# Web Services Foundations: SOAP, WSDL and UDDI

Helen Paik

School of Computer Science and Engineering
University of New South Wales

References used for the Lecture:

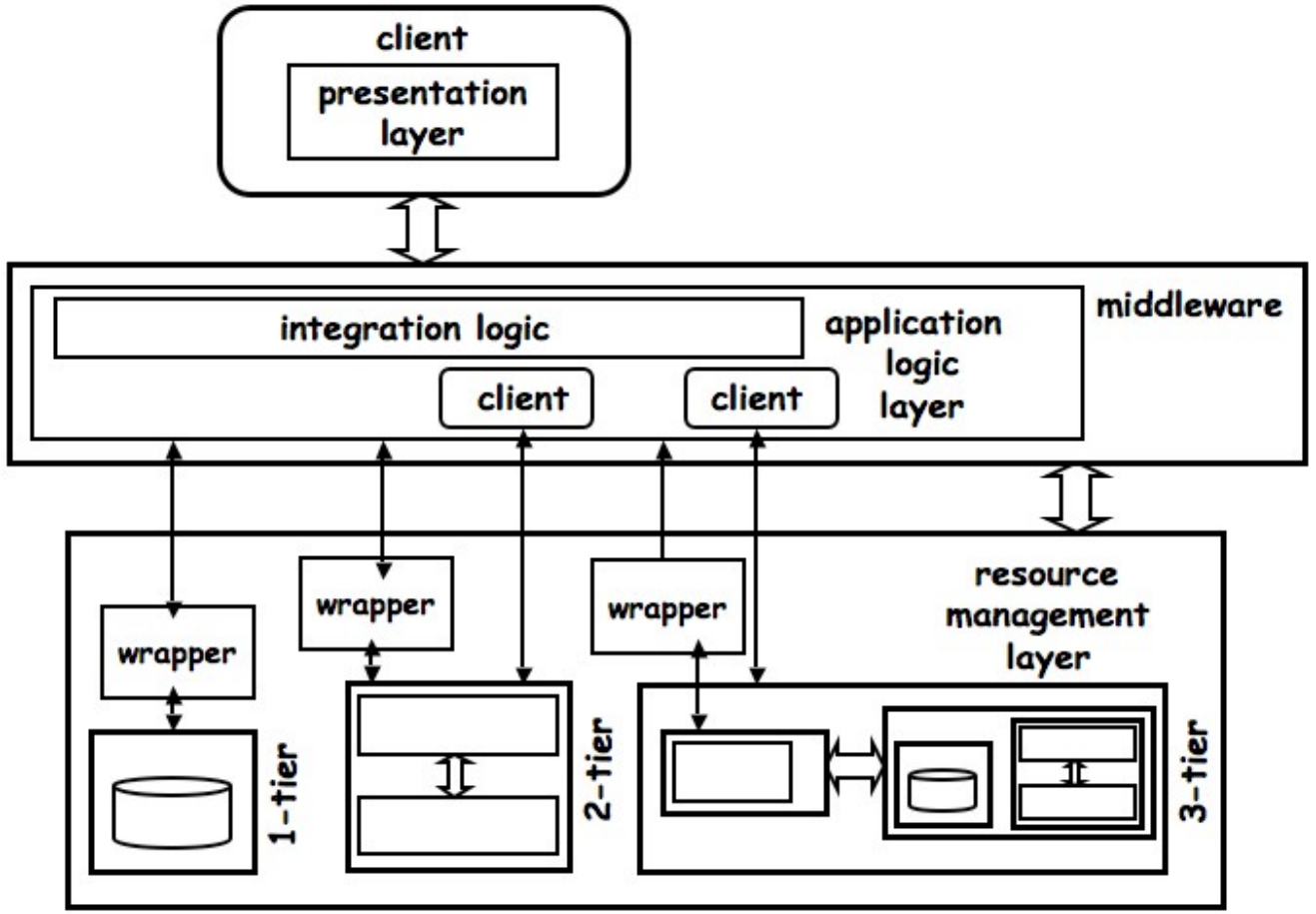
- Alonso Book Chapter 5-6
- Webber Book Chapter 3-4
- Mike Book Chapter 4-5

Acknowledgement: Some other materials is adapted from COMP9322 lectures in previous sessions, which are prepared by Dr. Helen H-Y Paik, Prof. Boualem Benatallah and Dr. Sherif Sakr. Some slides are prepared by Prof. Fabio Casati.

Week 2/3

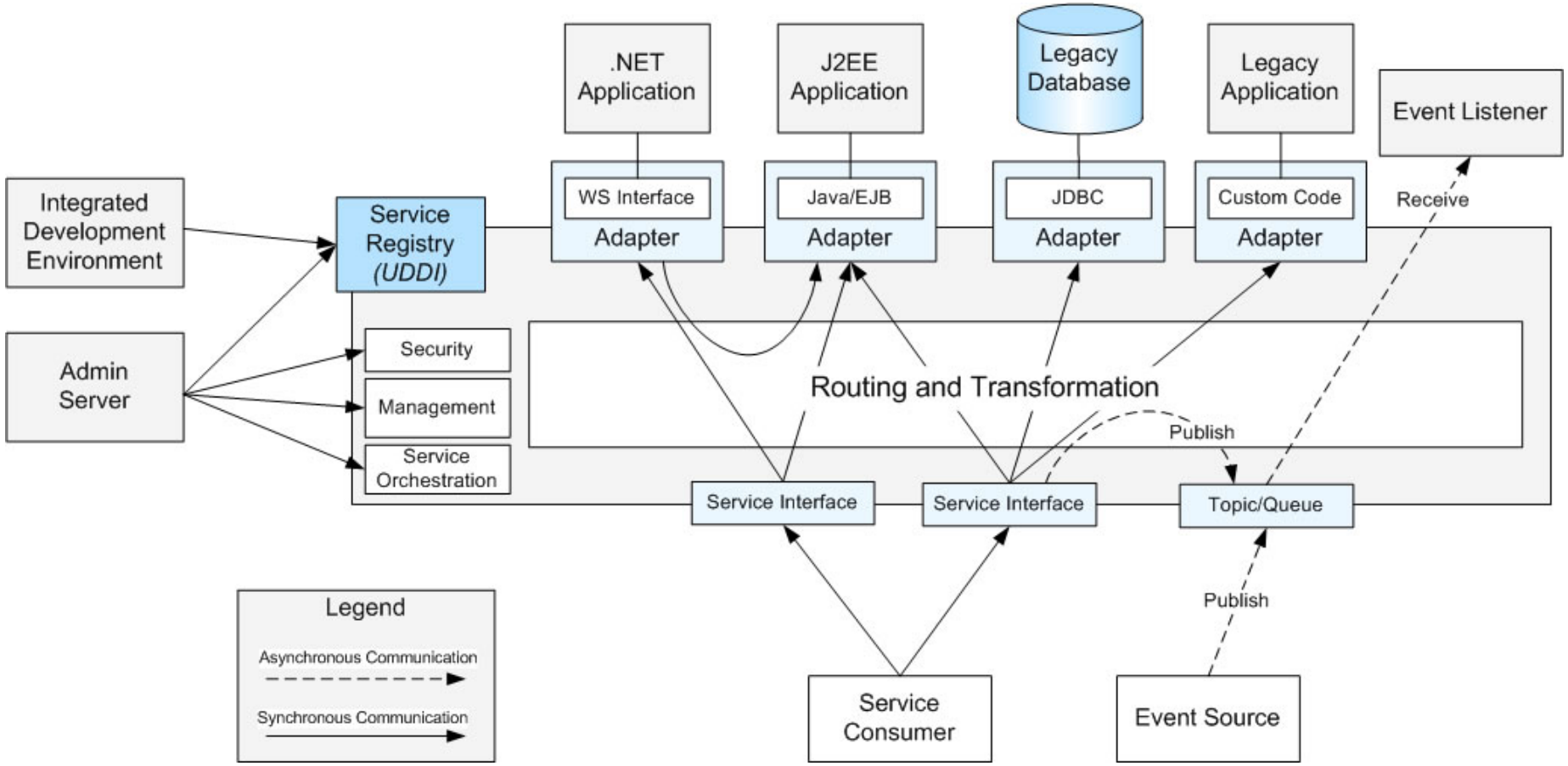
# SOA as a middleware solution

- information system architectures (1, 2, 3 and N-tiers)
- application integration layer: middleware



# SOA as a middleware solution: Service Bus

- ... often called Enterprise Service Bus[1]




[1] http://enterprisearchitecture.nih.gov/ArchLib/AT/TA/EnterpriseServiceBusPattern.htm

# Web Services

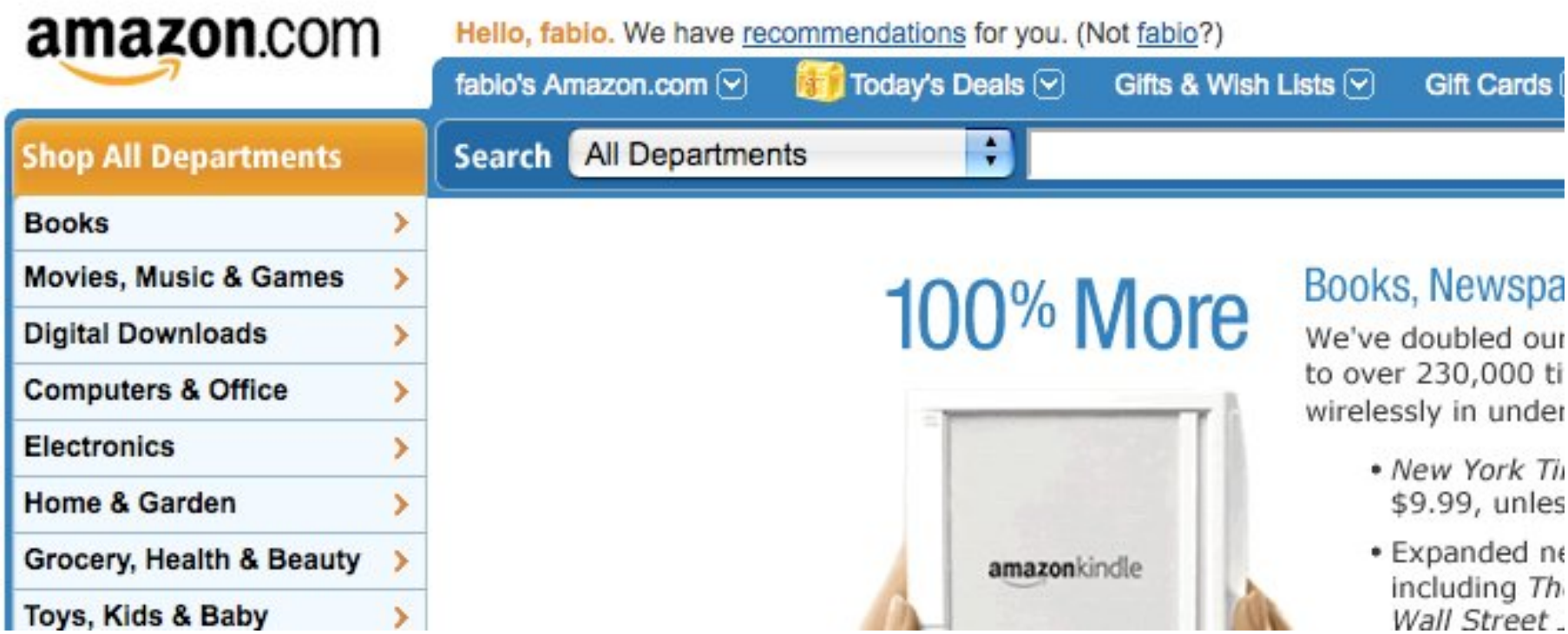
- In the context of middleware/service bus, we can see Web services as (individual) software component building technology that allows the underlying applications to communicate with client applications (both synchronised, or asynchronised).

- Outside the middleware context (i.e., more generically speaking), A Web service is a software component that can be accessed by another application (such as a client program, a server or another Web service)

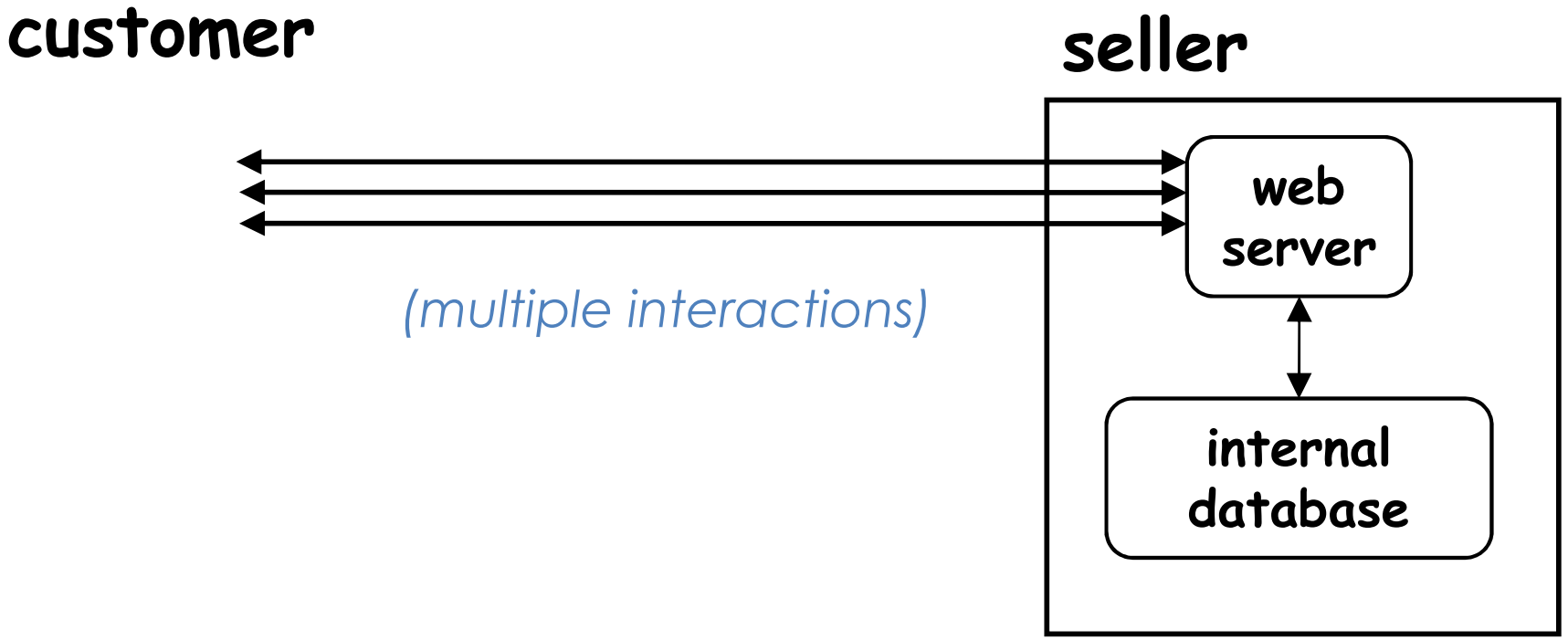
# Part I

# Web Services - Introduction

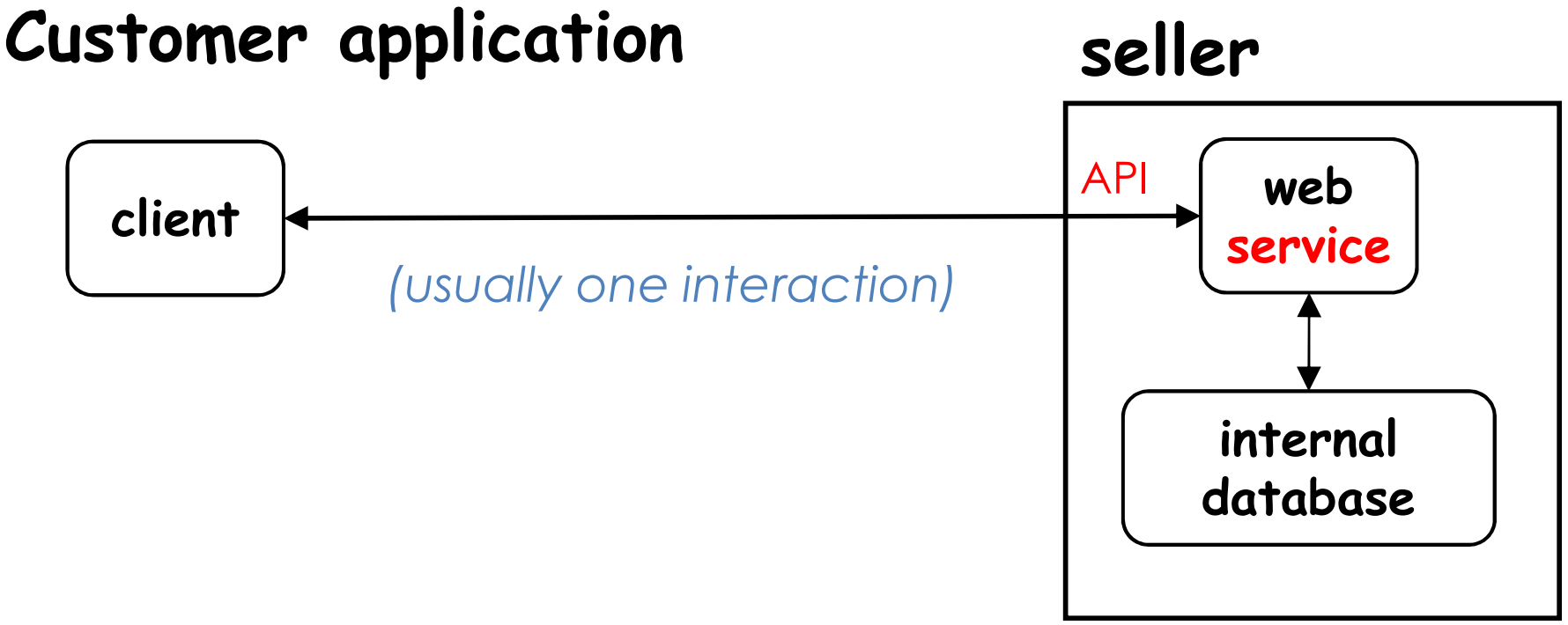
# Example - Amazon / Searching for books



# Amazon Search as an Application on the Web

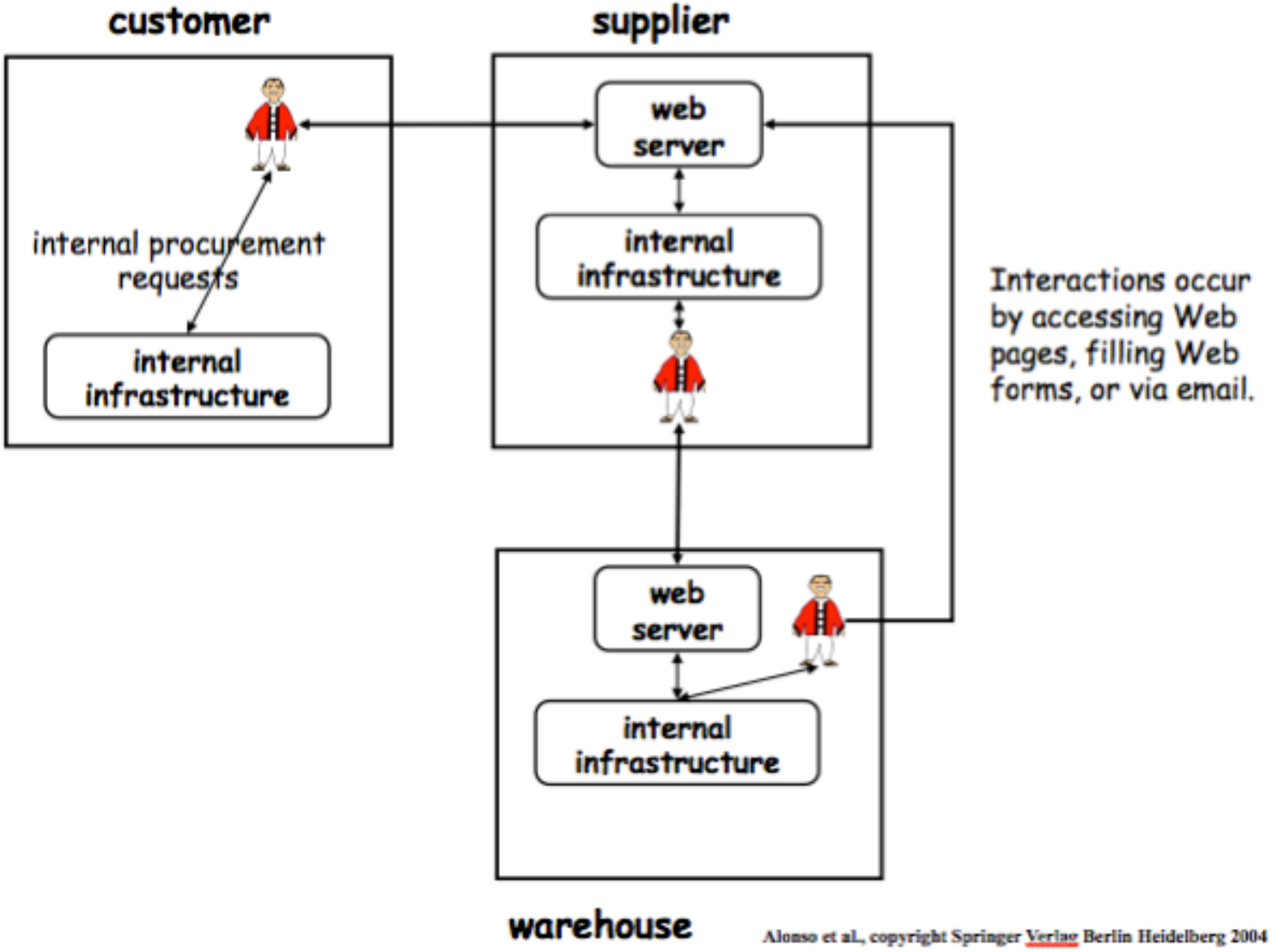


# Amazon Search as a Web Service

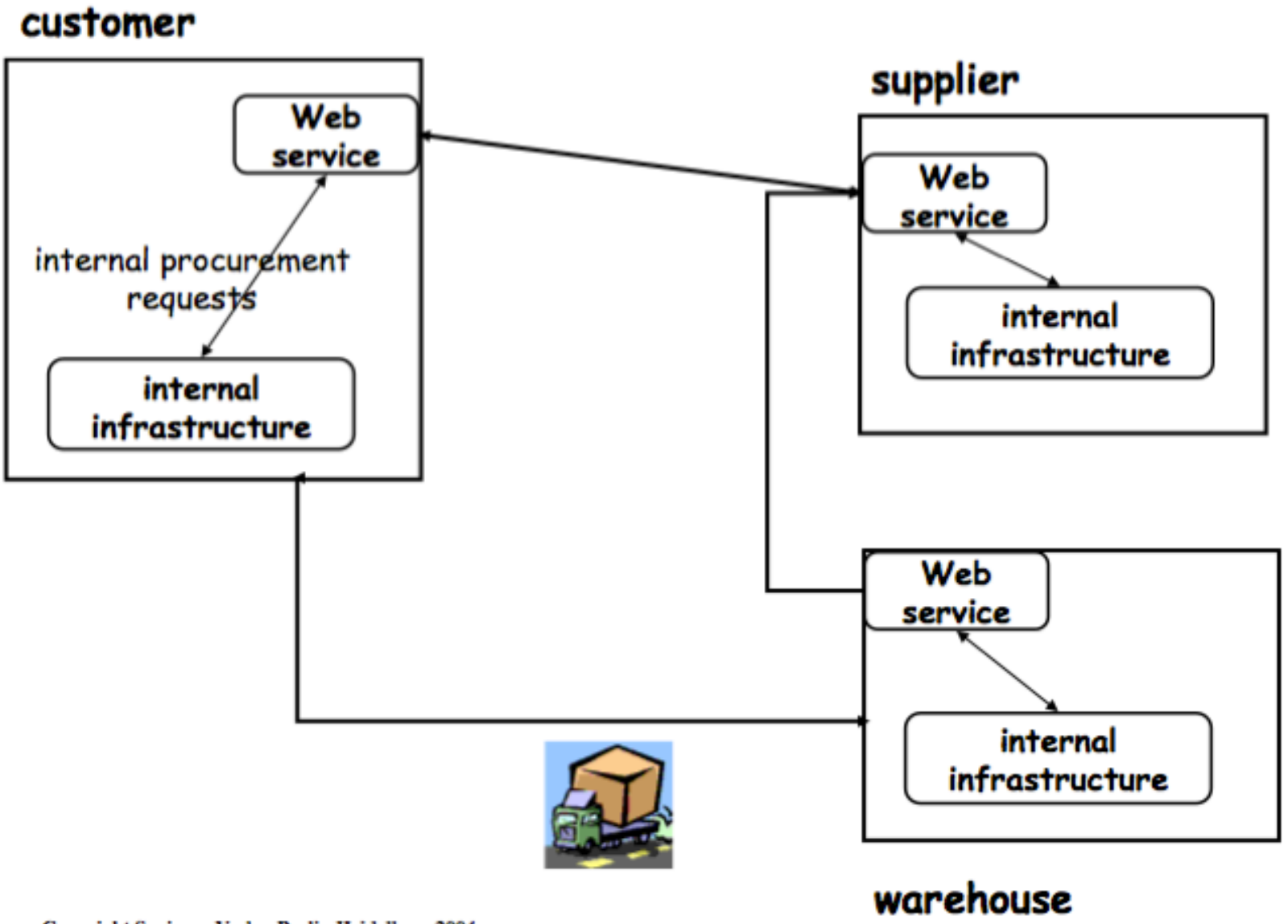



e.g., `http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl`

# Soft. System to Soft. System Communication - Manual



# Soft. System to Soft. System Comm. - Automated



# Fundamental Differences and Implications

Fundamental differences

- No human in the loop. The behaviour of the client is based on its code (pre-determined), no form-based interface to drive user input or interactions when things go wrong ....

