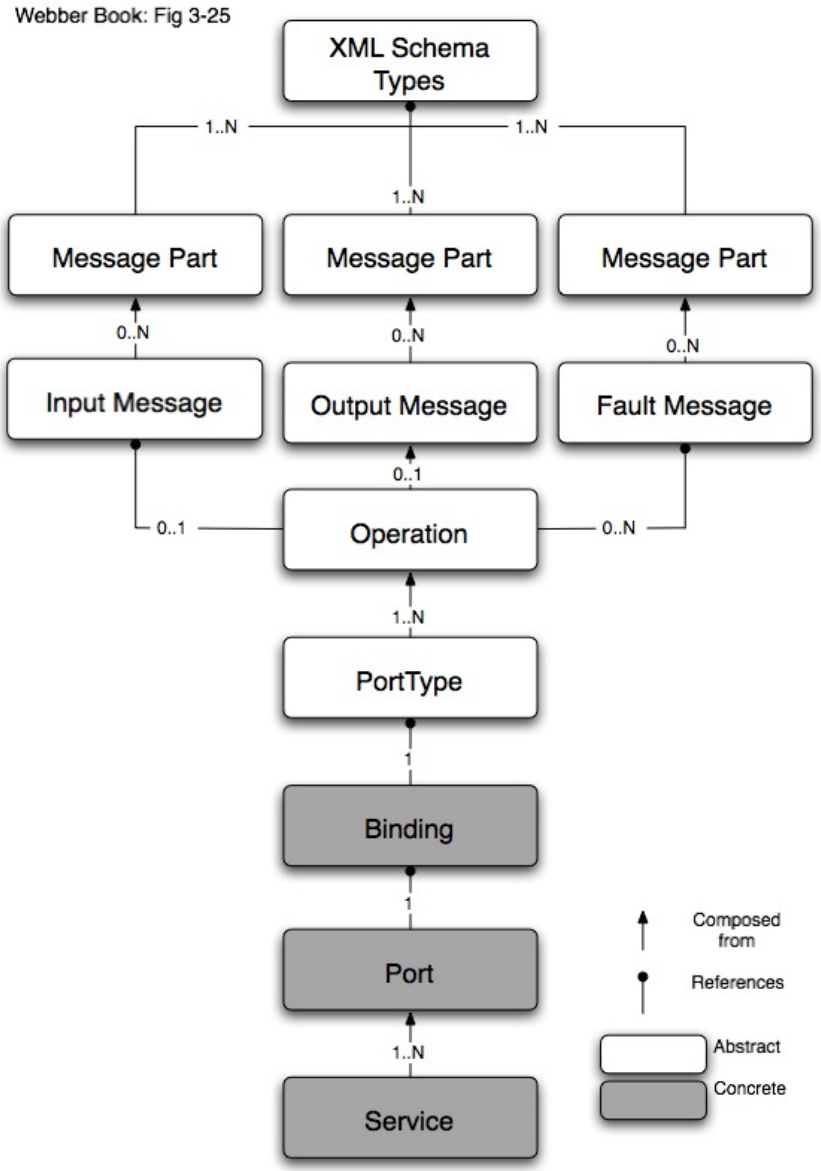
Web Services Description Language: Two parts





The split between abstract and concrete parts → useful for separating Web service design and Web service deployment environment details

WSDL Structure[5]



[5] Developing Enterprise Web Services: An Architect's Guide, S. Chatterjee and J. Webber, Prentice Hall, p.101

WSDL Main Elements: definitions

Say ... we want to describe a Web service that offers one operation:

```
double GetStockQuote(string symbol);
```

We start writing WSDL with ..

```
<wsdl:definitions targetNamespace="http://stock.example.org/wsdl"
                  xmlns:tns="http://stock.example.org/wsdl"
                  xmlns:stockQ="http://stock.example.org/schema"
                  xmlns:wsdl="http://www.w3.org/2003/02/wsdl">
    <!-- child elements -->
</wsdl:definitions>
```

- the parent for all other WSDL elements
- declare global namespaces in use

WSDL Main Elements: types

The types element encloses a number of types used in the interface description (XML Schema types: simple, complex, element . . .).

```
<wsdl:definitions ...>
   <wsdl:import namespace="http://stock.example.org/schema"
         location="http://stock.example/org/schema"/>
   <wsdl:types xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="stock_quote">
        <xs:complexType>
          <xs:sequence>
           <xs:element name="symbol" ref="stockQ:symbol"/>
           <xs:element name="lastPrice" ref="stockQ:price"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <!-- other type/XML elements definitions -->
   </wsdl:types>
</wsdl:definitions>
```

In another document, you can find definitions for symbol and price, for example:

```
<xsd:schema targetNamespace="http://stock.example.org/schema" ...>
    <xsd:element name="symbol" type="xsd:string"/>
    <xsd:element name="price" type="xsd:string"/>
</xsd:schema>
```

WSDL Main Elements: message

The message element declares the form of a message that the Web service sends/receives.

```
<wsdl:message name="StockPriceRequestMessage">
   <wsdl:part name="symbol" element="stockQ:symbol" />
</wsdl:message>
<wsdl:message name="StockPriceResponseMessage">
   <wsdl:part name="price" element="stockQ:stock_quote" />
</wsdl:message>
<wsdl:message name="StockSymbolNotFoundMessage">
   <wsdl:part name="symbol" element="stockQ:symbol" />
</wsdl:message>
```

- Defines the set of message types that can be used to declare input, output and fault messages of operations provided by this Web service
- Each message is constructed from a number of XML Schema-typed part elements.

WSDL Main Elements: portType/interface and operation

The portType element (WSDL 1.1) or interface element (WSDL 2.0) contains a named set of operations. It defines the functionality of the Web service (i.e., what the service does).

```
<wsdl:portType name="StockBrokerQueryPortType">
   <wsdl:operation name="GetStockPrice">
      <wsdl:input message="tns:StockPriceRequestMessage"/>
      <wsdl:output message="tns:StockPriceResponseMessage"/>
      <wsdl:fault name="UnknownSymbolFault"
                  message="tns:StockSymbolNotFoundMessage"/>
   </wsdl:operation>
   <!-- other operations -->
</wsdl:portType>
```

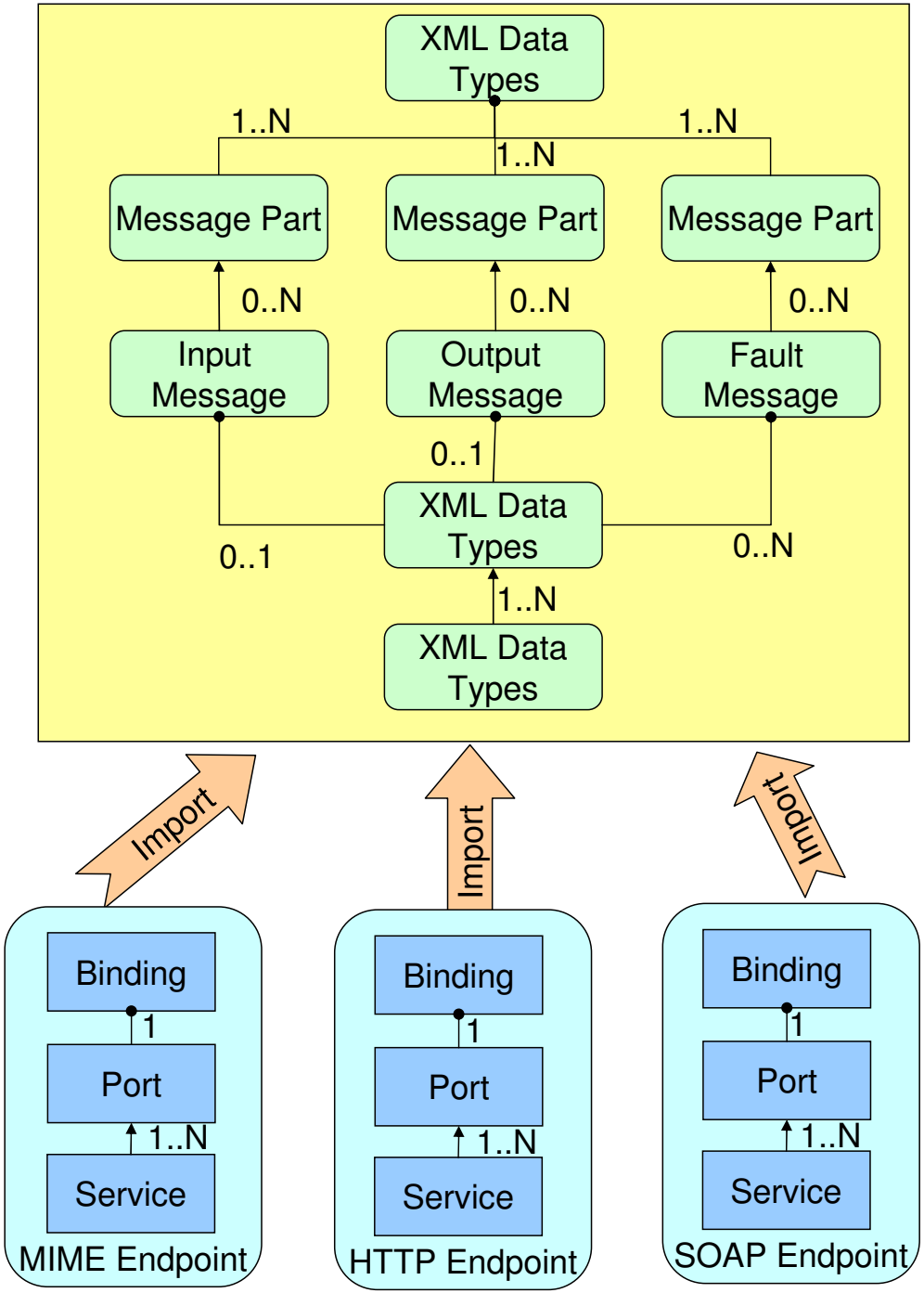
- Each operation combines input, output and fault messages from a set of message types defined previously
- Note: Multiple fault messages can be defined.

WSDL Main Elements: `binding` and `service`

The interface tells us what the service does – but how to call it?

- A `binding` defines *message encoding format* and *protocol details*.
- A `service` element defines one or more `port` elements with specific network endpoints (addresses) for a binding.

```
<wsdl:binding name="StockBrokerServiceSOAPBinding"
                                    type="tns:StockBrokerQueryPortType">
 <soap:binding style="document"
      transport="http://www.w3.org/2002/12/soap/bindings/HTTP/" />
   <wsdl:operation name="GetStockPrice">
     <soap:operation soapAction="http://stock.example.org/getStockPrice" />
     <wsdl:input>
        <soap:body use="literal" encodingStyle="http://stock.example.org/schema"/>
     </wsdl:input>
     <wsdl:output> [...] </wsdl:output>
     <wsdl:fault> [...] </wsdl:fault>
   </wsdl:operation>
</wsdl:binding>
<wsdl:service name="StockBrokerService">
  <wsdl:port name="StockBrokerServiceSOAPPort"
        binding="tns:StockBrokerServiceSOAPBinding>
    <soap:address location="http://stock.example.org/" />
  </wsdl:port>
</wsdl:service>
```



A single abstract definition can be exposed to the network via a number of protocols (having different bindings, offered on different ports, etc)

Part III

RESTful services

REST philosophy

What is REST?

- An *architectural style* of networked systems (not a protocol - not a specification)
- Objective: Expose resources on a networked system (the Web)
- Principles:
  - *Resource Identification* using a URI (Uniform Resource Identifier)
  - *Unified interface* to retrieve, create, delete or update resources
- REST itself is not an official standard or even a recommendation. It is just a *"design guideline"*

Reference: RESTful Web Services, L. Richardson and S. Ruby, O'Reilly.

What is a Resource?

- A **resource** is a thing that:
  - is **unique** (i.e., can be identified uniquely)
  - has at least one **representation**,
  - has one or more **attributes** beyond ID
  - has a potential **schema**, or definition
  - can provide **context**
  - is reachable within the **addressable** universe

Examples of resources

- Web Site
- Resume
- Aircraft
- A song
- A transaction
- An employee
- An application
- A Blog posting
- A printer
- Winning lottery numbers

Origins of REST

- **REST** is an acronym standing for *R*epresentational *S*tate *T*ransfer.
- First introduced by **Roy T. Fielding** in his PhD dissertation "*Architectural Styles and the Design of Network-based Software Architectures*"
- He focused on the rationale behind the design of the modern Web architecture and how it differs from other architectural styles.
  - REST is **client-server** where a representation of the resource is exposed to the client application.
  - The representation of resources places the client application in a **state**. Such state is evolving with each resource representation through traversing hyperlinks

REST principles: Resource Identification & Addressability

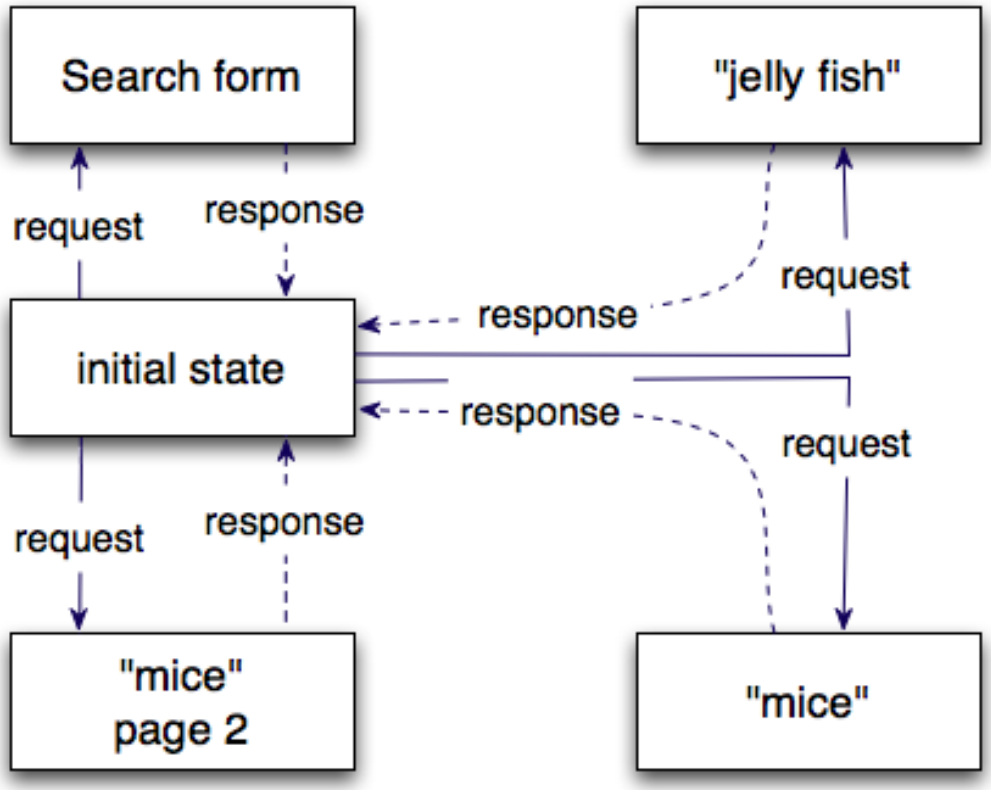
- Resources are identified by a URI (Uniform Resource Identifier)
  - http://www.example.com/software/release/1.0.3.tar.gz
- A resource has to have at least one URI
- Most known URI type are URL and URN
  - URN (Uniform Resource Name)
    - urn:isbn:0-486-27557-4
  - URL (Uniform Resource Locator)
    - file:///home/username/RomeoAndJuliet.pdf
- Every URI designates exactly one resource
- Make systems addressable:

  An application is 'addressable' if it exposes its data set as resources (i.e., usually an infinite number of URIs)

E.g., Google search: `http://www.google.com.au/search?q=unsw`

REST principles: Statelessness

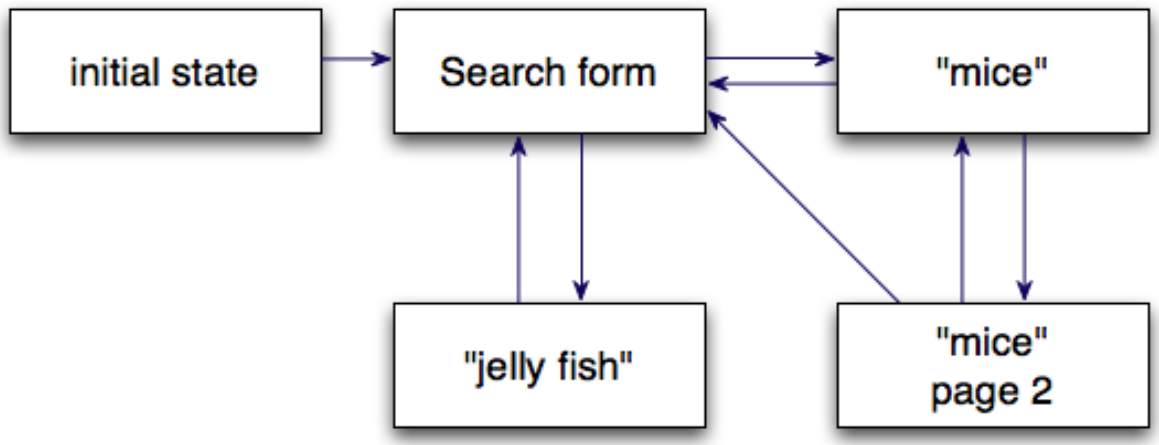
Stateless means every HTTP request happens in a complete isolation. When a client makes a HTTP request, it includes necessary information for the server to fulfil the request



A Stateless Search Engine: Page 88 RESTful Web Services

REST principles: Statelessness

There are many Web sites that expects you to make requests in a certain order: A, B, then C. (i.e., it gets confusing when you make request B a second time instead of moving onto request C)



A Statelful Search Engine: Page 88 RESTful Web Services

REST principles: Statelessness

The REST principle says: URI needs to contain the state within it, not just a key to some state stored on the server

- a RESTful service requires that the state stay on the client side
- client transmit the state to the server for every request that needs it
- server supports the client in navigating the system by sending 'next links' the client can follow
- server does not keep track of the state on behalf of a client
- pros / cons?

REST principles: Statelessness

Client / Application State vs. Resource State

What counts as 'state' exactly then? Think Flickr.com – would statelessness mean that I have to send every one of my pictures along with every request to flickr.com?

- Two *kinds* of states
  - – photos are resources
  - – a client 'application' accesses those resources
- Resource state live on the server (i.e., photos are stored on the server)
- Client application state should be kept off the server
- Statelessness in REST applies to the client application state
- *Resource state is the same for every client and its state is managed on the server.*

REST principles: Resource Representations

A resource needs a representation for it to be sent to the client

- a representation of a resource - some data from the 'current state' of a resource (i.e., a list of open bugs)
- a list of open bugs – either in XML, a web page, comma-separated-values, printer-friendly-format, or ...
- a representation of a resource may also contain metadata about the resource (e.g. for books: the book itself + metadata such as cover-image, reviews, stock-level, etc.)

Representation can flow the other way too: a client sends a 'representation' of a new resource and the server creates the resource.

REST principles: Resource Representations

Offering / deciding between multiple representations

Option 1.

- Have a distinct URI for each representation of a resource:
  - http://www.example.com/release/104.en (English)
  - http://www.example.com/release/104.es (Spanish)
  - http://www.example.com/release/104.fr (French)

This also meets the addressability and statelessness principles in that the client has all the info necessary for the server to fulfill the request.

REST principles: Resource Representations

Offering / deciding between multiple representations

Option 2.

- Use HTTP HEAD (metadata) for content negotiation:
  - expose, e.g., http://www.example.com/release/104
  - client HTTP request contains `Accept-Language`

Other types of request metadata can be set to indicate all kinds of *client preferences*, e.g., file format, payment information, authentication credentials, caching directives, and so on.

Option 1 or 2 are both acceptable REST-based solutions ... but the addressable URI option is preferred. E.g., say you want to use a HTML page translation service which accepts URIs ... which representation offer will give you the most flexibility?

REST principles: Links and Connectedness

In REST, resource representations are hypermedia: resources (data itself) + links to other resources

e.g., Google Search representation:

REST principles: Links and Connectedness

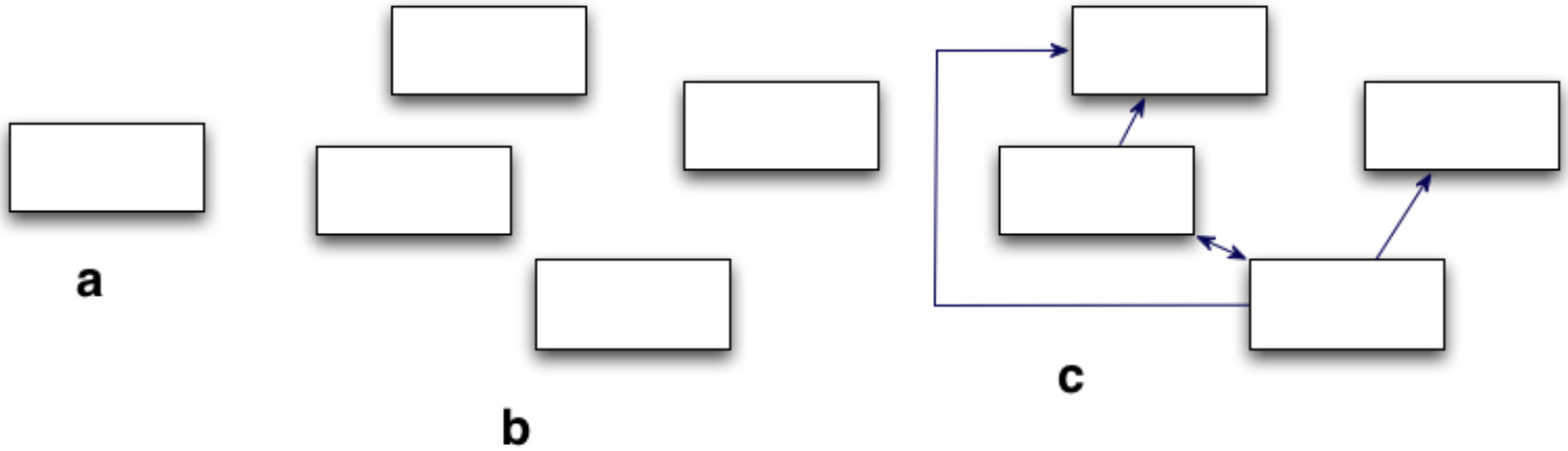
R. Fielding about "Hypermedia as the engine of application state"

The current state of an HTTP 'session' is not stored on the server as a resource state, but tracked by the client as an application state, and created by the path the client takes through the Web. The server guides the client's path by serving hypermedia: links and forms inside resource representations.

The server sends the client guidelines about *which states are near to current one*.

REST principles: Links and Connectedness

Connectedness in REST



- All three services expose the same functionality, but their usability increases towards the right
- Service A is typical RPC-style, exposing everything through a single URI. Neither addressable, nor connected.
- Service B is addressable but not connected: there is no indication of the relationships between resources. A hybrid style ...
- Service C is addressable and connected: resources are linked to each other in ways that make sense. A fully RESTful service.

REST principles: Uniform Interface

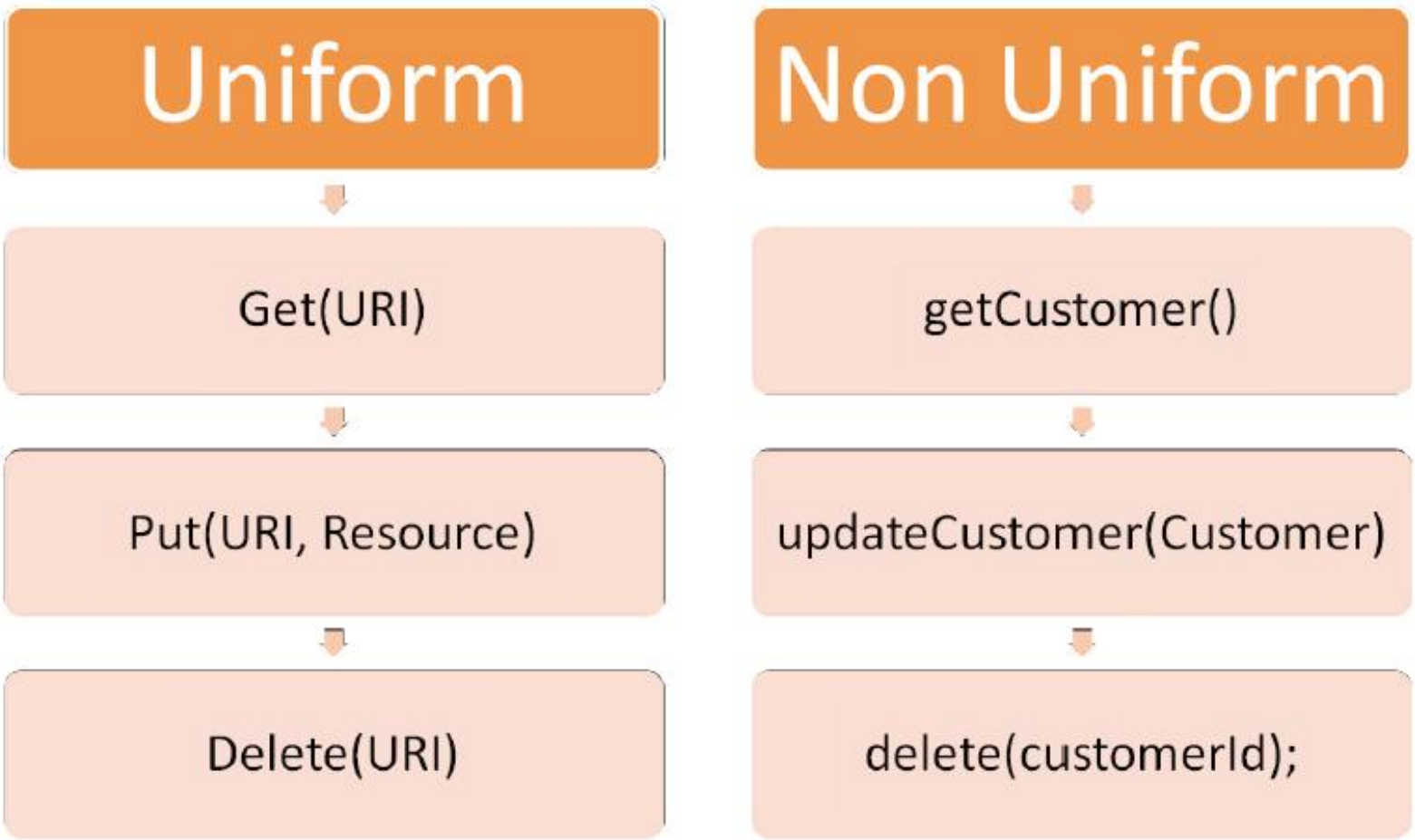
REST Uniform Interface Principle uses 4 HTTP methods on resources

- **PUT**: Update a resource (existing URI).
- **GET**: Retrieve a representation of a resource.
- **POST***: Create a new resource or modify the state of a resource. POST is a read-write operation and may change the state of the resource and provoke side effects on the server. Web browsers warn you when refreshing a page generated with POST.
- **DELETE**: Clear a resource, afterwards the URI is no longer valid
- HEAD and OPTIONS

Similar to the CRUD (Create, Read, Update, Delete) databases operations

*POST: a debate about its exact best-practice usage... Some people claim 'create' should be done via PUT.

REST principles: Uniform Interface



REST principles: Safety and Idempotence

REST Uniform Interface, if properly followed, gives you two properties:

being **safe**

Read-only operations ... The operations on a resource do not change any server state. The client can call the operations 10 times, it has no effect on the server state. (Server state may change during the 10 calls though, for other reasons.)

being **idempotent**

Operations that have the same "effect" whether you apply them once or more than once. An effect here may well be a change of server state. An operation on a resource is idempotent if making one request is the same as making a series of identical requests.

