



Internet of Things and Services

Service-oriented architectures

SOA & Web services

Department of Computer Science
Faculty of Electronic Engineering, University of Nis

Internet of Things and Services
Computing and informatics

Prof. dr Dragan Stojanović

References

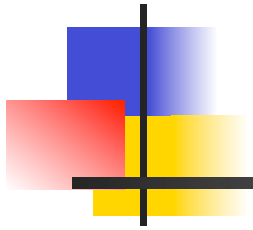
Books

- Michael Papazoglou, [Web Services and SOA: Principles and Technology, 2 edition](#), Pearson 2013.
- Gustavo Alonso, Fabio Casati, Harumi A. Kuno, Vijay Machiraju, [Web Services - Concepts, Architectures and Applications](#), Data-Centric Systems and Applications, Springer 2004.

Courses

- [Introduction to Service Design and Engineering](#)
 - *University of Trento, Italy*
- [Service Technologies](#)
 - *Politechnico di Milano, Italy*

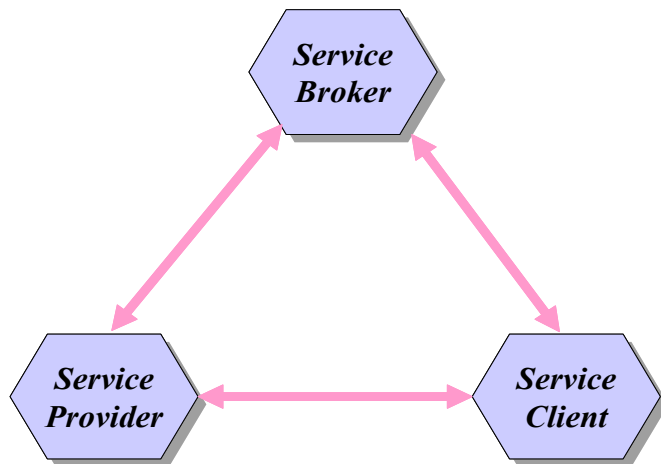
**Adapted from teaching material shared by
Helen Paik, Boualem Bentallah,
Marcello La Rosa, Fabio Casati, Maurizio Marchese**



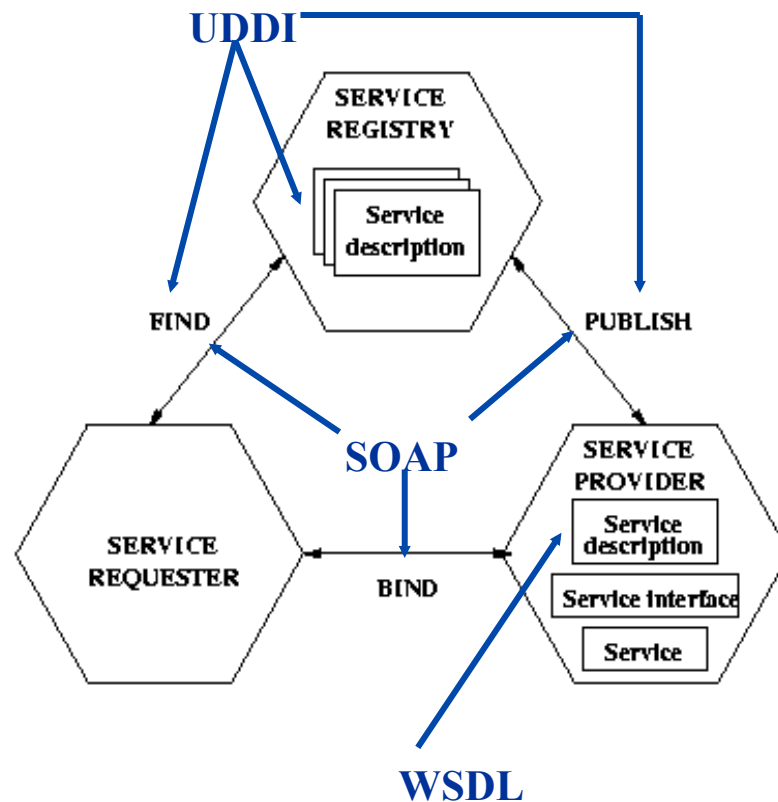
Principles of Service Oriented Architectures

Basic SOA

Maurizio Marchese – Intro SDE



Service Roles in an SOA



Web Service implementation of SOA



What is SOA? – Purpose

- *SOA is a **paradigm** for organizing and utilizing distributed **resources** and **capabilities** that may be under the control of different ownership domains.*

OASIS SOA Reference Model

Maurizio Marchese – Intro SDE

- SO systems will cross domain ownership:
 - development and maintenance is autonomous
 - need an agreement (mutual benefit) on how we are going to talk
- SOA enables communication among different parts of an organization or different organizations that need to work together.
- SOA does not imply only “technology”, but it can help to structure how technology is deployed and organised.



What is SOA? – Second try...

SOA is a software system composed of a collection of software services.

What is a *software service*?

- A software **asset** that is deployed at an **endpoint** and is **continuously maintained** by a provider for use by one or multiple clients.
- Services have explicit **contracts** that establish their purpose and how they should be used (SLAs..)
- Software services are (supposed to be) **reusable**.



SOA Principles

- **Loose-coupling**

SOAs should minimize dependencies between services and make these dependencies explicit

- **Abstraction**

Services adhere to contracts/interfaces and hide their inner workings

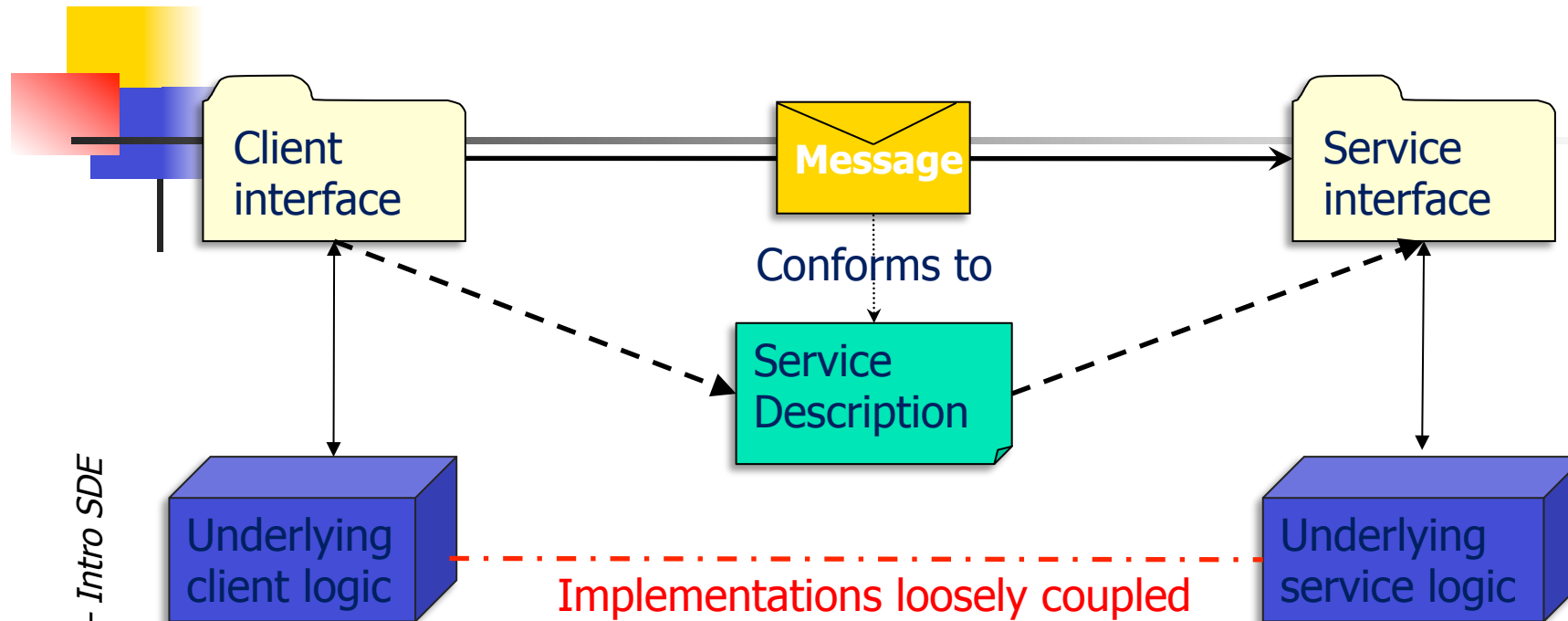
- **Reusability**

Services are created with the intent of promoting reuse

- **Composability**

Services can be coordinated and assembled to form composite services

Achieving Loose Coupling & Abstraction



Maurizio Marchese – Intro SDE

- Client is aware of service description only (*service oblivious*)
- Service needn't know about client (*client oblivious*).
- Interaction is purely through messaging infrastructure:
 - messages are self contained and self governing.
 - once a message is sent, the client has no control over it.



Achieving Reuse and Composability

- Align Services with Business Processes and Business Entities:
 - Non-technical domain experts should be able to easily understand and clearly see the business relevance of all services and service operation
 - Design services that are business oriented in scope and interface; avoid “technical” interfaces.
 - Avoid fine-grained patterns of interaction.
- Avoid “millions” of very simple service invocations
 - try to aggregate into larger operations: document-centric messages
 - strike a balance between atomic operations and one single service application

Conclusion: SOA Value Proposition

Organise systems into reusable services with explicit dependencies and contracts



Understand the value of each service, its touch-points, interactions, performance metrics, contract violations...



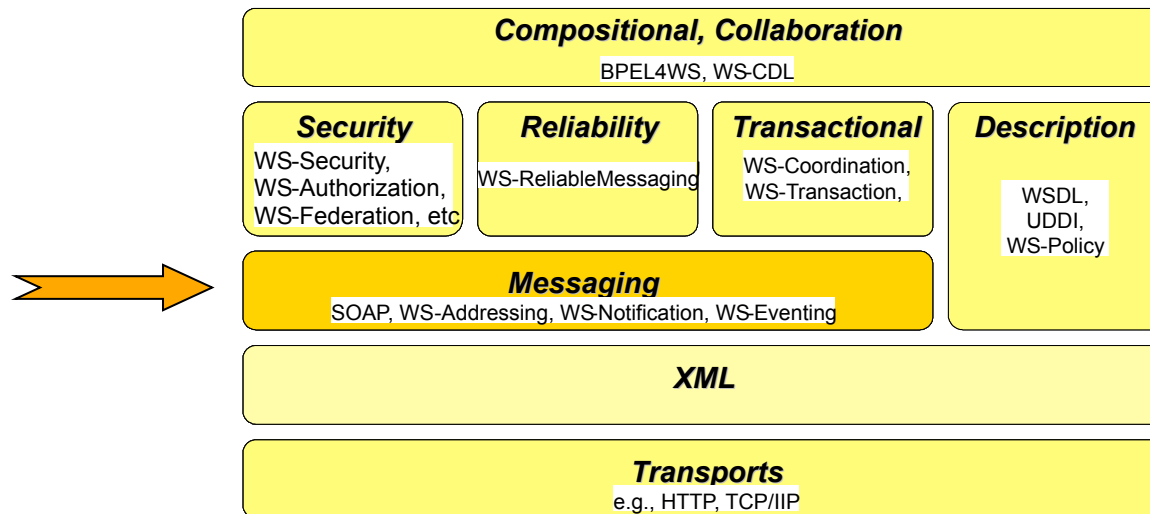
Increase reuse & sharing
Increase flexibility in business process
Reduce integration costs
Reduce time to market
Increase customer satisfaction...



