
INFOH509 XML & Web Technologies
Lecture 11



BIGWS-* WEB SERVICES

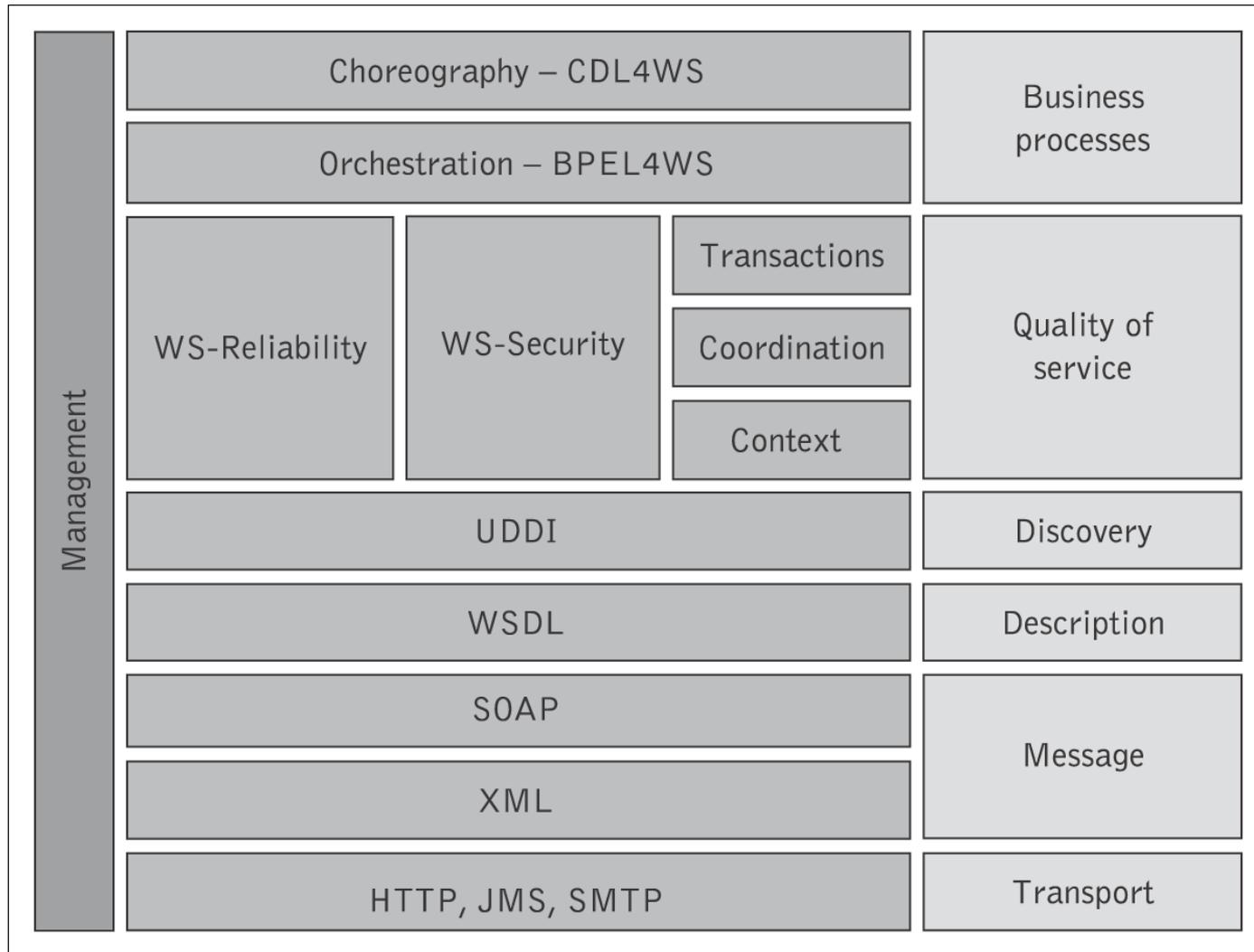
SERVICES



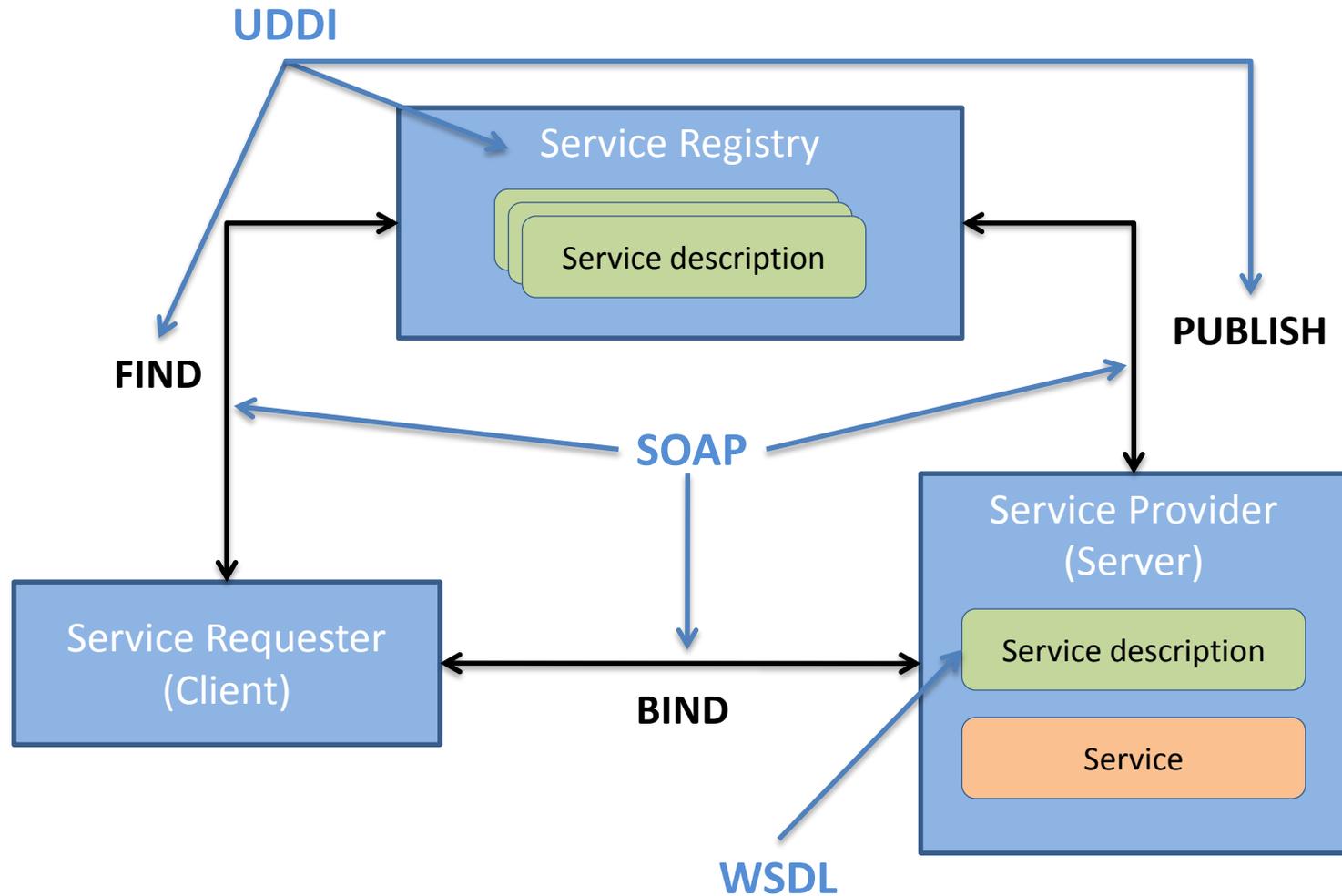
Two competing technology stacks

- **Big Web Services (WS-*)**  **This lecture**
 - Various (complex) protocols on top of HTTP (SOAP, UDDI, WSDL, WS-Addressing, ...)
 - Is mostly used to implement RPC-style services, but can be used to implement any of the three
 - Lots of standards! Primarily meant to create web services that involve more than 2 peers.
- **RESTful Web Services**  **Previous lecture**
 - Use ONLY HTTP and standard media types
 - Restricted to Resource-style services
 - Conceptually simpler, but mainly restricted to web services that are limited to two endpoints

WS-* Technology Stack



WS-* Message, Description & Discovery



- While SOAP & WSDL are frequently used, UDDI has never caught on.

Part I

SOAP



The history of SOAP

- Originally, SOAP was conceived as a minimalistic infrastructure to perform **remote procedure calls** (RPC) over the Web:
 - SOAP messages use XML as a format to exchange data between systems.
 - SOAP messages can be sent over HTTP (but also other protocols like SMTP) as a transport protocol. (This avoids system heterogeneity & firewall issues.)
- Currently, SOAP is no longer limited to RPC.

The history of SOAP (cont.)

- Originally proposed by Microsoft (then: the Object Access Protocol).
- Version 1.0 standardized by the W3C in 1999
- Currently at version 1.2

What is SOAP?