REST principles: Safety and Idempotence

In math terms:

- multiplying a number by zero is idempotent: 4x0x0x0x0 is the same as 4x0
- multiplying a number by one is both safe and idempotent: 4x1x1x1x1 is the same as 4x1 (idempotence) but also it does not change 4 (safe)

- GET, HEAD, OPTIONS: safe (and thus idempotent)
- PUT: idempotent
  - If you create a resource with PUT, and then resend the PUT request, the resource is still there and it has the same properties you gave it when you created it.
  - If you update a resource with PUT, you can resend the PUT request and the resource state won't change again
- DELETE: idempotent
  - If you delete a resource with DELETE, it's gone. You send DELETE again, it is still gone !
- POST: neither

REST principles: Uniform Interface

Why Safety and Idempotence matter

- The two properties let a client make reliable HTTP requests over an unreliable network.
- Your GET request gets no response? make another one. It's safe.
- Your PUT request gets no response, make another one. Even if your earlier one got through, your second PUT will have no side-effect.

There are many applications that misuse the HTTP uniform interface. e.g.,

- GET https://api.del.icio.us/posts/delete
- GET www.example.com/registration?new=true&name=aaa&ph=123

Many applications expose unsafe operations as GET, and there are many uses of the POST method which are neither safe nor idempotent. Repeating them has consequences ...

REST principles: Uniform Interface

Why Uniform Interface matters

Uniformity

The point about REST Uniform Interface is in the 'uniformity': that every service uses HTTP's interface the *same* way.

- It means, for example, GET does mean 'read-only' across the Web no matter which resource you are using it on.
- It means, we do not use methods in place of GET like doSearch or getPage or nextNumber.
- But, it is not just using GET in your service, it is about using it the way it was meant to be used.

REST example 1: POST request

Starbucks example for REST API.
Based on: ICSOC 2008 Summer School "Get Connected": REST-based Services, J. Webber and H. Skogsrud

"I would like a latte, please"

```
POST /order HTTP 1.1
Host: starbucks.example.org
Content-Type: application/xml
Content-Length: ...

<order xmlns="urn:starbucks">
 <drink>latte</drink>
</order>
```

REST example 2: POST response

```
201 Created
Location: http://starbucks.example.org/order?1234
Content-Type: application/xml
Content-Length: ...

<order xmlns="urn:starbucks">
<drink>latte</drink>
<link rel="payment"
  href="https://starbucks.example.org/payment/order?1234"
  type="application/xml"/>
</order>
```

REST example 3: OPTIONS

"I made a mistake, can I still change my order?"

Request:

```
OPTIONS /order?1234 HTTP 1.1
Host: starbucks.example.org
```

Response:

```
200 OK
Allow: GET, PUT
```

"Yes, for now!"

REST example 4: PUT Request

```
PUT /order?1234 HTTP 1.1
Host: starbucks.example.org ...

<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
  <link rel="payment"
    href="https://starbucks.example.org/payment/order?1234"
    type="application/xml"/>
</order>
```

REST example 5: PUT Response

```
200 OK
Location: http://starbucks.example.org/order?1234 ...

<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
  <link rel="payment"
    href="https://starbucks.example.org/payment/order?1234"
    type="application/xml"/>
</order>
```

REST example 6: OPTIONS 2

"How do I pay?"

Request:

```
OPTIONS /payment/order?1234 HTTP 1.1
Host: starbucks.example.org
```

Response:

```
200 OK
Allow: GET, PUT
```

REST example 7: PUT 2

Request:

```
PUT /payment/order?1234 HTTP 1.1 ...
<payment xmlns="urn:starbucks">
  <cardNo>123456789</cardNo>
  <expires>07/07</expires>
  <name>John Citizen</name>
  <amount>4.00</amount>
</payment>
```

Response:

```
201 Created
Location: https://starbucks.example.org/payment/order?1234 ..
<payment xmlns="urn:starbucks"> ... </payment>
```

REST example 8: GET

Request:

```
GET /order?1234 HTTP 1.1
Host: starbucks.example.org ...
```

Response:

```
200 OK
Content-Type: application/xml ...
<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
</order>
```

(I paid, so there is no link to outstanding payment anymore)

REST example 9: done.

Source: http://images.businessweek.com/ss/06/07/top_brands/image/starbucks.jpg

REST example: lessons learned

- HTTP has a header/status combination for most occasions
- APIs are expressed in terms of links
- XML is fine, but we could also use formats like Atom, JSON, or even XHTML
- Form encoding still works – "application/x-www-form-urlencoded" media content type
- State machines can be implemented through links
- BUT: no interface definition!

Positive aspects of REST

- Faster response time (caching)
- Reduced server load (caching)
- Improved server scalability and easier load-balancing (no session state)
- Client software can be reused (uniform interface)
- Can be implemented with any server-side technology
- HTTP client libraries are widespread
- Easy to serve different types of content such as images, videos, office docs, etc.
- The "Web" in "Web services" is for real!

Negative aspects of REST

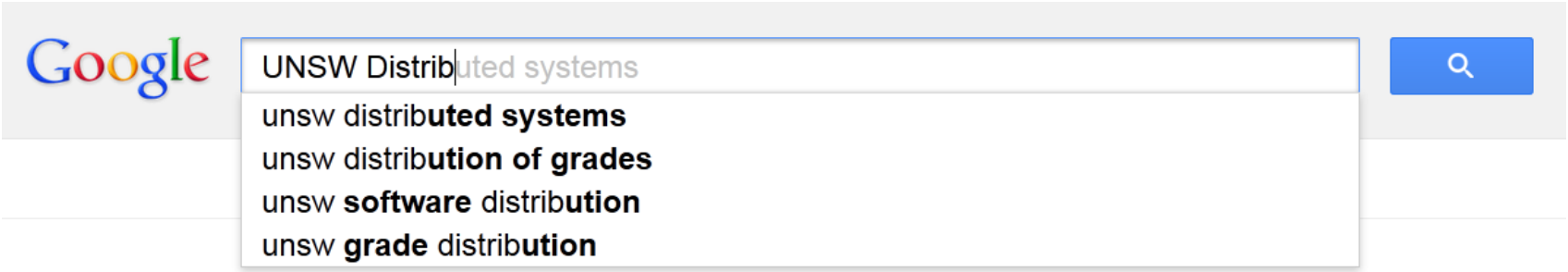
- No accepted interface description language like WSDL
  - proposals include using WSDL 2.0 (`http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/`), but breaks the spirit of REST
  - usually need to "read" the doc to understand how to interact ...
  - e.g., typical example `http://developers.facebook.com`
- Less tooling support for automation than SOAP/WSDL
- Most web browsers don't know PUT and DELETE

When can I use REST?

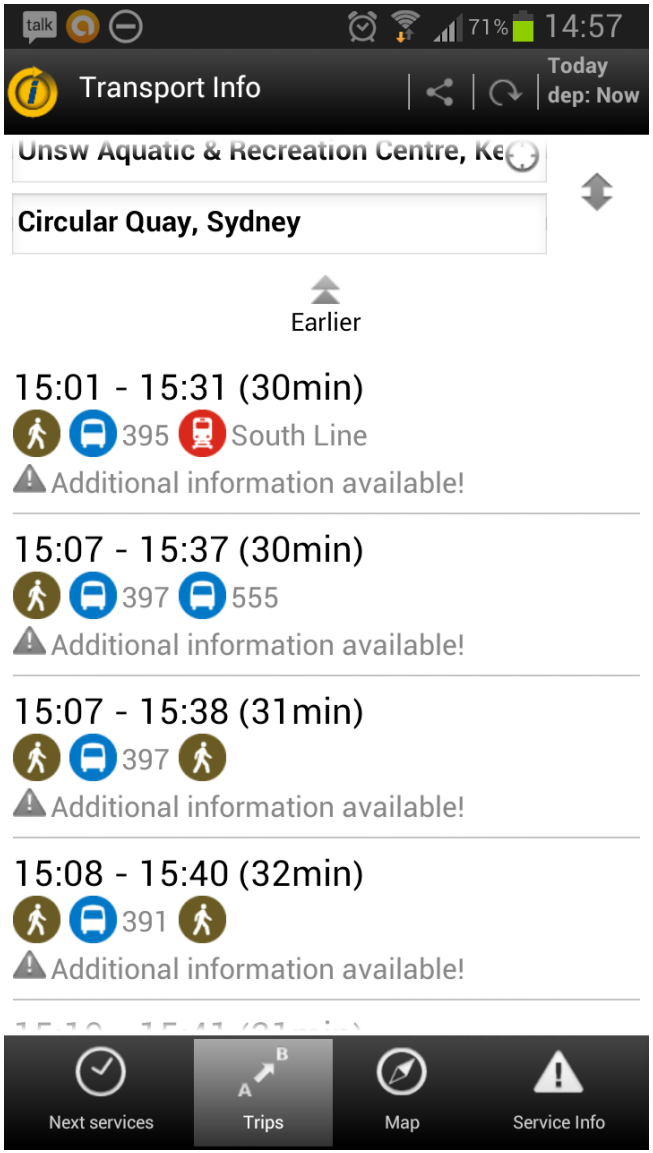
- For Web Services
  - build your web service using the REST style
  - alternative to some of WS-*, not a replacement for WS-*
- Build clients interfacing to public REST APIs
  - e.g. Amazon S3 REST API, Google Data APIs, Facebook
  - Many other public APIs have a REST-like interface
- For Rich Internet Applications (RIAs)?
  - client sends AJAX requests to a REST interface using a JavaScript library e.g. jQuery, or a framework like JavaFX or Silverlight
  - response (JSON, XML etc) is displayed on the client

Uses of WS in RIA/AJAX 1

- RIA: Rich Internet Applications
- AJAX: Asynchronous JavaScript and XML (often using JSON instead of XML)



Uses of WS in RIA/AJAX 2



REST vs. SOAP/WSDL - Technology

REST

- User-driven interaction via rich Web client application
- Few operations on many resources, not always a "natural" fit
- URI - consistent identification of resources
- No interface definition, textual documentation & 'browsing'
- Focus on scalability, performance, simplicity
- Composition = mashups (usually)

SOAP/WSDL

- Orchestrated, reliable event/message flows
- Many operations (WSDL interface), no explicit resources
- Lack of standard identification mechanism
- WSDL interfaces: can code against them without invoking the service
- Focus on design of integrated enterprise applications
- Composition = business processes

REST vs. SOAP/WSDL - Protocol

- XML in – XML out (with POST)
- URI in – XML out (with GET)

- Self-Describing" XML

- HTTP only

- HTTP/SSL is enough – no need for more standards
- HTTP is an application protocol
- Synchronous

- Do-it-yourself when it comes to "reliable message delivery", "distributed transactions"

- SOAP in – SOAP out (with POST)

- Strong Typing (XML Schema)

- "Transport independent"
- Heterogeneity in QoS needs.
- Different protocols may be used
- HTTP as a transport protocol
- Synchronous and Asynchronous

- Foundation for the whole WS* advanced protocol stack

REST vs. SOAP/WSDL Design Methodology

- Identify resources to be exposed as services (e.g., book catalog, purchase order)
- Define "nice" URLs to address them
- Distinguish read-only and side-effects free resources (GET) from modifiable resources (POST, PUT, DELETE)
- Relationships (e.g., containment, reference) between resources correspond to hyperlinks that can be followed to get more details
- Implement and deploy on Web server
- List what are the service operations (the "verbs") into the service's WSDL document
- Define a data model for the content of the messages exchanged by the service (XML Schema data types)
- Choose an appropriate transport protocol to bind the operation messages and define the corresponding QoS, security, transactional policies
- Implement and deploy on the Web service container (note the corresponding SOAP engine end-point)

Part IV

Business Process Management & Web Service Composition

Business Process Management

What is a **business process**? → According to Weske[6]:

> *A business process consists of a* **set of activities** *that are performed in coordination in an organizational and technical environment. These activities jointly realize a* **business goal**. *Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations.*

e.g., travel booking, tax return, procurement, expense reimbursement, hiring an employee, applying for a job, ...

In a business process, there is often a sense of "state" and changing state.

[6] M. Weske, Business Process Management: Concepts, Languages, Architectures, Springer, 2007.

Business Process Management

Business Process Management (BPM, Weske):

> *Business process management includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes.*

- Value-adding management of a business process requires streamlining the collection of activities within the process.
- The activities in a single process are often performed by various business units in an organisation.

Business Process Modelling

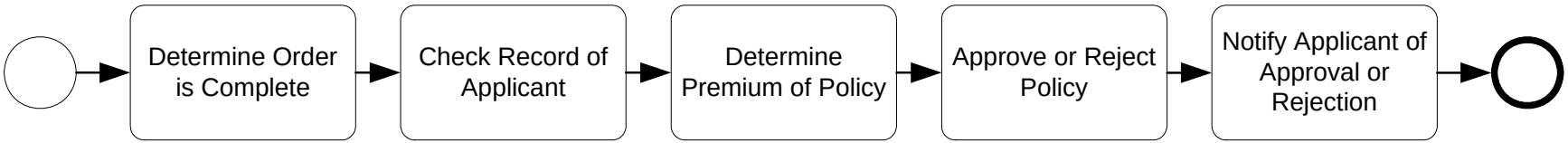
Business Process (BP) Modelling (Weske):

> *A business process model consists of a* **set of activity models** *and* **execution constraints** *between them. A business process* **instance** *represents a concrete case in the operational business of a company, consisting of activity instances. Each business process model acts as a* **blueprint** *for a set of business process instances, and each activity model acts as a blueprint for a set of activity instances.*

(*model vs. instances* is basically like *class vs. objects*)
BP modelling means creating a process model, i.e. describing the dynamic behaviour of organisations, businesses.
Purposes of process modeling include: Training and communication, simulation and analysis, costing and budgeting, documentation, knowledge management, and quality, system development, organizational design, management information, and **enactment**, i.e., execution of the models.