Adapted from teaching material shared by
Helen Paik, Boualem Bentallah,
Marcello La Rosa, Fabio Casati, Maurizio Marchese



"Big" Web Services: SOAP





Problems to solve

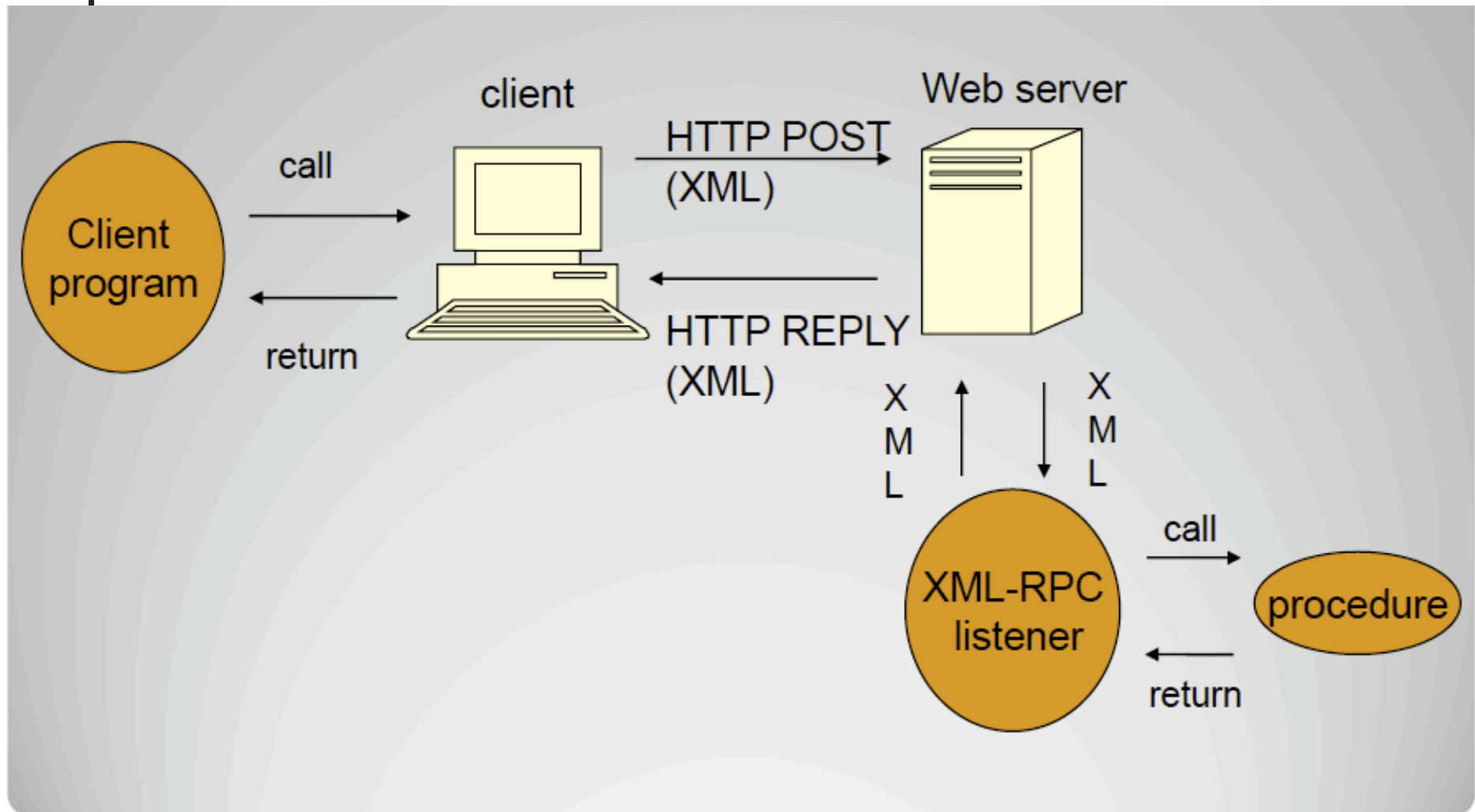
- How to make the **service invocation** part of the language in a more or less transparent manner.
 - Don't forget this important aspect: whatever you design, others will have to program and use
- How to **exchange data between machines** that might use different representations for different data types. This involves two aspects:
 - data type formats (e.g., byte orders in different architectures)
 - data structures (need to be flattened and the reconstructed)
- How to **find the service** one actually wants among a potentially large collection of services and servers.
 - The goal is that the client does not necessarily need to know where the server resides or even which server provides the service.
- How to **deal with errors (QoS)** in the service invocation in a more or less elegant manner:
 - server is down,
 - communication is down,
 - server busy,
 - duplicated requests ...



One solution: XML-RPC

- XML-RPC emerged in early 1998
 - Precursor of REST services
- Permits programs to make function or procedure calls across a network.
- Uses the HTTP protocol to pass information from a client computer to a server computer.

Sample XML-RPC Conversation





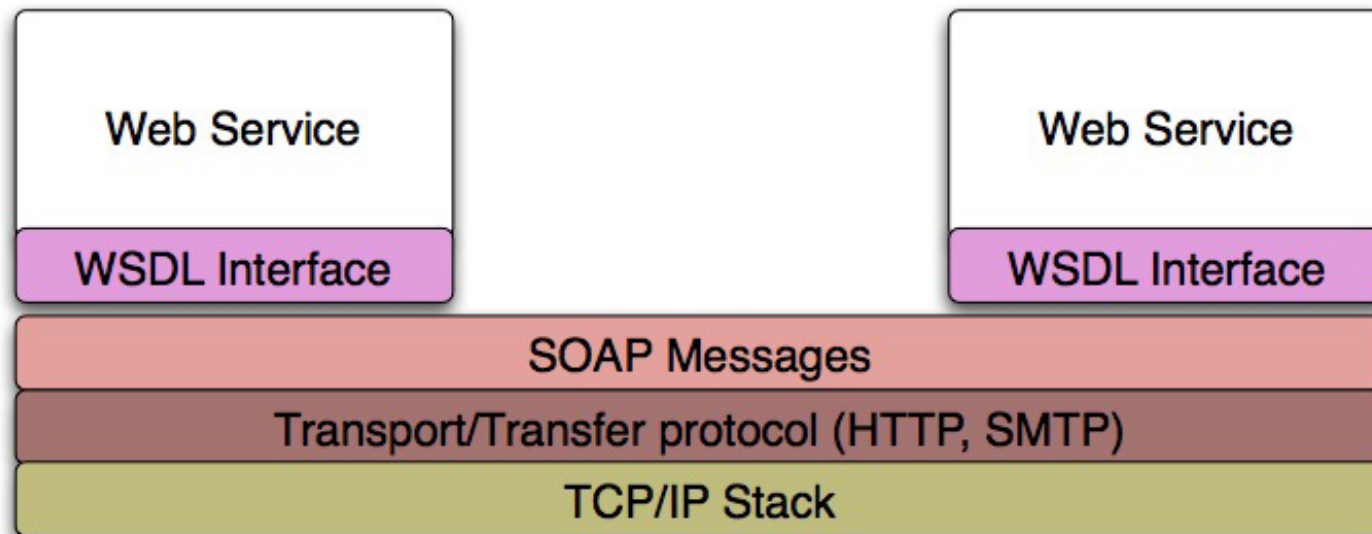
Limitation of XML-RPC / REST

- Only one protocol: HTTP
- Only RPC style communication and usage
- No standard XML language for data/resources representation

SIMPLE OBJECT ACCESS PROTOCOL: INTRO

What is SOAP?

- SOAP (Simple Object Access Protocol) is a standard 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 exchange





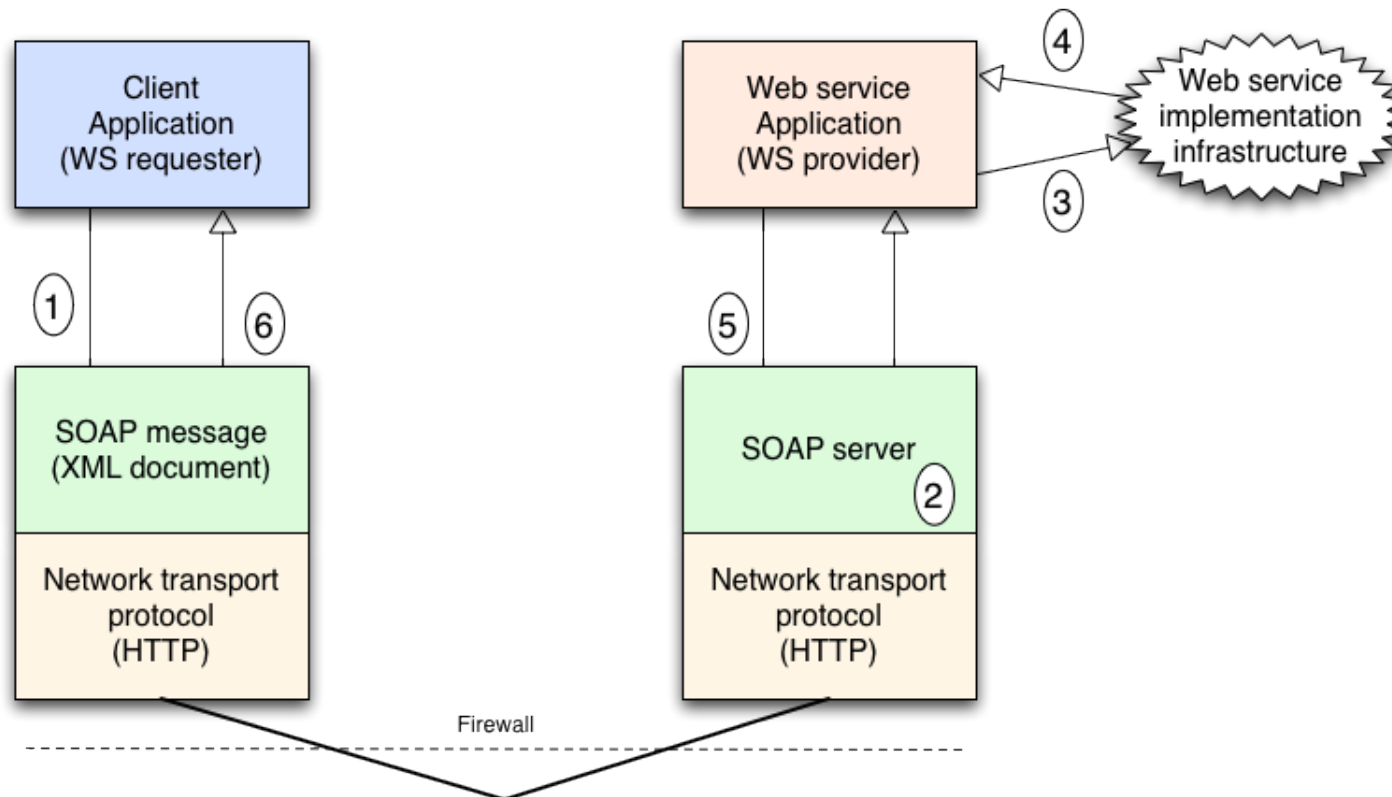
What is SOAP?