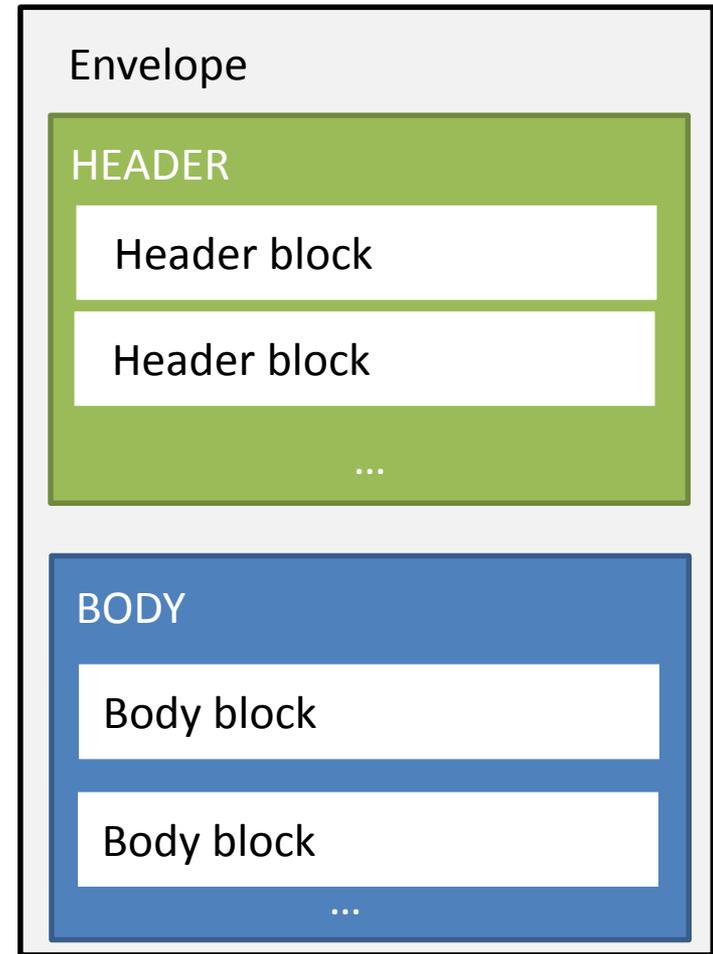
- SOAP = a messaging framework (much like HTTP)
- The SOAP specification covers the following four main areas:
 - **Message format:** A format for one-way communication describing how a message can be packed into an XML document.
 - **Processing model:** A set of rules that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message. Describes what parts of the messages should be read by whom and how to react in case of failure.
 - **Extensibility model:** How the basic message constructs can be extended with application-specific constructs
 - **Protocol binding framework:** A description of how SOAP messages can be transported using different protocols (e.g., HTTP, SMTP, ...).
 - A set of conventions on how to turn an RPC call into a SOAP message and back. (Since version 1.1. SOAP is not restricted to the RPC programming model).

What is SOAP?

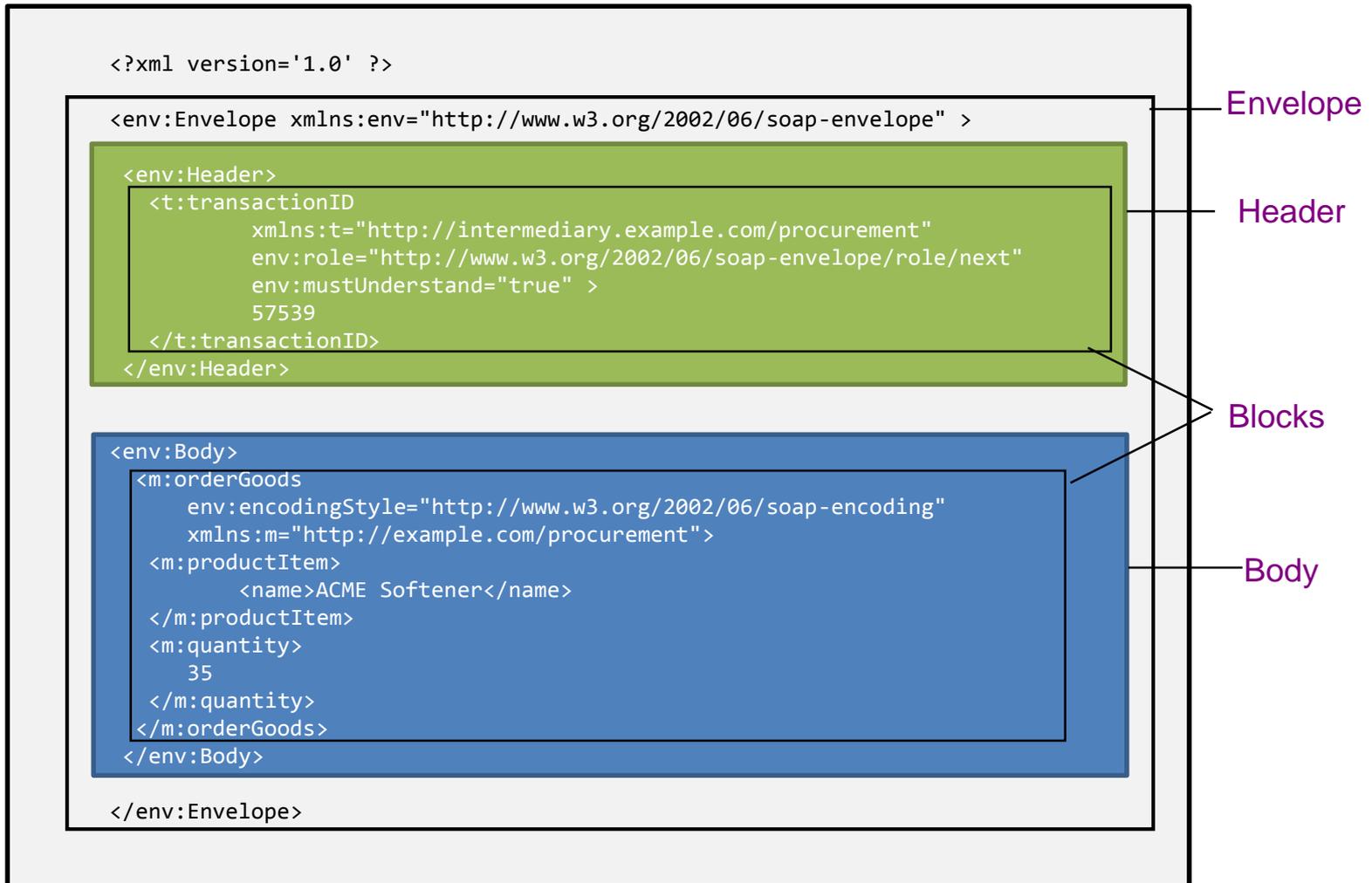
- SOAP = a messaging framework (much like HTTP)
- The SOAP specification covers the following four main areas:
 - **Message format:** A format for one-way communication describing how a message can be packed into an XML document.
 - **Processing model:** A set of rules that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message. Describes what parts of the messages should be read by whom and how to react in case of failure.
 - **Extensibility model:** How the basic message constructs can be extended with application-specific constructs
 - **Protocol binding framework:** A description of how SOAP messages can be transported using different protocols (e.g., HTTP, SMTP, ...).
 - A set of conventions on how to turn an RPC call into a SOAP message and back. (Since version 1.1. SOAP is not restricted to the RPC programming model).