Business Processing Modelling: basic terminology

BPMN example: insurance application processing



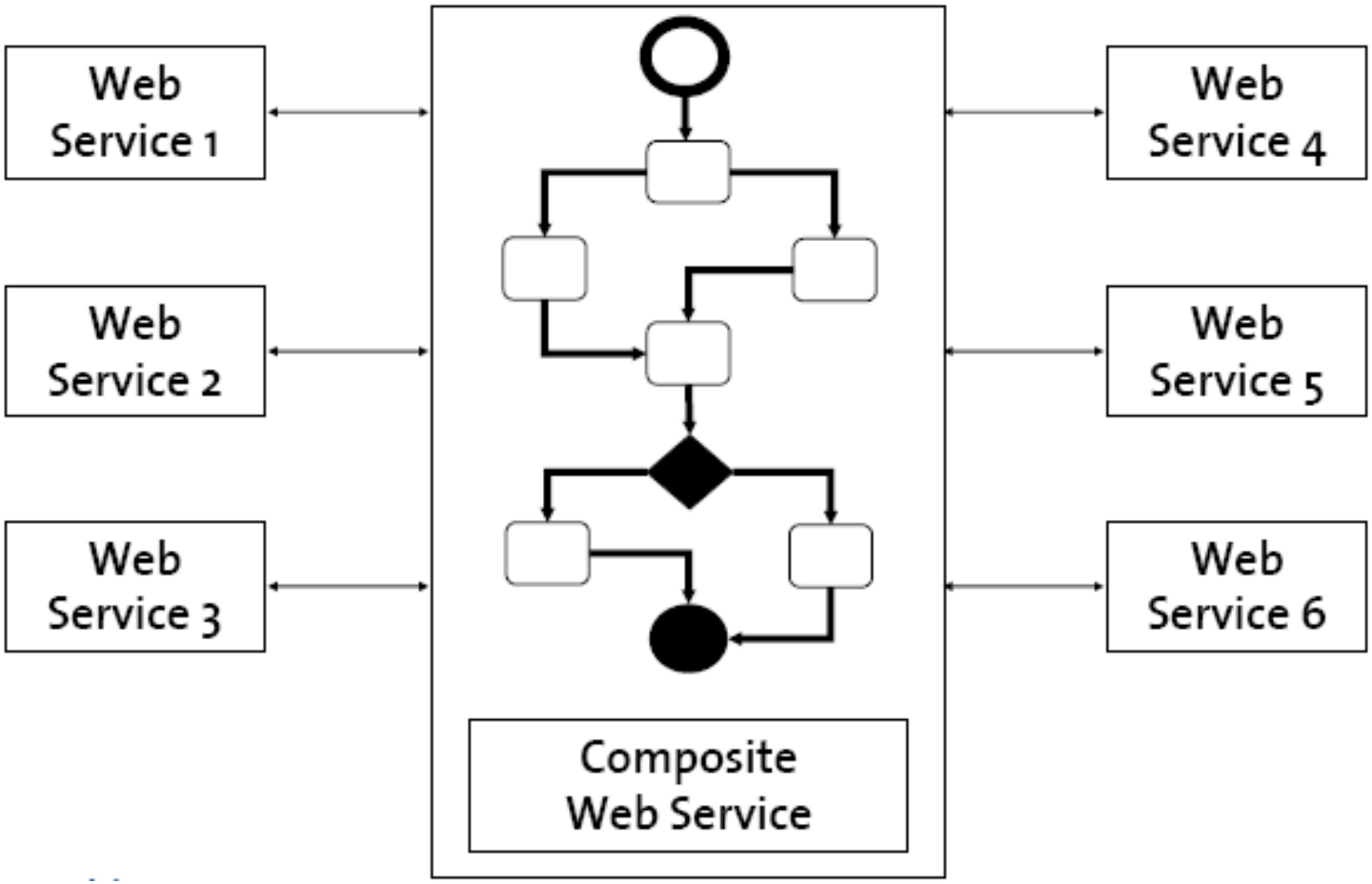
- **Tasks**: an activity that cannot be subdivided (unit of work)
- **Sequence**: some tasks need to be performed in a strict order
- **Choice**: certain tasks do not always need to be carried out
- **Parallel:** some tasks can be performed in parallel
- **Synchronisation:** some tasks need to wait for the result of previous tasks
- **Iteration:** some tasks need to be repeated

So far through the Web services topics ...

- We looked at Web services as a standard way of invoking remote components (SOAP, WSDL, REST)
- Application integration requires more than the ability to conduct simple interactions via invoking remote operations.
- Application integration requires coordinating complex interactions among the applications involved $\rightarrow$ such coordination represents the application integration logic.
- In modern Enterprise Application Integration (EAI) platforms, workflow engines and business process modelling have been used as technology for expressing and executing the application integration logic.
- For Web services to be a complete EAI platform, it needs a standard model that can express/execute the complex interactions among the Web services.

Web Services and Business Processes

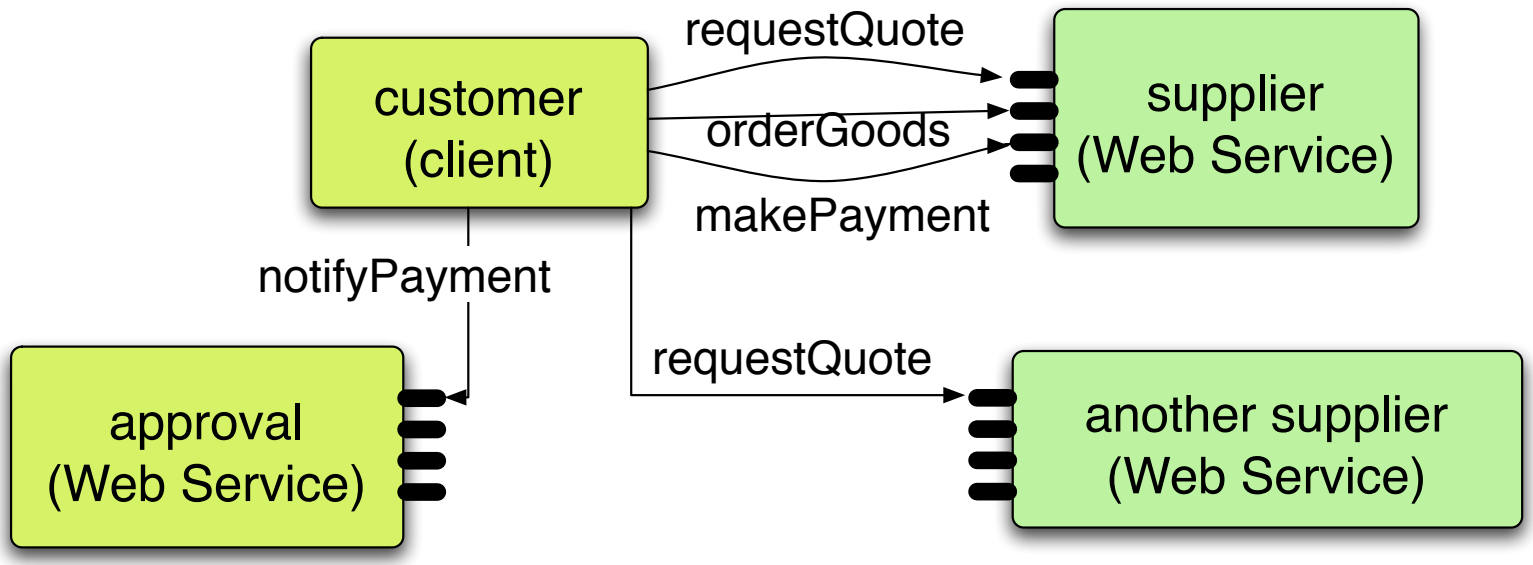
- Workflow models define how different Web services are composed.
- Workflows describe conversation rules and protocols between Web services.



Web Service Composition

e.g., a procurement scenario: a client...

- sends requestQuotes to two different suppliers.
- chooses the supplier that offers a better deal.
- obtains an approval from its finance department
- sends an order and then makes the payment



Web Service Composition

Important observations:

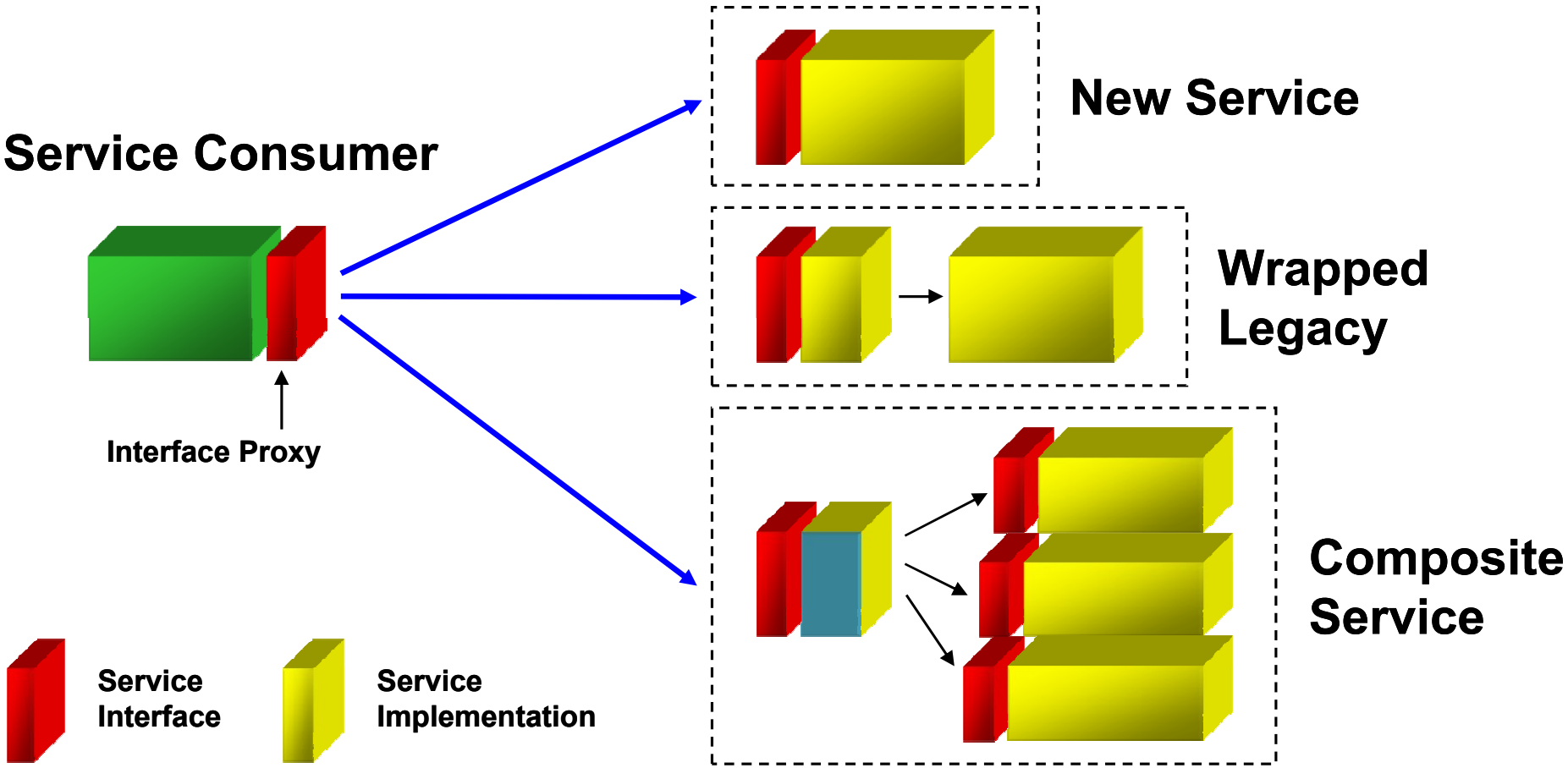
- The suppliers are Web services, as is the internal application that performs approval.
- Hence, the business logic of the client is realised by 'composing' multiple services (i.e., coordinating the interaction among multiple services)
- This is the core concept of Web service composition: *Write a program by calling on other services* (i.e., a new application implemented by integrating other applications)

Web Service Composition

Web Service composition can be iterated;

- Consider Web Services as building blocks that can be assembled.
- It allows building of a complex applications by progressively aggregating components.
- This allows to maintain higher levels of abstraction.
  - You do not necessarily know the "inside" of each component

Web Service Composition



WS Composition Styles: Orchestration vs. Choreography

- **Orchestration** describes how Web services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of a single endpoint (single party).

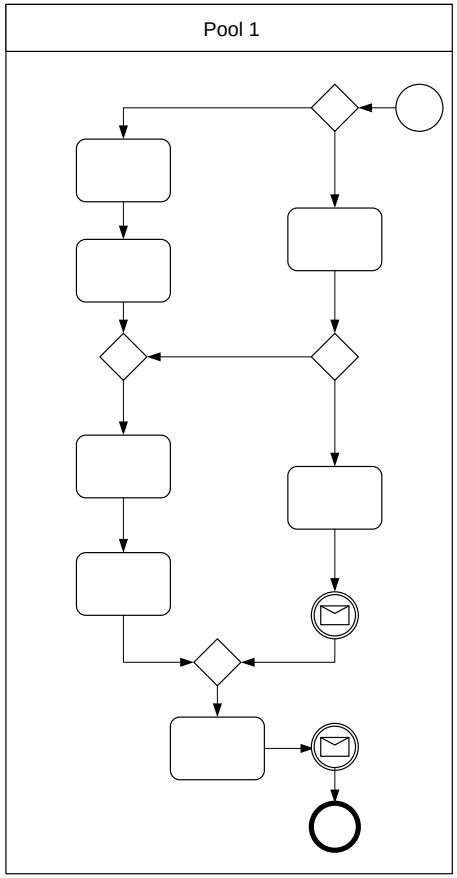
- **Choreography** is associated with the public (globally visible) message exchanges, rules of interaction and agreements that occur between multiple business process endpoints.