Implications (among other things):

- Specified and 'self-explanatory' programming interface
- Specified interaction style ... expected consumer-provider behaviour (e.g., can I search before login?)

$\rightarrow$ We need standardised approach to addressing these issues ...

# Web Services

Key concept: a Web service is a software component designed to support interoperable machine-to-machine interaction over a network.

- Allow applications to share data and invoke capabilities from other applications

Key facts:

- No need to consider how the 'other' applications were built, what operating system or platform they run on
- A standardised way of application-to-application communication based on XML open standards (ie., SOAP, WSDL and UDDI) over an Internet protocol backbone.
- Unlike traditional client/server models, web services do not require GUI or browser - not meant for human consumption.

# Web services commercial frameworks

- **Microsoft**

  (`http://msdn.microsoft.com/en-us/library/ms950421.aspx`)

  "XML Web services are the fundamental building blocks in the move to distributed computing on the Internet."

- **IBM**

  (`http://www.ibm.com/software/solutions/soa`)

  "Self-contained, modular applications that can be described, published, located and invoked over a network (generally the Internet)."

- **Oracle**

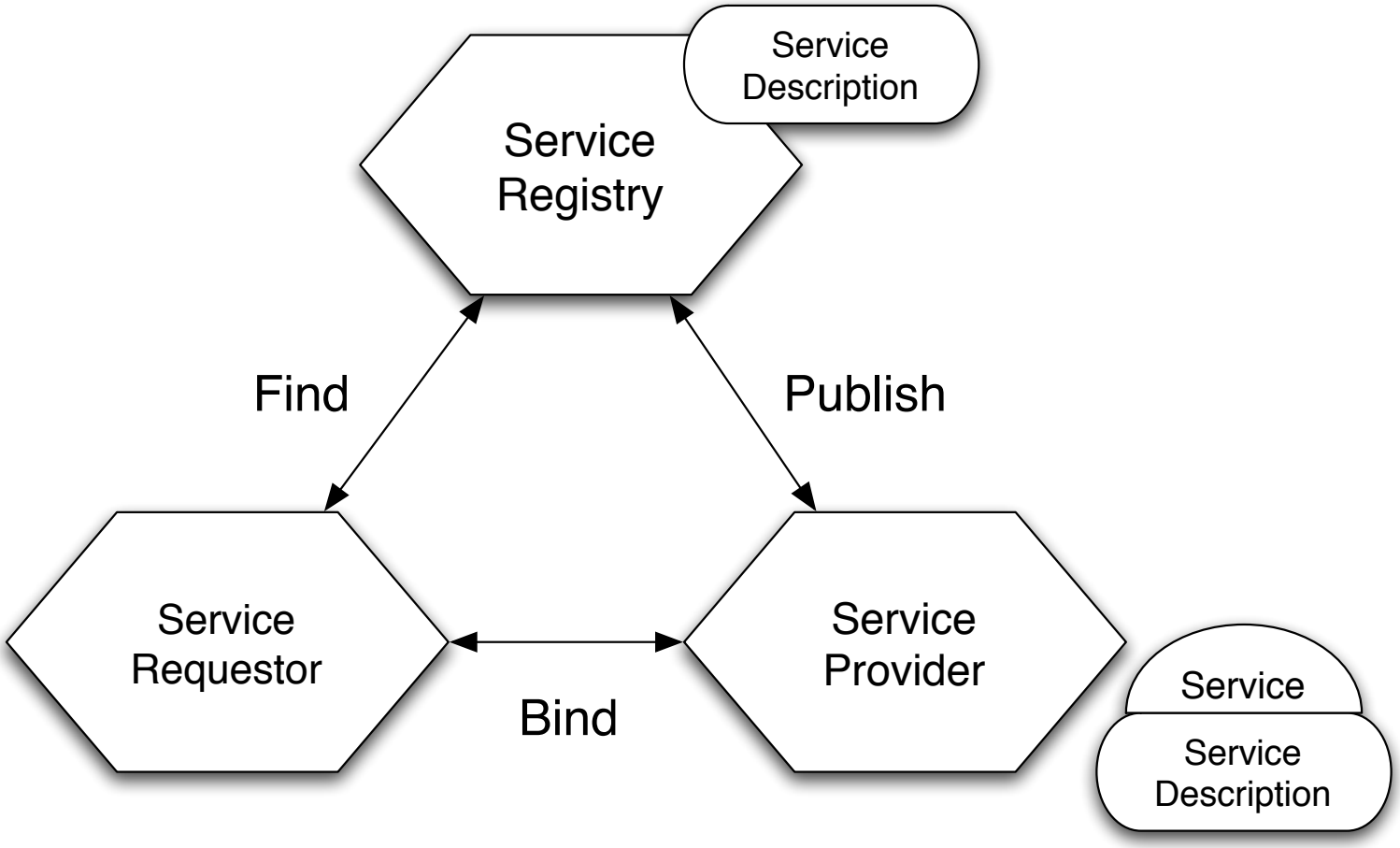
  (`http://www.oracle.com/us/technologies/soa/index.html`)

  "Service-oriented architecture ... is the cornerstone design principle upon which organizations are building and integrating modern business applications."

- and so on ...

# Web Services Conceptual Architecture (by IBM)

Basic Web Services Architecture



# Web Services Conceptual Architecture (by IBM)

**Three Roles**:

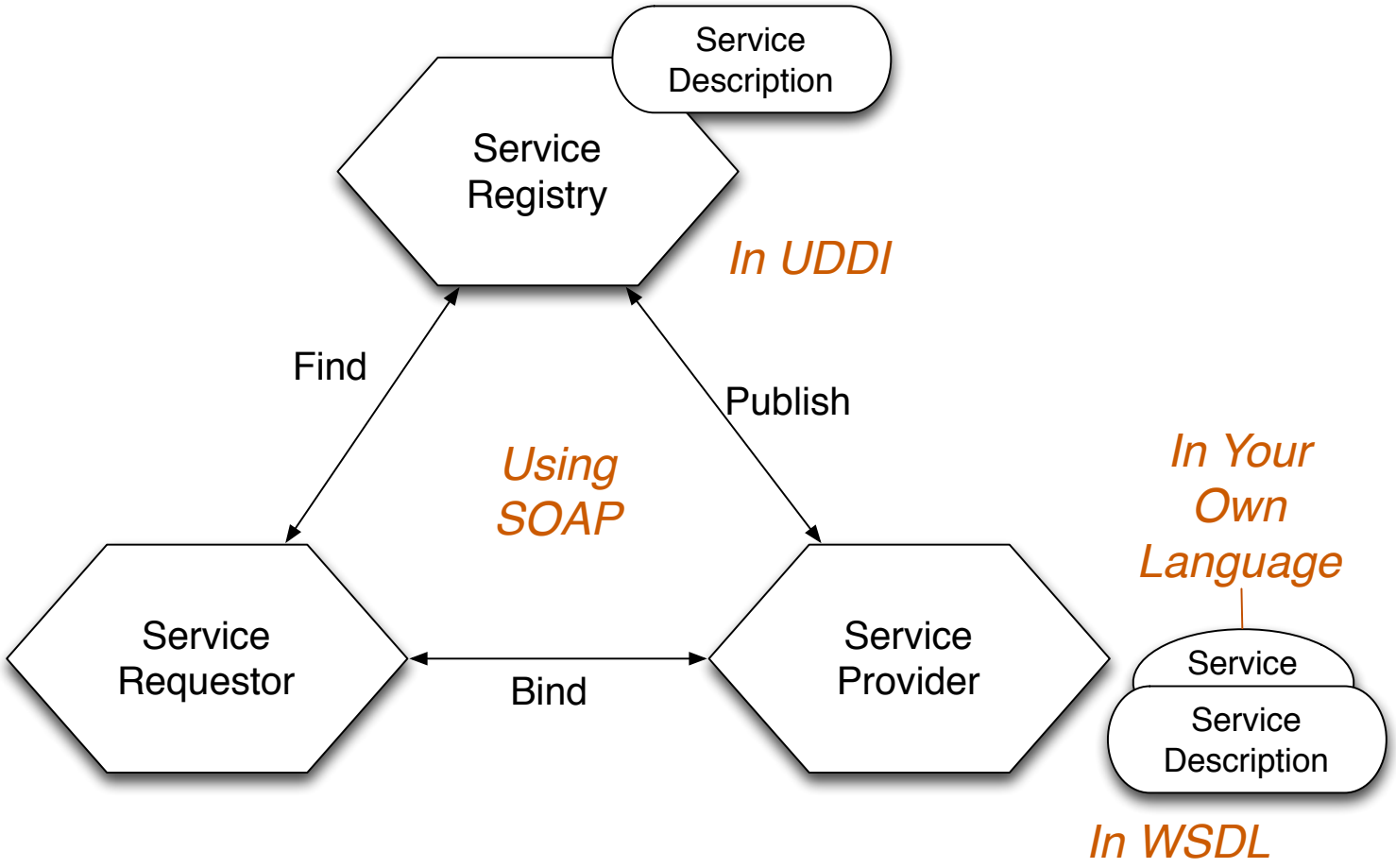
- service provider: develops an electronic service and registers its description at a publicly accessible service registry.
- service registry: store/manage web services details
- service requestor: query the registry to find an electronic service that meets his or her requirements. A binding occurs between the service provider and the service requestor.

**Main Web Services Standards**:

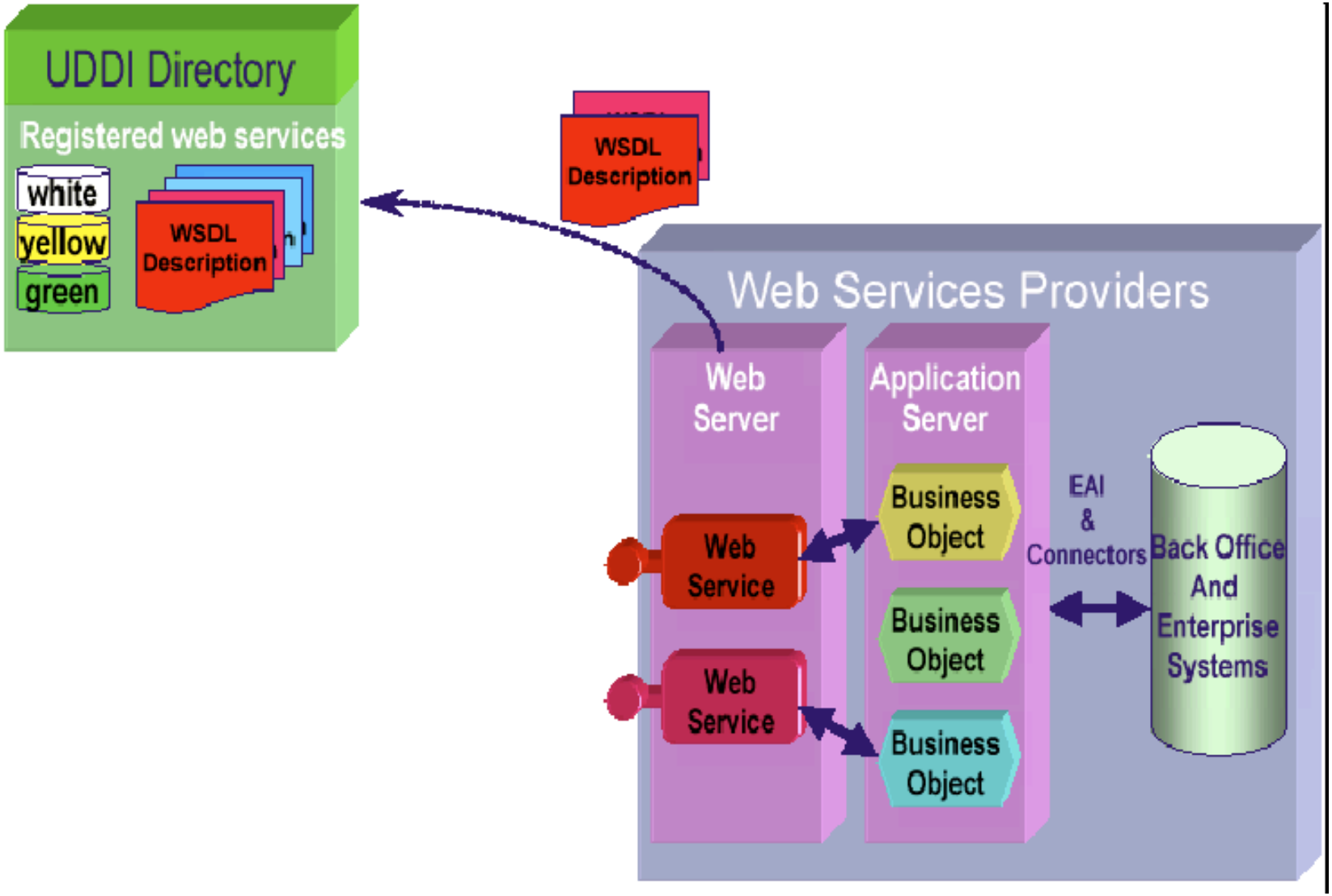
- For service registry: UDDI, Universal Description, Discovery and Integration (`http://uddi.xml.org/`)
- For service description: WSDL, Web-services Description Language (`www.w3.org/TR/wsdl/`)
- For messages: SOAP, Simple Object Access Protocol (`www.w3.org/TR/SOAP/`)

# Web Services Conceptual Architecture (by IBM)

Basic Web Services Architecture - supported by standards

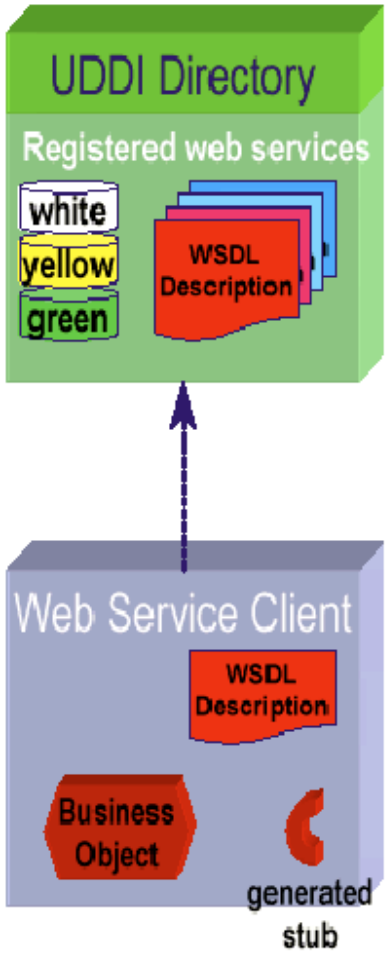


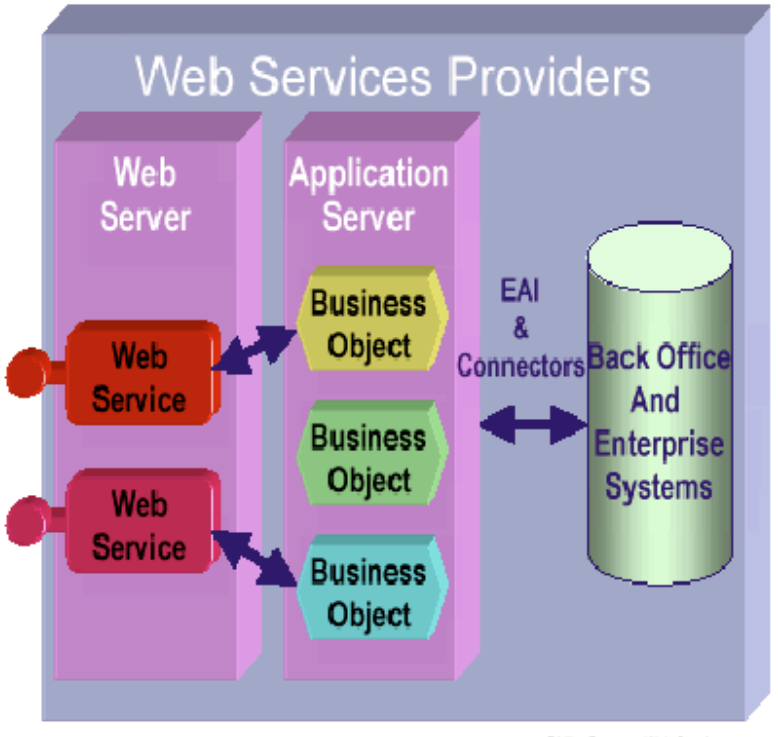
# Interactions between WS and Client




The web service provider declares his services within the UDDI register.

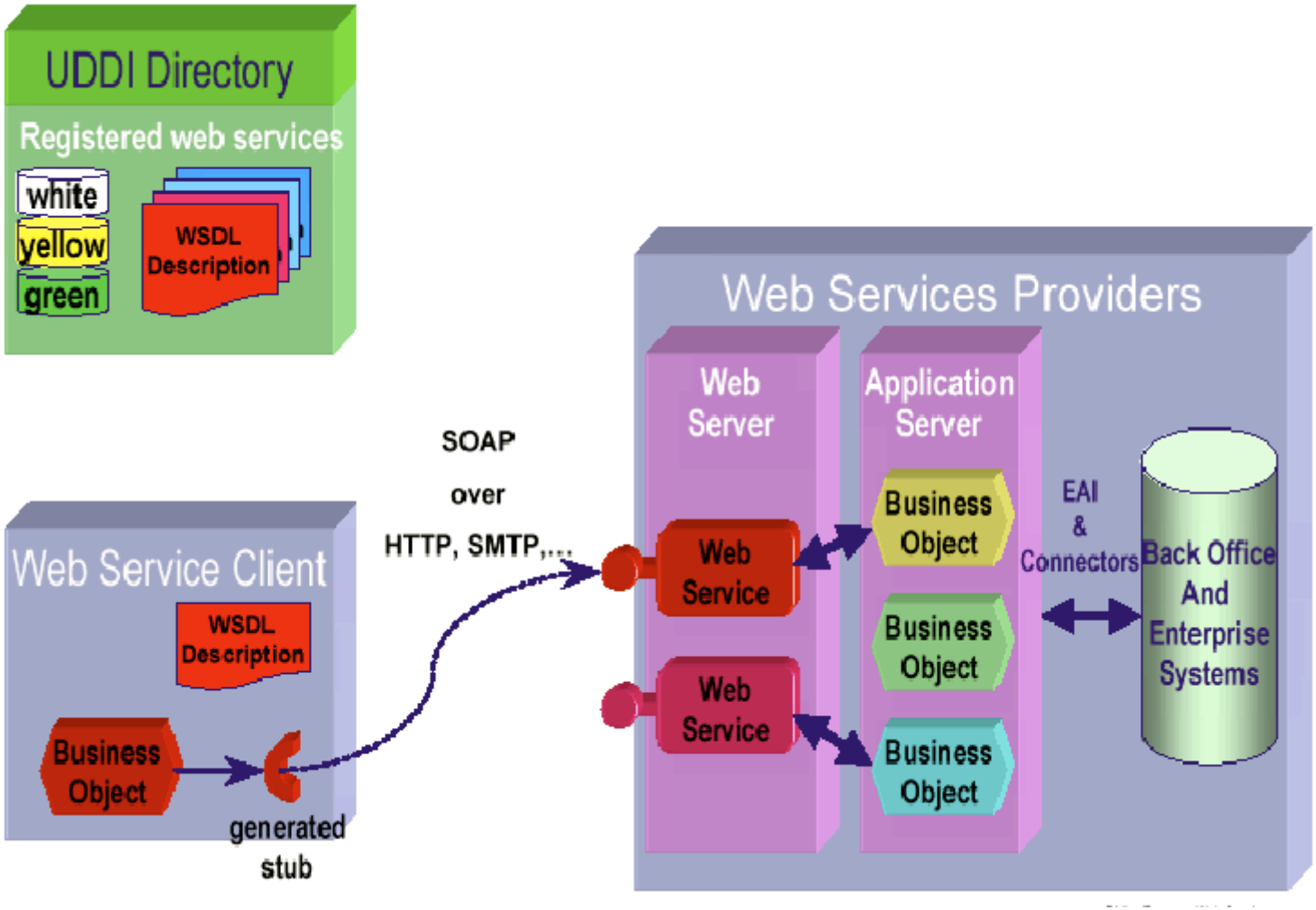
# Interactions between WS and Client




- The client looks for a Web service in the UDDI register
- Downloads the Web service's WSDL to build or generate a proxy (stub) for the web service

# Interactions between WS and Client




- The client invokes the Web service through the proxy sending/receiving SOAP messages

# So ... to summarise ...

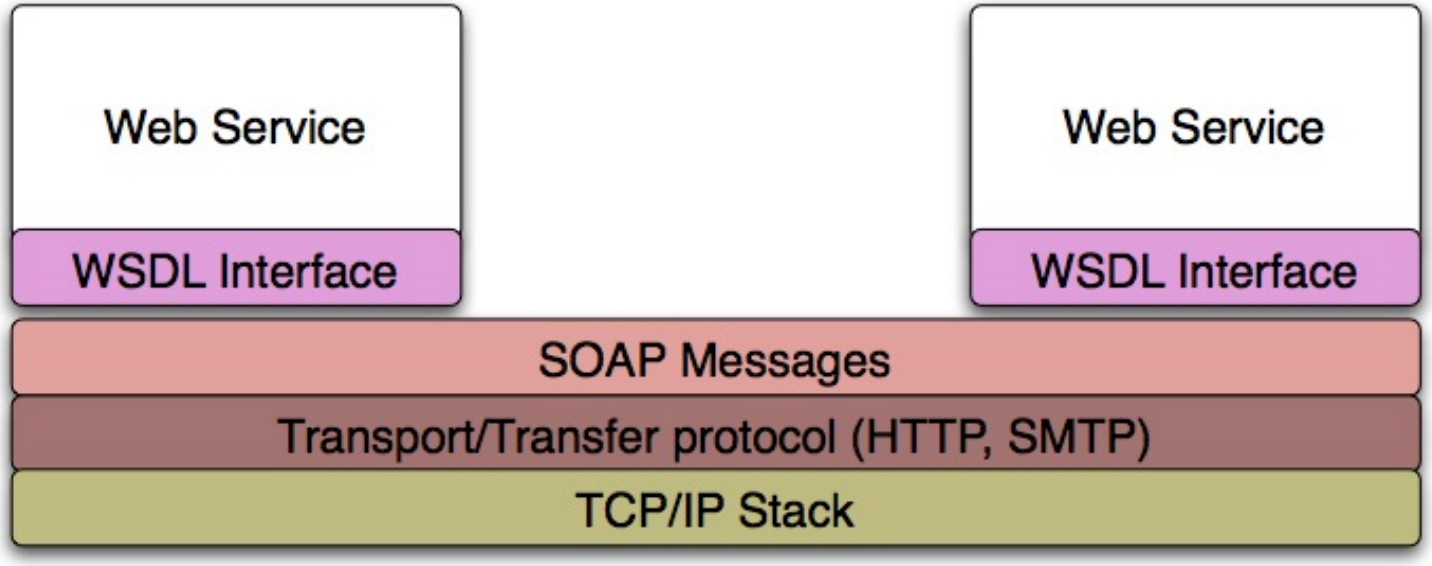
- What would you say the fundamental differences are between a Web site/app and Web service?
- Many middleware solutions have been proposed in the past ... What's different about Web services?
- Can you name three roles in the basic Web service architecture?
- Can you name the standards supporting the basic Web service architecture?
- Can you describe the basic interactions between WS and its client in the basic Web service architecture?