The SOAP message structure

- SOAP Message
= envelope in XML format
- Two parts:
 - Headers (optional)
“The stickers”
Contains meta data (security, transaction data, ...)
 - Body
“Application data”
- SOAP only specifies the structure of the message, not the semantics of header/body



An example SOAP message



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 (e.g., for 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:
 - Role attribute: who should process that particular header block.
 - Boolean mustUnderstand attribute: indicates whether it is mandatory to process the header. If a header is directed at a node (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

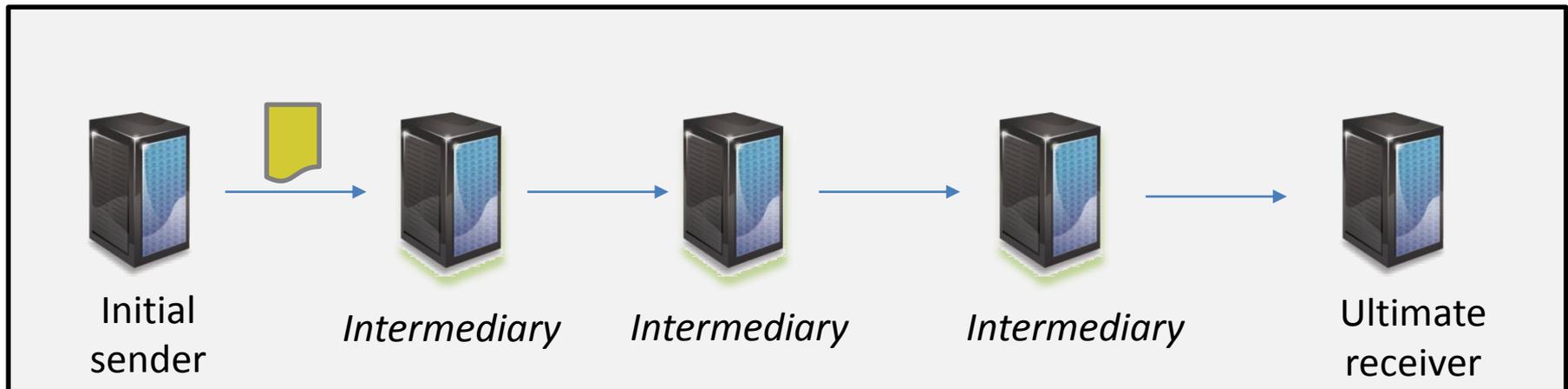
- The SOAP body is the area of the SOAP message, where the application specific XML data (payload) being exchanged in the message is placed.
- The **<Body> element** must be present and is an immediate child of the envelope. It may contain a number of child elements, called body entries, but it may also be empty. The **<Body> element** contains either of the following:
 - **Application-specific data** is the information that is exchanged with a Web service. The SOAP <Body> is where the method call information and its related arguments are encoded. It is where the response to a method call is placed, and where error information can be stored.
 - 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.

What is SOAP?

- SOAP = a messaging framework (much like HTTP)
- The SOAP specification covers the following four main areas:
 - **Message format:** A format for one-way communication describing how a message can be packed into an XML document.
 - **Processing model:** 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. Describes what parts of the messages should be read by whom and how to react in case of failure.
 - **Extensibility model:** How the basic message constructs can be extended with application-specific constructs
 - **Protocol binding framework:** A description of how SOAP messages can be transported using different protocols (e.g., HTTP, SMTP, ...).
 - A set of conventions on how to turn an RPC call into a SOAP message and back. (Since version 1.1. SOAP is not restricted to the RPC programming model).