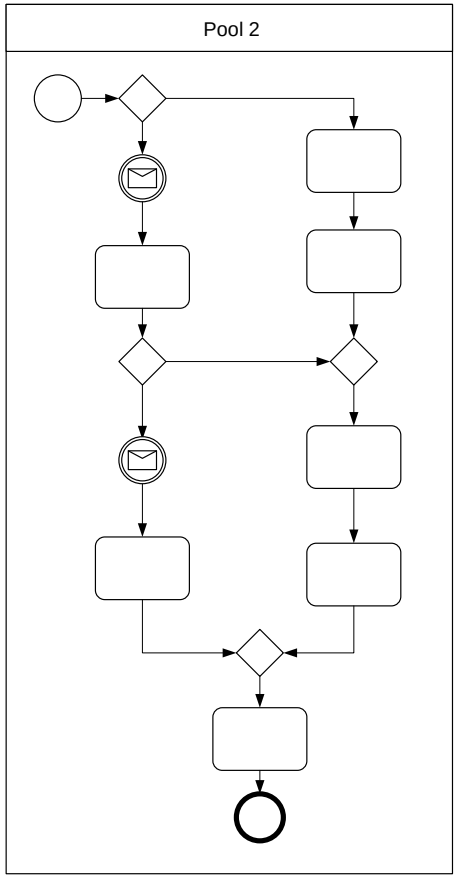
- **Choreography** tracks the sequence of messages that may involve multiple parties and multiple sources, and described from the perspectives of all parties (common view).

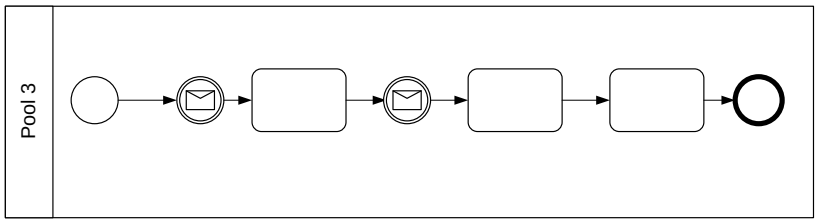
WS Choreography vs. Orchestration

Private process

- Internal details
- (3 separate processes)
- Synonyms: private view (on some process)



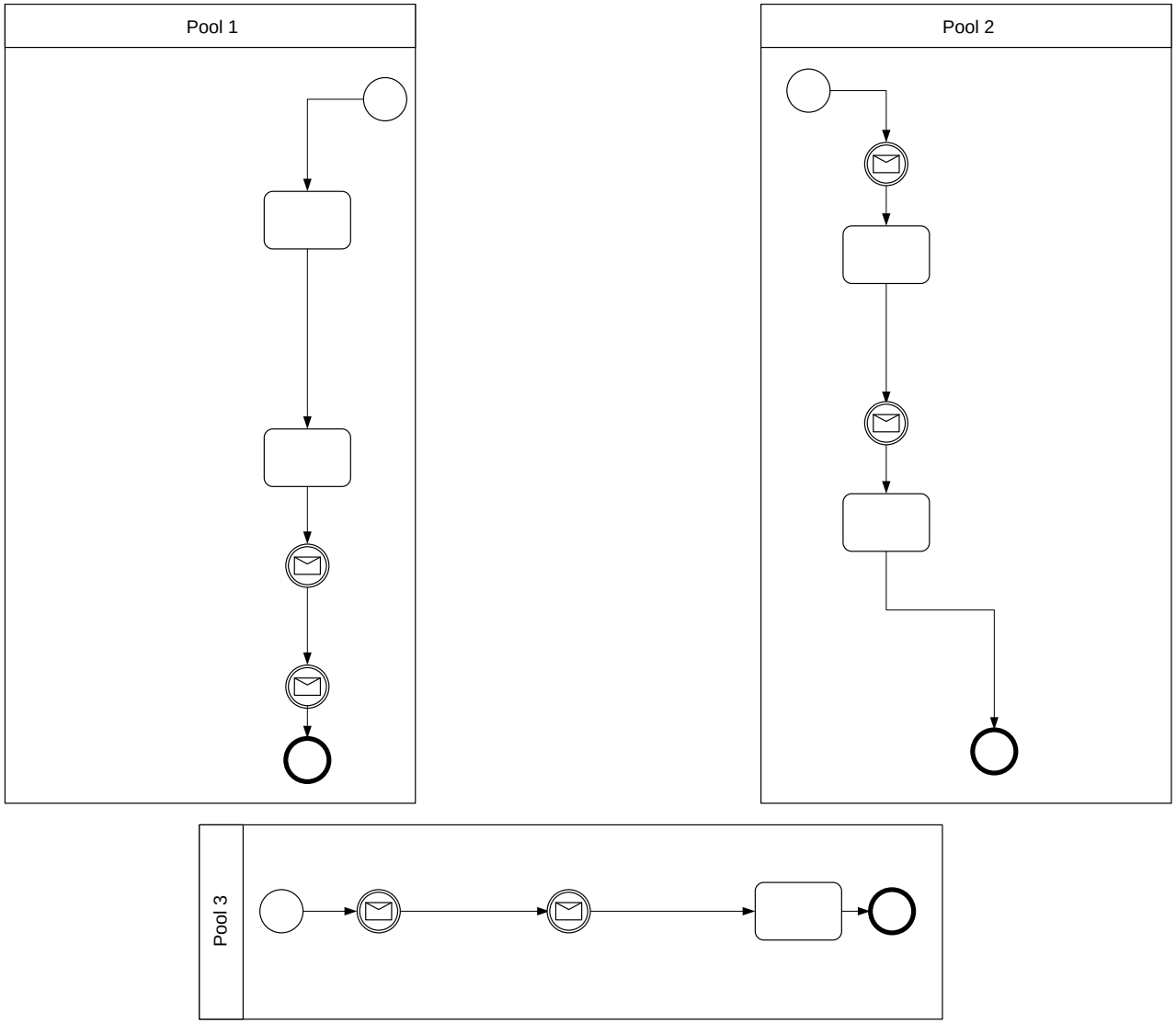




WS Choreography vs. Orchestration

Business Protocol

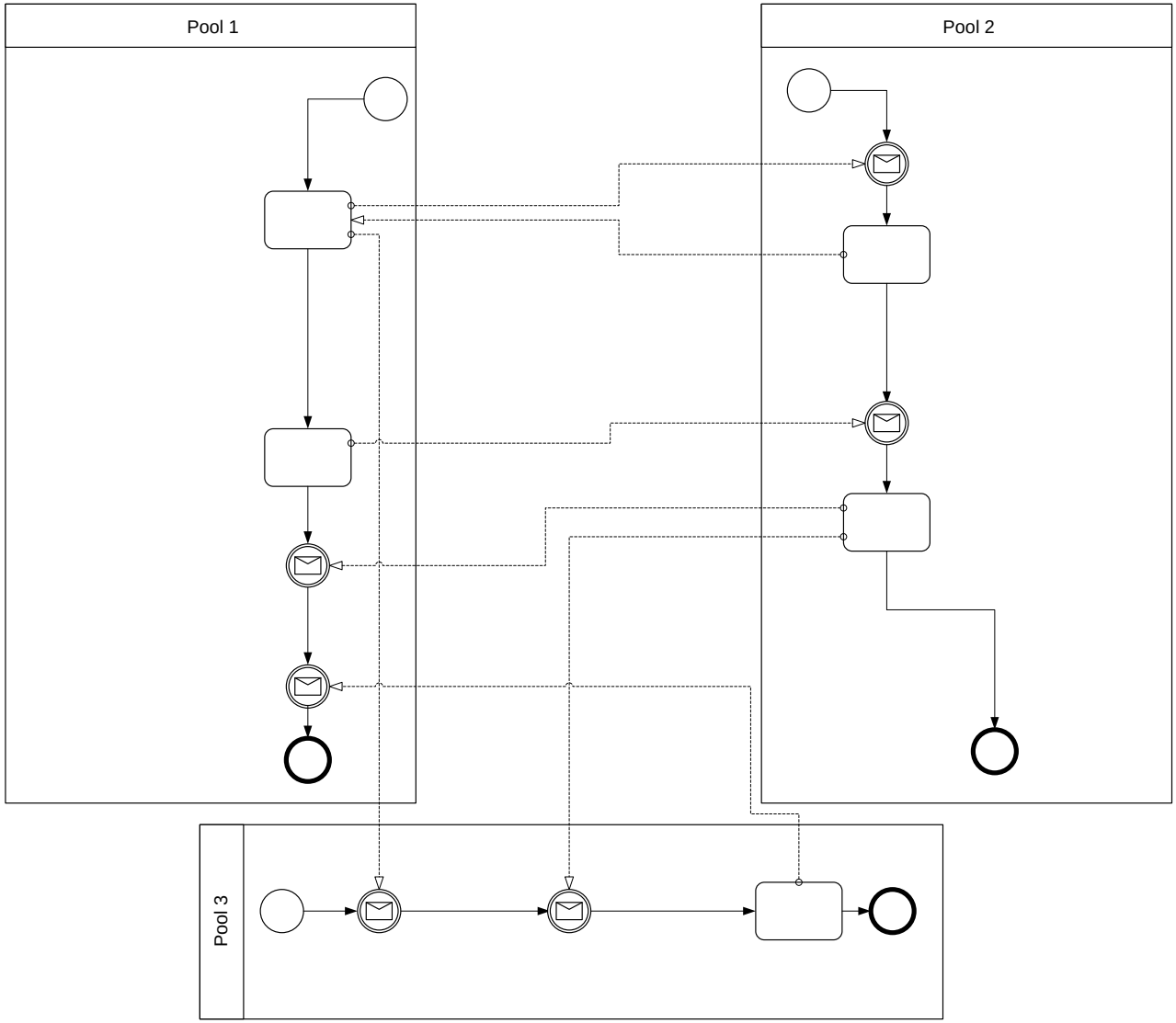
- No internal details
- Just external interface
- Synonyms: protocol, behavioural interface, public view (on some process)
- E.g., *Abstract BPEL*



WS Choreography vs. Orchestration

Choreography

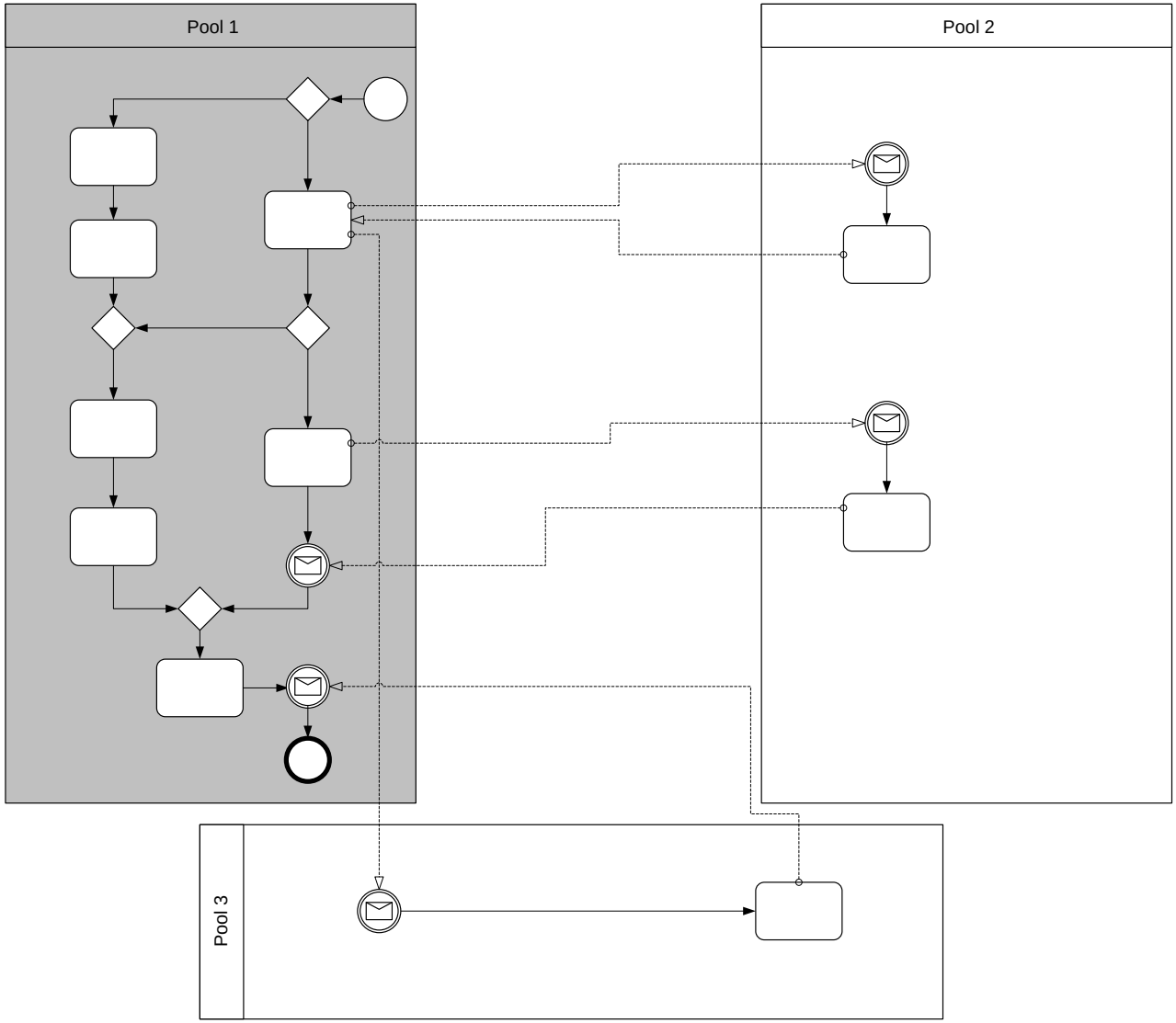
- Global view on protocols + data exchange
- All parties of equal importance



WS Choreography vs. Orchestration

Orchestration (1)