- The W3C started working on SOAP in 1999. The current W3C recommendation is Version 1.2
- SOAP is a **lightweight protocol** intended for **exchanging structured information** in a decentralized, distributed environment. **It uses XML technologies** to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.
- SOAP covers the following four main areas:
 - **A message format** for one-way communication describing how a message can be packed into an XML document
 - A **description of how a SOAP message** (or the XML document that makes up a SOAP message) **should be transported** using different protocols: HTTP (for Web based interaction), SMTP (for e-mail based interaction), others..
 - A **set of rules** that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message.
 - A **set of conventions** on how to turn an RPC call into a SOAP message and back as well as how to implement the RPC style of interaction

Distributed messaging using SOAP

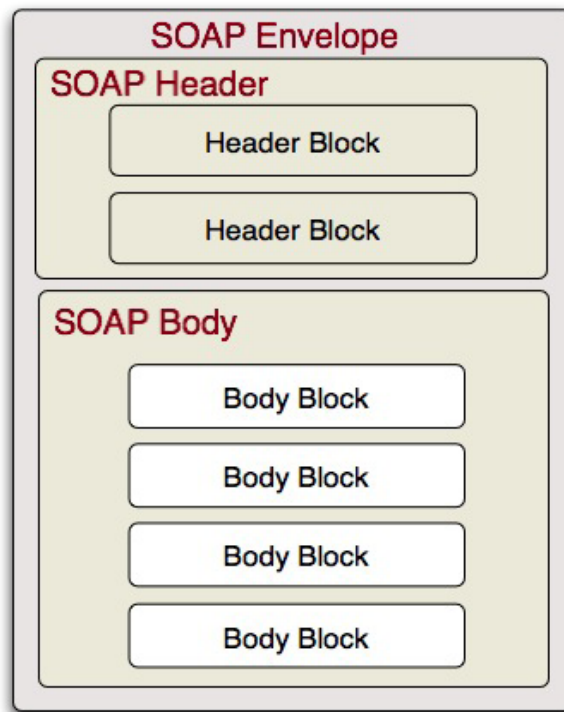
- SOAP server: Simply special code that listens for SOAP messages and acts as a distributor and interpreter of SOAP documents

Maurizio Marchese – IntroSDE



SOAP messages

- 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



The SOAP header


- The **header** is intended as a generic place holder for **information that is not necessarily application dependent**
 - the application may not even be aware that a header was attached to the message).
- Typical uses of the header are: **coordination information**, identifiers (for, e.g., **transactions**), **security information** (e.g., certificates)
- SOAP provides mechanisms to specify who should deal with headers and what to do with them. For this purpose it includes:
 - **SOAP actor attribute**: who should process that particular header entry (or header block). The actor can be either: **none**, **next**, **ultimateReceiver**.
 - **mustUnderstand attribute**: with values 1 or 0, indicating whether it is mandatory to process the header. If a node can process the message (as indicated by the actor attribute), the mustUnderstand attribute determines whether it is mandatory to do so.
 - **SOAP 1.2** adds a **relay attribute** (forward header if not processed)



The SOAP body

- The SOAP body is the area of the SOAP message where the application specific XML data (payload) being exchanged in the message is placed.
- The **<Body> element** must be present and must be an immediate child of the envelope. It may contain an arbitrary number of child elements, called body entries, but it may also be empty.
-
- The **<Body> element** contains either the application-specific data or a fault message.
 - **Application-specific data** is the information that is exchanged with a web service. It can be arbitrary XML data or parameters to a method call.
 - A **fault message** is used only when an error occurs.. A SOAP message may carry either application-specific data or a fault, but not both.

For the XML fans (the request, body only)



XML name space identifier for SOAP serialization
XML name space identifier for SOAP envelope

`<SOAP-ENV:Envelope`
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

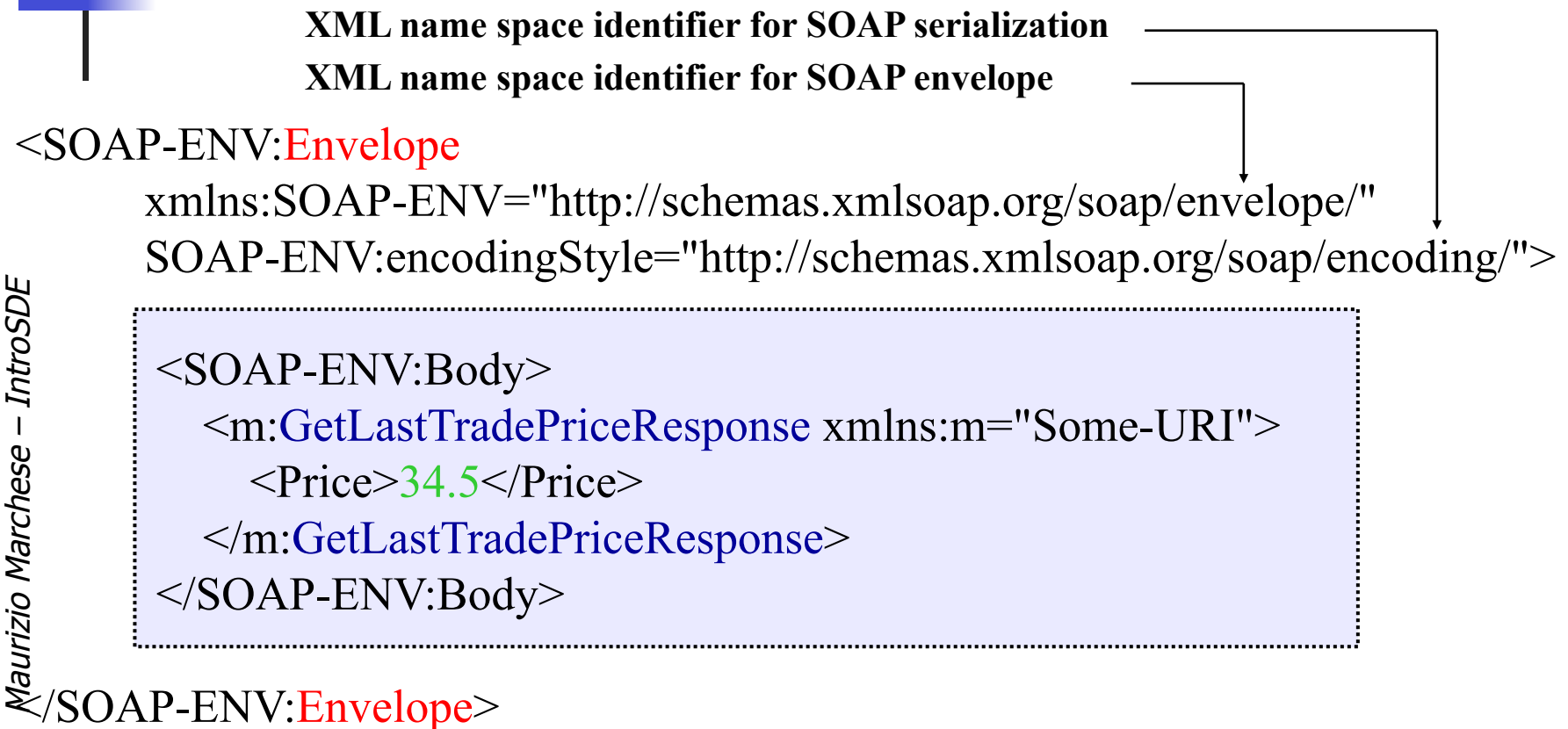
`<SOAP-ENV:Body>`
 <m:GetLastTradePrice xmlns:m="Some-URI">
 <symbol>DIS</symbol>
 </m:GetLastTradePrice>
 </SOAP-ENV:Body>