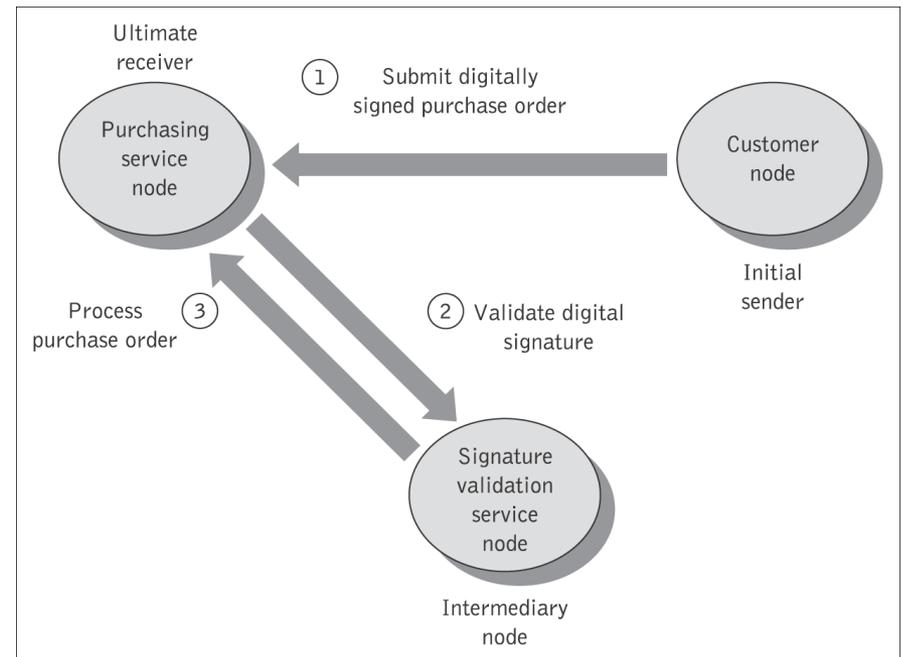
The SOAP message path

- A SOAP message can pass through multiple hops on the way from the initial sender to the ultimate receiver
- The entities involved in transporting the message are called **SOAP nodes**
- **SOAP Intermediaries** can manipulate and forward the message
- Every SOAP node assumes a certain **role** which influences how the message is processed at the node



SOAP Intermediaries

- SOAP headers have been designed in anticipation of participation of other SOAP processing nodes – called **SOAP intermediaries** – along a **message's path** from an initial SOAP sender to an ultimate SOAP receiver.
- A SOAP message travels along the message path from a sender to a receiver.
- All SOAP messages start with an initial sender, which creates the SOAP message, and end with an ultimate receiver.



Message Processing Model

- For each message received, every SOAP node on the message path must **process the message** as follows
 - Decide in which roles to act (standard roles: `next` or `ultimateReceiver`, or other application-defined roles). These roles may also depend on the contents of the message.
 - Identify the mandatory header blocks targeted at the node (matching role, `mustUnderstand=true`)
 - If a mandatory header block is not understood by the node, a fault must be generated. The message must not be processed further.
- Process the mandatory header blocks and, in case of the ultimate receiver, the body. Other header blocks targeted at the node may be processed. The order of processing is not significant.
- SOAP intermediaries will finally forward the message
 - Processed header blocks may be removed depending on the specification for the block.
 - Header blocks which were targeted at the intermediary but not processed are relayed only if the `relay` attribute is set to `true`.
- Active SOAP intermediaries may also change a message in ways not described here (e.g., encrypt the message).