# Part II

# SOAP

# SOAP and Web Services

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

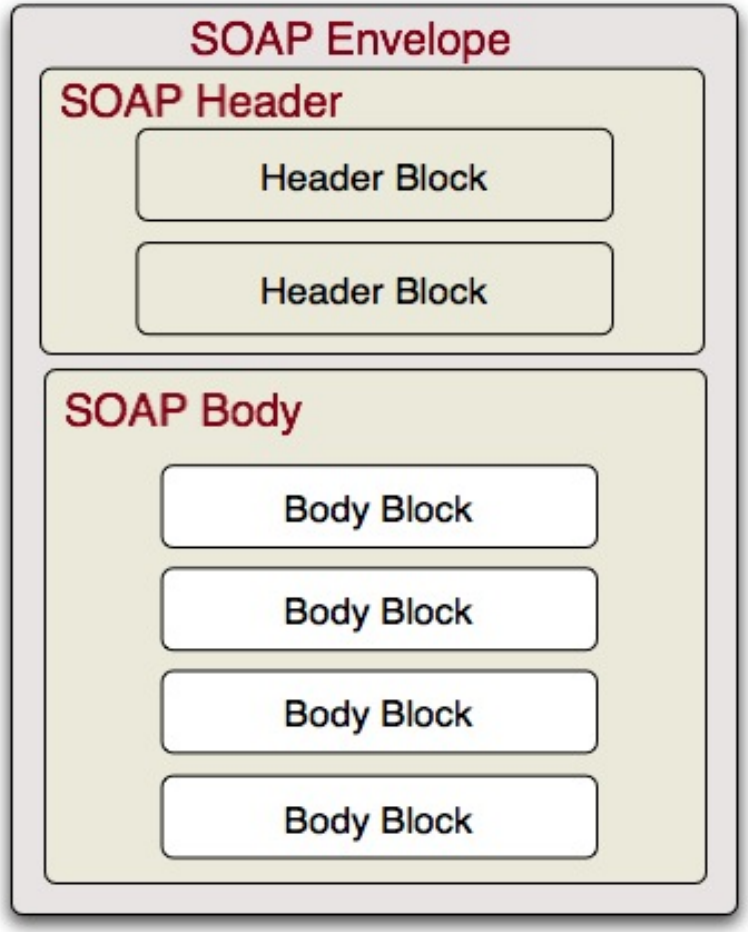



2

[2] Webber book, p72

# Simple Object Access Protocol (SOAP)

SOAP defines a standard message format for communication, describing how information should be packaged into an XML document.




```
<?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

# SOAP Message Example

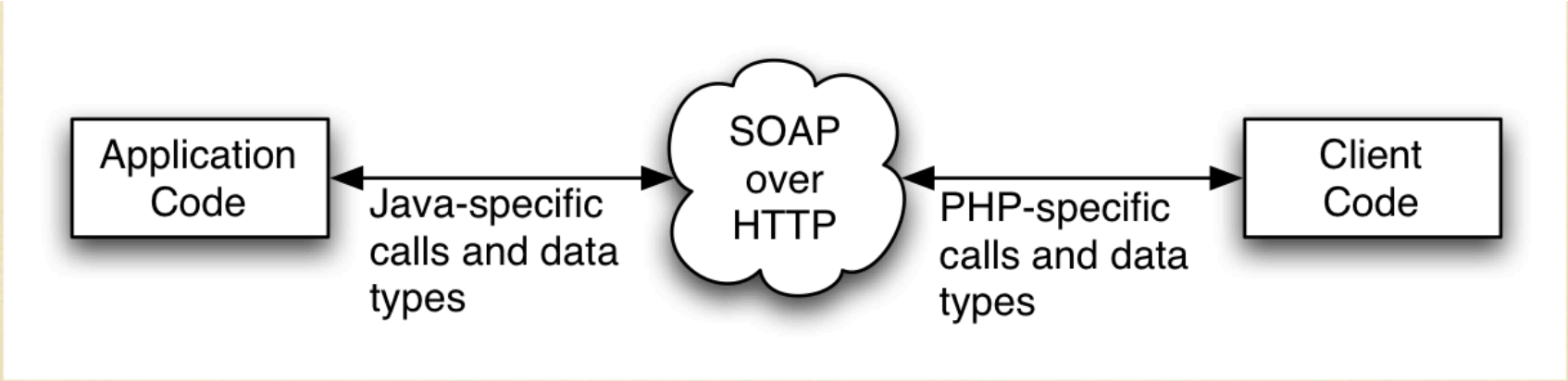
Sample SOAP Request and Response Message for Google's Web Service Interface (illustration purpose only)

`http://www.w3.org/2004/06/03-google-soap-wsdl.html`

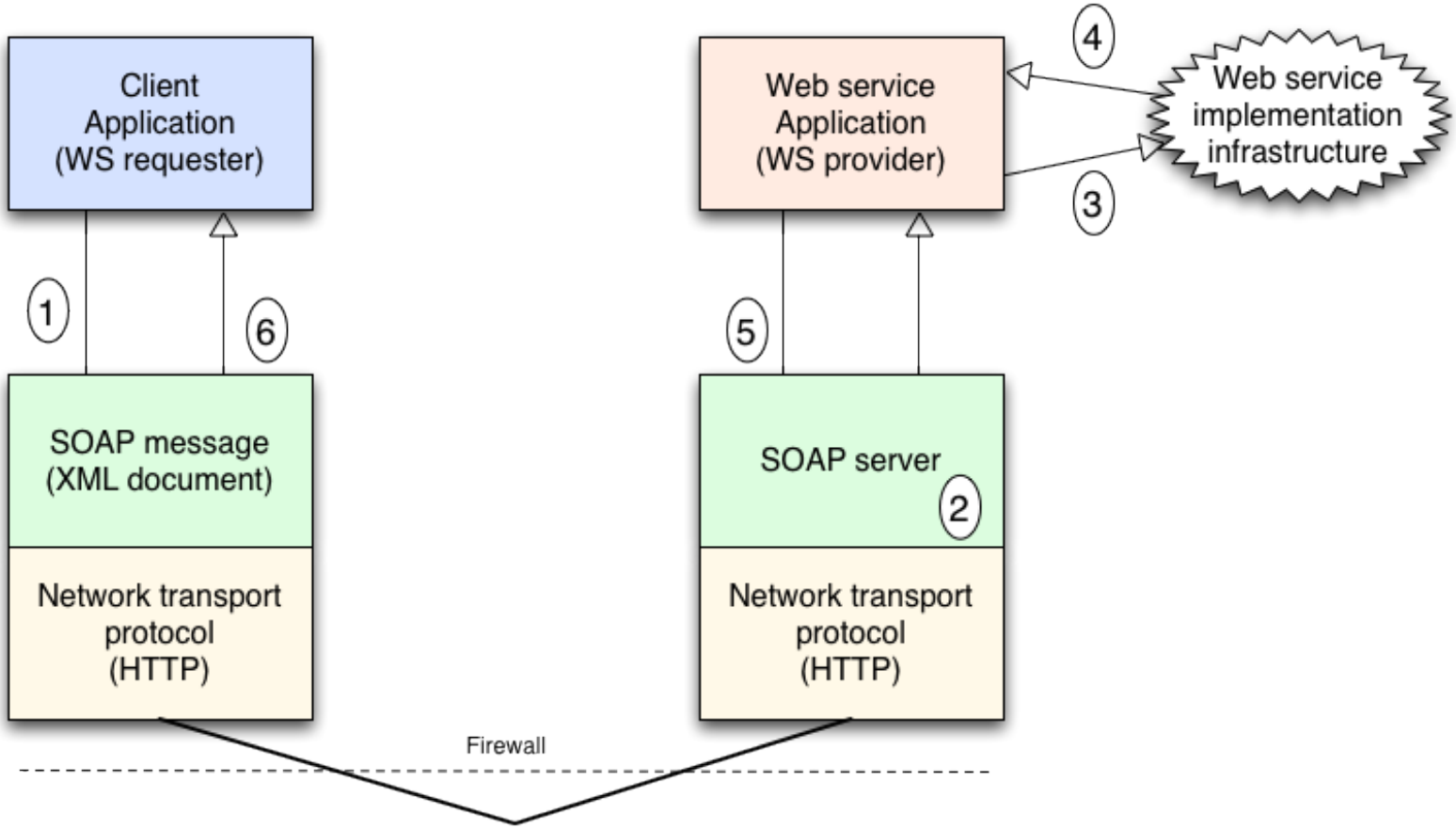
# Very Basic SOAP Architecture

- SOAP was initially adopted by Microsoft for inter-application communication within .NET framework
- now it is adopted by many other languages ...
  - J2EE 1.5 (onwards) supports JAX-WS (Java implementation of SOAP)
  - PHP 4 and over, Perl, Python, etc.

This means ... in SOAP, the following is possible:

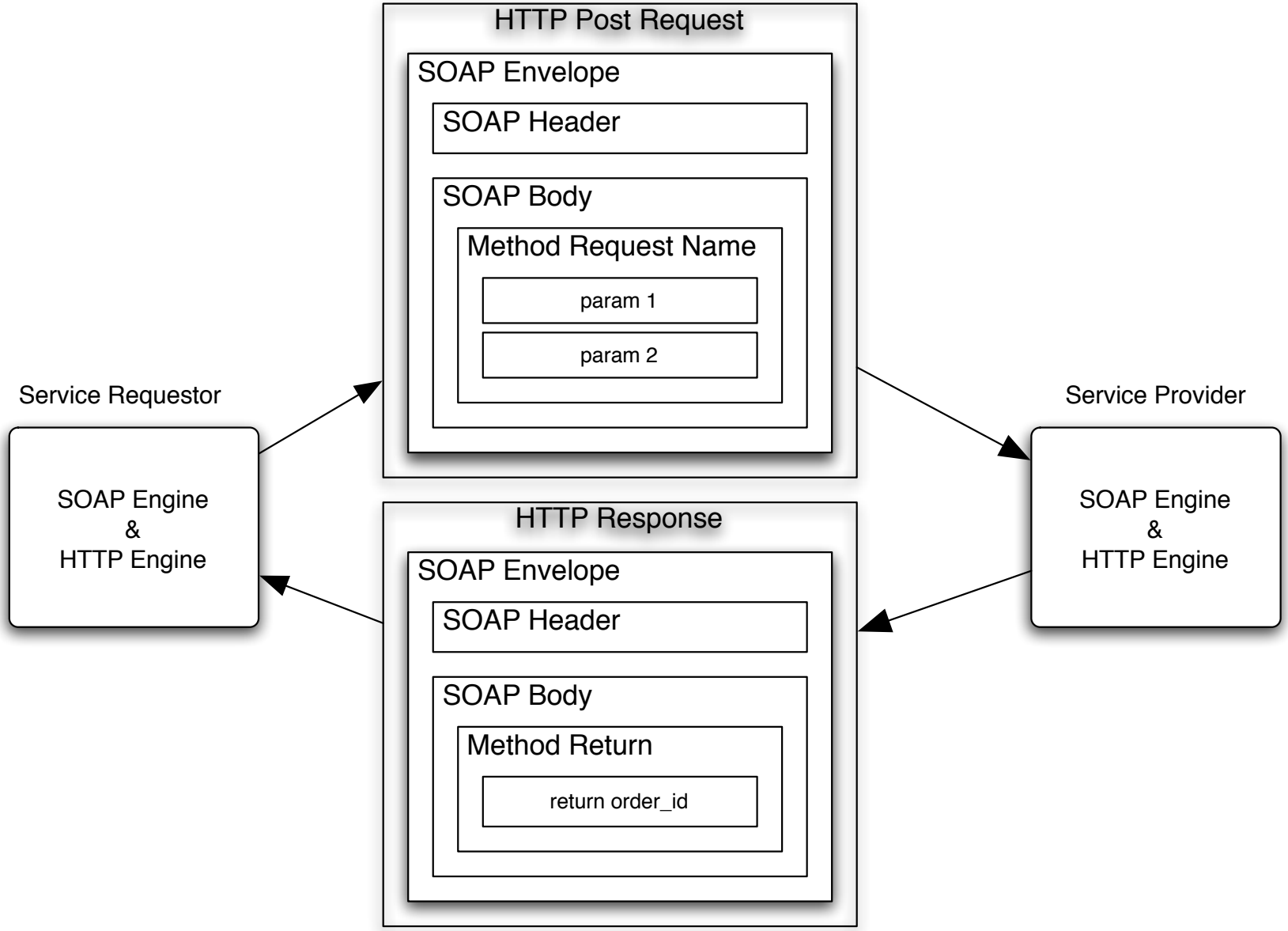


# Distributed messaging using SOAP (p.124 Mike Book)




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

# 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>
```

# An example of SOAP Binding over SMTP

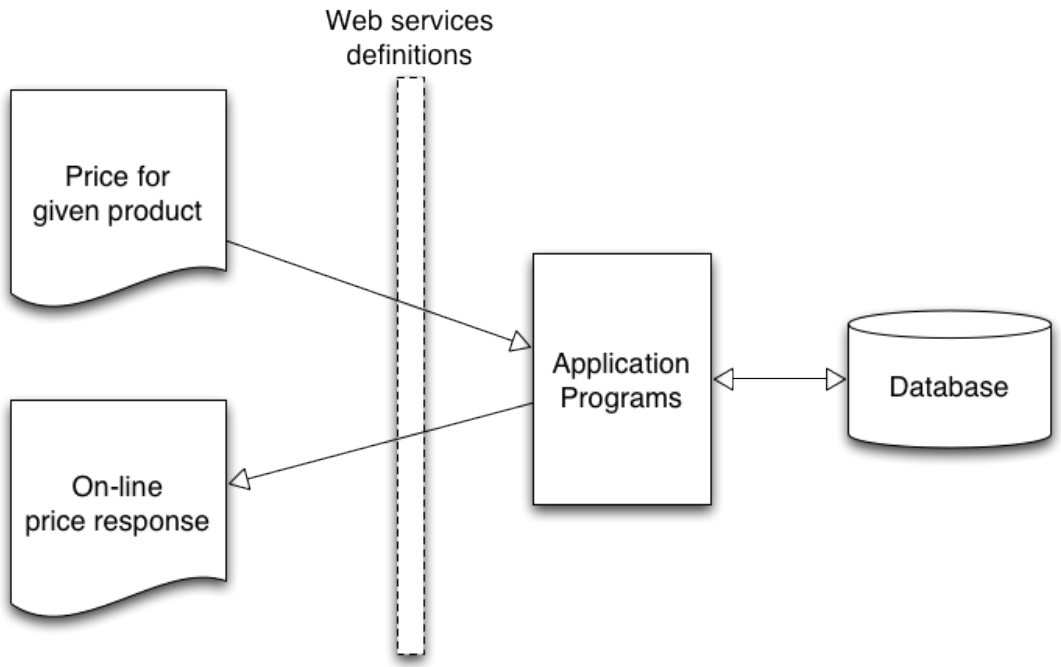
e.g., http://www.pocketsoap.com/

```
To: <soap@example.org>
From: <soap@client.com>
Reply-To: <soap@client.com>
Date: Tue, 15 Nov 2001 23:27:00 -0700
Message-Id: <1F75D4D515C3EC3F34FEAB51237675B5@client.com>
MIME-Version: 1.0
Content-Type: text/xml; charset=utf-8
Content-Transfer-Encoding: QUOTED-PRINTABLE

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
    <m:echoString xmlns:m="http://soapinterop.org/">
        <inputString>get your SOAP over SMTP here !</inputString>
    </m:echoString>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# An example of SOAP Binding over SMTP

```
To: <soap@client.com>
From: <soap@example.org>
Date: Tue, 13 Nov 2001 23:27:00 -0700
In-Reply-To: <1F75D4D515C3EC3F34FEAB51237675B5@client.com>
Message-Id: <FF75D4D515C3EC3F34FEAB51237675B5@soap.example.org>
MIME-Version: 1.0
Content-Type: TEXT/XML; charset=utf-8
Content-Transfer-Encoding: QUOTED-PRINTABLE

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
   <m:echoStringResponse xmlns:m="http://soapinterop.org/">
      <return>get your SOAP over SMTP here !</return>
   </m:echoStringResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# SOAP communication model: RPC-style




- An RPC-style WS appears as a remote object to a client
- Client express their request as a *method call* with a set of *parameters*
- Service returns a response containing a return value
- Request-response (tightly coupled and synchronous communication model)

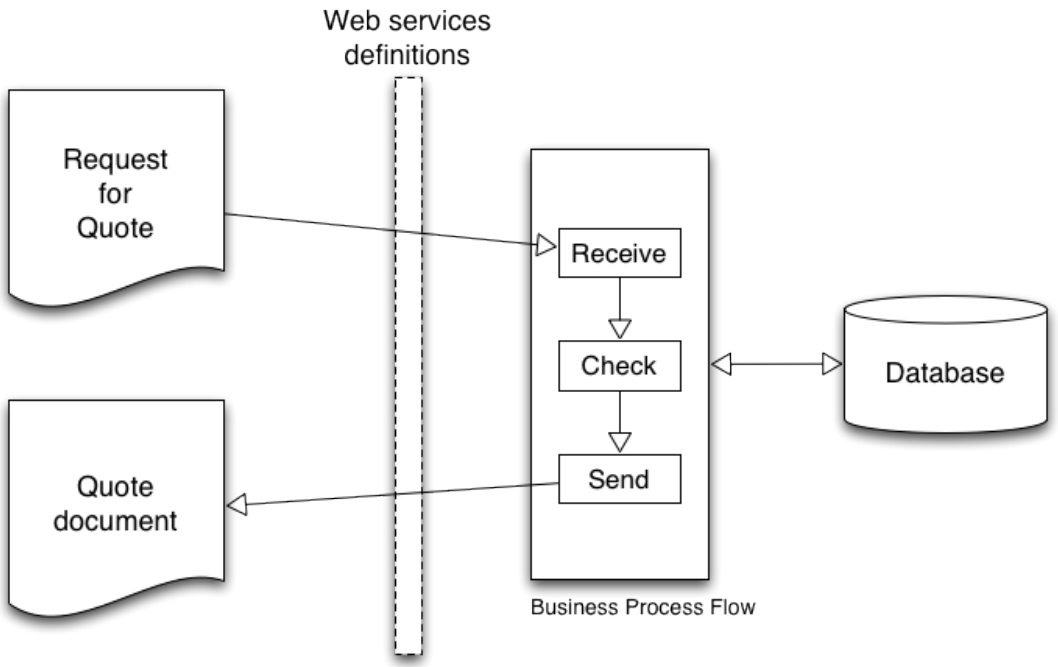
# SOAP communication model: RPC-style

Example of an RPC-style SOAP communication (p.135-138 Mike Book)

```
<env:Envelope ...>
      <env:Header> some header </env:Header>
      <env:Body>
         <m:GetProductPrice>
             <product-id> 450R60P </product-id>
         </m:GetProductPrice>
      </env:Body>
</env:Envelope>

<env:Envelope ...>
      <env:Header> some header </env:Header>
      <env:Body>
         <m:GetProductPriceResponse>
             <product-price> 134.32 </product-price>
         </m:GetProductPriceResponse>
      </env:Body>
</env:Envelope>
```

Key point: SOAP `Body` must conform to a structure that indicates the method name and contains a set of parameters

# SOAP communication model: Document-style




- SOAP actually contains 'XML data' (not a method call)
- Client sends an entire document (e.g., purchase order) rather than a discrete set of parameters
- Message driven, asynchronous communication model

# SOAP communication model: Document-style

Example of an Document-style SOAP (p.135-138 Mike Book)

```
<env:Envelope ...>
      <env:Header> some header </env:Header>
      <env:Body>
          <po:PurchaseOrder orderDate="2004-12-02">
             <po:from>
                 <po:accountName> RightPlastics </po:accountName>
                 <po:accountNumber> PSC-0343-02 </po:accountNumber>
             </po:from>
             <po:to>
                 <po:supplierName> Plastic Supplies Inc.</po:supplierName>
                 <po:supplierAddress> Yarra Valley Melbourne</po:supplierAddress>
             </po:to>
             <po:product>
                <po:product-name> injection molder </po:product-name>
                <po:product-model> G-100T </po:product-model>
                <po:quantity> 2 </po:quantity>
             <po:product>
          </po:PurchaseOrder>
      </env:Body>
</env:Envelope>
```

Key point: SOAP `Body` contains well-formed XML documents

# Summary

- The communication protocol used by all participants in WS architecture is: (...)
- Among other things, SOAP defines the message format - can you describe the main skeleton/structure of a SOAP message?
- How would you describe what SOAP engine/server is?
- SOAP can be transmitted over Internet based protocols like (...) and (...)
- Can you name the two SOAP communication styles?
- Which one of the two is suitable for asynchronous communication?

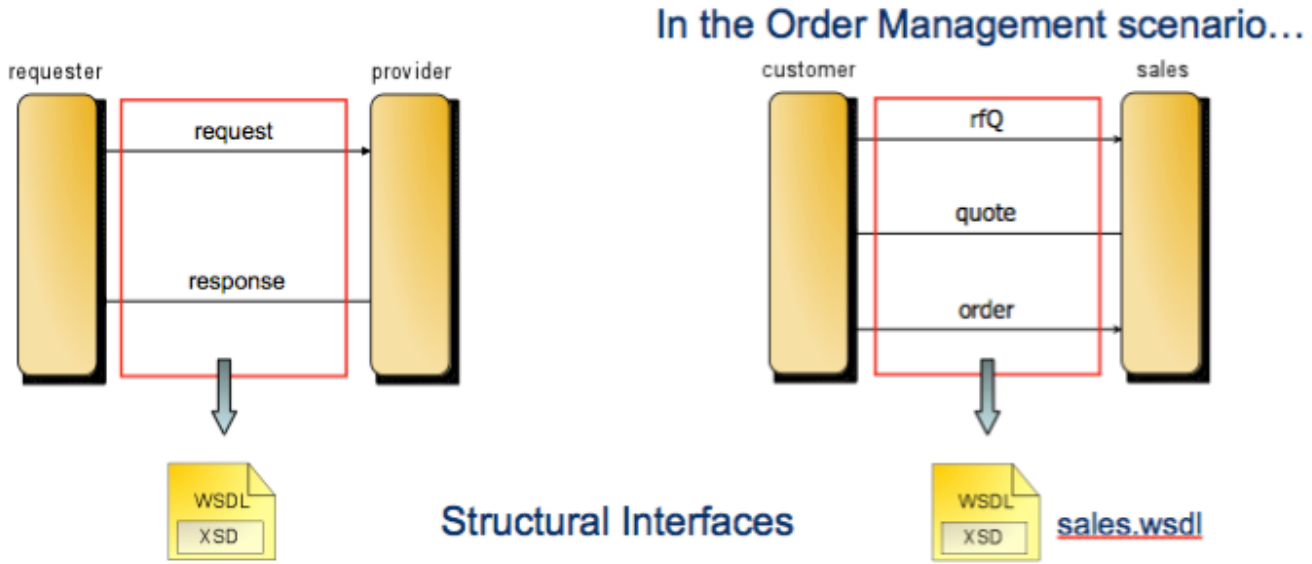
# Part III

# WSDL

# Web Services Description Language

A WS interaction typically involves two roles[3]

- requester: the initiator, who requests the service sending the first message,
- provider: the follower, who provides the service replying to the request.



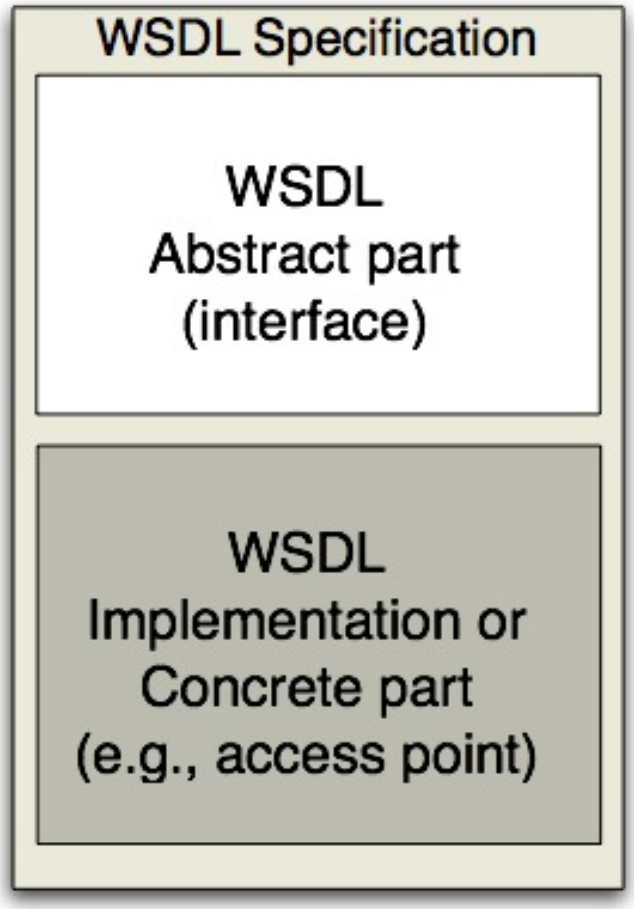

[3] Dr Marcello La Rosa, QUT, INB/N374 Intro to Web Services

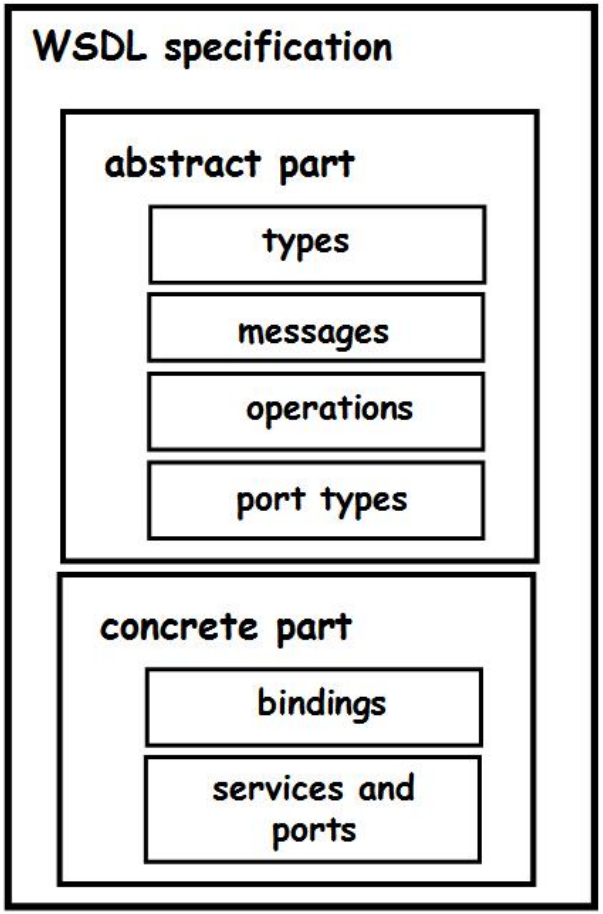
# Web Services Description Language

WSDL – pronounced "Whiz Dull"

- It is a machine-processable specification of the Web service's interface, written in Web Service Description Language (WSDL).
- Specification: `http://www.w3.org/TR/wsdl`
- It uses XML syntax. It describes a service in terms of the operations that make up the service, the messages that each operation requires, and the parts from which each message is composed.
- It is used by the client to generate a proxy (client stub) to the Web service. The proxy is then acts as a go-between between the WS and the client. This is usually done by a tool.