`</SOAP-ENV:Envelope>`

Maurizio Marchese – IntroSDE

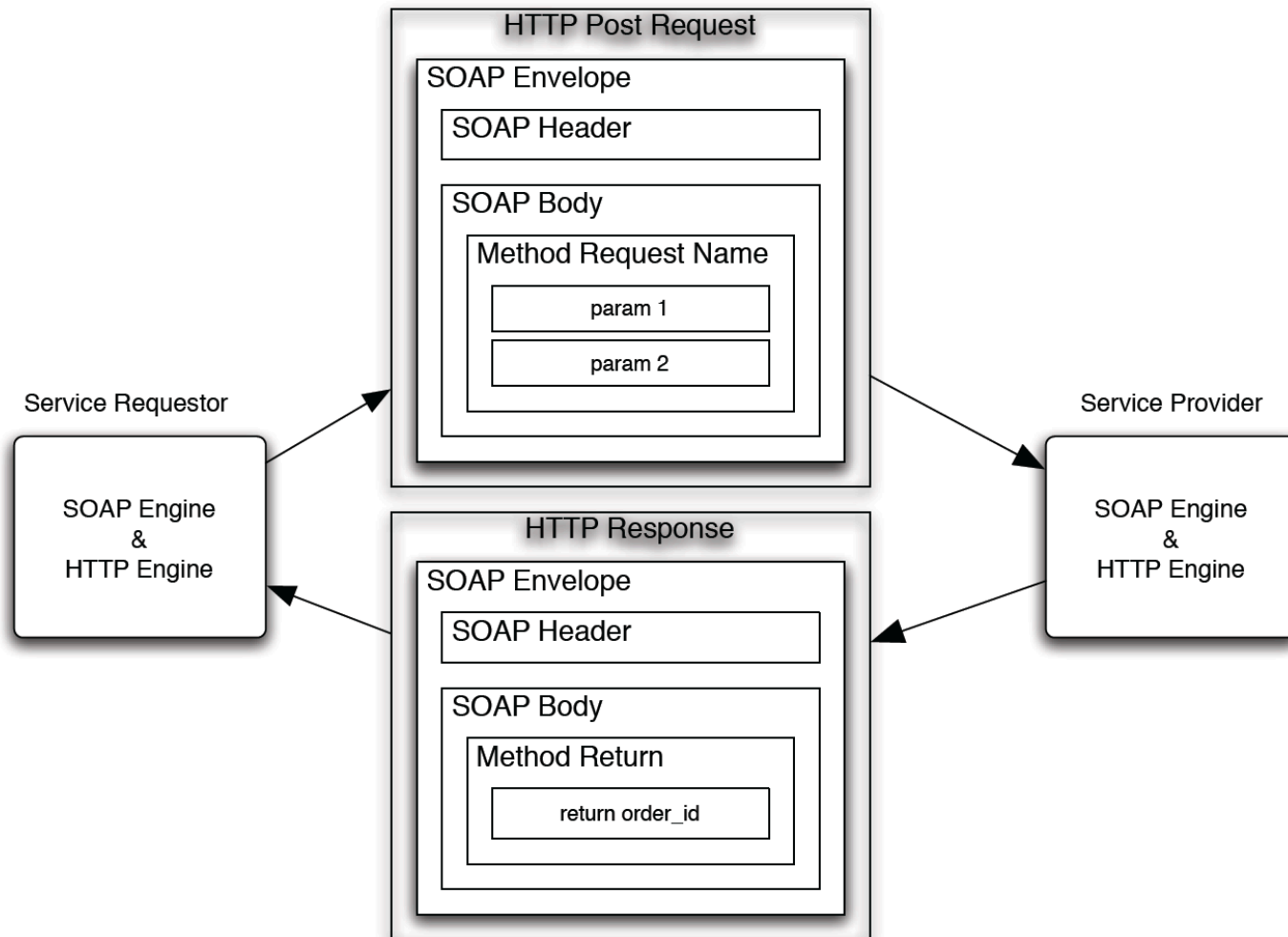
For the XML fans (the response)

Maurizio Marchese – IntroSDE



Binding SOAP with a Transfer Protocol

Maurizio Marchese – IntroSDE



An example of SOAP Binding over HTTP

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

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>

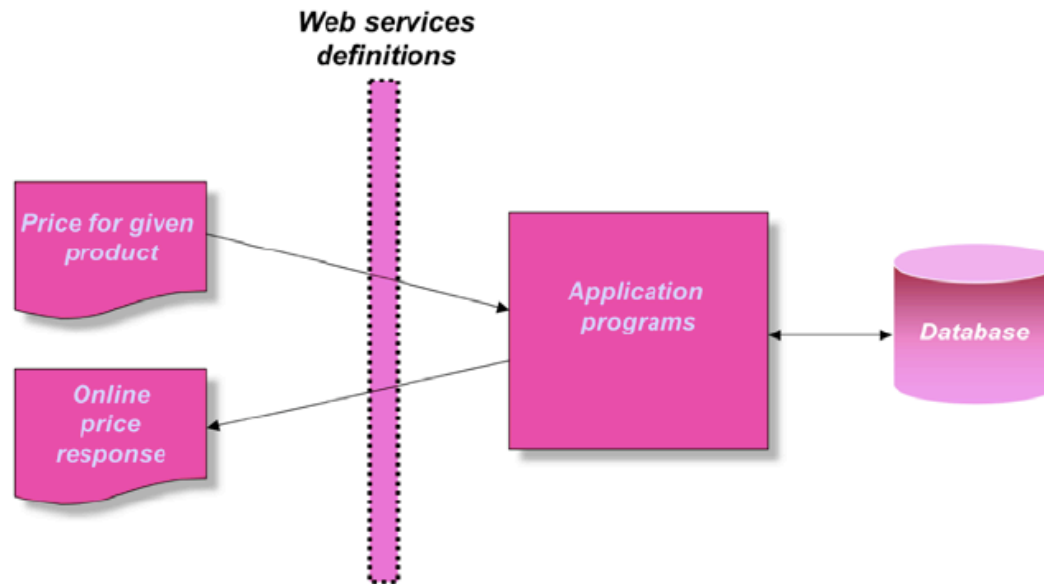
</SOAP-ENV:Envelope>



The SOAP Communication Model

- The **SOAP communication style** conveys information about how the contents of a particular element in the header blocks or the <Body> element of a SOAP message are passed between clients/services
- SOAP supports two possible communication styles:
 - **remote procedure call** (RPC) and
 - **document** (or message).
- The **SOAP communication styles** come in four flavours:
 1. RPC/Literal,
 2. Document/Literal,
 3. RPC/Encoded, and
 4. Document/Encoded

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 (synchronous communication model)

SOAP communication model: RPC-style

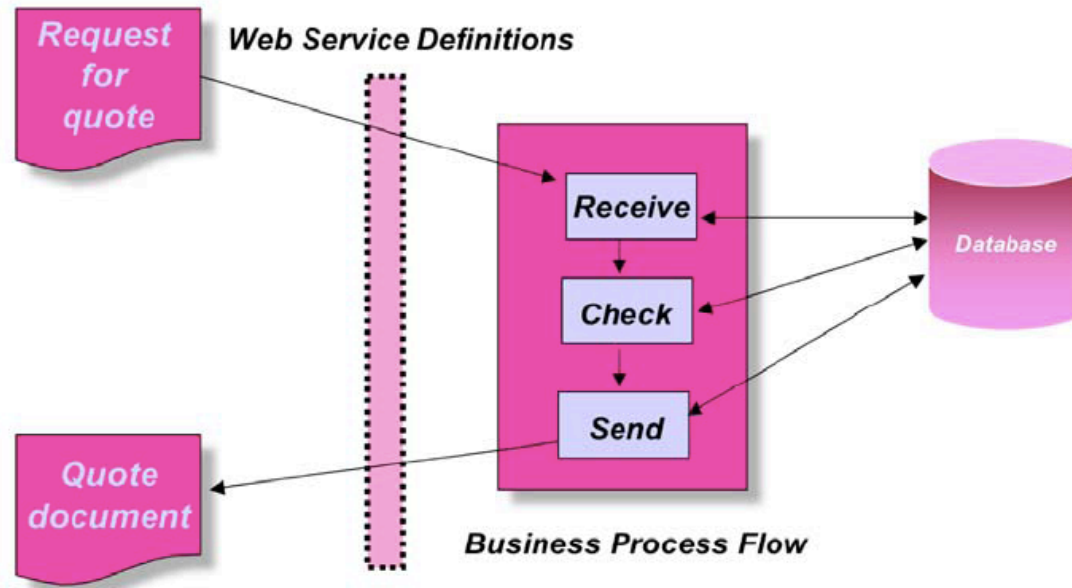
- Example of an RPC-style SOAP communication

```
<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 RPC-like structure that indicates the method name and contains a set of parameters

SOAP communication model: document-style



- SOAP actually contains generic 'XML data' (not a method call)
- Client sends an entire document (e.g., purchase order) rather than a discrete set of parameters
- The document is processed (most of the time within an internal workflow) at the service provider
- Message driven, asynchronous communication model

SOAP communication model: document-style

- Example of an Document-style SOAP

```
<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



SOAP Fault element (v 1.1 & 1.2)

- The fault entry has four elements (in 1.1):
 - **fault code**: indicating the class of error (version, mustUnderstand, client, server)
 - **fault string**: human readable explanation of the fault (not intended for automated processing)
 - **fault actor**: who originated the fault
 - **detail**: application specific information about the nature of the fault
- In version 1.2, the fault element is specified in more detail. It must contain two mandatory sub-elements:
 - **Code**: containing a value (the code for the fault) and possibly a subcode (for application specific information)
 - **Reason**: same as fault string in 1.1
- and may contain a few additional elements:
 - **detail**: as in 1.1
 - **node**: the identification of the node producing the fault (if absent, it defaults to the intended recipient of the message)
 - **role**: the role played by the node that generated the fault
- Errors in understanding a mandatory header are responded using a fault element but also include a special header indicating which one of the original headers was not understood.