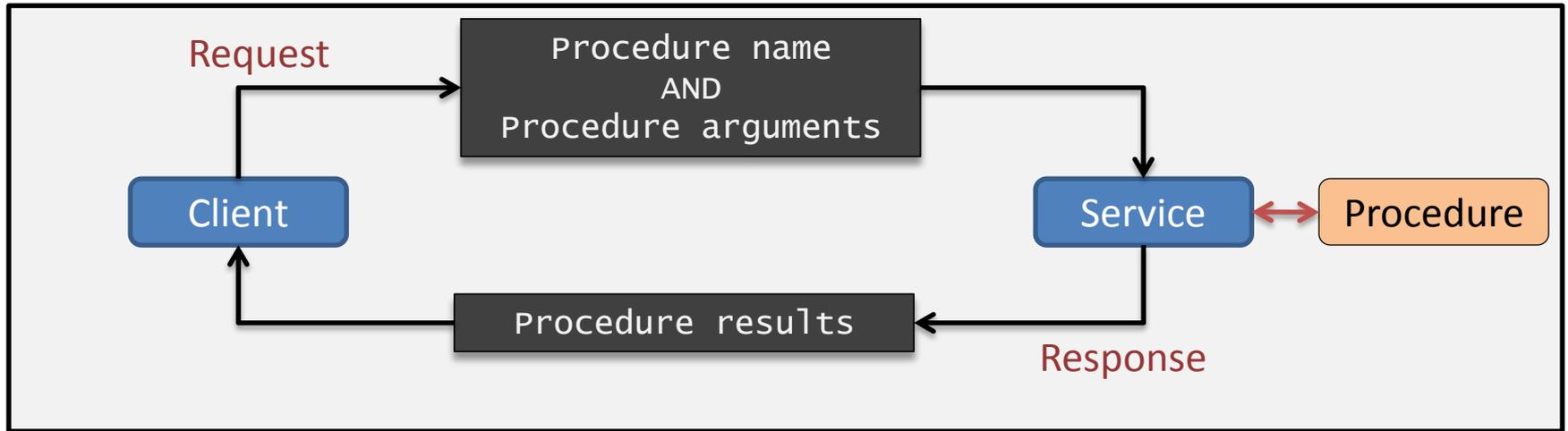
Example of a header with routing

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:order xmlns:m="http://www.plastics_supply.com/purchase-order"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:order-no >uuid:0411a2daa</m:order-no>
      <m:date>2004-11-8</m:date>
    </m:order>
    <n:customer xmlns:n="http://www.supply.com/customers"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name> Marvin Sanders </n:name>
    </n:customer >
  </env:Header>
  <env:Body>
    <!-- Payload element goes here -->
  </env:Body>
</env:Envelope>
```

What is SOAP?

- SOAP = a messaging framework (much like HTTP)
- The SOAP specification covers the following four main areas:
 - **Message format:** A format for one-way communication describing how a message can be packed into an XML document.
 - **Processing model:** 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. Describes what parts of the messages should be read by whom and how to react in case of failure.
 - **Extensibility model:** How the basic message constructs can be extended with application-specific constructs
 - **Protocol binding framework:** A description of how SOAP messages can be transported using different protocols (e.g., HTTP, SMTP, ...).
- A set of conventions on how to turn an **RPC call** into a SOAP message and back. (Since version 1.1. SOAP is not restricted to the RPC programming model).

RPC Style (1/2)



- Client sends message to a remote server and blocks while waiting for response
- Request message identifies the procedure to be executed and its arguments