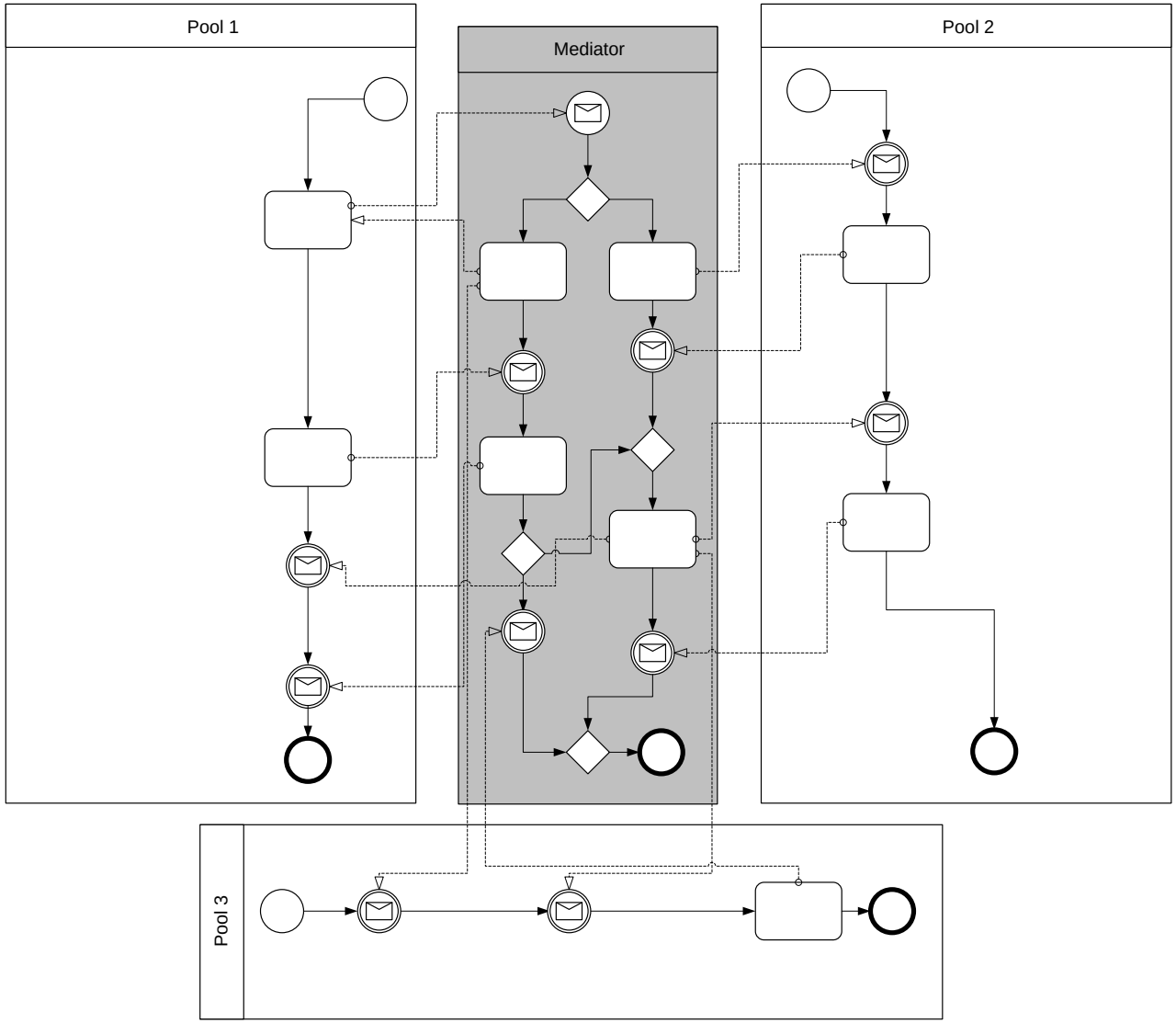
- Like private process, but with connections to other processes
- From the viewpoint of one partner
- Focus on Web services
- E.g., *Executable BPEL*



WS Choreography vs. Orchestration

Orchestration (2)

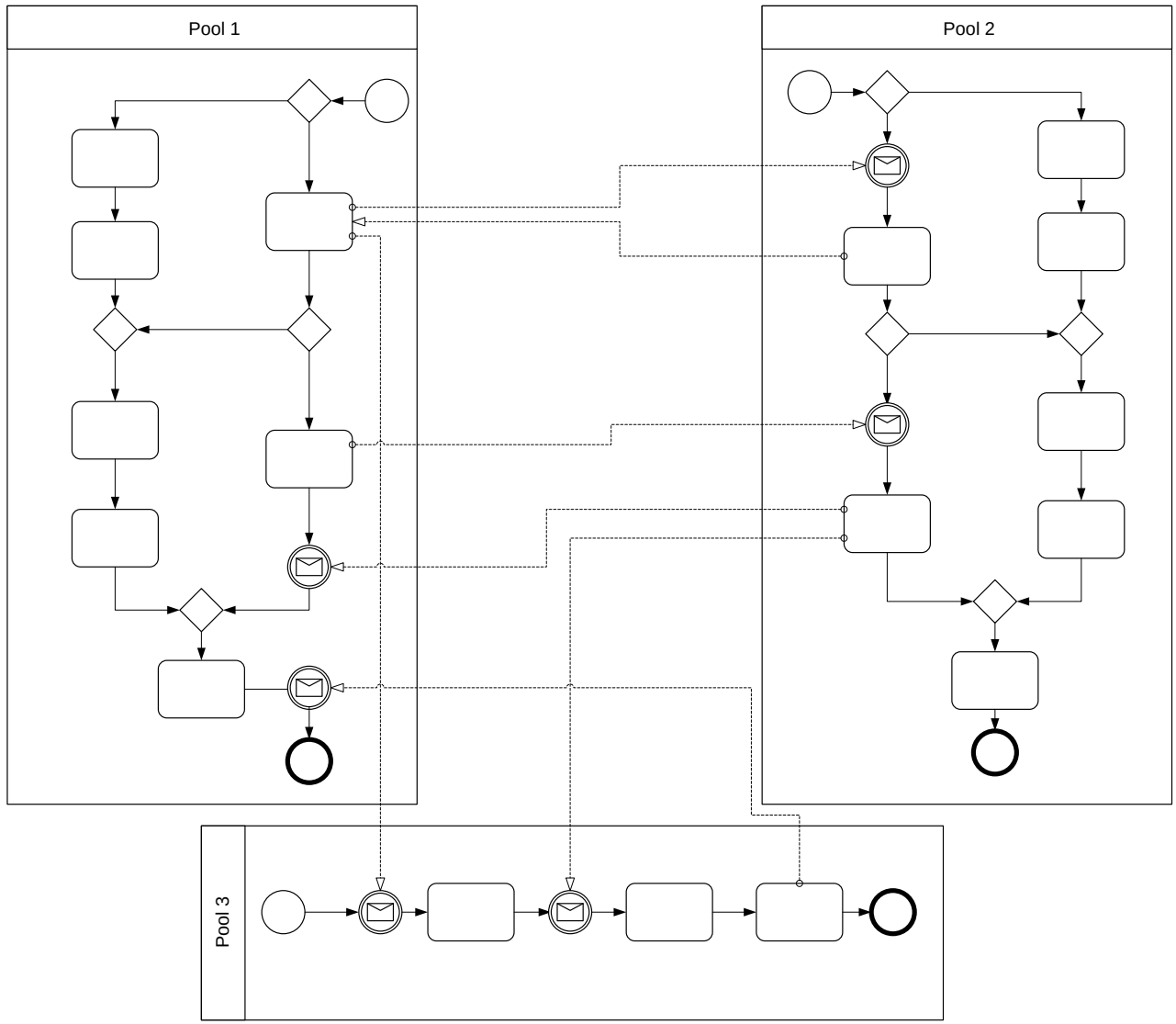
- Orchestration as mediation between different protocols



WS Choreography vs. Orchestration

Global collaborative process

- Combination of all other views
- Theoretical, since usually internal parts of processes are not shared



Web Service Composition Development

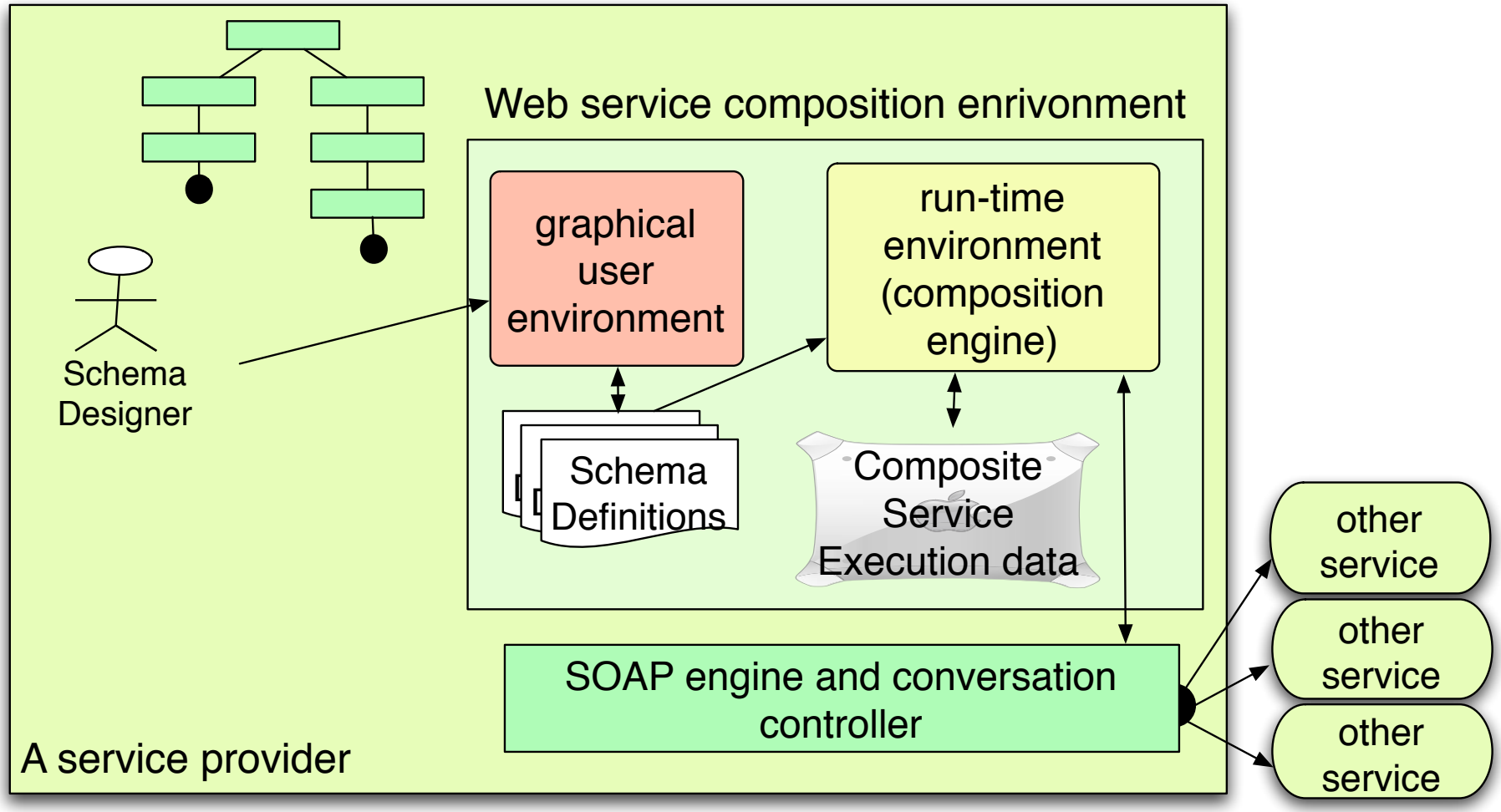
Features of a Web service composition development environment:

- **composition model and language**: enabling the specification of the services to be combined, the order in which the different services are to be invoked, and the way in which service invocation parameters are determined. The specification is referred to as *composition schema*.

- **graphical user interface**: an interface through which designers can specify a composition schema by dragging and dropping Web services into a canvas. The graphs and other descriptive information are them translated into textual specifications (i.e., the composition schema)

- **run-time environment:** composition engine that interprets the composition schema and executes its business logic

Web Service Composition Development



Part V

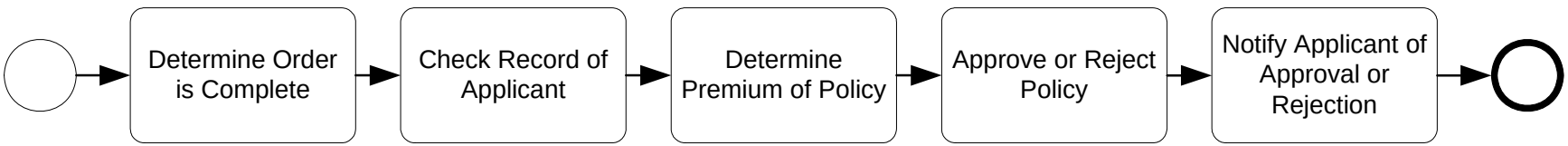
WS Composition Technologies: BPMN and BPEL

The need for BPMN

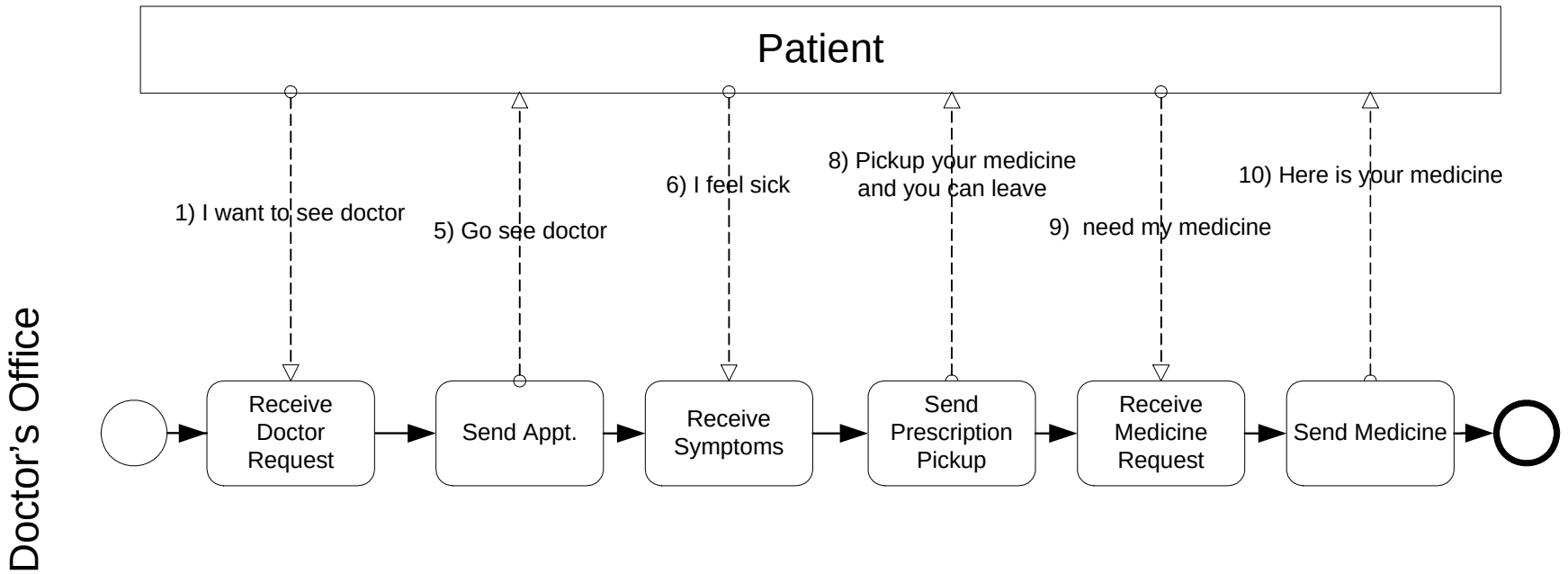
- Focus of earlier workflow languages: clear execution semantics (e.g., Petri Nets), expressivity (e.g., YAWL), interoperability (e.g., BPEL)
- Separate set of languages: conceptual process modelling notations without clear execution semantics, etc.
- But no notation that appeals to business users and has (somewhat) clear execution semantics

- $\Rightarrow$ Business Process Model and Notation (BPMN) addresses this "niche"
- OMG Spec 2.0 released in 2011 (v1.0 in 2006). Number of Google hits for BPMN: $\sim$ 1.2 million (7/5/2013).