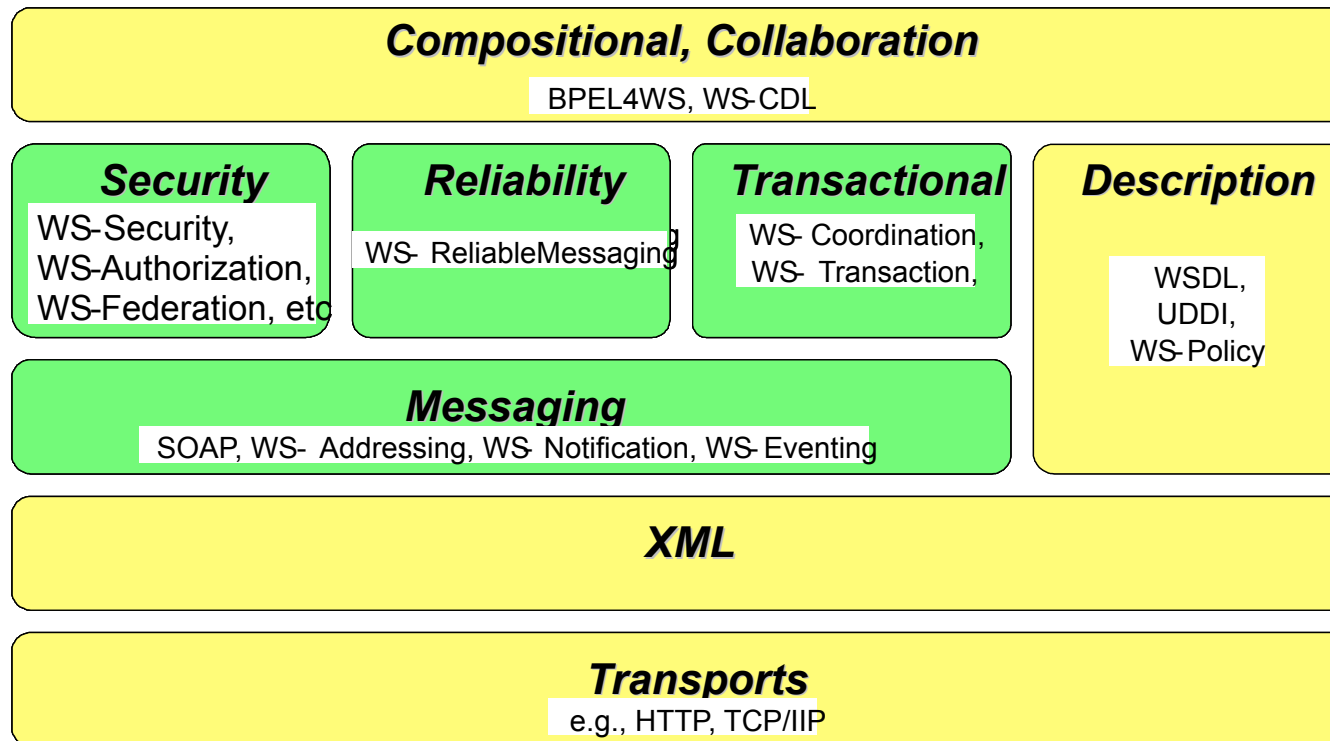


Example of Fault SOAP message

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
      <faultstring xsi:type="xsd:string">
        Failed to locate method (ValidateCreditCard) in class
        (examplesCreditCard) at /usr/local/ActivePerl-5.6/lib/
        site_perl/5.6.0/SOAP/Lite.pm line 1555.
      </faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP + WS-"extensions"

Maurizio Marchese – IntroSDE

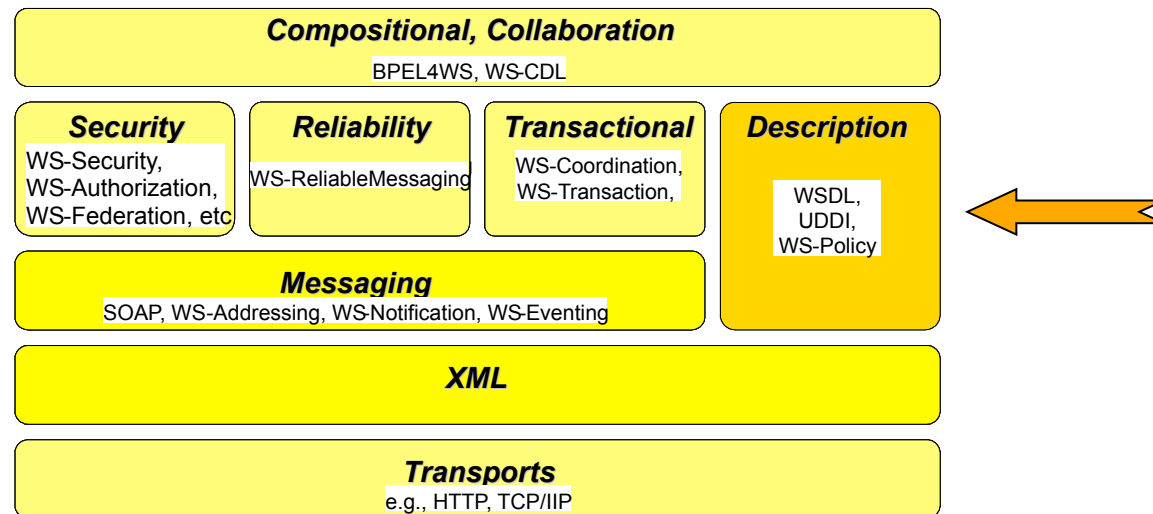


Adapted from teaching material shared by
Helen Paik, Boualem Bentallah,
Marcello La Rosa, Fabio Casati, Maurizio Marchese



Web services: WSDL

Web Service Description Language





Problems to solve

- How to make the **service invocation** part of the language in a more or less transparent manner.
 - Don't forget this important aspect: whatever you design, others will have to program and use
- How to **exchange data between machines** that might use different representations for different data types. This involves two aspects:
 - data type formats (e.g., byte orders in different architectures)
 - data structures (need to be flattened and the reconstructed)
- How to **find the service** one actually wants among a potentially large collection of services and servers.
 - The goal is that the client does not necessarily need to know where the server resides or even which server provides the service.
- How to **deal with errors (QoS)** in the service invocation in a more or less elegant manner:
 - server is down,
 - communication is down,
 - server busy,
 - duplicated requests ...



Outline

- | ■ Web Service Description Language
 - Structure of WSDL documents
 - WSDL interface definition part
 - WSDL interface implementation part
 - WSDL Message Exchange Patterns
 - WSDL Usage

WEB SERVICE DESCRIPTION LANGUAGE



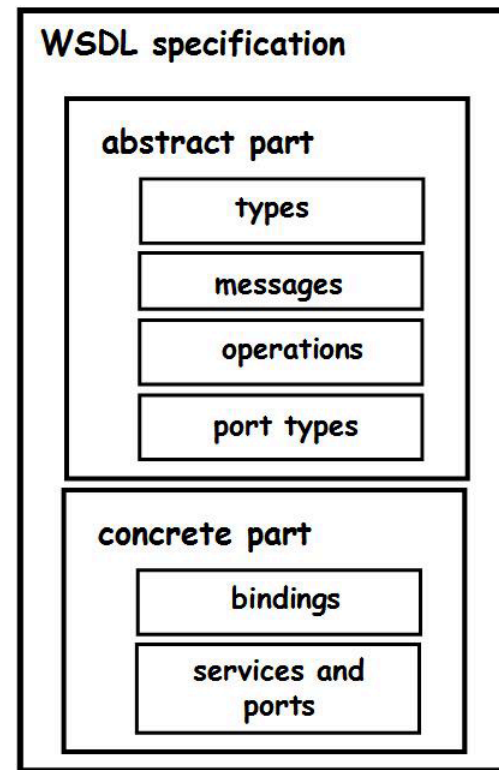
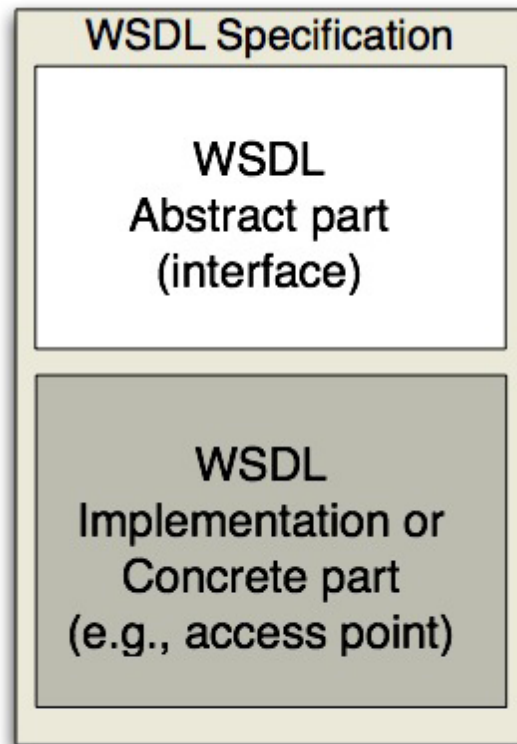
Web Services Description Language

- 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 as well as the data **types**
- 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.

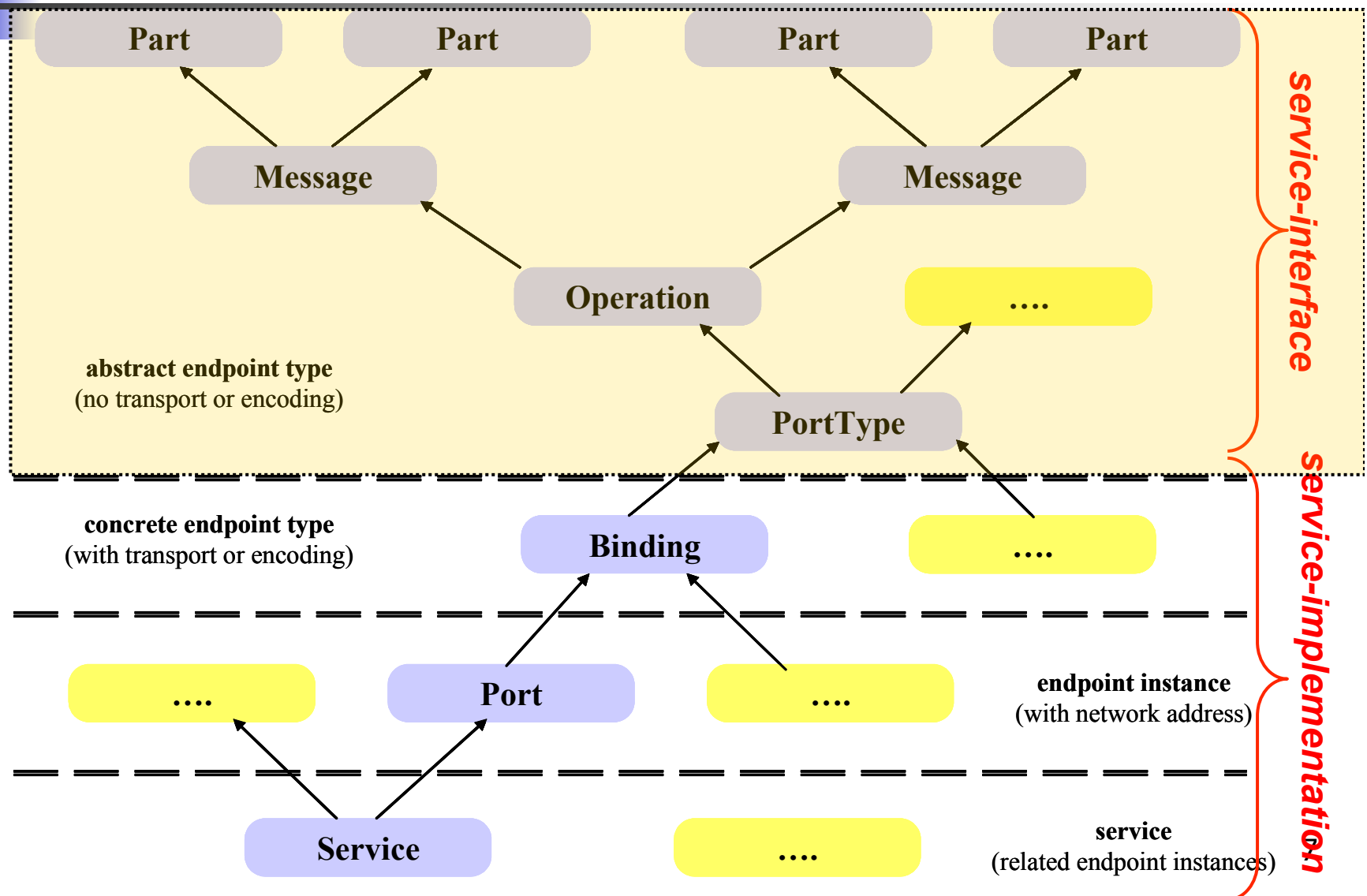
Web Services Description Language: Two parts

Maurizio Marchese – IntroSDE



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

WSDL elements hierarchy



WSDL STRUCTURE: ABSTRACT SERVICE INTERFACE



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
  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 (<http://stock.example/org/schema>), you can find definitions for symbol and price, for example:

```
<xsd:schema...>
  <xsd:element name="symbol" type="xsd:string"/>
  <xsd:element name="price" type="xsd:string"/>
</xsd:schema>
```



WSDL Main Elements: **part** & **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: **operations** & **portType**

- The **portType** element contains a named set of **operations**. It defines the functionality of the Web service (i.e., what the service does).