- Server decodes message, maps message arguments directly to input parameters, executes procedure, and sends (serialized) results back to client

RPC Style (2/2)

- **Pros:**

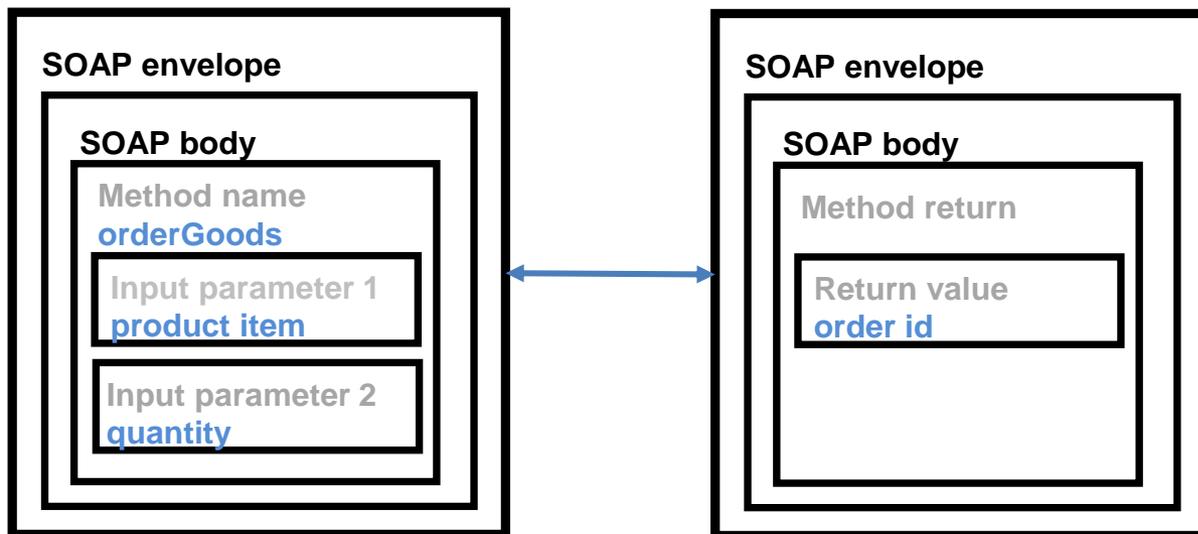
- Very easy to implement (lots of frameworks that automate the process, e.g. AX-WS framework for Java)

- **Cons:**

- Usually inflexible and fragile: tight coupling between client and service, if procedure needs to change (e.g., number of arguments), all clients need to be rewritten.
- Usually restricted to **synchronous communication** (client blocks while waiting for response)

RPC style SOAP services

- In a remote procedure call (RPC)-style Web service clients express their request as a method call with a set of arguments, which returns a response containing a return value.



RPC style SOAP services (cont.)

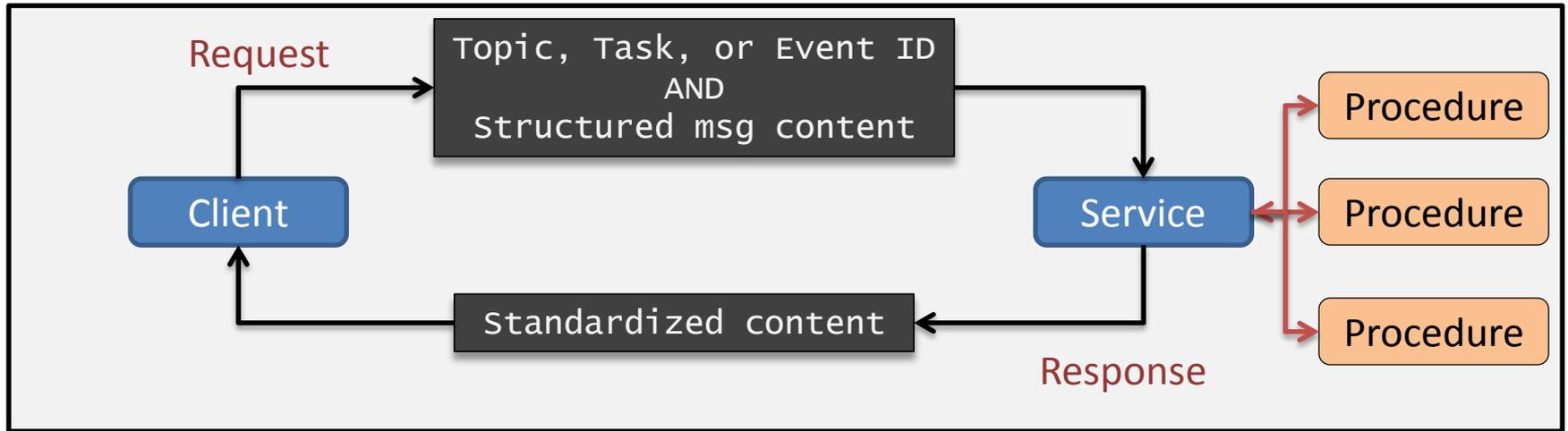
```
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:m="http://www.plastics_supply.com/product-prices">
  <env:Header>
    <tx:Transaction-id xmlns:t="http://www.transaction.com/transactions"
      env:mustUnderstand="true">
      512
    </tx:Transaction-id>
  </env:Header>
  <env:Body>
    <m:GetProductPrice>
      <product-id> 450R60P </product-id >
    </m:GetProductPrice >
  </env:Body>
</env:Envelope>
```

Example
Request

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:m="http://www.plastics_supply.com/product-prices">
  <env:Header>
    <!-- Optional context information -->
  </env:Header>
  <env:Body>
    <m:GetProductPriceResponse>
      <product-price> 134.32 </product-price>
    </m:GetProductPriceResponse>
  </env:Body>
</env:Envelope>
```