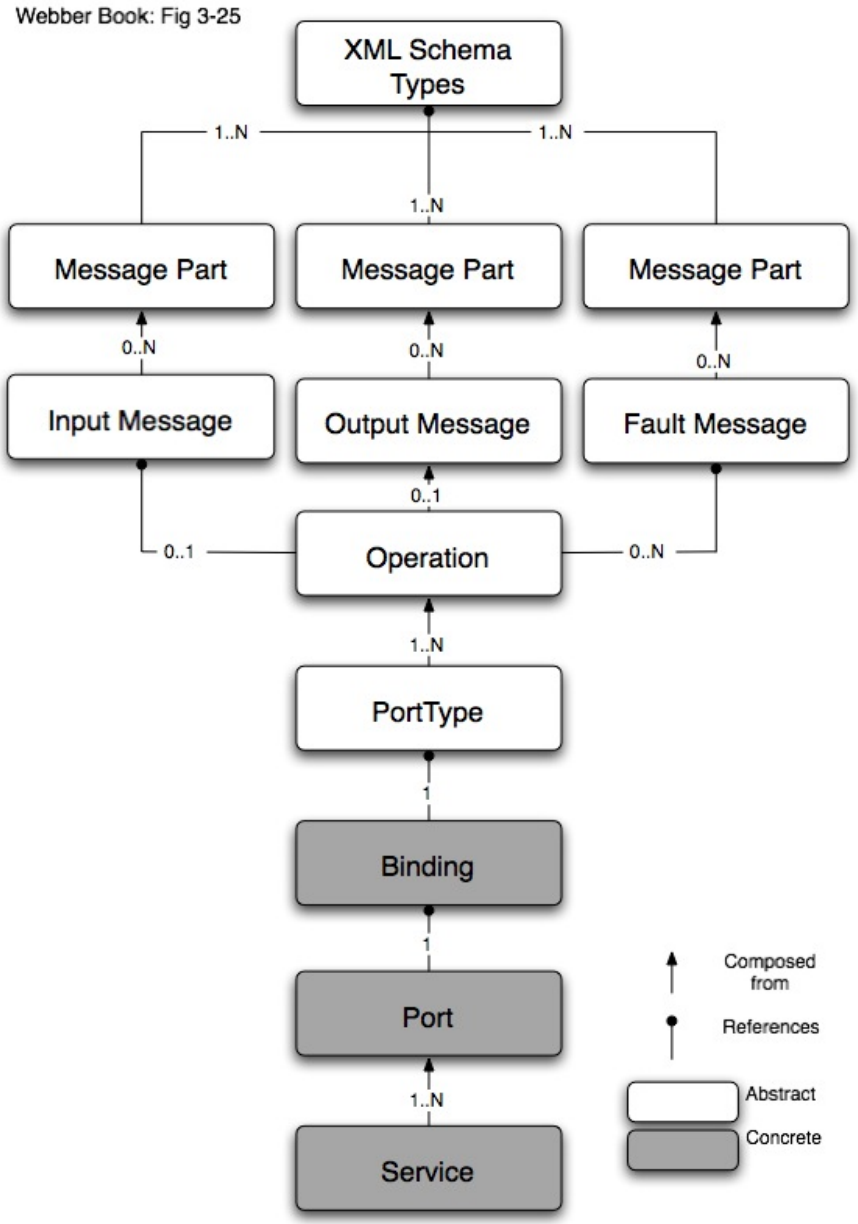
# Web Services Description Language: Two parts




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

# WSDL Structure: Webber book (p.101)



# WSDL Main Elements: `definitions`

Say ... we want to desribe 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 types/XML elements used in the interface description (XML Schema types).

```
<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>
```

- It defines what kind of messages is expected as input, output and fault by this Web service
- each message is constructed from a number of XML Schema-typed part element.

# WSDL Main Elements: portType

The portType element contains a named set of operations. It defines the functionality of the Web service (ie., 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 messages defined previously
- Note: WSDL 1.2 changes portType to interface

# WSDL Main Elements: `operation`

Not all operations will have a single input, output and fault.

`operation` indicate a message exchange pattern (transmission primitives):

- **Request-response** (i.e., Input-Output): The Web service receives a message, and sends a correlated message (or fault).
- **One-way** (i.e., Input only): The service receives a message. The service consumes the message and does not produce any output or fault message.
- **Solicit-response** (i.e., Output-Input): The service generates a message, and receives a correlated message (or fault) in return.
- **Notification** (i.e., Output only): The service sends a message. It does not expect anything in return.

Synchronous interactions are defined using request-response, and solicit-response, while asynchronous interactions are defined using one-way and notifications operations.

# Transmission primitives (general syntax)

## One-way Operation:

```
<wsdl:definitions .... >
   <wsdl:portType .... > *
        <wsdl:operation name="nmtoken">
           <wsdl:input name="nmtoken"? message="qname"/>
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

## Notification Operation:

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken">
           <wsdl:output name="nmtoken"? message="qname"/>
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

example scenarios?

# Transmission primitives (general syntax)

## Request-response Operation:

```
<wsdl:portType .... >
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
       <wsdl:input name="nmtoken"? message="qname"/>
       <wsdl:output name="nmtoken"? message="qname"/>
       <wsdl:fault name="nmtoken" message="qname"/>
    </wsdl:operation>
</wsdl:portType >
```

## Solicit-response Operation:

```
<wsdl:portType .... >
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
       <wsdl:output name="nmtoken"? message="qname"/>
       <wsdl:input name="nmtoken"? message="qname"/>
       <wsdl:fault name="nmtoken" message="qname"/>
    </wsdl:operation>
</wsdl:portType >
```

example scenarios?

# WSDL Main Elements: `binding`

The second portion of a service description involves specific (concrete) statements about how to use the service:

- A `binding` defines *message encoding format* and *protocol details* for operations and messages defined by a particular portType.

```
<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>
        <soap:body use="literal" encodingStyle="http://stock.example.org/schema"/>
     </wsdl:output>
     <wsdl:fault>
        <soap:fault name="StockSymbolNotFoundMessage"/>
     </wsdl:fault>
   </wsdl:operation>
</wsdl:binding>
```

# WSDL Main Elements: `binding`

`wsdl:binding` –

- name attribute: name of the binding (use any name you want),
- type attribute: points to the portType for the binding

`soap:binding` –

- the purpose of the SOAP binding element is to signify that the binding is bound to the SOAP protocol format: *Envelope*, *Header* and *Body*. This element makes no claims as to the encoding or format of the message.
- style attribute: this can be 'rpc' or 'document'. It determines the how the SOAP body is constructed (rpc-style or document-style).
- transport attribute: transport protocol to use (eg. HTTP, SMTP).

# WSDL Main Elements: `binding`

`wsdl:operation` –

- It maps each operation in the portType to a binding
- For each operation in the portType, specify how the input and output messages/data are encoded

`soap:operation` –

- soapAction attribute specifies the value of the SOAPAction header for this operation. For the HTTP protocol binding of SOAP, this is value required (it has no default value).

`soap:body` –

- The soap:body element specifies how the message parts appear inside the SOAP Body element.
- `http://www.w3.org/TR/wsdl#_soap:body`
- use attribute: literal (literally follow an XML Schema definition) or encoded (follow SOAP encoding specification)

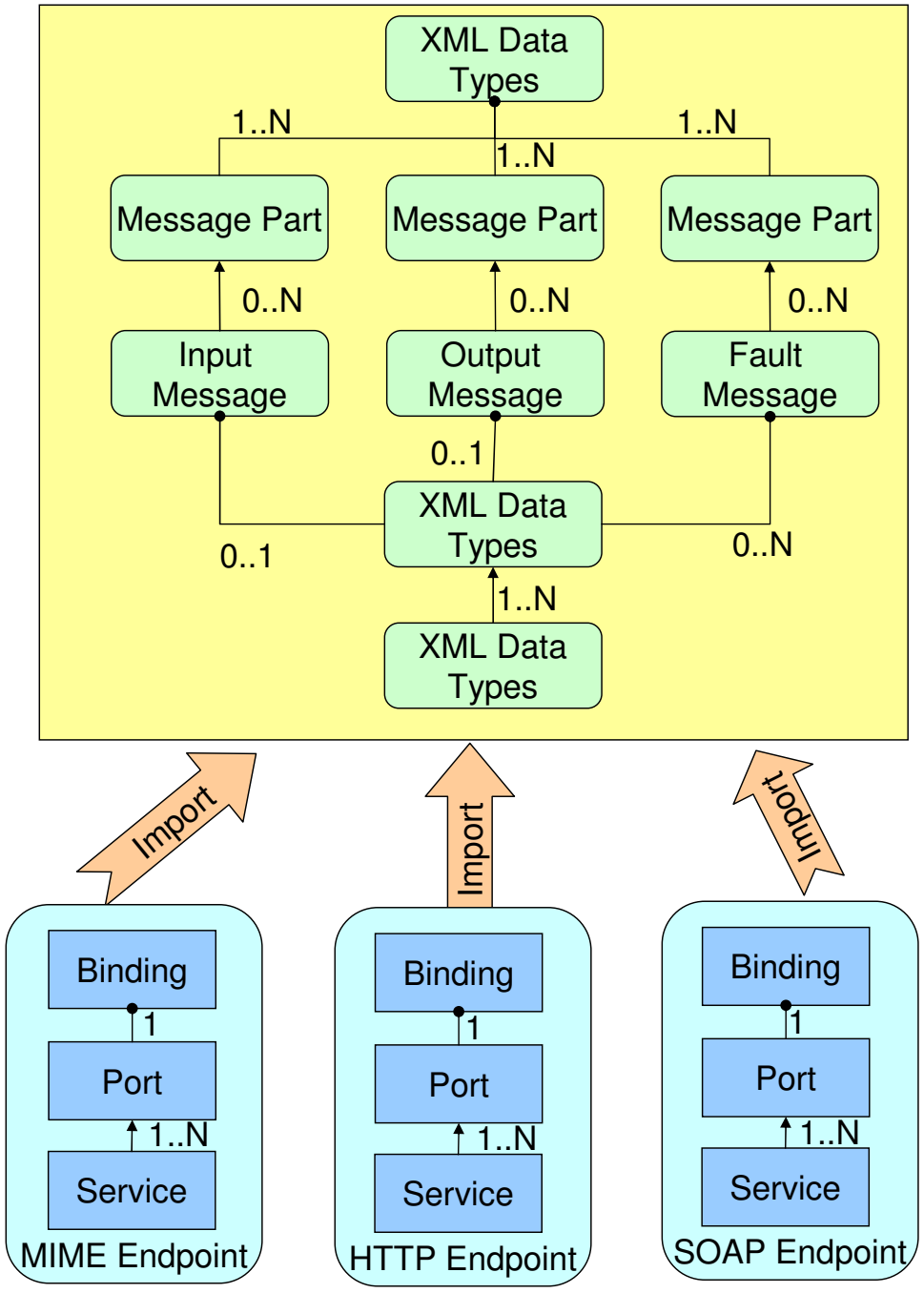
# WSDL Main Elements: `service`

The service element:

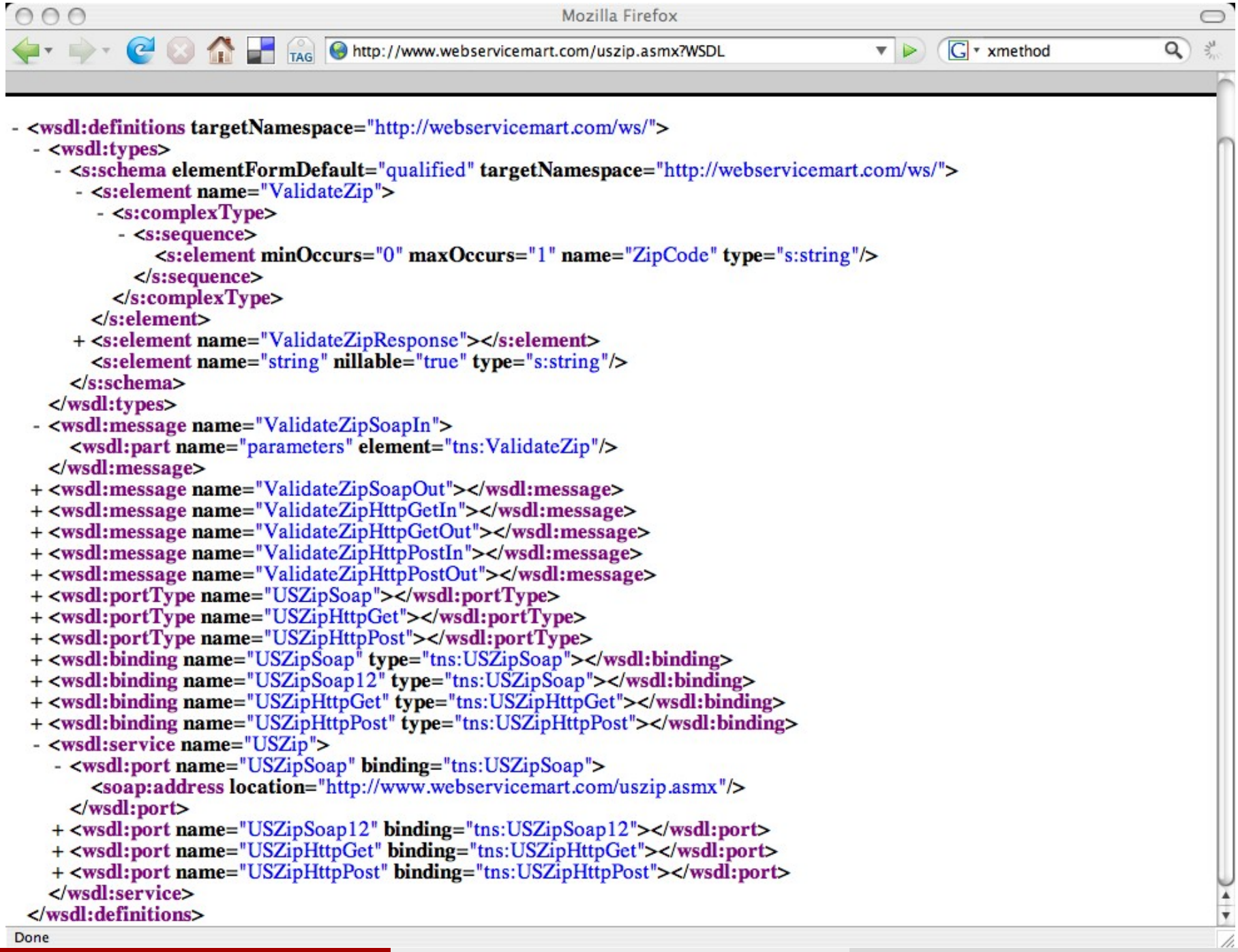
- A `port` element defines an specific network endpoint (address) for a binding.