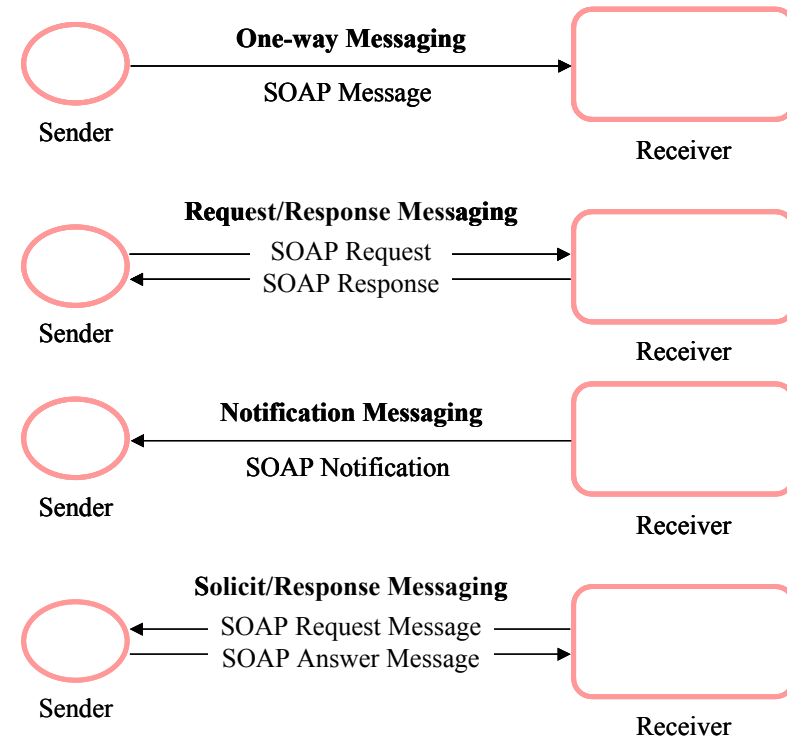
```
<wsdl:portType name="StockBrokerQueryPortType">
  <wsdl:operation name="GetStockPrice">
    <wsdl:input message="tns:StockPriceRequestMessage"/>
    <wsdl:output message="tns:StockPriceResponseMessage"/>
    <wsdl:fault name="UnknownSymbolFault"
      message="tns:StockSymbolNotFoundMessage"/>
 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

- Each operation combines input, output and fault messages from a set of message types defined previously
- Note: WSDL 2.0 changes portType to interface
- Note: Multiple fault messages can be defined.

WSDL MESSAGE EXCHANGE PATTERNS

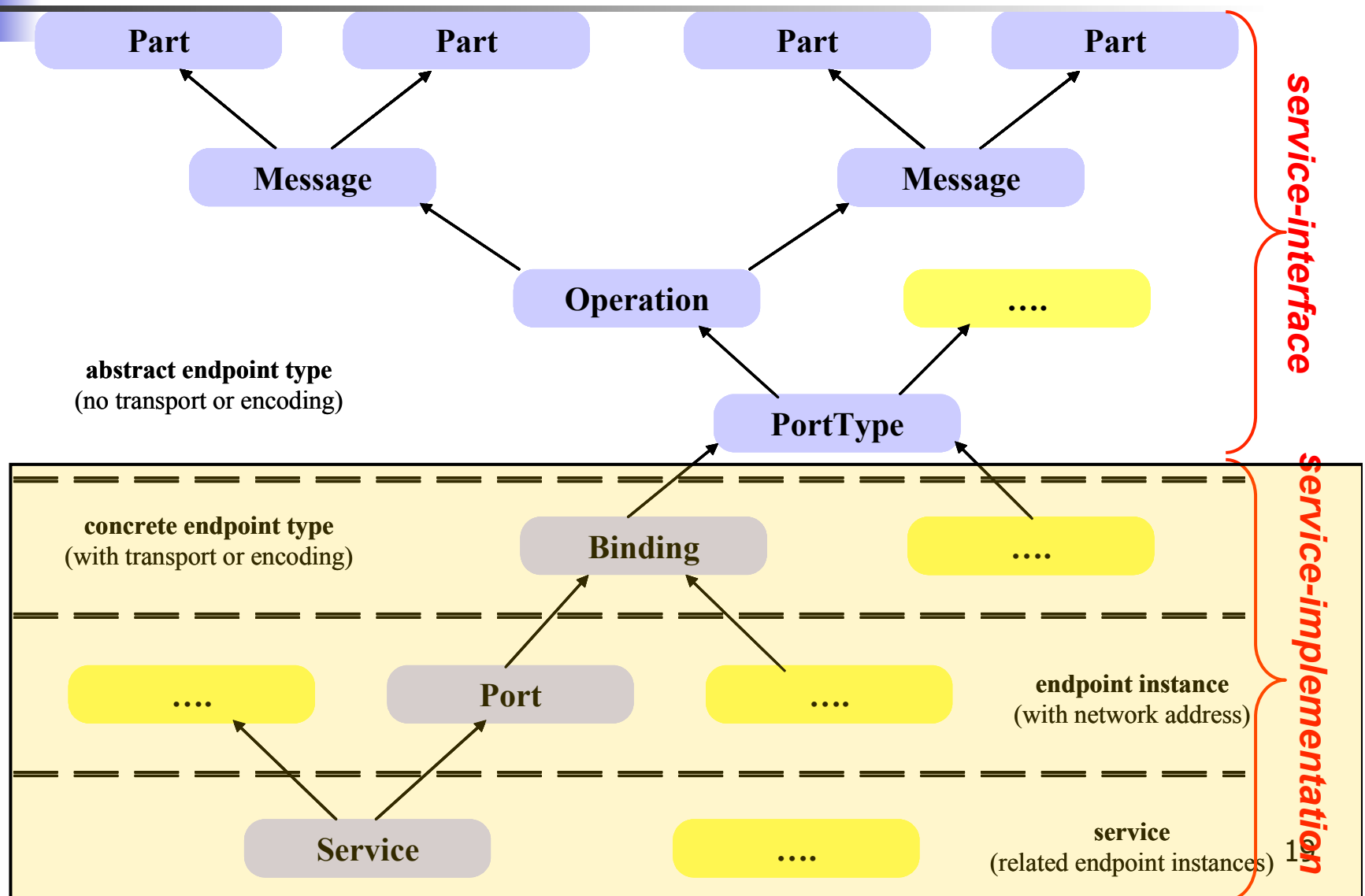
WSDL Message Exchange Patterns

- Not all operations will have a single input, output and fault.
- The **operation** element indicates also 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, consumes it 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**
- asynchronous interactions are defined using **one-way** and **notifications** operations
- Variations of the above exist (e.g., Input-only with potential fault message); cf. WSDL 2.0 Message Exchange Patterns (MEP).



WSDL STRUCTURE: SERVICE IMPLEMENTATION

WSDL elements hierarchy





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**

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: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 value is required (no default value).

`soap:body` & `soap:fault`

- 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: **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: **port** & **service**

The **service** element:

- Aggregates **port** elements that define specific network endpoints (addresses) 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 and network endpoints together.

WSDL USAGE



WSDL and proxy class - 1

■ How to use WSDL ?

- Developers can implement web services logic within their applications by incorporating available web services
- The mechanism that makes this possible is called the **proxy class**.
 - Proxy classes enable developers to reference remote web-services and use their functionality within a local application, as if the data the services return were generated locally.
 - The application developer communicates with any remote objects by sending messages to these local objects, which are commonly known as **proxy objects**.

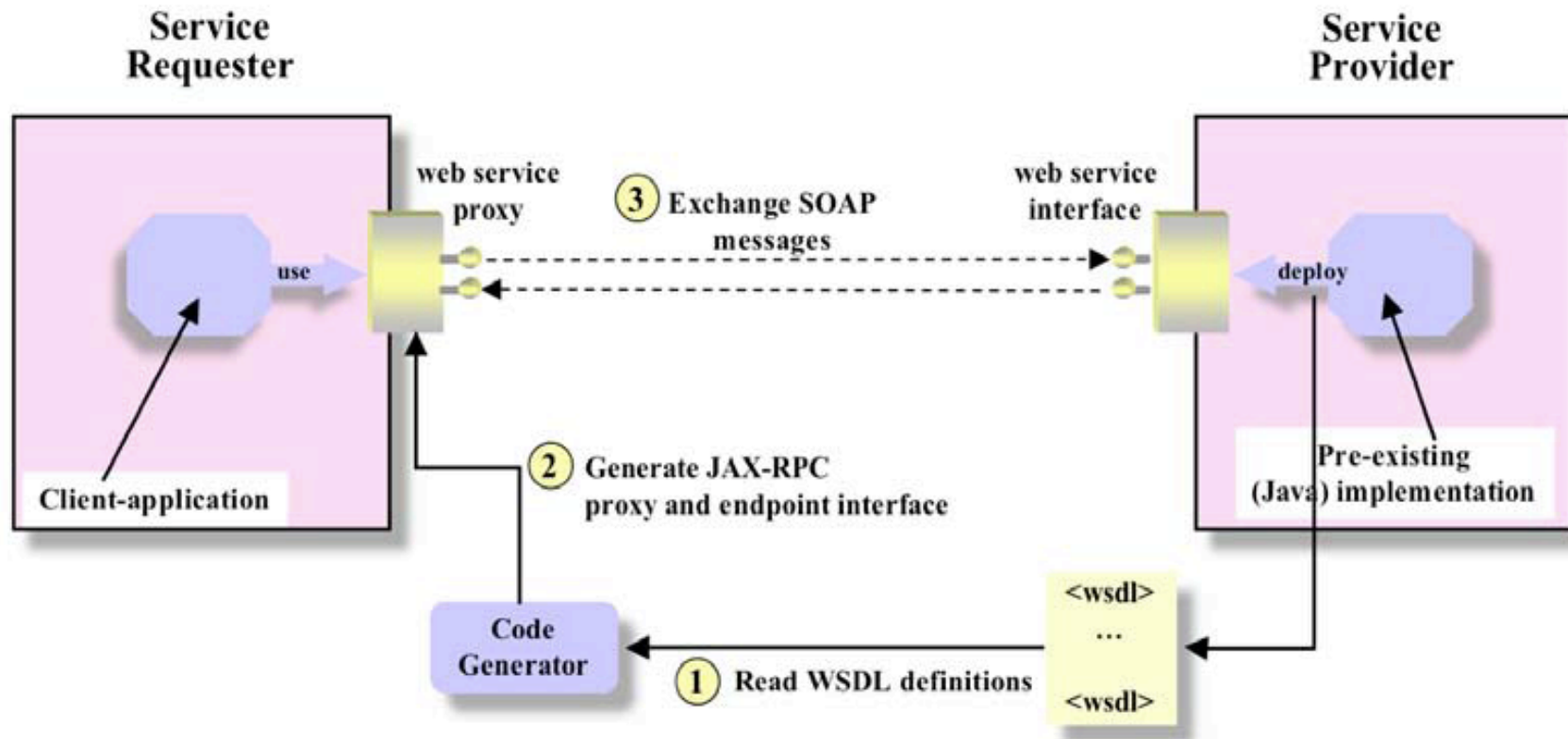


WSDL and proxy class - 2

- The **proxy classes (or *stub* classes)** are **client-side images of the remote (provider) object classes** that implement the web services.