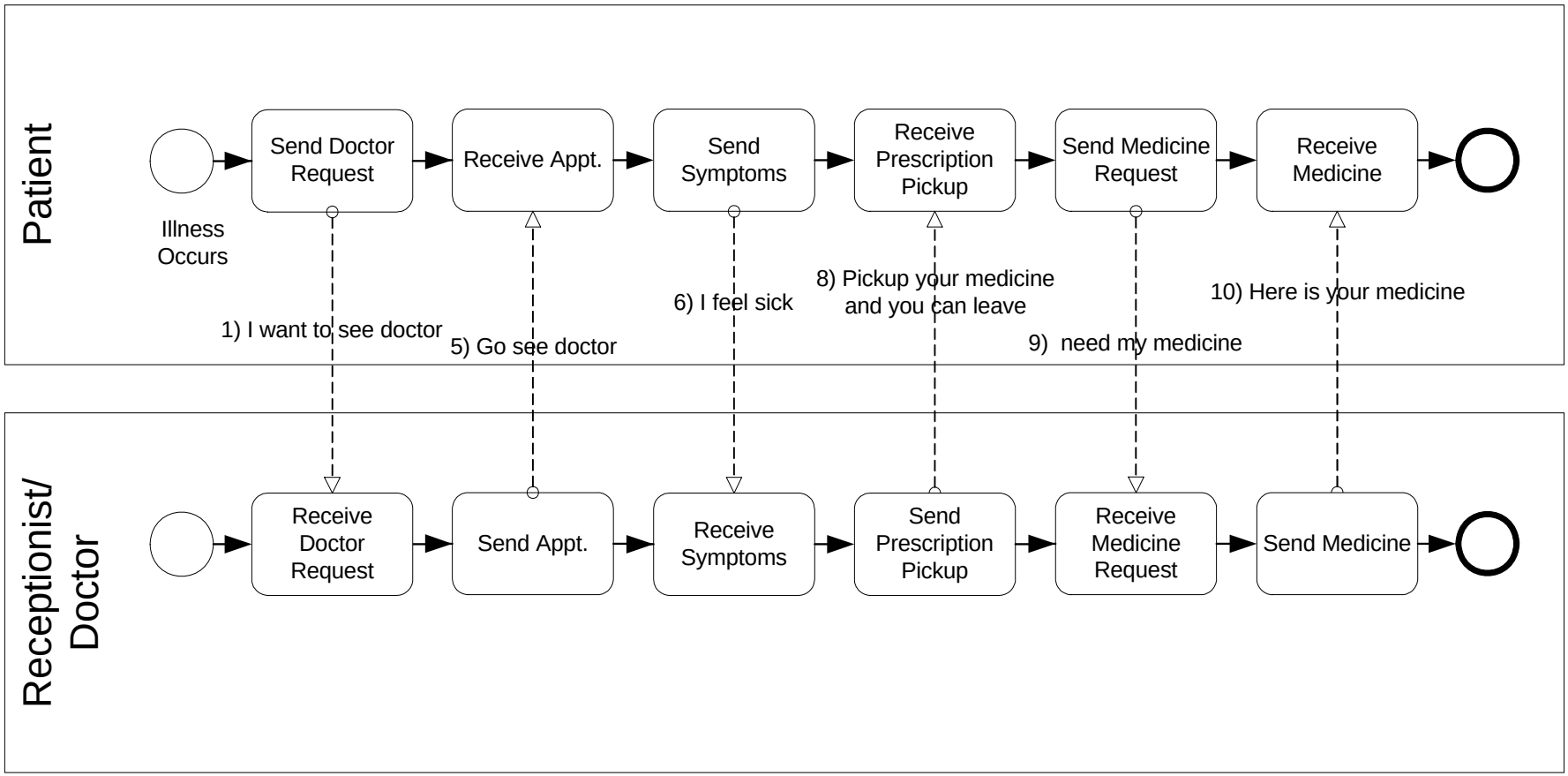
BPMN example: private process



BPMN example: abstract process



BPMN example: collaborative process



BPMN diagram elements

- Flow Objects
  - Events
  - Activities
  - Gateways
- Connecting Objects
  - Sequence Flow
  - Message Flow
  - Association
- Swimlanes
  - Pools
  - Lanes
- Artefacts
  - Data Object
  - Group
  - Annotation

BPMN particulars

For details, check the spec:

`http://www.omg.org/spec/BPMN/2.0/PDF`

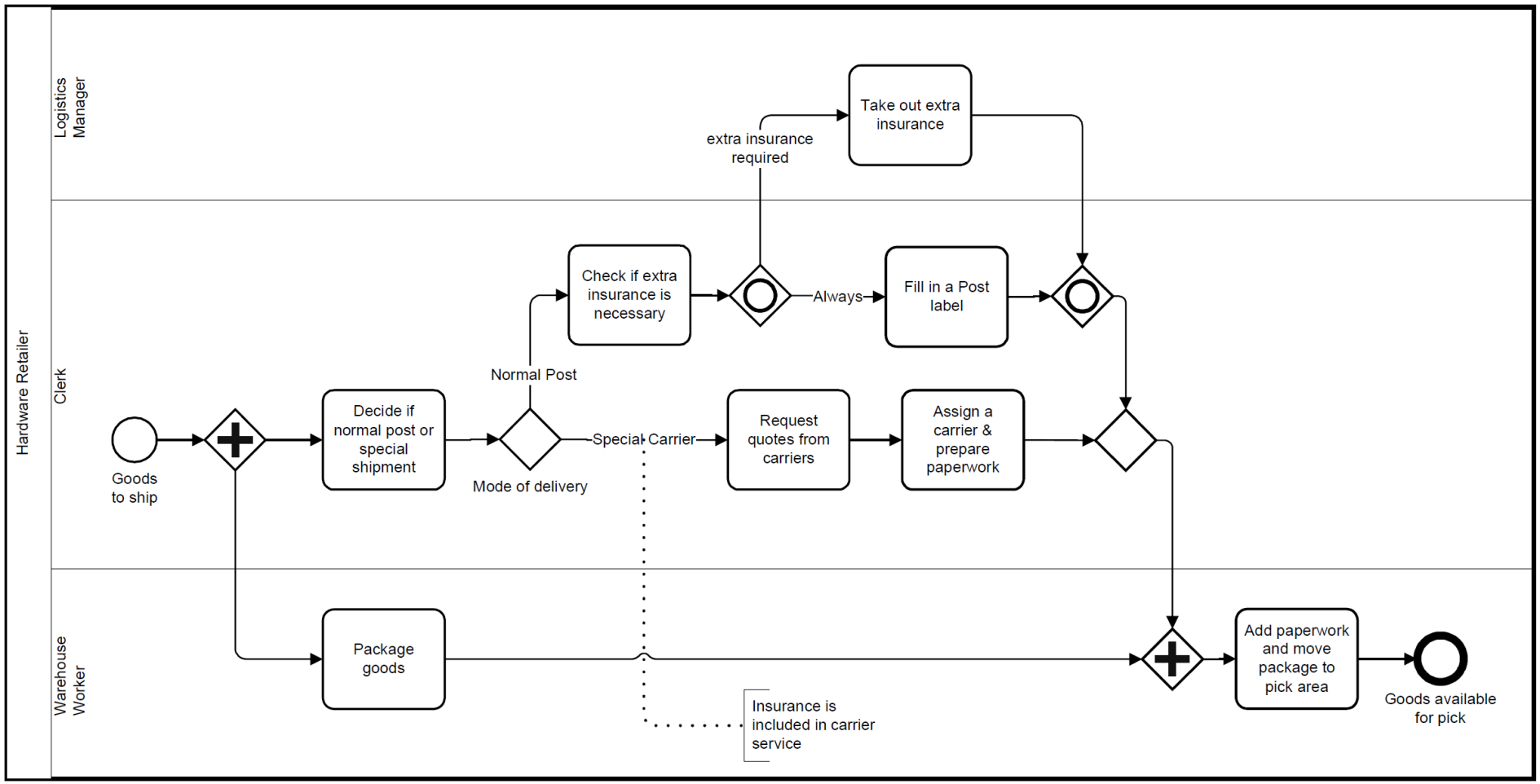
From p.59, the elements are explained.

Two things to note:

- Within a single pool: only sequence flows and associations. No sequence flow can leave a pool, but they can connect activities in different lanes of the same pool.
- Message flows only between different pools!

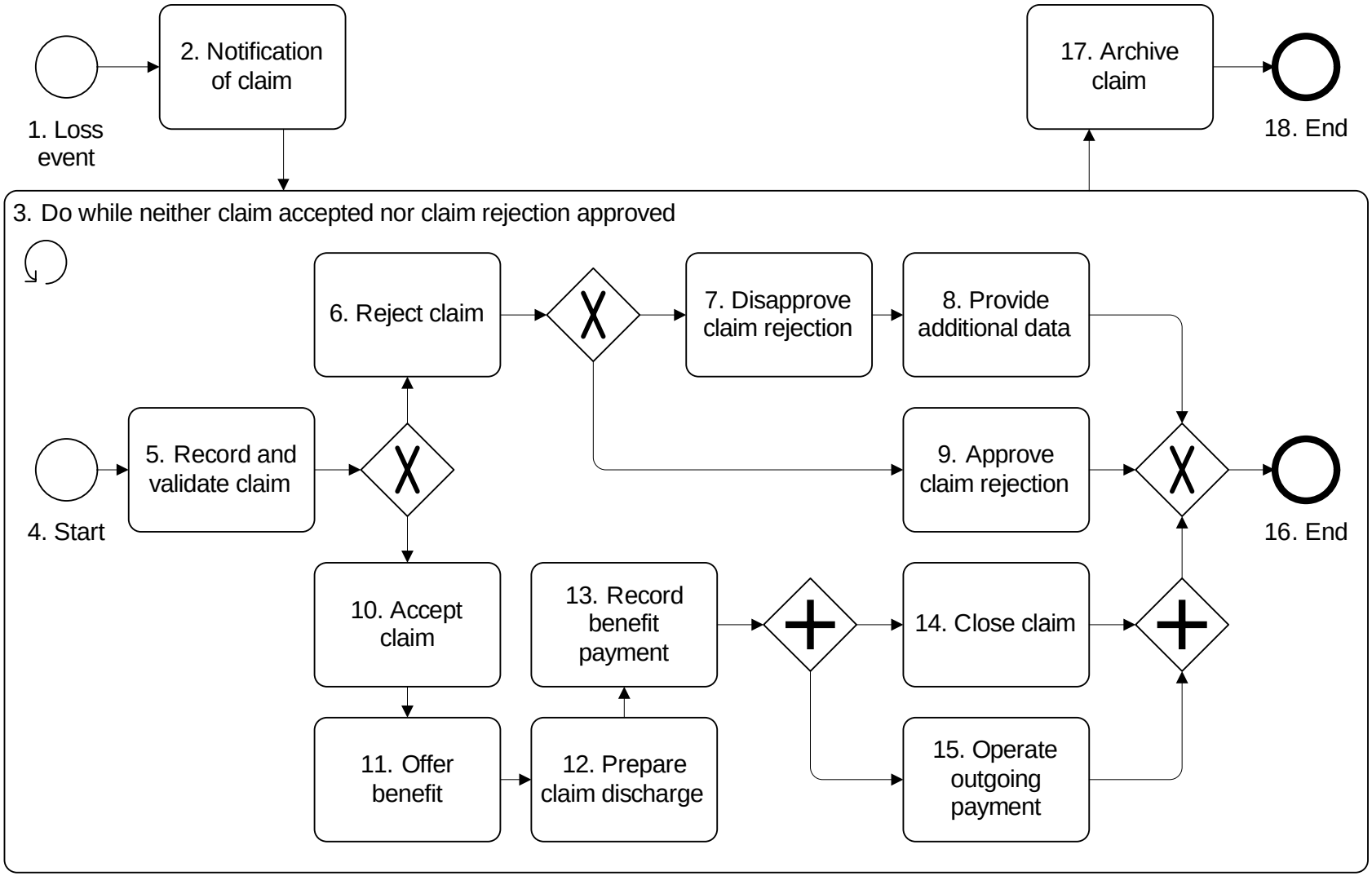
BPMN can be used to model choreographies, orchestrations, protocols, collaborative processes...