Example
Response

Message-based style



- In a message-based API, messages are not derived from the signatures of remote procedures.
- Instead, messages may carry information on specific topics, tasks to execute, and events.
- The server selects the correct procedure to execute based on the message content

Message-based Style (2/2)

- **Pros:**

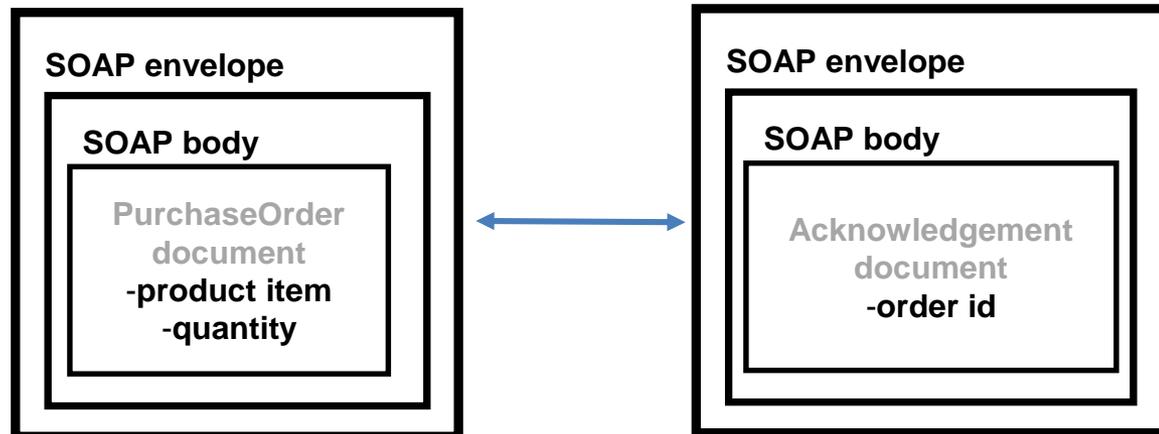
- Looser coupling between clients and servers
- Support for asynchronous communication [necessary on web-scale networks]

- **Cons:**

- Messages must be standardized somehow. This is easy if communication is within the same organization, but more difficult when many parties are involved.

Message-style SOAP Services

- In the document-style of messaging, the SOAP <Body> contains an XML document fragment. The <Body> element reflects no explicit XML structure.
- The SOAP run-time environment accepts the SOAP <Body> element as it stands and hands it over to the application it is destined for unchanged. There may or may not be a response associated with this message.



Message-style SOAP Services (cont.)