- The **server-side object classes are commonly known as *skeletons*** in the distributed computing systems domain
- The proxy object is simply a local object with methods that are merely a pass-through to the web service it is representing.
- The role of the proxy class is to act as the local representative for the remote object and basically is, to the client, the remote reference. However, instead of executing an invocation, the proxy forwards it in a message to a remote object.

Generating proxies from WSDL code generators (example Java)

Maurizio Marchese – IntroSDE





WSDL code generators

- WSDL is well suited (**XML expressiveness**) for code generators that can read WSDL definitions and generate a programming interface for accessing a web service.
 - For instance, a JAX-RPC provider may use WSDL 1.1 to generate Java RMI interfaces and network stubs, which can be used to exchange messages with a web service interface.
- WSDL code generator tools allow automatic creation of web services, automatic generation of WSDL files and invocation of web services.
- These toolkits speed the creation of web-services
 - by generating the service implementation template code from the WSDL specifications, leaving only the application-specific implementation details to the developer.
 - They also simplify the development of client applications by generating service proxy code from the WSDL specification.
- Several code generators can generate interfaces and network stubs from WSDL documents. These include amongst others IBM Websphere Studio Application Developer, Microsoft .NET Studio, and **Apache Axis**.

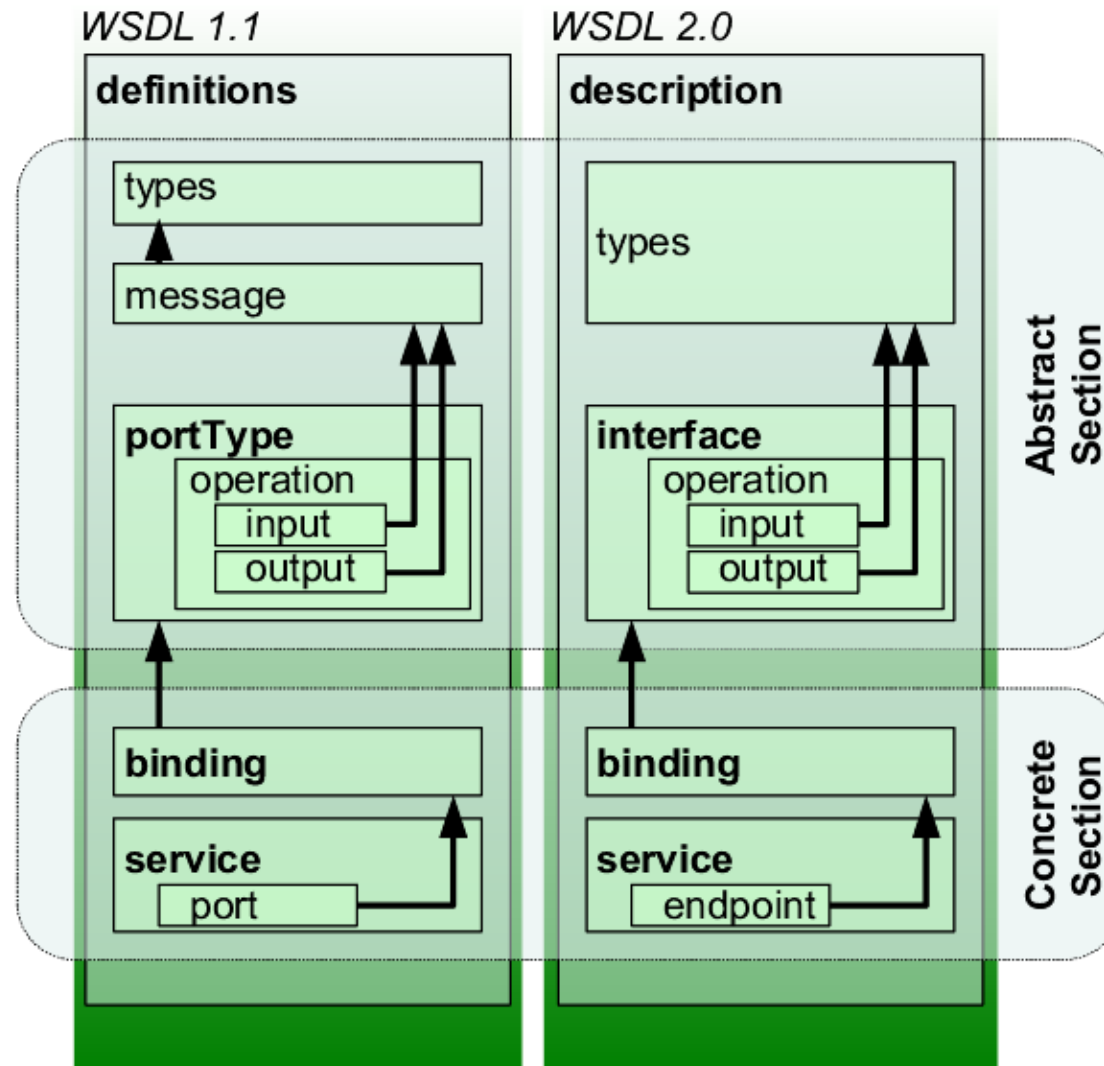


WSDL State-of-the-art

- The World Wide Web Consortium (W3C) has issued Web Services Description Language (WSDL) 1.2 and WSDL 1.2 Bindings as W3C Public Working Drafts.
- WSDL version 1.2/2.0 enhancements to version 1.1 include the following:
 - **Language clarifications** that make it easier for developers to understand and use WSDL.
 - **Support** for **XML** Schemas and **XML** Information Set.
 - **Better definition** for the HTTP 1.1 binding and a binding for SOAP 1.2, which allows description of services using the most current version of SOAP.
 - Adopting a conceptual **framework approach** to define a description model of WSDL. This **conceptual model** comprises a set of components with attached properties, which collectively describe a web service. Each subsection in this conceptual model describes a different type of component, its defined properties, and its representation as an XML Information Set.