BPMN example: shipment process of a hardware retailer[7]



[7] Source: BPMN 2.0 by Example, OMG Document Number dtc/2010-06-02

BPMN example: insurance claims processing[8]



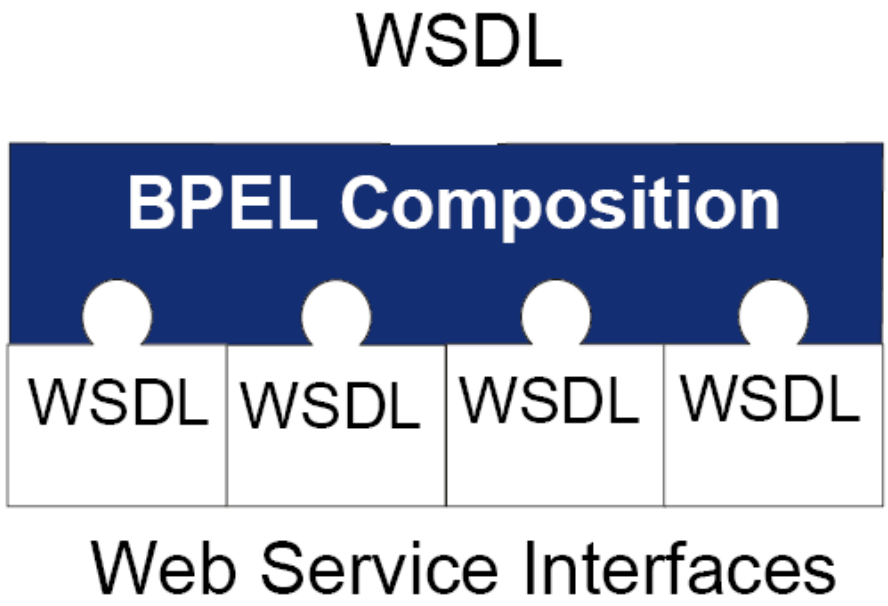
[8] Source: I. Weber, J. Hoffmann, J. Mendling. Beyond soundness: On the verification of semantic business process models. Distributed and Parallel Databases (DAPD), 27(3):271-343, June 2010

Business Process Execution Language (BPEL)

- BPEL is one of the most popular Web Service composition standards, nowadays at least on the execution side.
- BPEL can be used to;
  - implement *executable* business processes (cf. orchestration)
  - describe non-executable *abstract* processes (cf. protocol)
- BPEL provides the Web service composition environment
  - Language: describes business logic of the composite services as structured activities, XML-based
  - GUI environments (from vendors)
  - Run-time: BPEL engines (from vendors)
- Officially called WSBPEL (Web Services BPEL)
- OASIS Spec 2.0: `http://www.oasis-open.org/committees/wsbpel/`

Composing Executable Processes

- Executable Processes describe an orchestration of different Web service interfaces (WSDL Port Types)
- Abstract Processes describe basically a business protocol
- The result of a composition using BPEL is recursively published as a Web service.



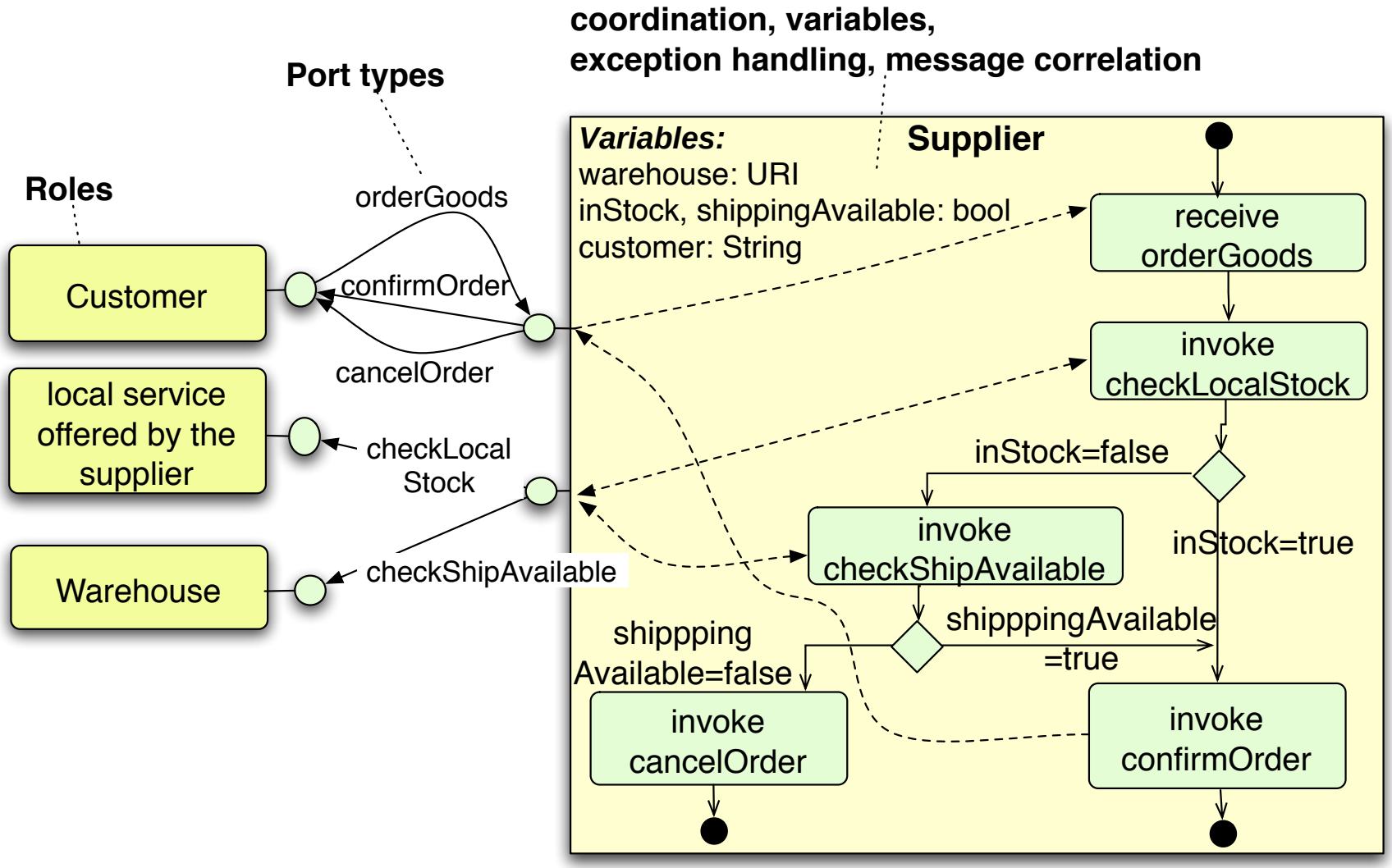
BPEL in a nutshell

BPEL processes are XML documents that define the following aspects of the process:

- The different roles that take part in the message exchanges with the process.
- The port types that must be supported by the different roles and by the process itself.
- The coordination of control flows and the other aspects (e.g., transaction, exception handling)
- Message correlation information, defining how messages can be routed to the correct composition instance.

BPEL in a nutshell

Scope of BPEL processes (orchestration for a supplier):



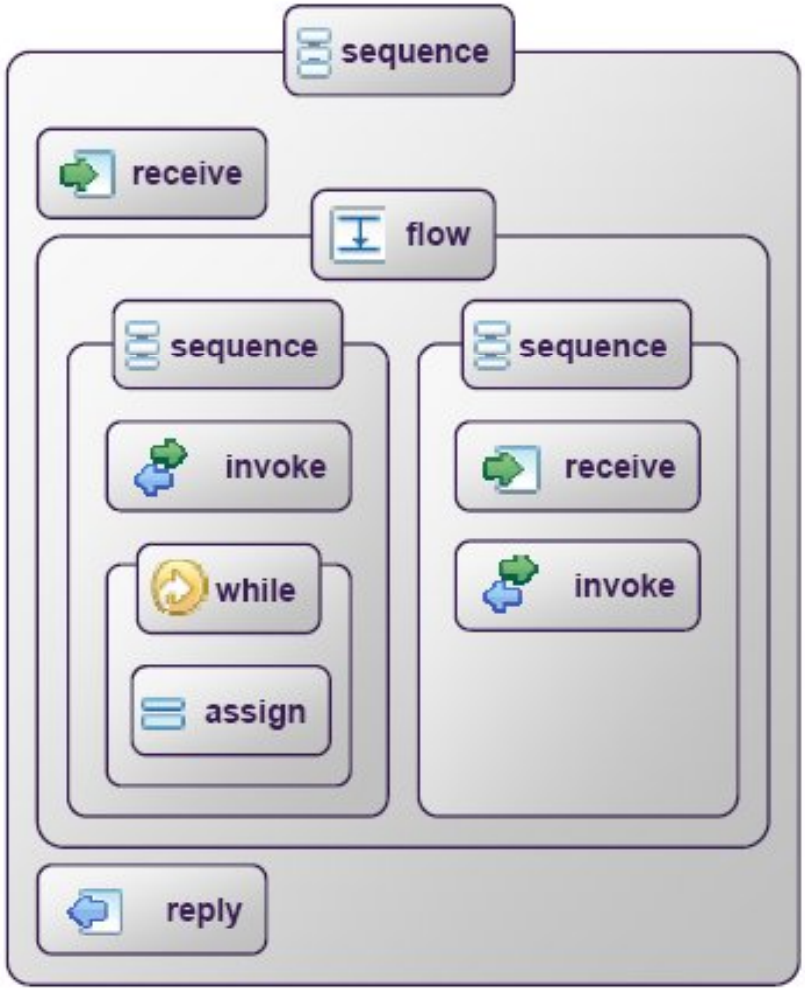
BPEL activities

The composition is based on BPEL Activities:

- Communication:
  - <invoke>: invoking an operation of a WS
  - <receive>: waiting for a message to an operation
  - <reply>: generating a message through an operation
- Structured Activities:
  - <sequence>: ordered steps of activities
  - <switch>: branching (like case-statement)
  - <while>: loop
  - <pick>: waiting for one of several events to occur
  - <flow>: parallel activities
- Misc.:
  - <wait>: waiting for some time
  - <assign>: copy data from one place to another
- The activities represent the well-known business process constructs such as AND-split, OR-split, AND-join etc.

BPEL Control-flow Style 1: Nesting Activities

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

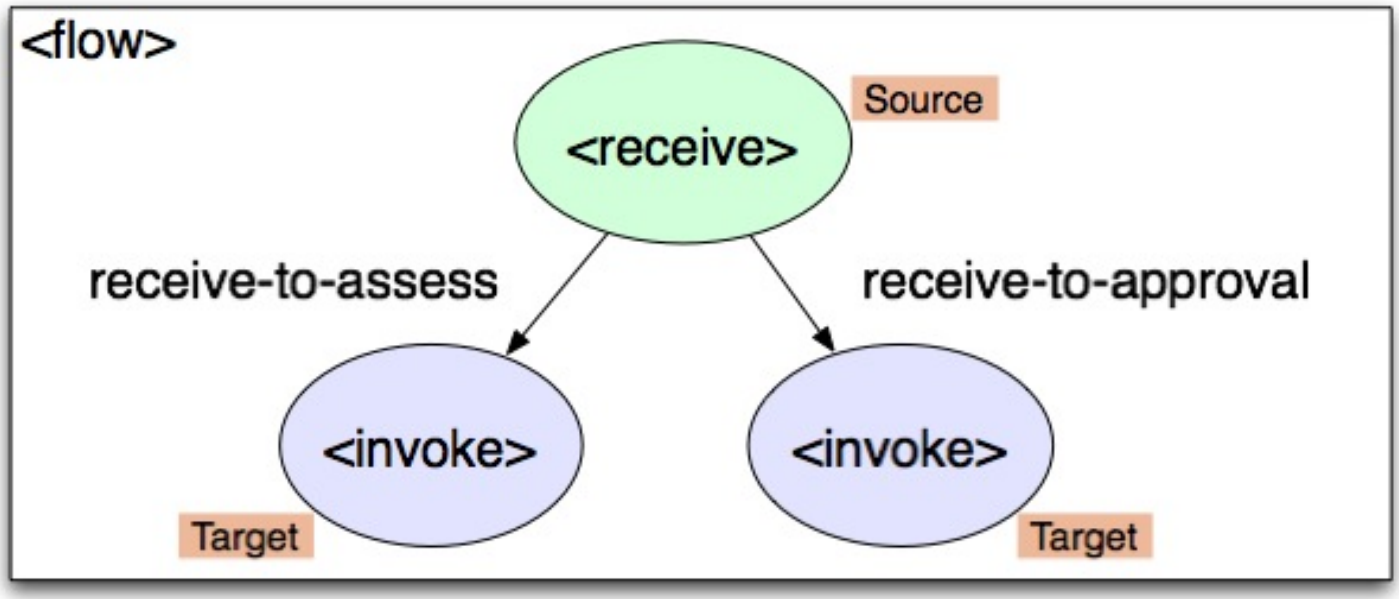


Disclaimer: (proprietary notation)

BPEL Control-flow Style 2: Flows + Links

Linking activities ...

An activity `A` may have outgoing links (when `A` is the source of the links) and incoming links (when `A` is the target of the links)



At the beginning, all links are inactive and only those with no dependencies (those that are not declared as target) can execute.

BPEL Control-flow Style 2: Flows + Links

```
<flow>
   <links>
      <link name="receive-to-assess"/>
      <link name="receive-to-approval"/>
   </links>
  <receive name="receive1" partner="customer"
         portType="apns:loanApprovalPT"
         operation="approve" variable="request" createInstance="yes">
    <source linkName="receive-to-assess"  .../>
    <source linkName="receive-to-approval"  .../>
  </receive>
  <invoke name="invokeAssessor" partner="assessor"
        portType="asns:riskAssessmentPT"
        operation="check" inputVariable="request"
        outputVariable="riskAssessment">
    <target linkName="receive-to-assess"/>
    <source linkName="assess-to-setMessage"  .../>
    <source linkName="assess-to-approval"  .../>
  </invoke>
  <invoke name="invokeapprover" partner="approver"
        portType="apns:loanApprovalPT"
        operation="approve"
        inputVariable="request"
        outputVariable="approvalInfo">
    <target linkName="receive-to-approval"/>
 </invoke>
</flow>
```

End of lecture: Web services

Summary

- Service-Orientation
- SOAP/WSDL Web service
- REST-ful services
- Web service composition, BPMN, BPEL

Next lecture

- Middleware

Assignments

- Bonus Assignment due Sunday, 12 May 2013*
- Assignment 2 due Sunday, 19 May 2013