```
<wsdl:service name="StockBrokerService">
  <wsdl:port name="StockBrokerServiceSOAPPort"
        binding="tns:StockBrokerServiceSOAPBinding>
    <soap:address location="http://stock.example.org/" />
  </wsdl:port>
</wsdl:service>
```

A `service` is a generic wrapper that groups various methods together.




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


Mozilla Firefox
http://www.webservicemart.com/uszip.asmx?WSDL
xmethod
- <wsdl:definitions targetNamespace="http://webservicemart.com/ws/">
- <wsdl:types>
- <s:schema elementFormDefault="qualified" targetNamespace="http://webservicemart.com/ws/">
- <s:element name="ValidateZip">
- <s:complexType>
- <s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="ZipCode" type="s:string"/>
</s:sequence>
</s:complexType>
</s:element>
+ <s:element name="ValidateZipResponse"></s:element>
<s:element name="string" nillable="true" type="s:string"/>
</s:schema>
</wsdl:types>
- <wsdl:message name="ValidateZipSoapIn">
<wsdl:part name="parameters" element="tns:ValidateZip"/>
</wsdl:message>
+ <wsdl:message name="ValidateZipSoapOut"></wsdl:message>
+ <wsdl:message name="ValidateZipHttpGetIn"></wsdl:message>
+ <wsdl:message name="ValidateZipHttpGetOut"></wsdl:message>
+ <wsdl:message name="ValidateZipHttpPostIn"></wsdl:message>
+ <wsdl:message name="ValidateZipHttpPostOut"></wsdl:message>
+ <wsdl:portType name="USZipSoap"></wsdl:portType>
+ <wsdl:portType name="USZipHttpGet"></wsdl:portType>
+ <wsdl:portType name="USZipHttpPost"></wsdl:portType>
+ <wsdl:binding name="USZipSoap" type="tns:USZipSoap"></wsdl:binding>
+ <wsdl:binding name="USZipSoap12" type="tns:USZipSoap"></wsdl:binding>
+ <wsdl:binding name="USZipHttpGet" type="tns:USZipHttpGet"></wsdl:binding>
+ <wsdl:binding name="USZipHttpPost" type="tns:USZipHttpPost"></wsdl:binding>
- <wsdl:service name="USZip">
- <wsdl:port name="USZipSoap" binding="tns:USZipSoap">
<soap:address location="http://www.webservicemart.com/uszip.asmx"/>
</wsdl:port>
+ <wsdl:port name="USZipSoap12" binding="tns:USZipSoap12"></wsdl:port>
+ <wsdl:port name="USZipHttpGet" binding="tns:USZipHttpGet"></wsdl:port>
+ <wsdl:port name="USZipHttpPost" binding="tns:USZipHttpPost"></wsdl:port>
</wsdl:service>
</wsdl:definitions>
Done

# Part IV

# Service Implementation and Deployment

# Web Service Development

Web service development activities:

- design/obtain a service interface (WSDL document and related skeletons)
- implement service business logic / wrap an existing implementation
- deploy service to a runtime engine
- then ... clients write applications

Web Service Development Patterns

- Bottom-up Service Development
  - Start with (Java) implementation to produce WSDL
  - Targeted at exposing existing code as Web services
- Top-down Service Development
  - Start with WSDL to produce (Java) service implementation
  - Best option for developing new Web services

# Implementing Web services with Java

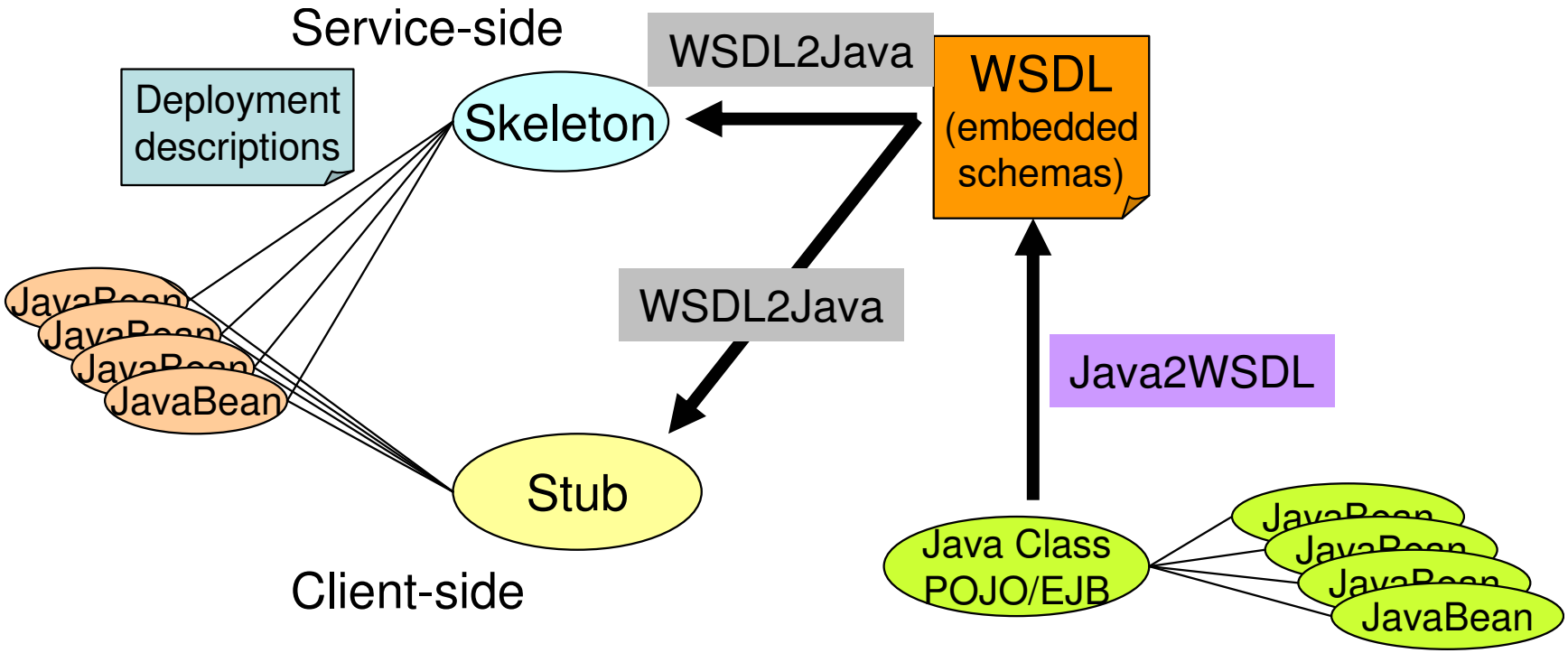
In Java, Web services are supported by two standard specifications: JAX-WS and JAX-RS

Popular options: Apache Axis2 (ws.apache.org/axis) or Apache CXF. But Both provide:

- a "SOAP engine" – a framework for clients and services to construct/process SOAP messages
- run/operated through Tomcat (as a Web application)
- provides various tools for dealing with WSDL
  - generating client (or server) code templates to access (or create) remote services from WSDL
  - data binding (XML to Java object and vice versa)
  - etc.

* note: We will use Apache CXF for the labs and assignments.

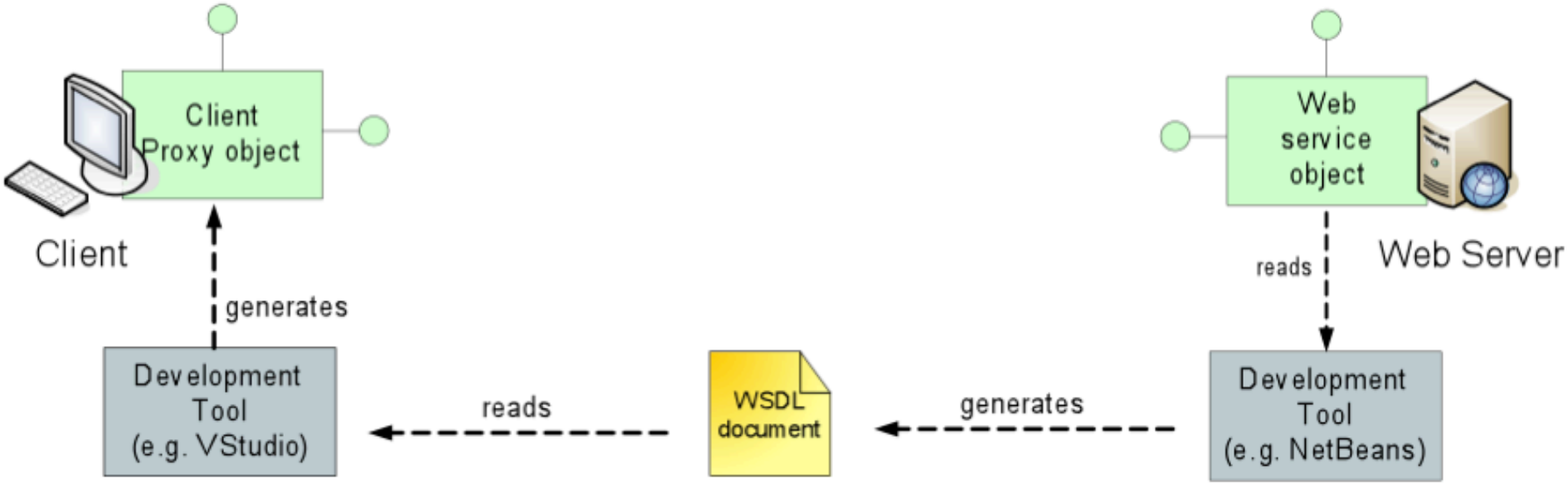
# Bottom-up Development Pattern (Summary)




- it is a quick way to expose legacy implementations as Web services
- it requires little or no knowledge of WSDL or XML
- Changes to the Java interface and package names are difficult to manage

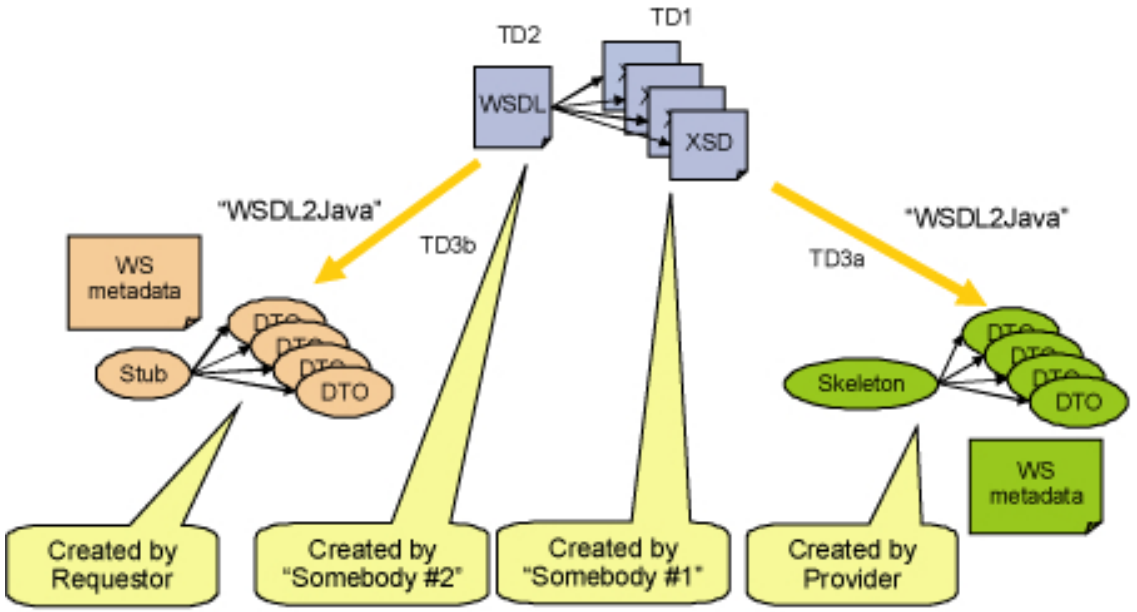
# Bottom-up Development Pattern (Summary)

Programming Language First Approach ...[4]




[4] Dr Marcello La Rosa, QUT, INB/N374 Intro to Web Services

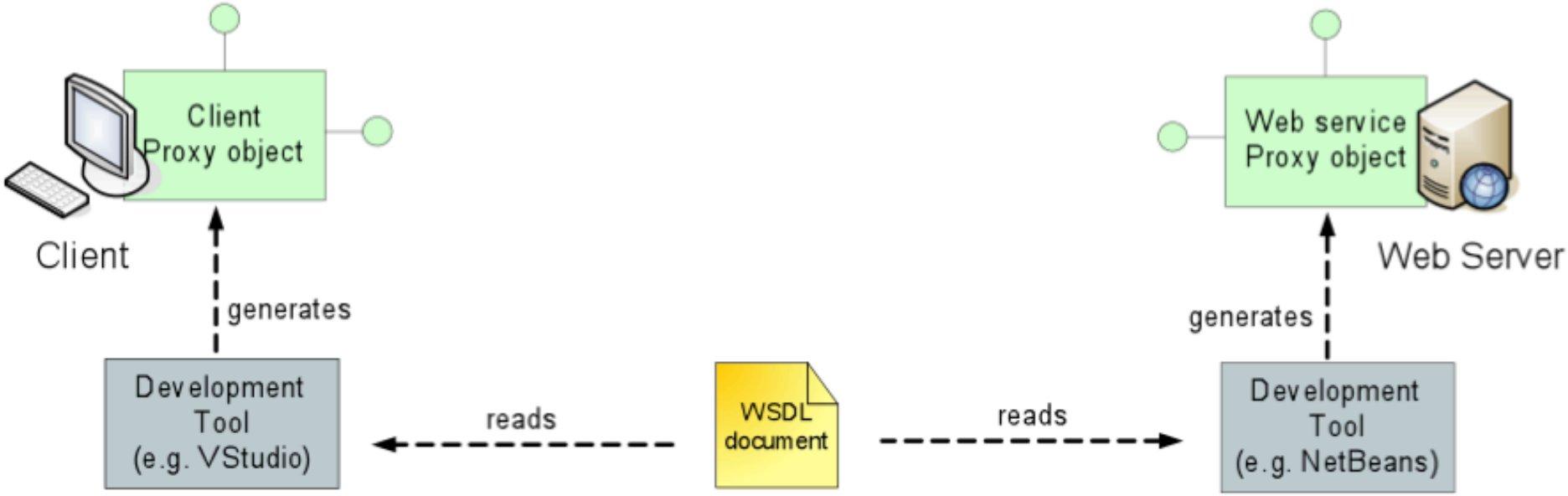
# Top-down Development Pattern (Summary)




- It supports the use of existing standards-based XSD types, allows for reuse and greater interoperability
- It requires knowledge of WSDL and XSD because both must be manually generated or manipulated

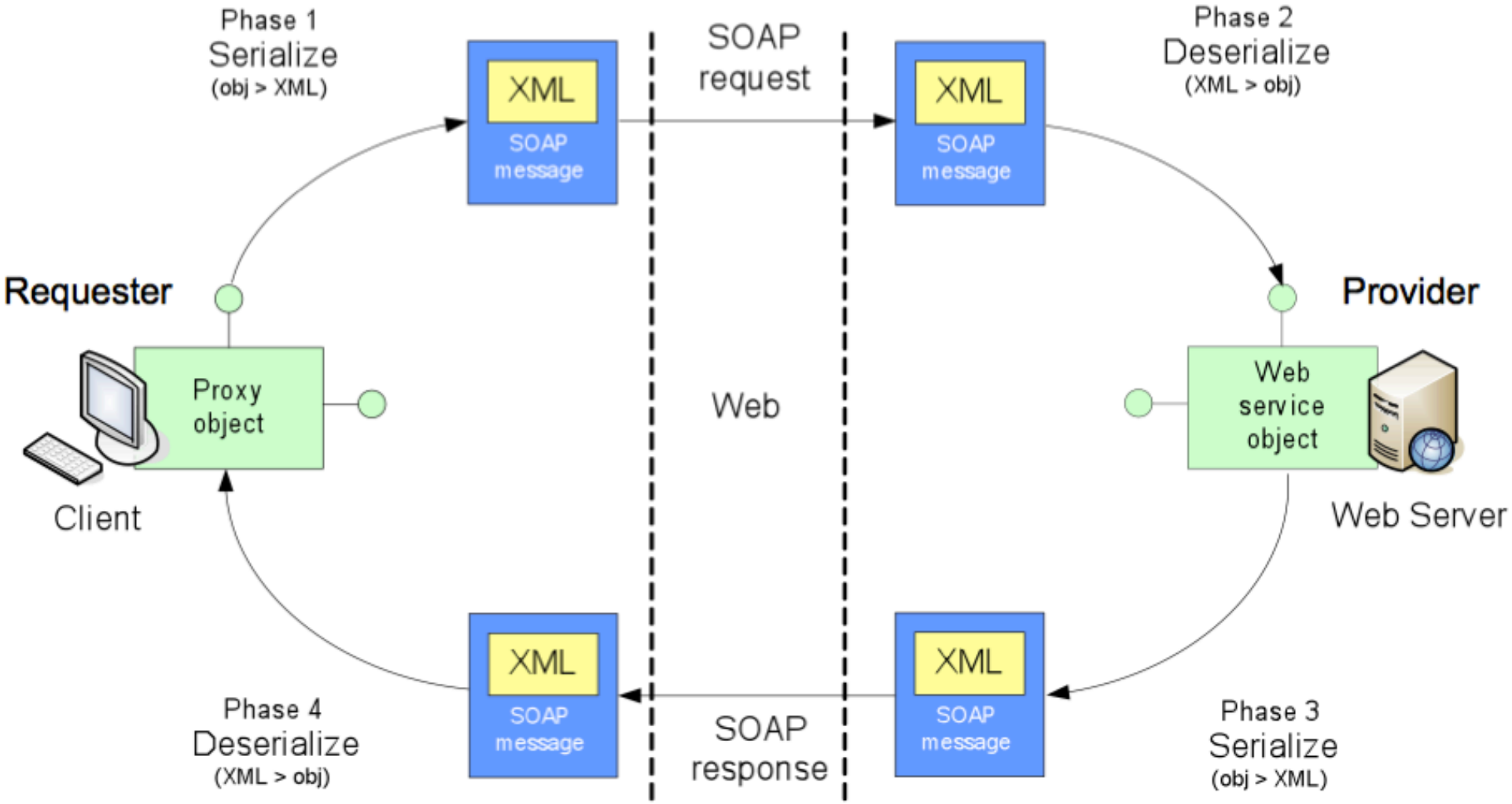
# Top-down Development Pattern (Summary)

WSDL (contract) First Approach ...[5]




[5] Dr Marcello La Rosa, QUT, INB/N374 Intro to Web Services

# SOAP/WSDL in action[6]

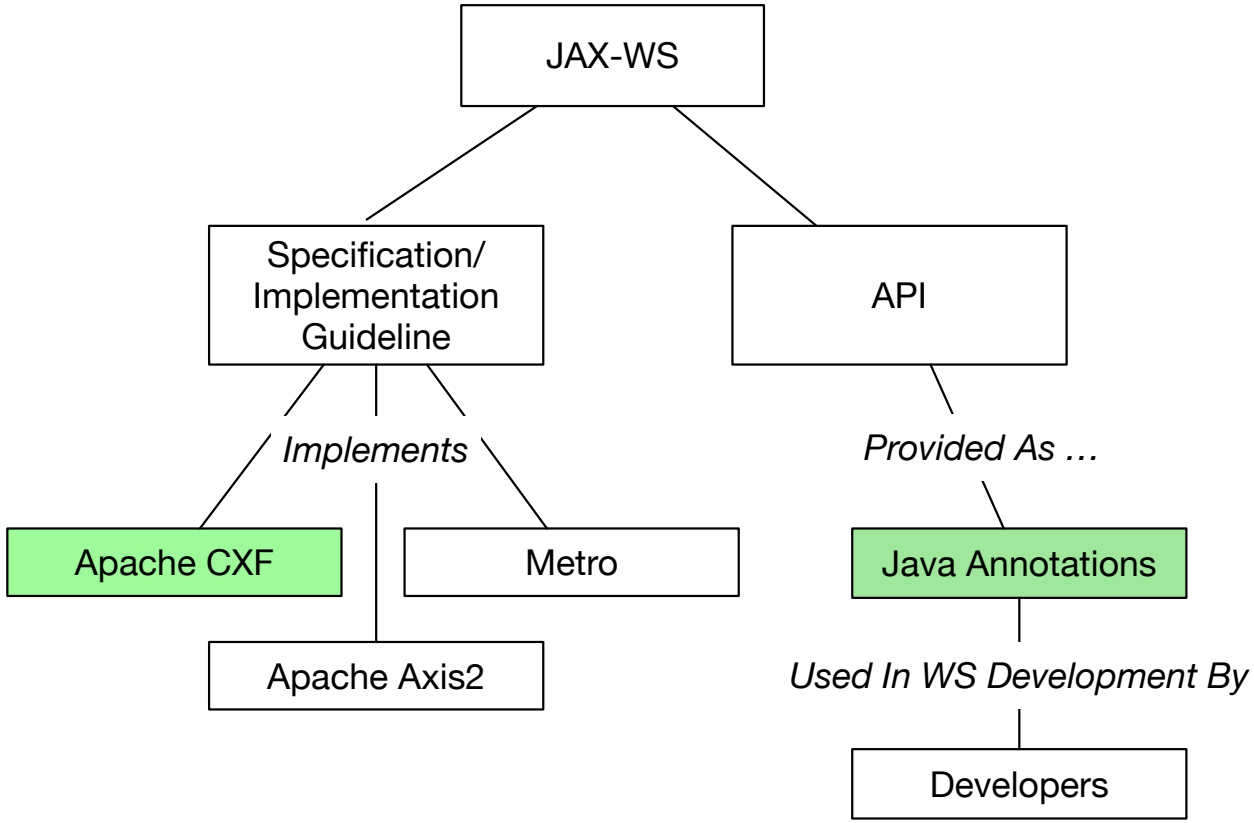



[6] Dr Marcello La Rosa, QUT, INB/N374 Intro to Web Services

# Code-first vs. Contract-first

- code-first approach sometimes can run into issues with type conversion. E.g., the use of Java Collections (such as Maps) - sometime the same semantics are not available in other languages
- benefit of logic-to-contract binding lengthens the lifespan of the service by blackboxing the implementation from the contract
  - after all, JAX-WS is referred to as 'FRONT-END' technology (i.e., application logic + Web service interface)
- re-use of 'business objects'

# On JAX-WS (Java API for XML Web Services)




- As a developer, you need to learn the JAX-WS API Annotations and mark your classes and methods with them
- At run time, Apache CXF will recognise/interpret the annotations and take proper actions

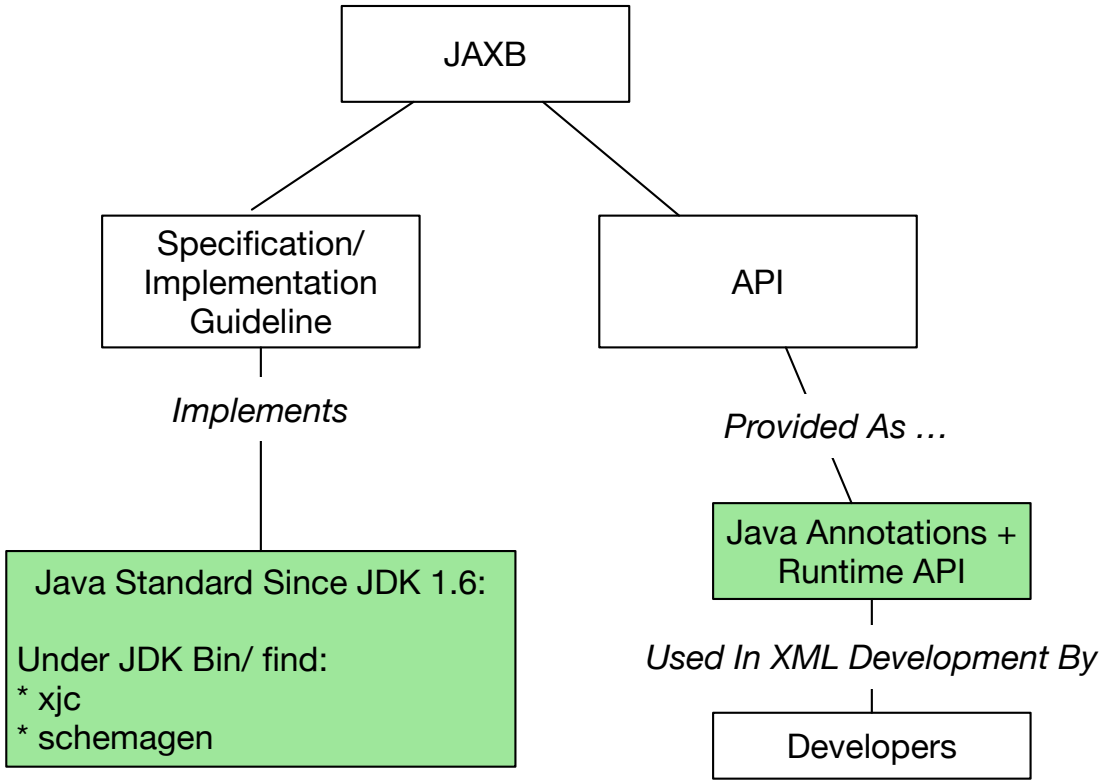
# Core JAX-WS Annotations: on the interface (match WSDL)

```
3  import javax.jws.WebMethod;
4  import javax.jws.WebParam;
5  import javax.jws.WebResult;
6  import javax.jws.WebService;
7  import javax.jws.soap.SOAPBinding;
8  import javax.xml.bind.annotation.XmlSeeAlso;
9
10 /**
11  * This class was generated by Apache CXF 3.0.4
12  * 2016-03-07T21:30:42.754+11:00
13  * Generated source version: 3.0.4
14  *
15  */
16 @WebService(targetNamespace = "http://topdown.soacourse.unsw.edu.au", name = "TopDownSimpleService")
17 @XmlSeeAlso({ObjectFactory.class})
18 @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
19 public interface TopDownSimpleService {
20
21     @WebResult(name = "downloadFileResponse", targetNamespace = "http://topdown.soacourse.unsw.edu.au", partName = "parameters")
22     @WebMethod(action = "http://topdown.soacourse.unsw.edu.au/downloadFile")
23     public DownloadFileResponse downloadFile(
24         @WebParam(partName = "parameters", name = "downloadFileRequest", targetNamespace = "http://topdown.soacourse.unsw.edu.au")
25         DownloadFileRequest parameters
26     );
27
28     @WebResult(name = "importMarketDataResponse", targetNamespace = "http://topdown.soacourse.unsw.edu.au", partName = "parameters")
29     @WebMethod(action = "http://topdown.soacourse.unsw.edu.au/importMarketData")
30     public ImportMarketDataResponse importMarketData(
31         @WebParam(partName = "parameters", name = "importMarketDataRequest", targetNamespace = "http://topdown.soacourse.unsw.edu.au")
32         ImportMarketDataRequest parameters
```

# Core JAX-WS Annotations: on the implementation class

```
3  import javax.jws.WebService;
4
5  @WebService(endpointInterface = "au.edu.unsw.soacourse.topdown.TopDownSimpleService")
6  public class TopDownSimpleServiceImpl implements TopDownSimpleService {
7
8          ObjectFactory factory = new ObjectFactory();
9          ImportMarketDataRequest req = factory.createImportMarketDataRequest();
10
11     public ImportMarketDataResponse importMarketData(ImportMarketDataRequest req) {
12         StringBuilder sbf = new StringBuilder();
13         sbf.append("Security Code: ").append(req.sec).append("\r\n");
14         sbf.append("Start date: ").append(req.startDate).append("\r\n");
15         sbf.append("End date: ").append(req.endDate).append("\r\n");
16         sbf.append("Data source: ").append(req.dataSource).append("\r\n");
17
18         ImportMarketDataResponse res = factory.createImportMarketDataResponse();
19         res.returnData = sbf.toString();
20         return res;
21     }
22
23     public DownloadFileResponse downloadFile(DownloadFileRequest req) {
24
25         DownloadFileResponse res = factory.createDownloadFileResponse();
26         res.returnData = "EventSet Id: " + req.eventSetId;
27
28         return res;
29
30     }
```

# On JAXB (Java Architecture for XML Binding)




Three tools in JAXB:

- XML Schema $\rightarrow$ Java Classes (XJC)
- Java Classes $\rightarrow$ XML Schema (SCHEMAGEN)
- Runtime API (marshalling/unmarshaling + Annotations)

# On JAXB (Java Architecture for XML Binding)

Borrowed example from Vogella.
`http://www.vogella.com/tutorials/JAXB/article.html`

# On JAXB (Java Architecture for XML Binding)

- JAXB and Apache CXF: wsdl2java, java2wsdl
- Refer to Apache CXF documentation - http://cxf.apache.org/docs/wsdl-to-java.html
- Relevant Maven Plug-ins

```
35    <plugins>
36        <plugin>
37                <groupId>org.apache.cxf</groupId>
38                <artifactId>cxf-codegen-plugin</artifactId>
39                <version>3.0.4</version>
40                <executions>
41                  <execution>
42                    <id>generate-sources</id>
43                    <phase>generate-sources</phase>
44                  <configuration>
45                      <sourceRoot>src/main/java-generated</sourceRoot>
46                      <wsdlOptions>
47                          <wsdlOption>
48                              <wsdl>${basedir}/src/main/resources/wsdl/TopDownSimpleService.wsdl</wsdl>
49                          </wsdlOption>
50                      </wsdlOptions>
51                  </configuration>
52                  <goals>
53                      <goal>wsdl2java</goal>
54                  </goals>
55                  </execution>
56                </executions>
57      </plugin>
58    </plugins>
```

# JAX-WS Service Client Development

Retrieve the target WSDL, run WSDL2Java → generated Java classes, obtain *Service* then obtain *Port* ... construct a request message and send.

```
1  package au.edu.unsw.soacourse.topdownclient;
2
3  import java.net.MalformedURLException;
4  import java.net.URL;
5
6  import au.edu.unsw.soacourse.topdown.TopDownSimpleService;
7  import au.edu.unsw.soacourse.topdown.TopDownSimpleServiceImplService;
8  import au.edu.unsw.soacourse.topdown.ObjectFactory;
9  import au.edu.unsw.soacourse.topdown.ImportMarketDataRequest;
10 import au.edu.unsw.soacourse.topdown.ImportMarketDataResponse;
11
12 public class TopDownServiceWSClient {
13
14     public static void main(String[] args) {
15         try {
16             TopDownSimpleServiceImplService service = new TopDownSimpleServiceImplService(new URL(
17                     "http://localhost:8080/SimpleServices/TopDownSimpleService?wsdl"));
18             TopDownSimpleService port = service.getTopDownSimpleServiceImplPort();
19             ImportMarketDataRequest request = new ObjectFactory().createImportMarketDataRequest();
20             request.setSec("ABC");
21             request.setStartDate("01-DEC-2014");
22             request.setEndDate("01-DEC-2014");
23             request.setDataSource("http://www.example.org.ss/marketfile");
24
25             ImportMarketDataResponse response = port.importMarketData(request);
26             System.out.println("Returned from the Service: \n" + response.getReturnData());
27         } catch (MalformedURLException e) {
28             e.printStackTrace();
29         }
30     }
```

# Part V

# More on SOAP/WSDL

# Designing WS Interface with SOAP/WSDL

## Given an operation:

```
Operation name: concat
Parameters: (st1: string, st2: string)
Return: string
```

## More precisely as WS operation ...

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: st1
  Type: string in http://www.w3.org/2001/XMLSchema
Part 2:
  Name: st2
  Type: string in http://www.w3.org/2001/XMLSchema
Output message:
Part 1:
   Name: return
   Type: string in http://www.w3.org/2001/XMLSchema
```

# Designing WS Interface with SOAP/WSDL - RPC style

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: st1
  Type: string in http://www.w3.org/2001/XMLSchema
Part 2:
  Name: st2
  Type: string in http://www.w3.org/2001/XMLSchema
Output message:
Part 1:
   Name: return
   Type: string in http://www.w3.org/2001/XMLSchema
```

```
<ese:concat xmlns:ese="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance">
    <st1 xsi:type="xsd:string">service</st1>
    <st2 xsi:type="xsd:string">foundry</st2>
</ese:concat>
```

```
<ese:concatResponse xmlns:ese ="http://sltf.unsw.edu.au/eg"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance">
   <return xsi:type="xsd:string">service foundry</return>
</ese:concatResponse>
```

# Designing WS Interface w/ SOAP/WSDL: Document-style

(note - We first should define the message types)

```
<xsd:schema targetNamespace="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="concatRequest">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="st1" type="xsd:string"/>
            <xsd:element name="st2" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

*Type of concatRequest defined as a schema*

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: concatRequest
  Element: concatRequest in http://sltf.unsw.edu.au/eg
Output message:
...
```

# Designing WS Interface w/ SOAP/WSDL: Document-style

```
<xsd:schema targetNamespace="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="concatRequest">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="st1" type="xsd:string"/>
            <xsd:element name="st2" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="concatResponse" type="xsd:string"/>
</xsd:schema>
```

*Type of concatResponse also defined as a schema*

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: concatRequest
  Element: concatRequest in http://sltf.unsw.edu.au/eg
Output message:
Part 1:
  Name: concatResponse
  Element: concatResponse in http://sltf.unsw.edu.au/eg
```

# Designing WS Interface w/ SOAP/WSDL: Document-style

```
<xsd:schema targetNamespace="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="concatRequest">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="st1" type="xsd:string"/>
            <xsd:element name="st2" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="concatResponse" type="xsd:string"/>
</xsd:schema>
```

*These SOAP messages can be validated against the schema*

```
<ese:concatRequest xmlns:ese="http://sltf.unsw.edu.au/eg">
    <st1>service</st1>
    <st2>foundry</st2>
</ese:concatRequest>
```

```
<ese:concatResponse xmlns:ese ="http://sltf.unsw.edu.au/eg">
   service foundry
</ese:concatResponse>
```

# RPC and Document Binding Options in WSDL

- Option 1: (see SimpleConcatRPC as example)
  - soap:binding, style=rpc, transport=http
  - soap:body, use=literal ('encoded' option not supported)
- Option 2: (see SimpleConcatDOC as example)
  - soap:binding, style=document, transport=http
  - soap:body, use=literal

Note: communication pattern – both could be done in RPC/synchronous style

# Fault Handling in SOAP

A SOAP `Fault` message is reserved for providing an extensible mechanism for transporting structured and unstructured information about problems that have arisen during the processing of SOAP messages.