WSDL 1.1 vs WSDL 2.0

Maurizio Marchese – IntroSDE

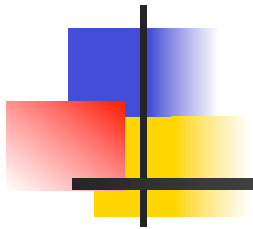




WSDL Outlook

- WSDL is in its current version an extension of the IDL model to support interaction through the Internet:
 - XML as syntax and type system
 - possibility of grouping operations into a service
 - different options for accessing the service (addresses and protocols)
- This is its great advantage ...
 - it is straightforward to adapt existing middleware platforms to use or support WSDL
 - automatic translation from existing IDLs to WSDL is trivial
- ... but also the disadvantage
 - electronic commerce and B2B interactions are not single service calls
 - WSDL does not reflect the structure of the procedures to follow to correctly interact with a service (conversations)
 - business protocol = set of valid conversations
- Without a business protocol, most of the development work for business applications is still manual

Adapted from teaching material shared by
Helen Paik, Boualem Bentallah,
Marcello La Rosa, Fabio Casati, Maurizio Marchese



Web services: UDDI

Universal Description, Discovery and Integration



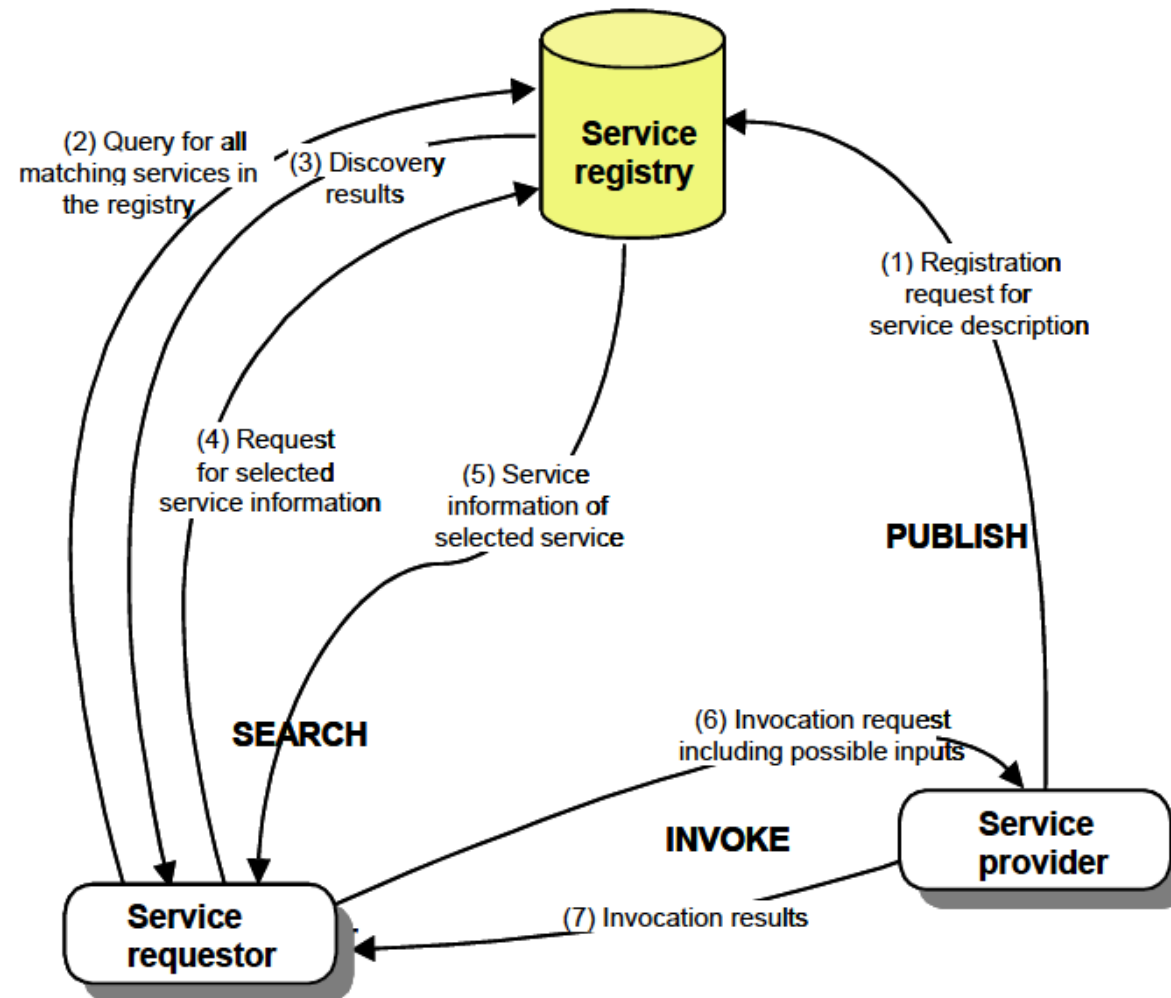
Service Registries

- One way to discover Web services is through a service registry. This requires describing and registering the Web service.

Alternatives?

- 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 higher level abstraction of the submitted document.

The overall idea





Service Discovery

- Service discovery is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.
- Interrogating services involves 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 providers.
- 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:

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

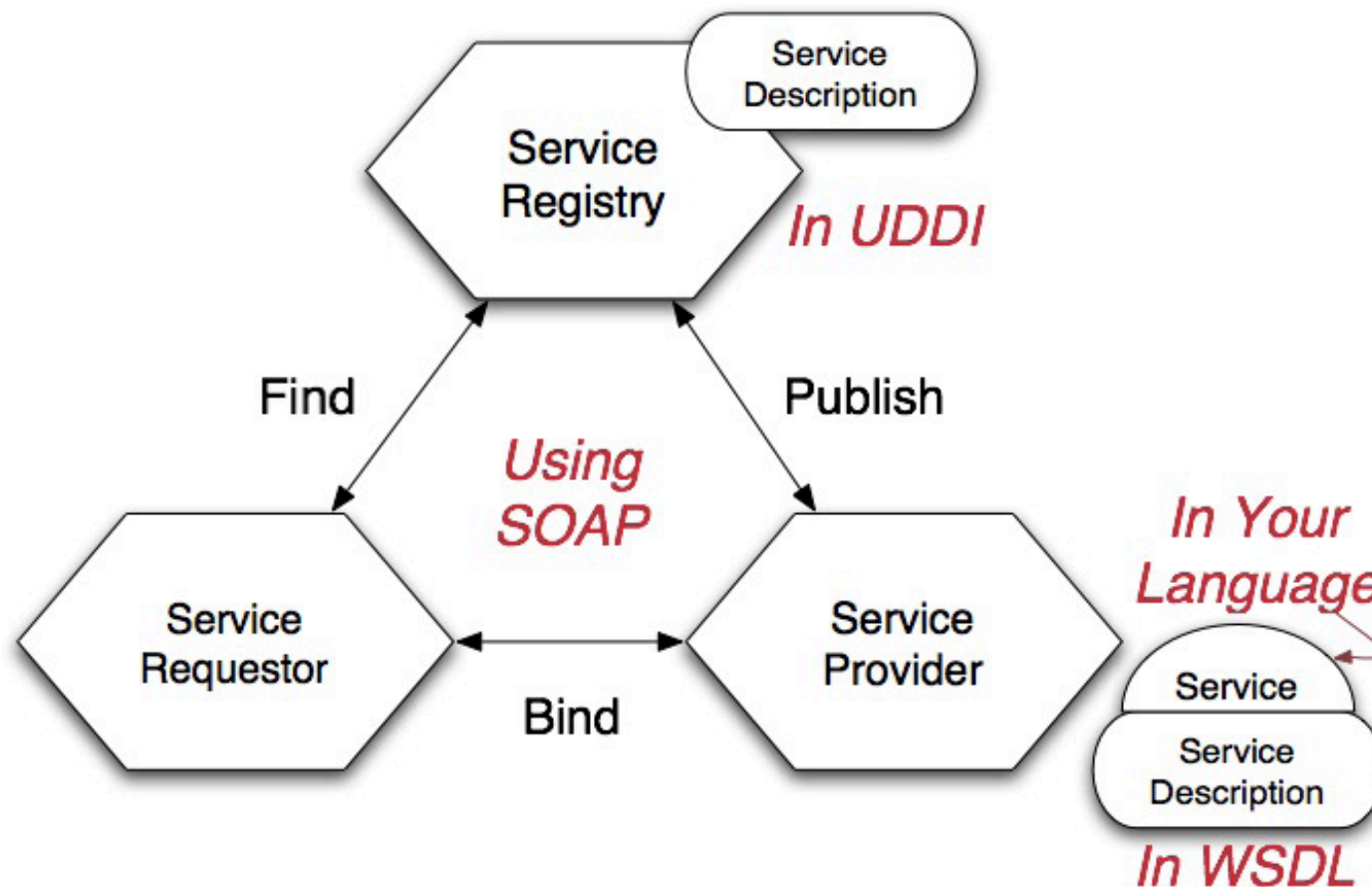
Dynamic:

- 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
- E.g. based on application logic's 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.

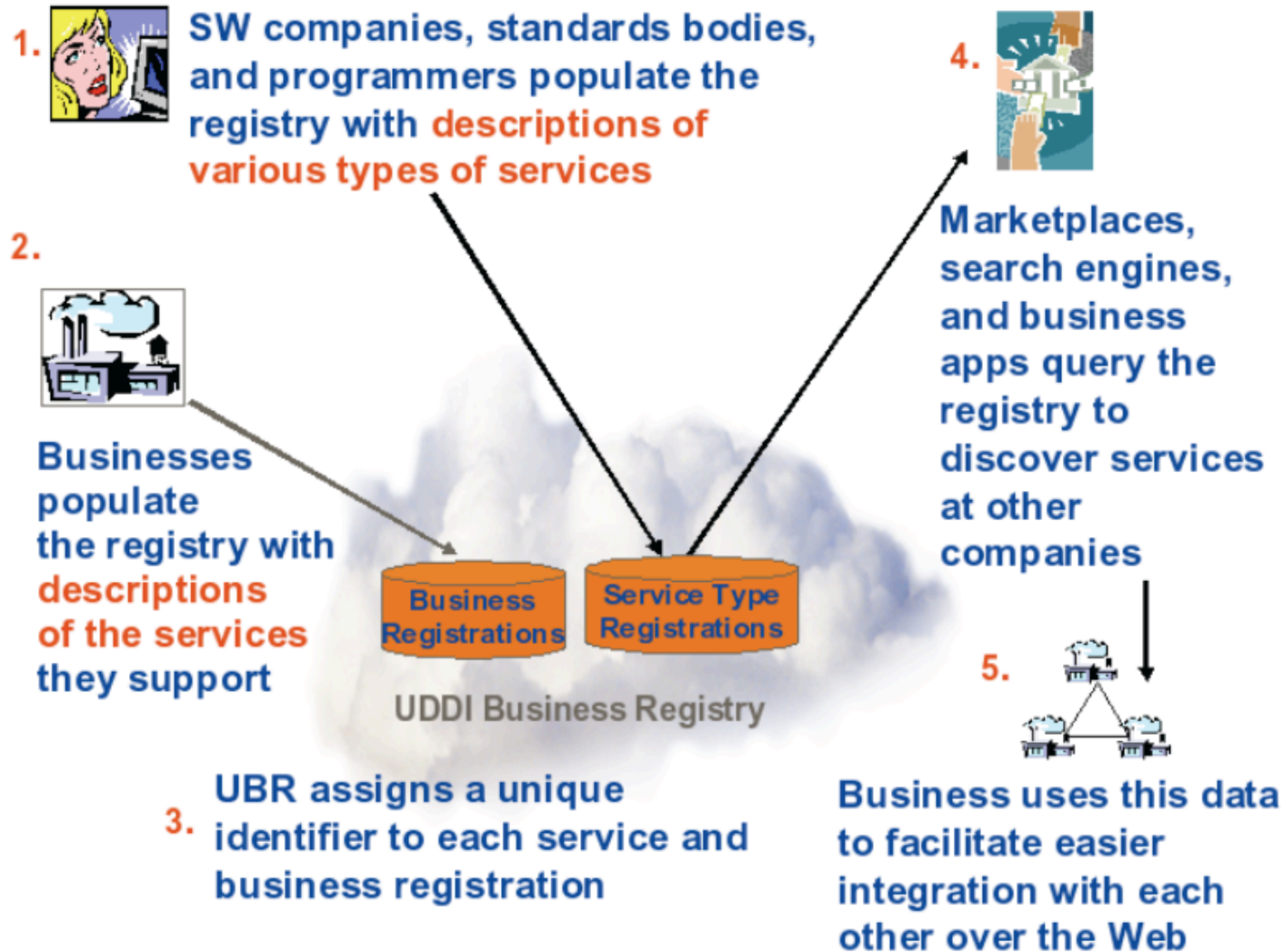
UDDI

UDDI

Universal Description, Discovery and Integration



UDDI and the big idea





UDDI

- UDDI is a registry (not repository) of Web services
- IBM, Microsoft and SAP *used* to host public UDDI registry
 - www.theserverside.net/news/thread.tss?thread_id=38136
- 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: /metro/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)
- Its main body now is OASIS: <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