Because clients can be written on a variety of platforms using different languages, there must exist a standard, platform-independent mechanism for communicating the error. SOAP provides a platform-independent way of describing the error within the SOAP message using a SOAP fault.

- `Fault` element is located inside body of the message
- Two mandatory elements: `Code` and `Reason`
- `Code` - e.g., `VersionMismatch`, `MustUnderStand`, `DataEncodingUnknown`, `Sender`
- `Reason` - Human readable description of the fault
- and other elements (see documents: JAX-WS SOAP Faults)

# SOAP faults appear in the SOAP body section

e.g., SOAP v1.2

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" ...
<env:Body>
   <env:Fault>
      <env:Code>
         <env:Value>env:Sender</env:Value>
         <env:Subcode>
            <env:Value>rpc:BadArguments</env:Value>
         </env:Subcode>
      </env:Code>
      <env:Reason>
           <env:Text xml:lang="en-US">Processing error</env:Text>
      </env:Reason>
      <env:Detail>
           <e:myFaultDetails xmlns:e="http://travelcompany.example.org/faults">
              <e:message>Name does not match card number</e:message>
              <e:errorcode>098</e:errorcode>
           </e:myFaultDetails>
      </env:Detail>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

# SOAP faults appear in the SOAP body section

e.g., SOAP v1.1 (simpler structure)

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap='http://schemas.xmlsoap.org/soap/envelope'>
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:VersionMismatch</faultcode>
         <faultstring, xml:lang='en">
            Message was not SOAP 1.1 compliant
         </faultstring>
         <faultactor>
            http://sample.org.ocm/jws/authnticator
         </faultactor>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

```
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://examples/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap…
   targetNamespace="http://examples/" name="HelloWorldService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://examples/" schemaLocation="http://localhost:…/>
    </xsd:schema>
  </types>
  <message name="sayHelloWorld">
    <part name="parameters" element="tns:sayHelloWorld" />
  </message>
  <message name="sayHelloWorldResponse">
    <part name="parameters" element="tns:sayHelloWorldResponse" />
  </message>
  <message name="MissingName">
    <part name="fault" element="tns:MissingName" />
  </message>
  <portType name="HelloWorld">
    <operation name="sayHelloWorld">
      <input message="tns:sayHelloWorld" />
      <output message="tns:sayHelloWorldResponse" />
      <fault message="tns:MissingName" name="MissingName" />
    </operation>
  </portType>
  <binding name="HelloWorldPortBinding" type="tns:HelloWorld">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="sayHelloWorld">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
       </output>
      <fault name="MissingName">
        <soap:fault name="MissingName" use="literal" />
      </fault>
    </operation>
  </binding>
  <service name="HelloWorldService">
    <port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
      <soap:address
        location="http://localhost:7001/HelloWorld/HelloWorldService" />
    </port>
  </service>
```

Modelled SOAP faults - in WSDL definition (example from Oracle Docs)

# Modelled SOAP faults - in service-side code

***Web services throws the custom Exception- MissingName***

```
package examples;
import javax.jws.WebService;

@WebService(name="HelloWorld", serviceName="HelloWorldService")
public class HelloWorld {
   public String sayHelloWorld(String message) throws MissingName {
        System.out.println("Say Hello World: " + message);
        if (message == null || message.isEmpty()) {
           throw new MissingName();
        }
      return "Here is the message: '" + message + "'";
      }
}
```

***Custom Exception (MissingName.java)***

```
package examples;
import java.lang.Exception;

public class MissingName extends Exception {
    public MissingName() {
        super("Your name is required.");
    }
}
```

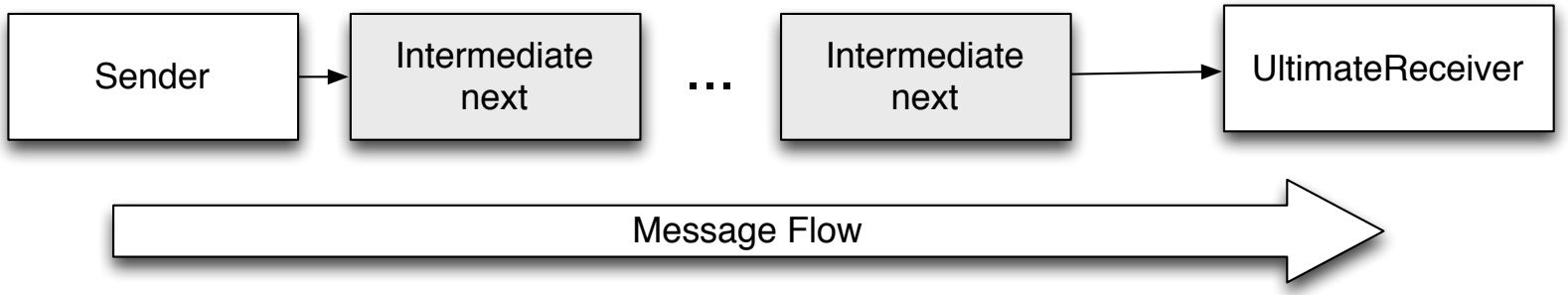
# Modelled SOAP faults - in SOAP

The following shows how the SOAP fault is communicated in the resulting SOAP message when the MissingName Java exception is thrown.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Your name is required.</faultstring>
         <detail>
            <ns2:MissingName xmlns:ns2="http://examples/">
               <message>Your name is required.</message>
            </ns2:MissingName>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

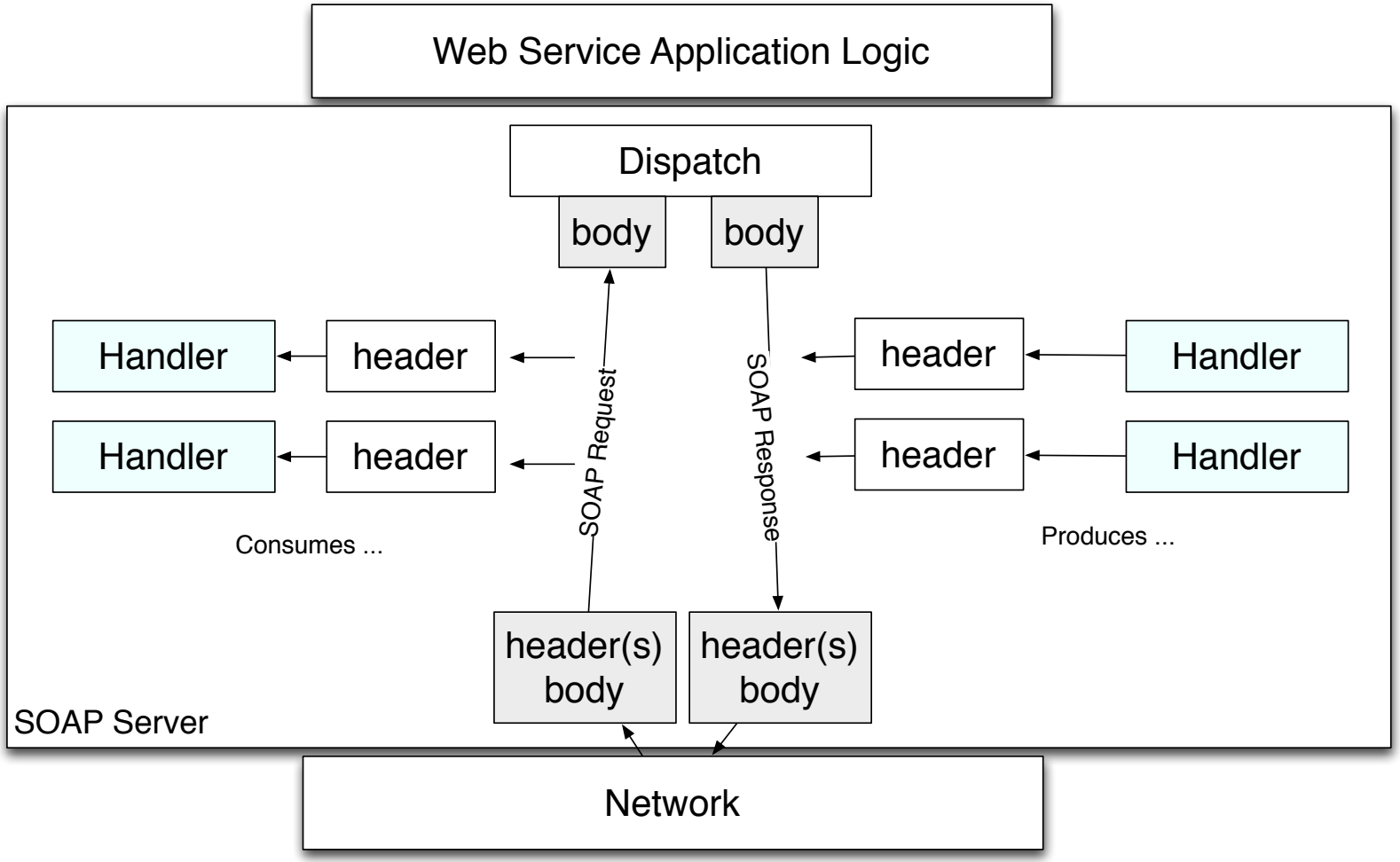
# SOAP Message Processing Model

- SOAP engines provide a message processing model that assumes that a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries.
- The messages pass through a number of intermediate *nodes* between the sender and the receiver.
  - Initial Sender: The message originator
  - Ultimate Receiver: The intended recipient
  - Intermediaries: Processing blocks that operate on the soap message before it reaches the ultimate receiver



# SOAP processing model

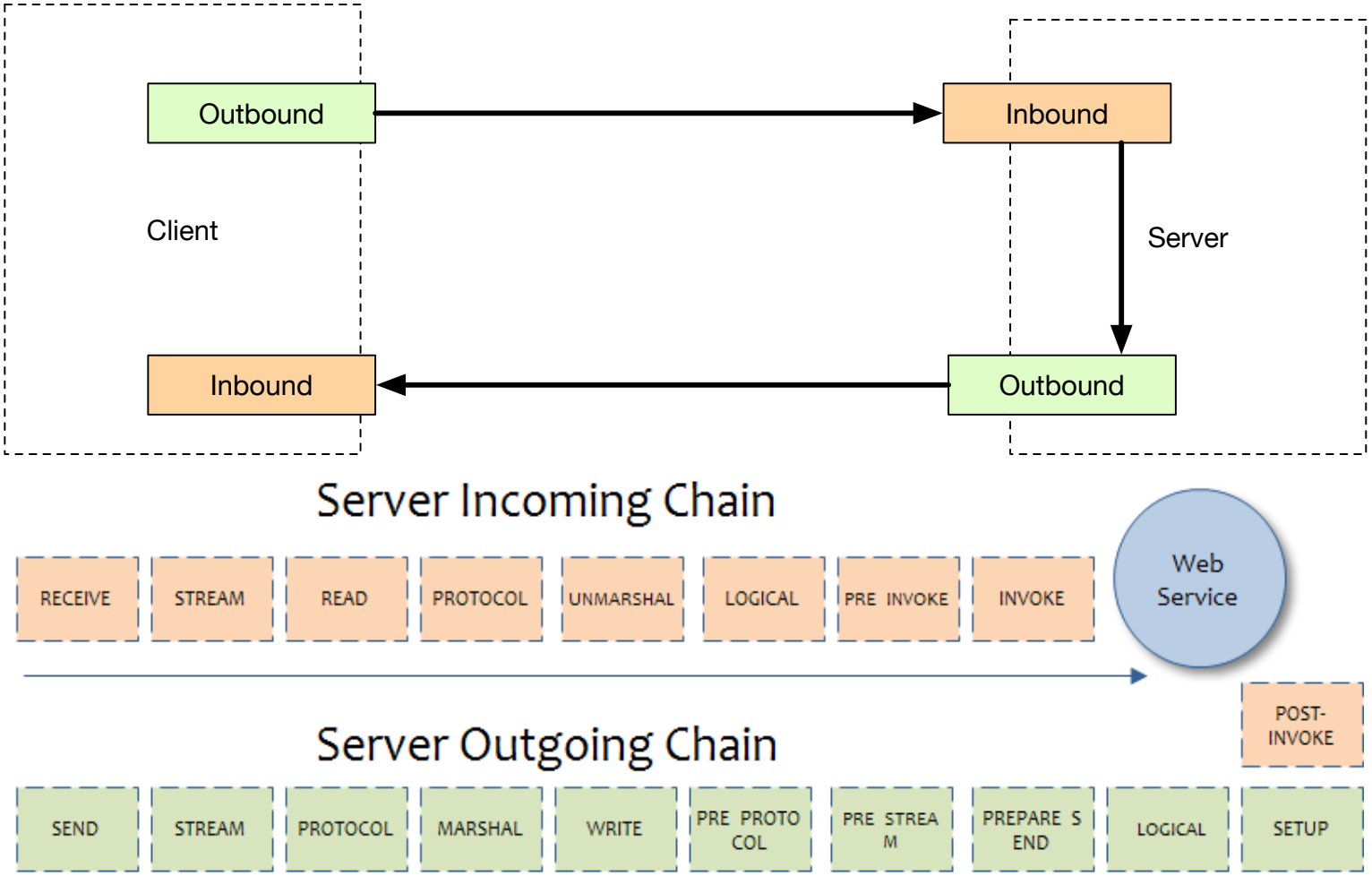
In a SOAP server (handlers == nodes):




Request: process the header blocks, Response: generates the header blocks

# e.g., Apache CXF - Interceptor model

A SOAP engine "implements" a model of their own based on these basics. Apache CXF uses Inbound, Outbound chains, Phases and Interceptors



# e.g., Apache CXF - Interceptor model

Developing an interceptor, regardless of its functionality, always follows the same basic procedure:

- Determine which abstract interceptor class to extend.
- Determine the phase in which the interceptor will run.

```
public class HelloWorldInterceptor extends AbstractPhaseInterceptor
  public HelloWorldInterceptor()
    super(Phase.INVOKE); // Put this interceptor in this phase

  public void handleMessage(Message message) throws Fault
     // do the intercepting here ...
```

- Implement the interceptor's message processing logic.
- Attach the interceptor to one of the endpoint's interceptor chains (configuration).

# SOAP Processing Model

The intermediaries work by intercepting messages, performing their function, and forwarding the (altered) message to the ultimate receiver. Common examples of intermediaries would be:

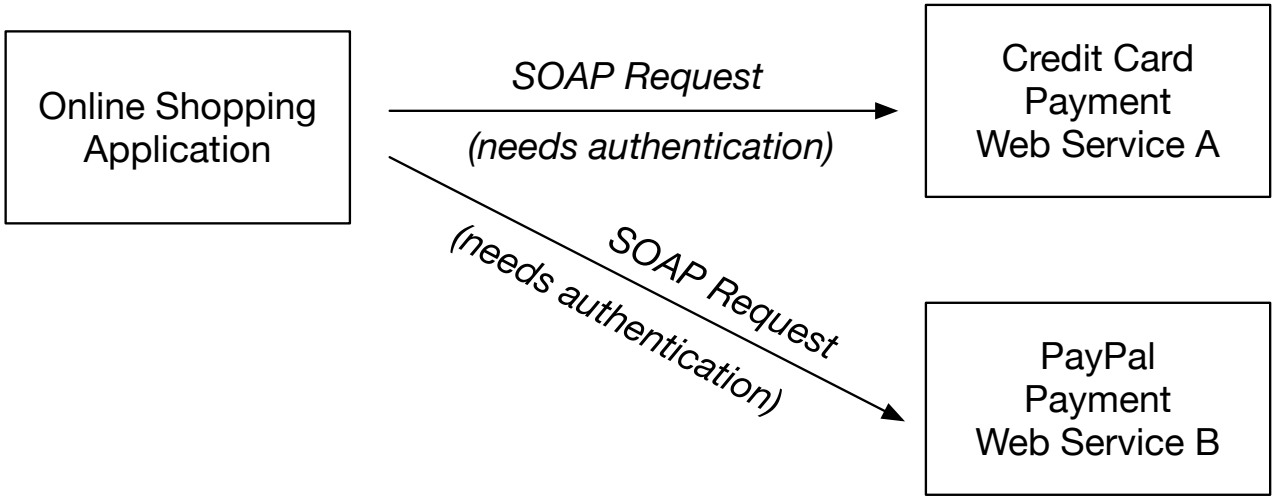
- logging
- encryption/decryption intermediary
- auditing purpose (e.g., persisting messages for billing or compliance)

Above all, using this model, it is possible to build 'extensions' to basic SOAP (e.g., supporting transaction, different security standards) ...

These extensions are called WS-* Standards.

# SOAP Processing Model and WS-* Standards

Why WS-* Standards? Take a security (e.g., user authentication) scenario.

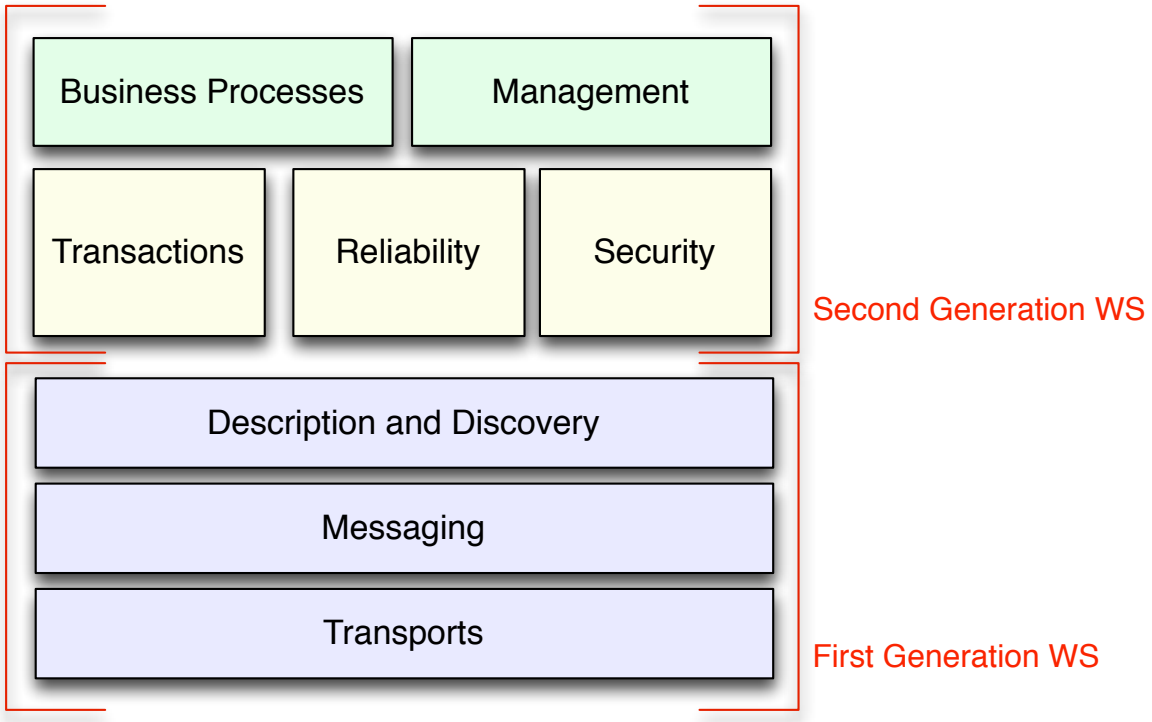



Where should the authentication information be placed? SOAP Body? SOAP Header? What should be the names of the XML elements? Could/should Web Service A and Web Service B require their own XML elements?

# Web Service Standards: WS-* extensions

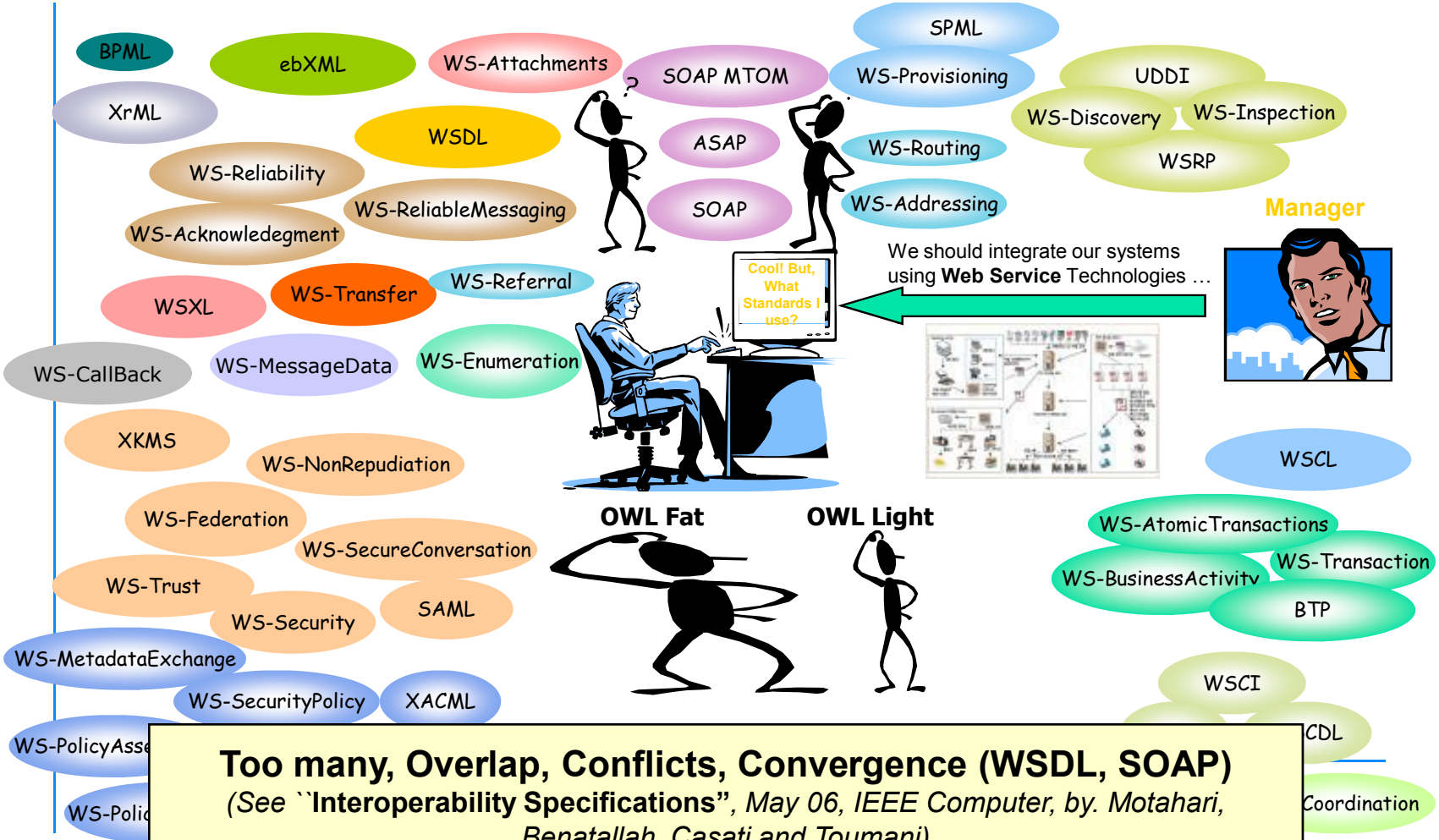
- The term "WS-*" refers to the second generation of Web services standards/specifications.
- On top of the basic standards (WSDL, SOAP and UDDI), these extensions focus on providing supports for various issues in enterprise computing environment

http://www.ibm.com/developerworks/webservices/standards/

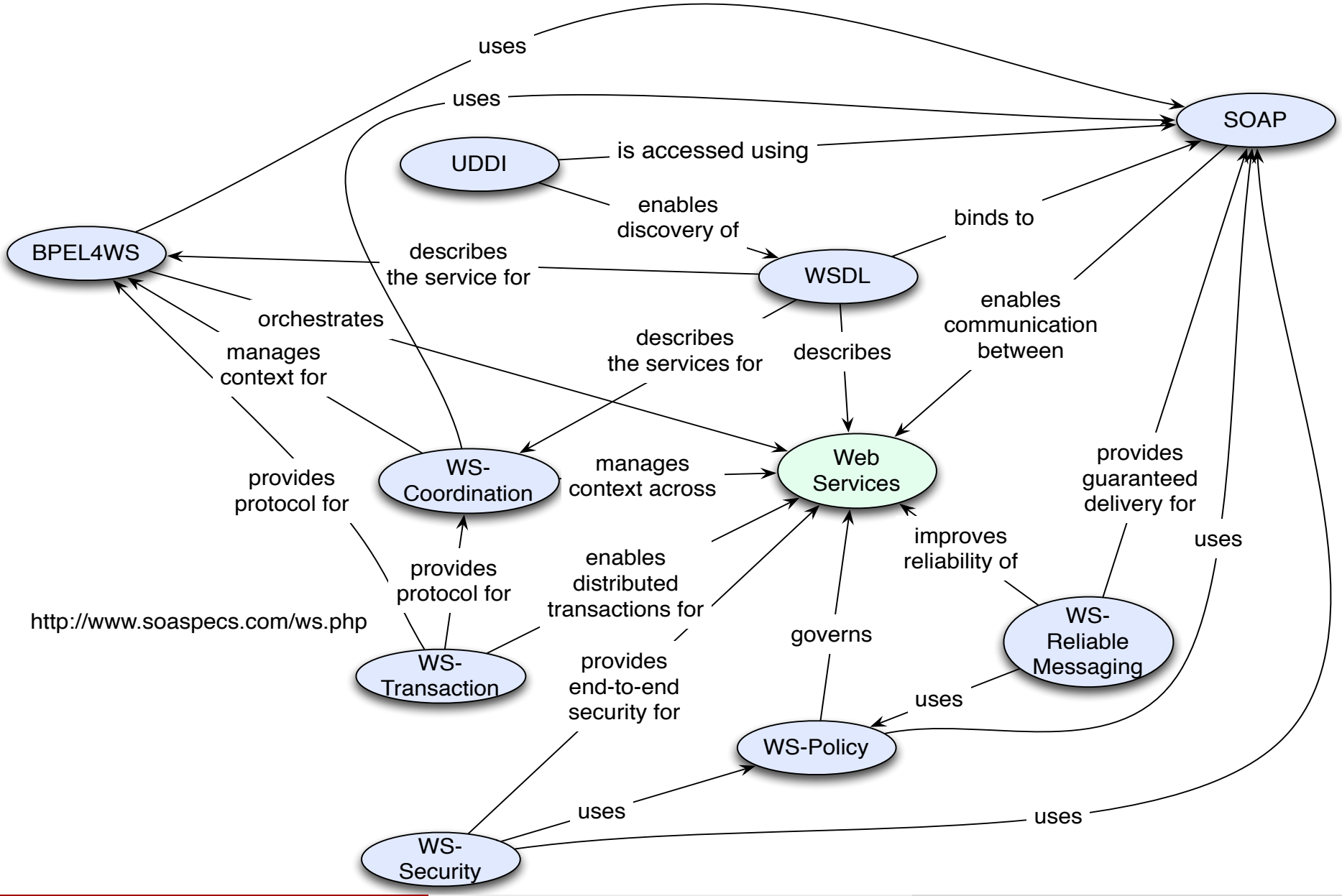


# Web Service Standards: WS-* extensions

e.g., http://www.ibm.com/developerworks/webservices/standards/



# WS-* extensions and their relationships



# WS-* extensions

Just a few "quick" examples:

- **WS-Security**
  http://www.ibm.com/developerworks/webservices/library/ws-security.html
  e.g., Apache Rampart

- **WS-ReliableMessaging**
  http://www.ibm.com/developerworks/library/specification/ws-rm
  e.g., Apache Sandesha2

- **WS-Policy**
  http://www.ibm.com/developerworks/webservices/library/ws-policy.html

# WS-* Standards

Header blocks could contain information that influences payload processing (e.g., WS-security standard: credentials info that helps control access to an operation)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
   <wsse:UsernameToken wsu:Id="sample"
       xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
    <wsse:Username>sample</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">oracle</wsse:Password>
    <wsu:Created>2004-05-19T08:44:51Z</wsu:Created>
   </wsse:UsernameToken>
  </wsse:Security>
  <wsse:Security soap:actor="oracle"
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
   <wsse:UsernameToken wsu:Id="oracle"
       xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
    <wsse:Username>oracle</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">oracle</wsse:Password>
    <wsu:Created>2004-05-19T08:46:04Z</wsu:Created>
   </wsse:UsernameToken>
  </wsse:Security>
 </soap:Header>
 <soap:Body>
  <getHello xmlns="http://www.oracle.com"/>
 </soap:Body>
```

# WS-* extensions: Reliable Messaging (Blue Book Chap. 7)

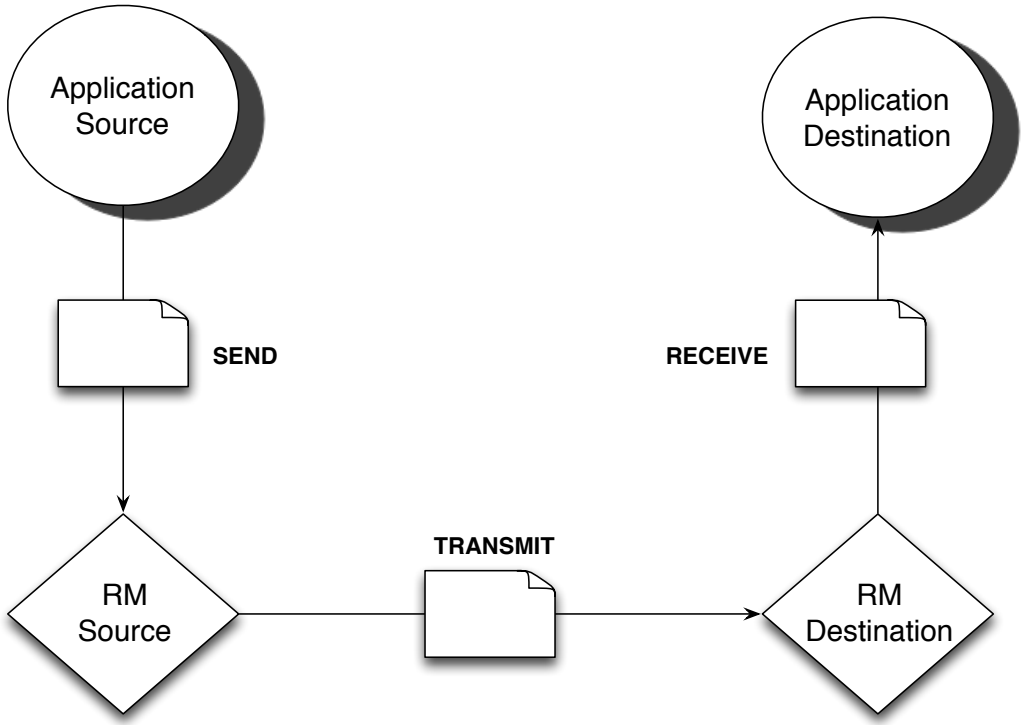
After a Web service transmit a message, it has no immediate way of knowing whether:

- the message successfully arrived at its destination
- the message failed to arrive and therefore requires a retransmission
- a series of messages arrived in the sequence they were intended to

Web service reliable messaging is a framework that enables an application running on one application server to reliably invoke a Web service running on another application server, assuming that both servers implement the WS-ReliableMessaging specification.

Reliable is defined as the ability to guarantee message delivery between the two endpoints (Web service and client) in the presence of software component, system, or network failures.

# Reliable Messaging (Blue Book Chap. 7)




- WS-RM separates/abstracts 'initiating messaging' from 'performing actual transmission'
- e.g., application source is the service that *sends* the message to the RM source (the physical processor/node that performs the actual wire *transmission*.

# Reliable Messaging (Blue Book Chap. 7)

Sequences:

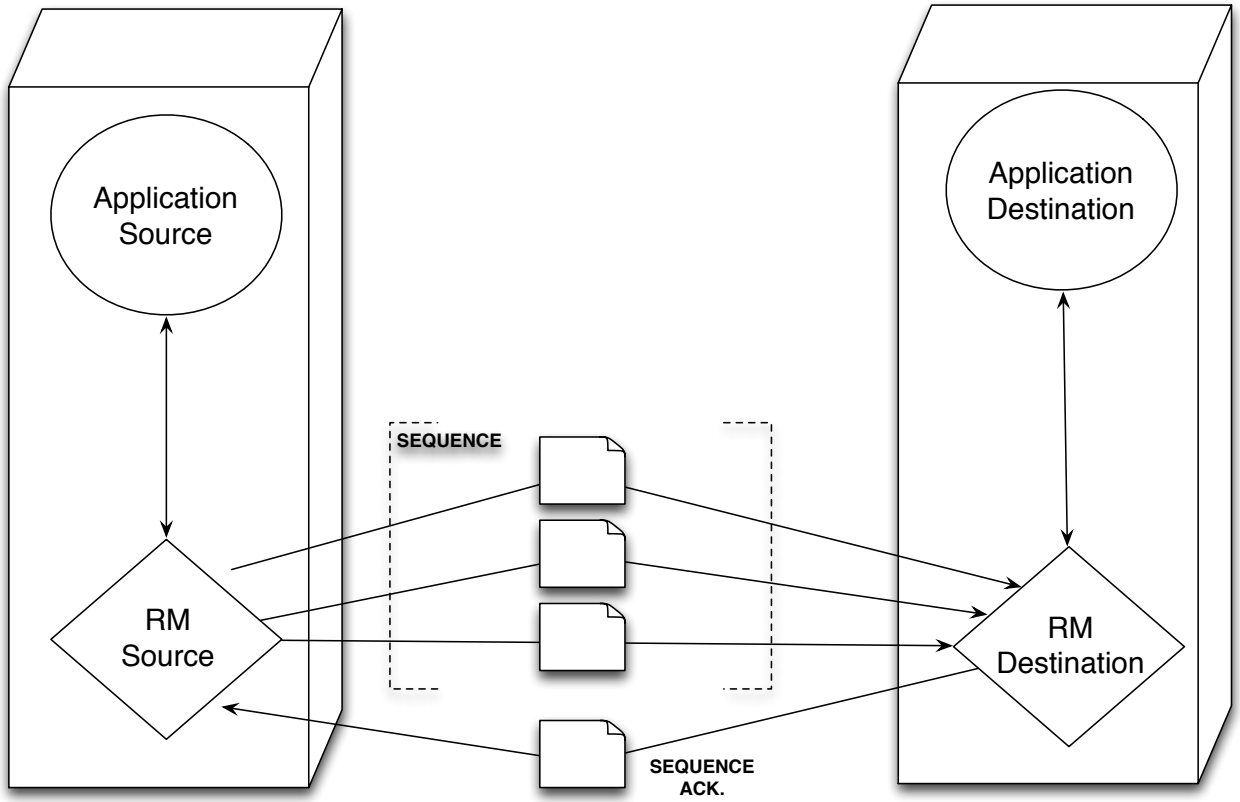
- A *sequence* establishes the order in which messages should be delivered
- Each message is labeled with a *message number*, the last one being a *last message* identifier.

Acknowledgements:

- A core part of reliable messaging is a notification system used to communicate conditions from the RM dest. to the RM source.
- The acknowledgement message indicates to the RM source which messages were received.

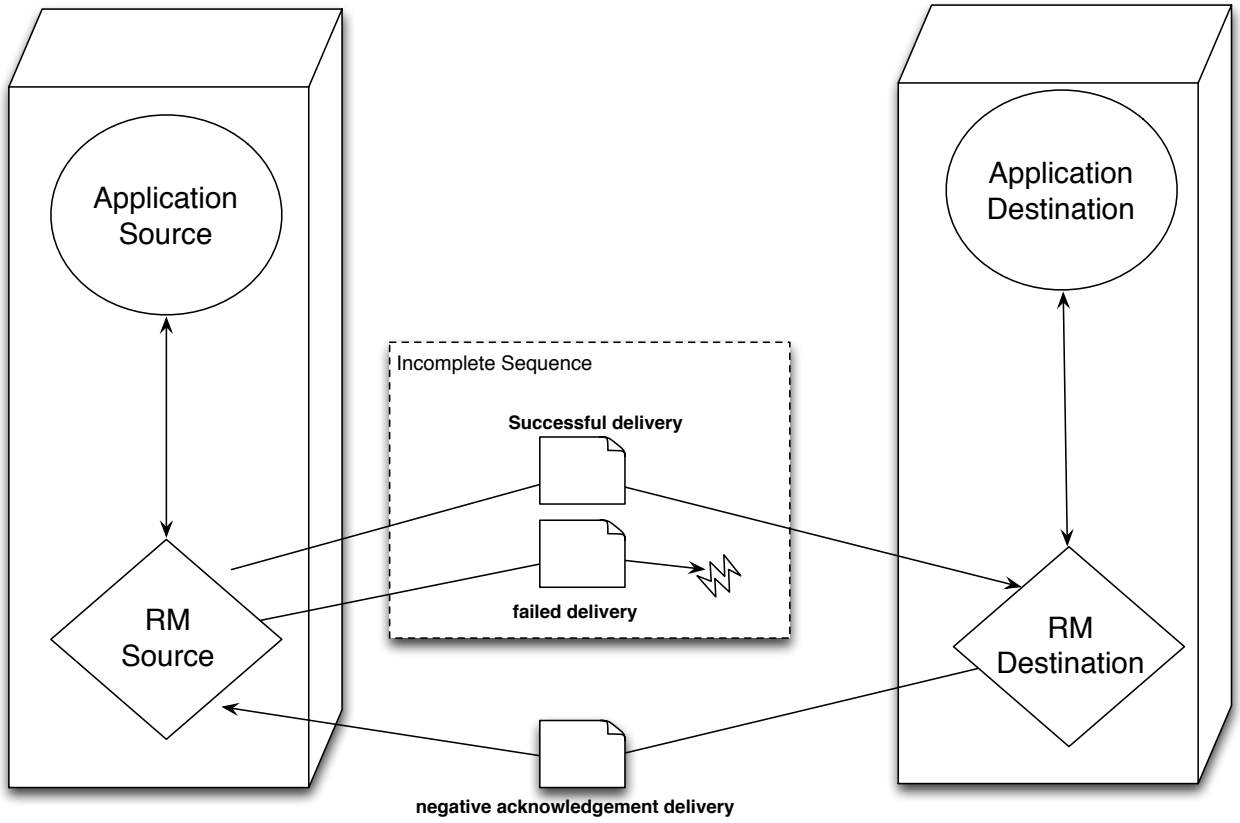
All this information is "injected" into SOAP headers within the messages themselves.

# Reliable Messaging (Blue Book Chap. 7)




A sequence acknowledgement sent by the RM dest. after the successful delivery of a sequence of messages.

# Reliable Messaging (Blue Book Chap. 7)




A negative ack. sent by the RM dest. to the RM source, indicating failed delivery prior to the completion of the sequence.

# Reliable Messaging (Blue Book Chap. 7)

Delivery assurances:

- the nature of a sequence is determined by a set of reliability rules known as *Delivery Assurances*.
- They are predefined message delivery patterns that establish a set of reliability policies

| Delivery Assurance | Description |
|---|---|
| At Most Once | Messages are delivered at most once, without duplication. It is possible that some messages may not be delivered at all. |
| At Least Once | Every message is delivered at least once. It is possible that some messages are delivered more than once. |
| Exactly Once | Every message is delivered exactly once, without duplication. |
| In Order | Messages are delivered in the order that they were sent. This delivery assurance can be combined with one of the preceding three assurances. |

# Reliable Messaging (Blue Book Chap. 17)

An example:

```
<Envelope
   xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wsrm="http://schemas.xmlsoap.org/ws/2004/03/rm">
   <Header>
      <wsrm:Sequence>
         <wsu:Identifier>
            http://www.xmlrc.com/railco/seq22231
         </wsu:Identifier>
         <wsrm:MessageNumber>
            15
         </wsrm:MessageNumber>
         <wsrm:LastMessage/>
       </wsrm:Sequence>
   </Header>
   <Body>
     ...
   </Body>
</Envelope>
```

# Reliable Messaging (Blue Book Chap. 17)

An example:

```
<Envelope
   xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wsrm="http://schemas.xmlsoap.org/ws/2004/03/rm">
   <Header>
      <wsrm:SequenceAcknowledgement>
         <wsu:Identifier>
            http://www.xmlrc.com/railco/seq22231
         </wsu:Identifier>
         <wsrm:AcknowledgementRange Upper="4" Lower="1"/>
         <wsrm:AcknowledgementRange Upper="8" Lower="6"/>
         <wsrm:AcknowledgementRange Upper="12" Lower="11"/>
         <wsrm:AcknowledgementRange Upper="15" Lower="14"/>
      </wsrm:SequenceAcknowledgement>
   </Header>
   <Body>
     ...
   </Body>
</Envelope>
```

# Summary

- Binding in WSDL defines (i) message encoding format (ii) transport protocol details. (...) and (...) are two options available in <soap:binding style='...'>.
- A SOAP server employs a pipeline based SOAP message processing model which includes: a sender, ultimate receiver and a series of (...).
- Each (...) is responsible for processing a (...)
- Can you roughly draw a diagram to illustrate how in-bound and out-bound SOAP messages are handled by a SOAP server?
- Why, would you say, is this type of processing model important in using SOAP for WS communication?
- SOAP message body can contain a normal response or a fault through (...)
- Where are the details of faults (if any) by a service declared ?

# Part VI

# UDDI - Advertising/Discovering Services

# Service Registries

- To *discover* Web services, a service registry is needed. This requires describing and registering the Web service.
- Publication of a service requires proper description of a Web service in terms of business, service, and technical information.
- Registration deals with persistently storing the Web service descriptions in the Web services registry.
- **Two types of registries can be used**:
  - **The document-based registry**: enables its clients to publish information, by storing XML-based service documents such as business profiles or technical specifications (including WSDL descriptions of the service).
  - **The meta-data-based service registry**: captures the essence of the submitted document.

# Service Discovery

- **Service discovery** is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.
- Interrogating services involve querying the service registry for Web services matching the needs of a service requestor.
  - A query consists of search criteria such as: the type of the desired service, preferred price and maximum number of returned results, and is executed against service information published by service provider.
- After the discovery process is complete, the service developer or client application should know the exact location of a Web service (URI), its capabilities, and how to interface with it.

# Types of service discovery

**Static:**

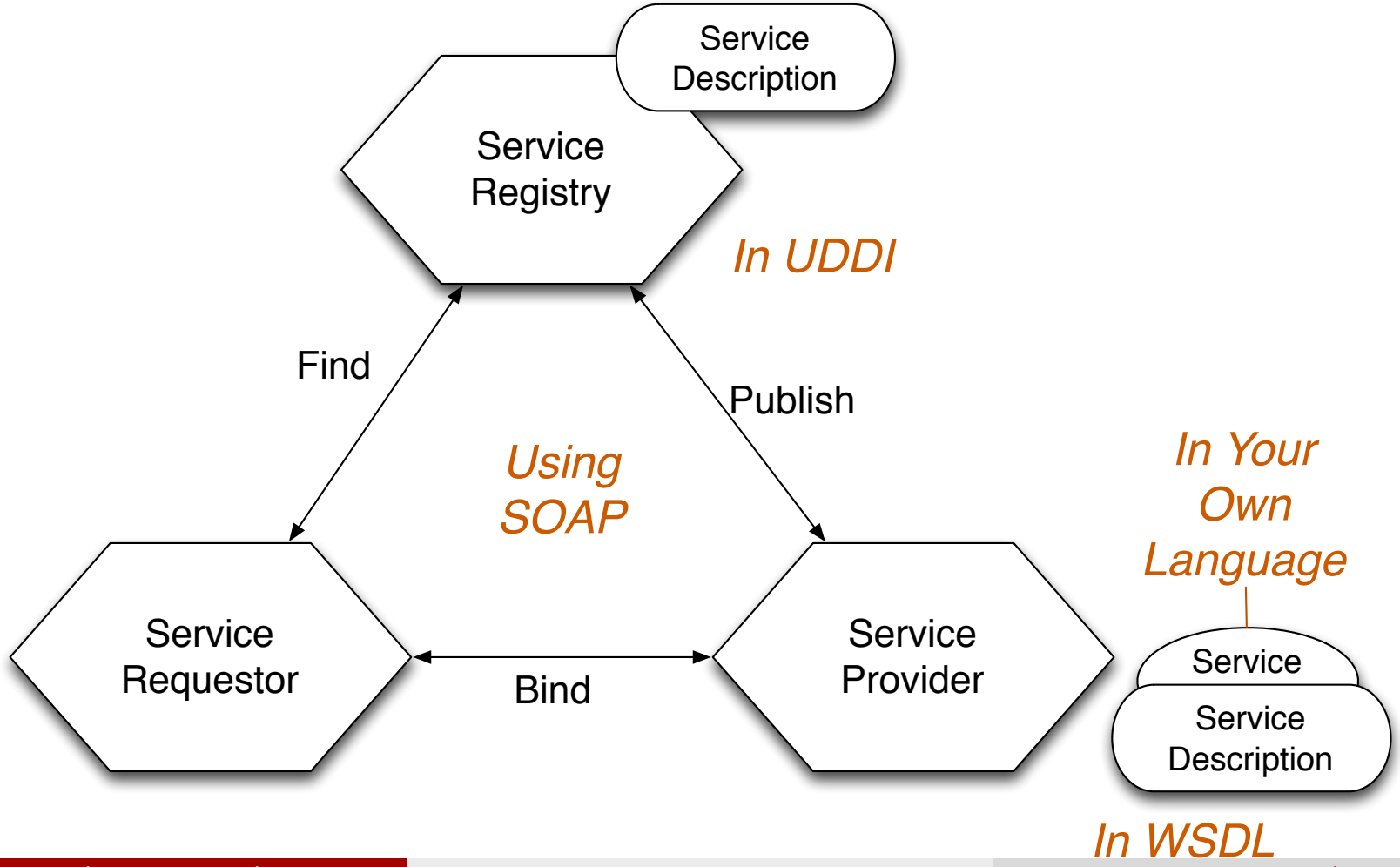
- The service implementation details are bound at design time and a service retrieval is performed on a service registry.
- The results of the retrieval operation are examined usually by a human designer and the service description returned by the retrieval operation is incorporated into the application logic.

**Dynamic:**

- The service implementation details are left unbound at design time so that they can be determined at run-time.
- The Web service requestor has to specify preferences to enable the application to **infer/reason** which Web service(s) to choose
- Based on application logic quality of service considerations such as best price, performance or security certificates. The application chooses the most appropriate service, binds to it, and invokes it.

# UDDI

## Universal Description Discovery and Integration

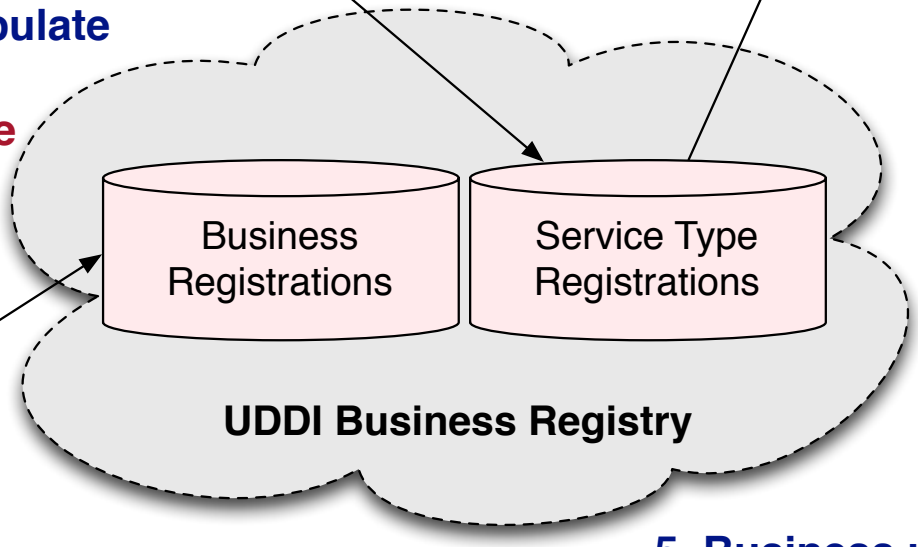


# UDDI and the big idea

**1. SW companies, standard bodies, and programmers populate the registry with description of various types of services**

**4. Marketplaces, search engines and business apps query the registry to discover services at other companies**


**2. Businesses populate the registry with descriptions of the services they support**



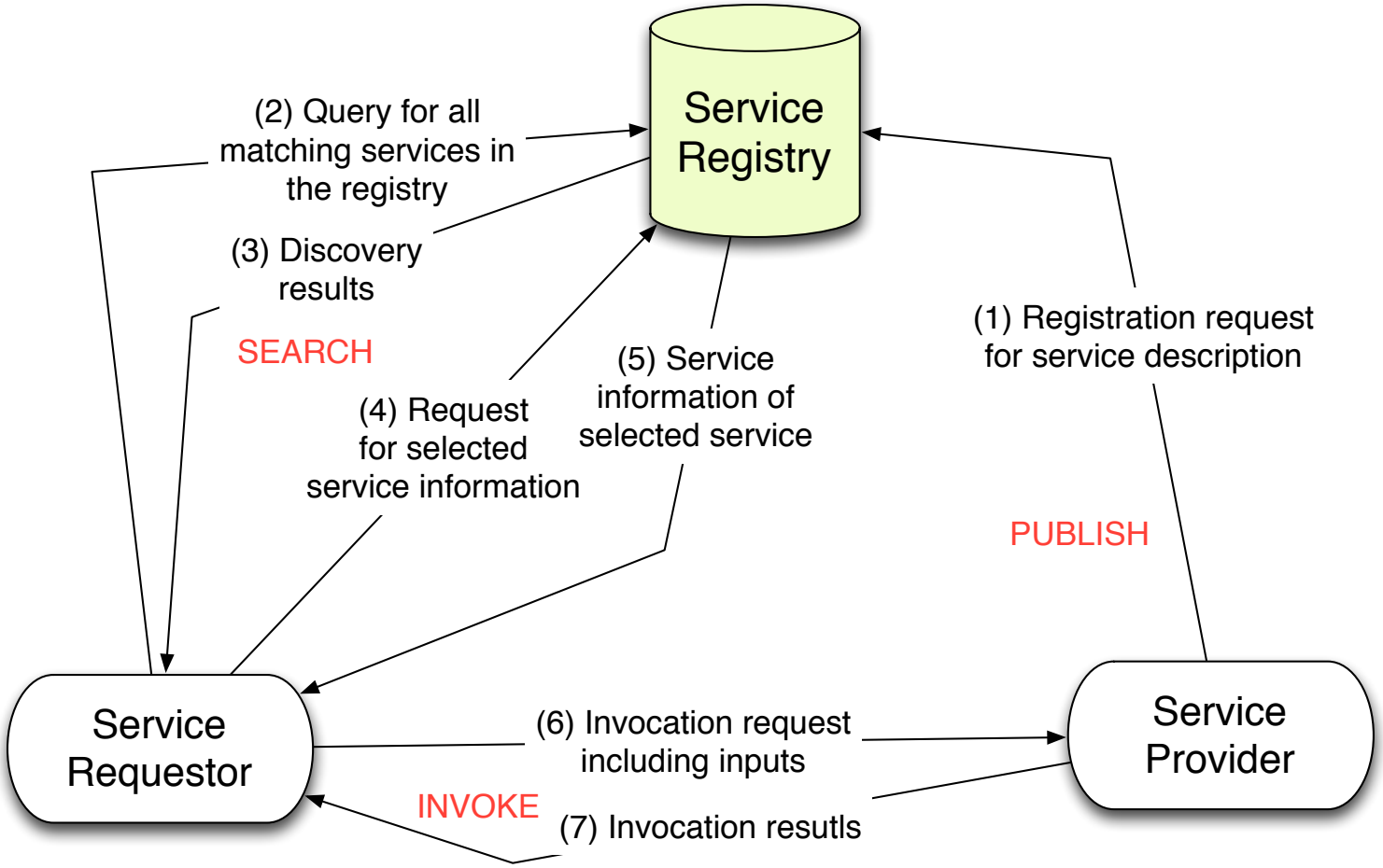


**3. UBR assigns a unique identifier to each service and business registration**

**5. Business uses this data to facilitate easier integration with each other over the web**

# UDDI and the big idea

How to find the service you want among a potentially large collection of services and servers. The client does not necessarily need to know a priori where the server resides or which server provides the service.



# UDDI

- UDDI is a registry (not repository) of Web services
- IBM and Microsoft *used* to host public UDDI registry
- Before UDDI, there was no standard way of finding documentation or the location of a particular remote object. Ad-hoc documentation may look like:

```
Contact person: John Smith
COM+ Object: GetWeatherInfo
COMP+ Server: http://bindingpoint.com
Relative URL: /metero/weather
Proxy Location: /Instal/GetWeatherInfo.dll
Description: Returns today's weather. It requires a zip code ...
```

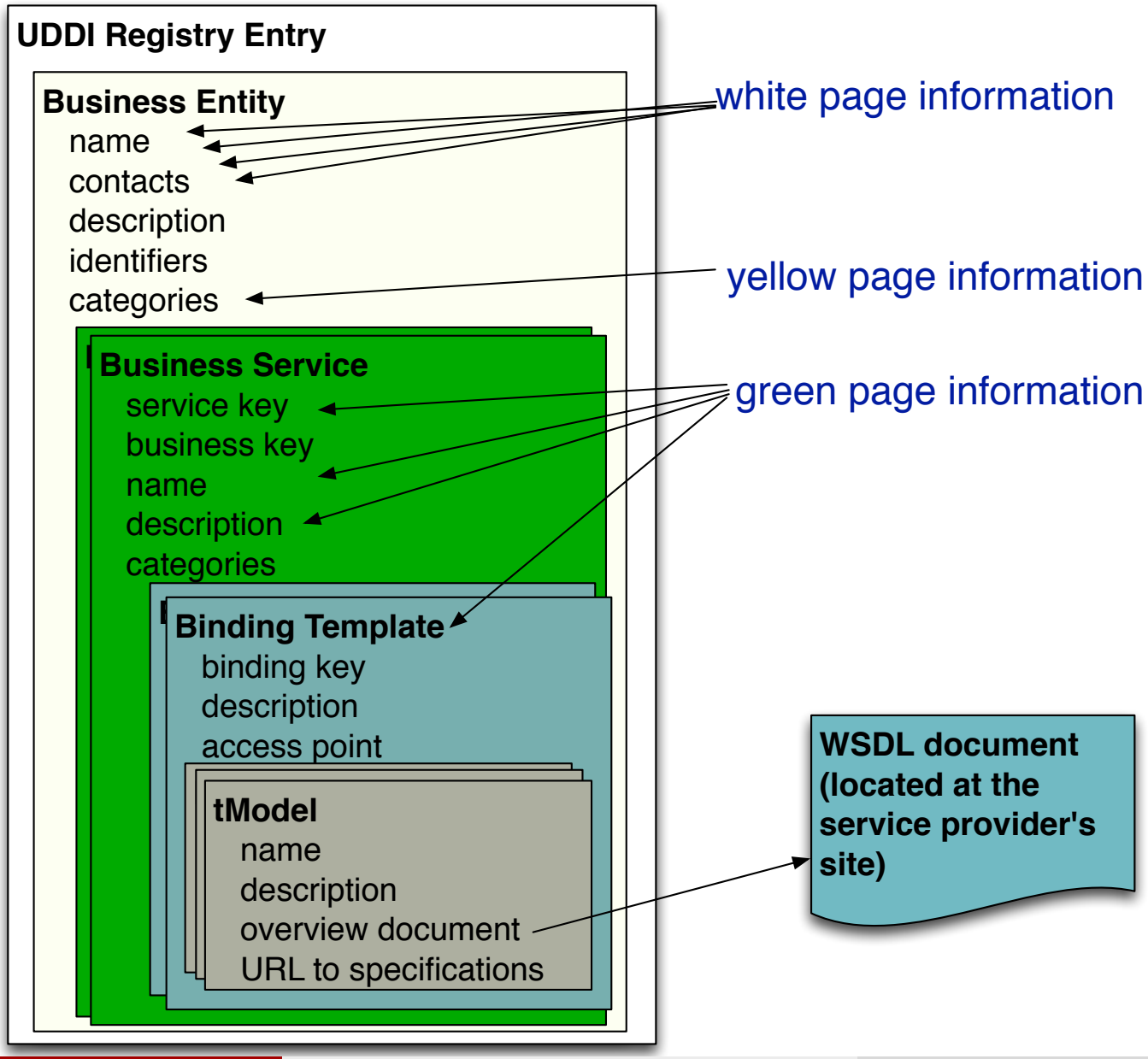
- UDDI is not part of W3C standard (unlike SOAP, WSDL)
- The main standard body for it is OASIS http://www.oasis-open.org, http://uddi.xml.org

# UDDI

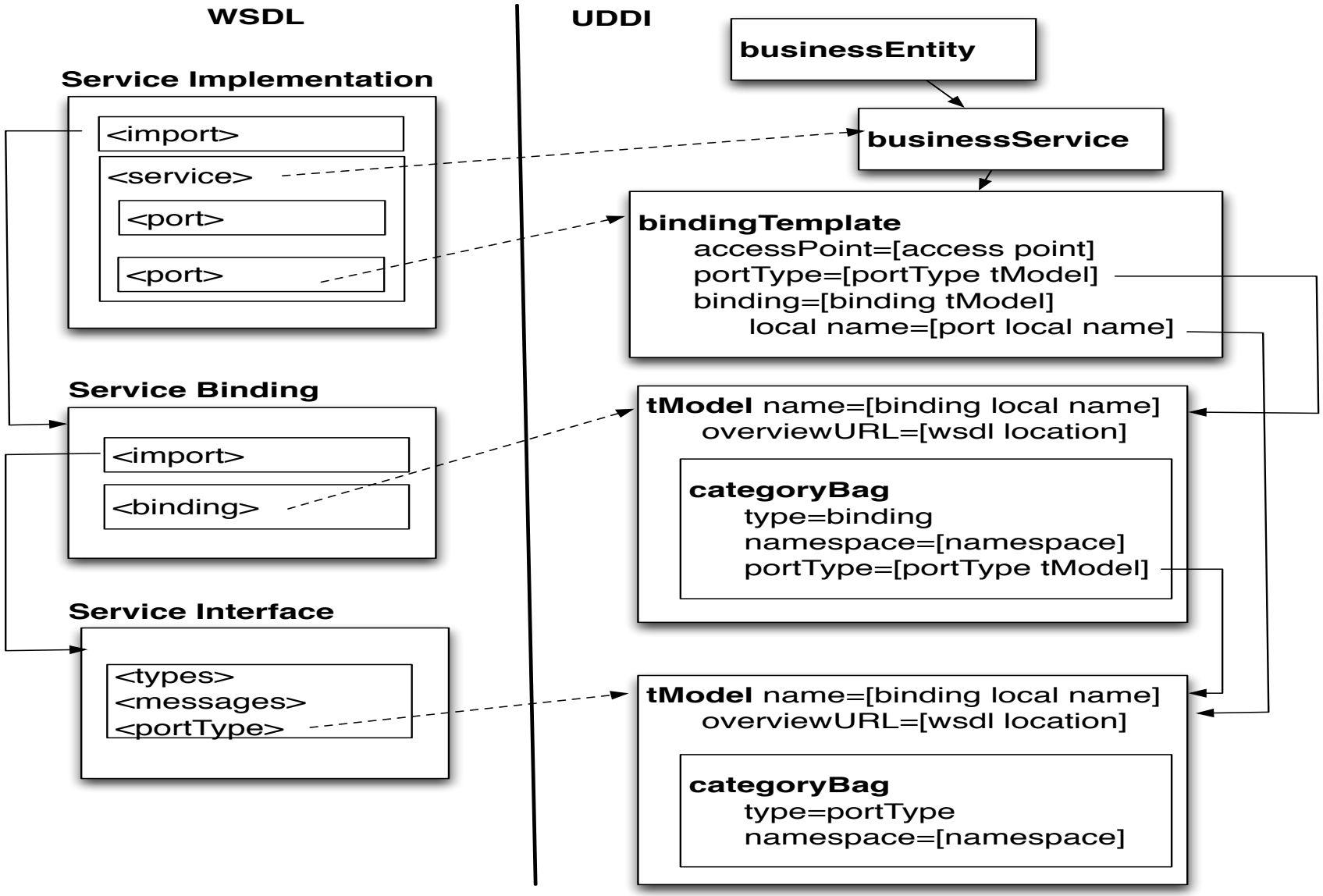
UDDI shares similarities with telephone directories.

- **White Pages**: Contact information about the service provider company. This information includes the business or entity name, address, contact information, other short descriptive information about the service provider, and unique identifier with which to facilitate locating this business
- **Yellow Pages**: Categories under which Web services implementing functionalities within those categories can be found
- **Green Pages**: Technical information about the capabilities and behavioral grouping of Web services

# UDDI - overview of its data structure



# WSDL and UDDI Mapping (e.g., jUDDI Apache Project)



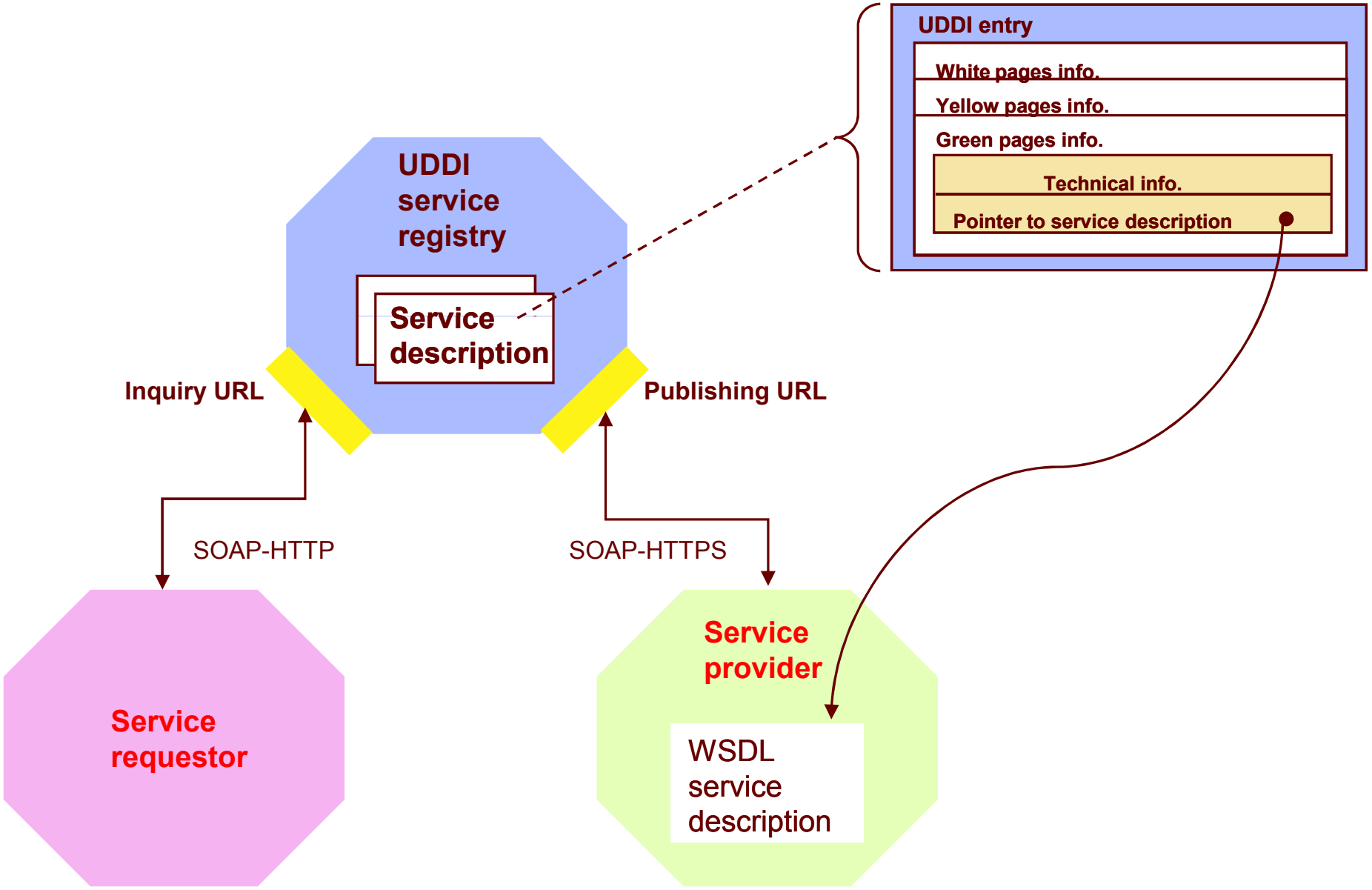
# UDDI

- UDDI registry can be browsed by human
- UDDI registry can be programmatically accessed
  - Inquiry API: enable lookup of registry information
  - Publishers API: allow applications to register services
  - an XML schema for SOAP message is defined
  - SOAP is used as the communication protocol
- An example implementation (jUDDI by Apache)
  - http://juddi.apache.org
  - http://juddi.apache.org/docs/3.2/juddi-client-guide/html/

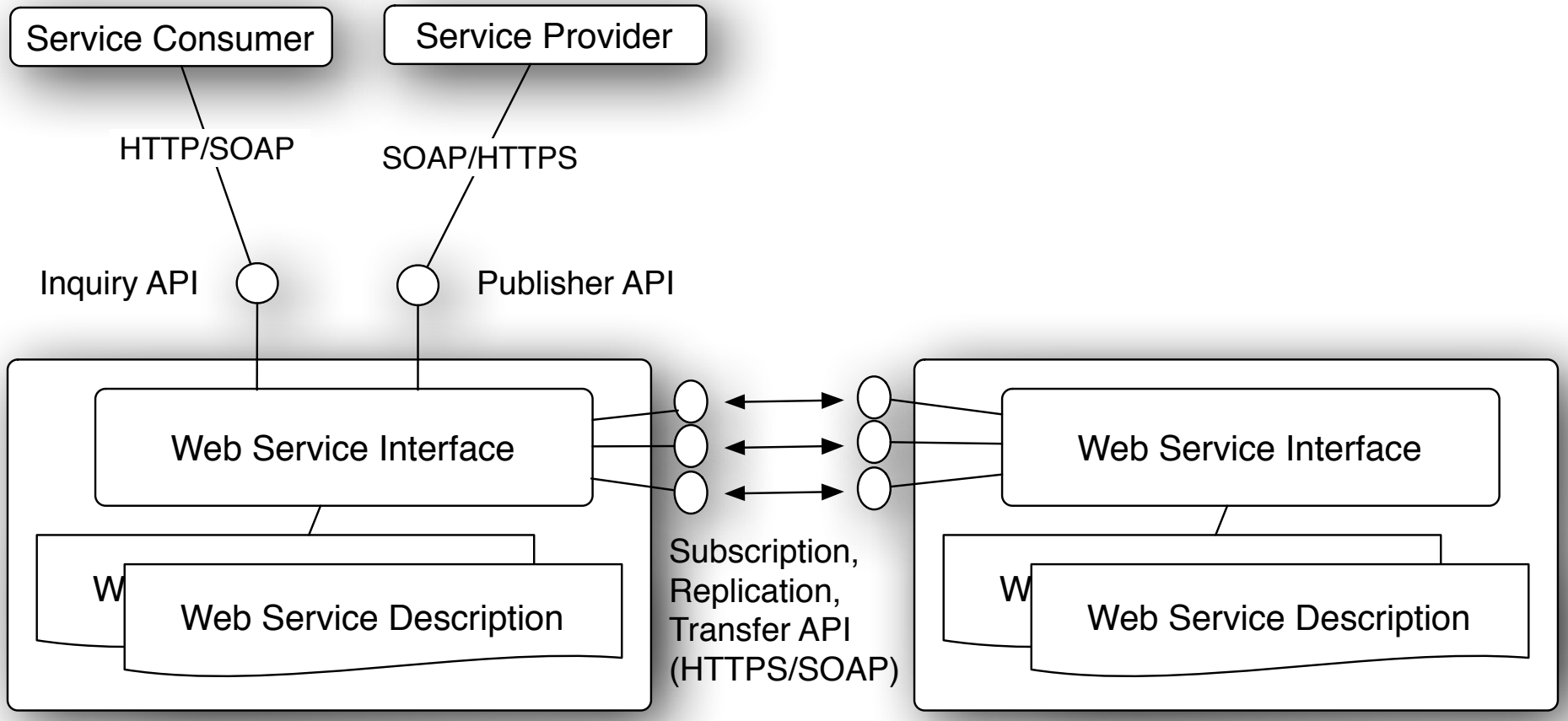
# UDDI's provided APIs

- **UDDI provides a SOAP-based API to the business registry.**
- **UDDI Inquiry APIs**: includes operations to find registry entries.
  - Browse, Drilldown, Invocation Patterns
- **UDDI Publish APIs**: operations to add, modify and delete entries in the registry.
- **UDDI Security API**: for access control to the UDDI registry.
- **UDDI Subscription API**: for clients to subscribe to changes of information in the UDDI registry.
- **UDDI Replication API**: to perform replication of information across nodes in a UDDI registry.

# UDDI's provided APIs



# Interaction with and between UDDIs



# How UDDI could play out: an opinion (Webber Book pp.136-141)

Question: Is it reasonable to assume that "people" will search a service registry using APIs to select a service during design time?

- Most likely no ... that's just not how "people" do business
- You will browse, ask around (word of mouth), google, etc.
- Oftentimes, the selection criteria can be tricky (e.g., existing business relationships, cutting deals, etc.)

If UDDI is not going to be useful in selecting services ... then what?

Note: Most of the UDDI registries in place today are private registries operating inside companies or maintained by a set of companies in a private manner

# How UDDI could play out: an opinion (Webber Book pp.136-141)

... UDDI might be useful at runtime ...

**Case 1: Service Life-cycle Management**

Consider the issues you have to deal with after Web services are deployed and clients are using them

- Overtime, some changes might have to be made (not only the code, but also the physical environment that the service is deployed in)
- e.g., migration to a new server, multiple mirror servers, routine maintenance on the server ...
- Applications that rely on Web services need to stay updated with the latest access end-point information
- How do we propagate the changes to the access point?

UDDI can play the runtime broker/middleman in handling and propagating these changes ...

# How UDDI could play out: an opinion (Webber Book pp.136-141)

Scenario: Service Life-cycle management with UDDI

- A Web service is selected for use (searched in or outside UDDI)
- Save (in your local database) the bindingTemplate information of the service from UDDI
- Develop an application using the service
- If the service call fails (or times out):
  - query UDDI for the lastest information
  - compare the info. with the saved info.
  - if different, try calling the service again with the new info.
  - update your local bindingTemplate if needed

# How UDDI could play out: an opinion (Webber Book pp.136-141)

... UDDI might be useful at runtime ...

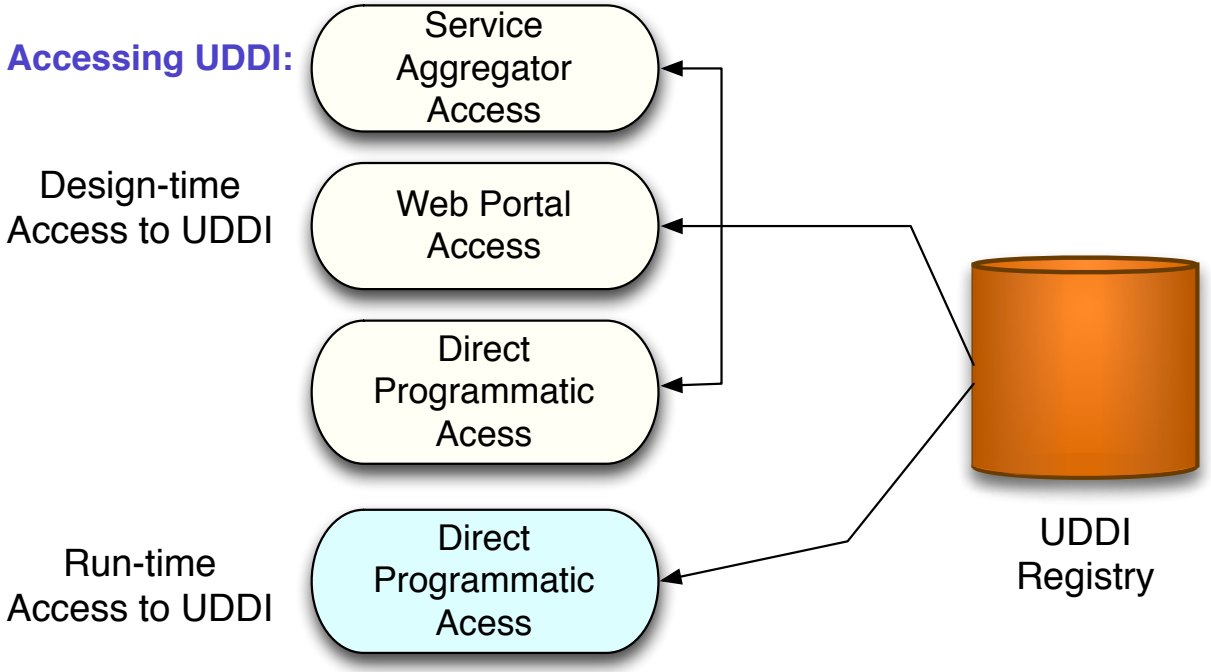
**Case 2: Dynamic access point management**

Not only when a service call is failed, you may want to dynamically manage and select the most appropriate access point for a service.

- A service may be available from multiple geographical locations
- The client application may have been developed in one country and later used in another county

The concept is similar to downloading files from different mirror sites. The access point can be hardwired in the client application, but by dynamically selecting the most appropriate access point (based on certain criteria) may lead to increased performanace.

# How UDDI could play out: an opinion (Webber Book pp.136-141)




Desgin time access – via manual search or direct API acccess, to search and discover services during the application design phase

Runtime access (UDDI playing a brokering/middleman role) – via direct API access, it offers possibilities to build more robust and flexible applications

# Static Discovery of Web Services ...

There are some public Web service registries operating (not following UDDI):

- XMETHODS (seems to be offline nowadays): http://www.xmethods.net/ve2/Directory.po
- WebserviceX.NET: http://www.webservicex.net/WS/wscatlist.aspx
- WebServiceList: http://www.webservicelist.com (with user rating info)

# Summary

- Can you describe a service registration process and a service discovery process?
- What is the purpose of a WSDL to UDDI mapping model?
- Can you list some of the operations in UDDI API?
- Alternative/suggested use of UDDI ...?