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">

  <env:Header>
    <tx:Transaction-id xmlns:t="http://www.transaction.com/transactions"
      env:mustUnderstand="true">
      512
    </tx:Transaction-id>
  </env:Header>
  <env:Body>
    <po:PurchaseOrder oderDate="2004-12-02"
      xmlns:po="http://www.plastics_supply.com/POs">
      <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> Yara 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>
```

SOAP Fault Element

- SOAP provides a model for handling faults arise.
- It distinguishes between the conditions that result in a fault, and the ability to signal that fault to the originator of the faulty message or another node. The SOAP <Body> is the place where fault information is placed.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:m="http://www.plastics_supply.com/product-prices">
  <env:Header>
    <tx:Transaction-id xmlns:t="http://www.transaction.com/transactions"
                      env:mustUnderstand='1'>
      512
    </tx:Transaction-id>
  </env:Header>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode> <env:Value> m:InvalidPurchaseOrder </env:Value> </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-UK"> Specified product did not exist </env:Text>
      </env:Reason>
      <env:Detail> arbitrary XML here </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

What is SOAP?

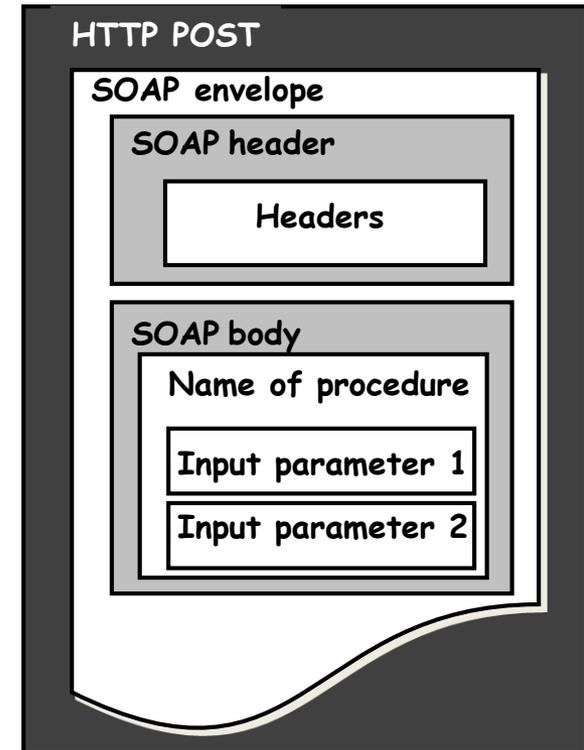
- SOAP = a messaging framework (much like HTTP)
- The SOAP specification covers the following four main areas:
 - **Message format:** A format for one-way communication describing how a message can be packed into an XML document.
 - **Processing model:** A set of rules that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message. Describes what parts of the messages should be read by whom and how to react in case of failure.
 - **Extensibility model:** How the basic message constructs can be extended with application-specific constructs
 - **Protocol binding framework:** A description of how SOAP messages can be transported using different protocols (e.g., HTTP, SMTP, ...).
 - A set of conventions on how to turn an RPC call into a SOAP message and back. (Since version 1.1. SOAP is not restricted to the RPC programming model).

SOAP Protocol Binding Framework

- SOAP messages can be transferred using any protocol,
- A binding of SOAP to a transport protocol is a description of how a SOAP message is to be sent using that transport protocol.
- A binding specifies how response and request messages are correlated.
- The SOAP binding framework expresses guidelines for specifying a binding to a particular protocol

SOAP HTTP Binding

- SOAP Messages are typically transferred using HTTP.
- The binding to HTTP defined in the SOAP specification
- SOAP can use GET or POST. With GET, the request is not a SOAP message but the response is a SOAP message, with POST both request and response are SOAP messages (in version 1.2, version 1.1 mainly considers the use of POST).
- SOAP uses the same error and status codes as those used in HTTP so that HTTP responses can be directly interpreted by a SOAP module.



SOAP over HTTP Example

```
POST /order/billing HTTP/1.1
```

```
Host: billing.eserver.com
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: 12354
```

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<env:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<env:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:env="http://www.w3.org/2003/05/soap-envelope">
```

```
<env:Body>
```

```
<orderCreated xmlns="http://orders.com/" />
```

```
</env:Body>
```

```
</env:Envelope>
```

Part II

WSDL:

WEB SERVICE DESCRIPTION LANGUAGE



Service Description

- Any service that you provide, others will have to program and use.
- The purpose of a service = abstract from system & programming language heterogeneity
 - For instance, neither the service requestor nor the provider should be aware of each other's technical infrastructure, programming language, or distributed object framework (if any).
- **A Service description** is a machine understandable document that
 - describes the operations of a web service,
 - describes how the web service should be accessed (transport protocol, access point),
 - describes what the content of the messages that are exchanged is

It hence reduces the amount of required common understanding and custom programming and integration.

WSDL

- WSDL = Web Service Description Language
- A WSDL document is an XML document that specifies the details of the complete interfaces exposed by web services:
 - **what** a service does, i.e., the operations the service provides,
 - **where** it resides, i.e., details of the protocol-specific address (e.g., its URI)
 - **how** to invoke it, i.e., details of the data formats (e.g., XML) and protocols (e.g., SOAP over HTTP) necessary to access the service's operations.

WSDL History

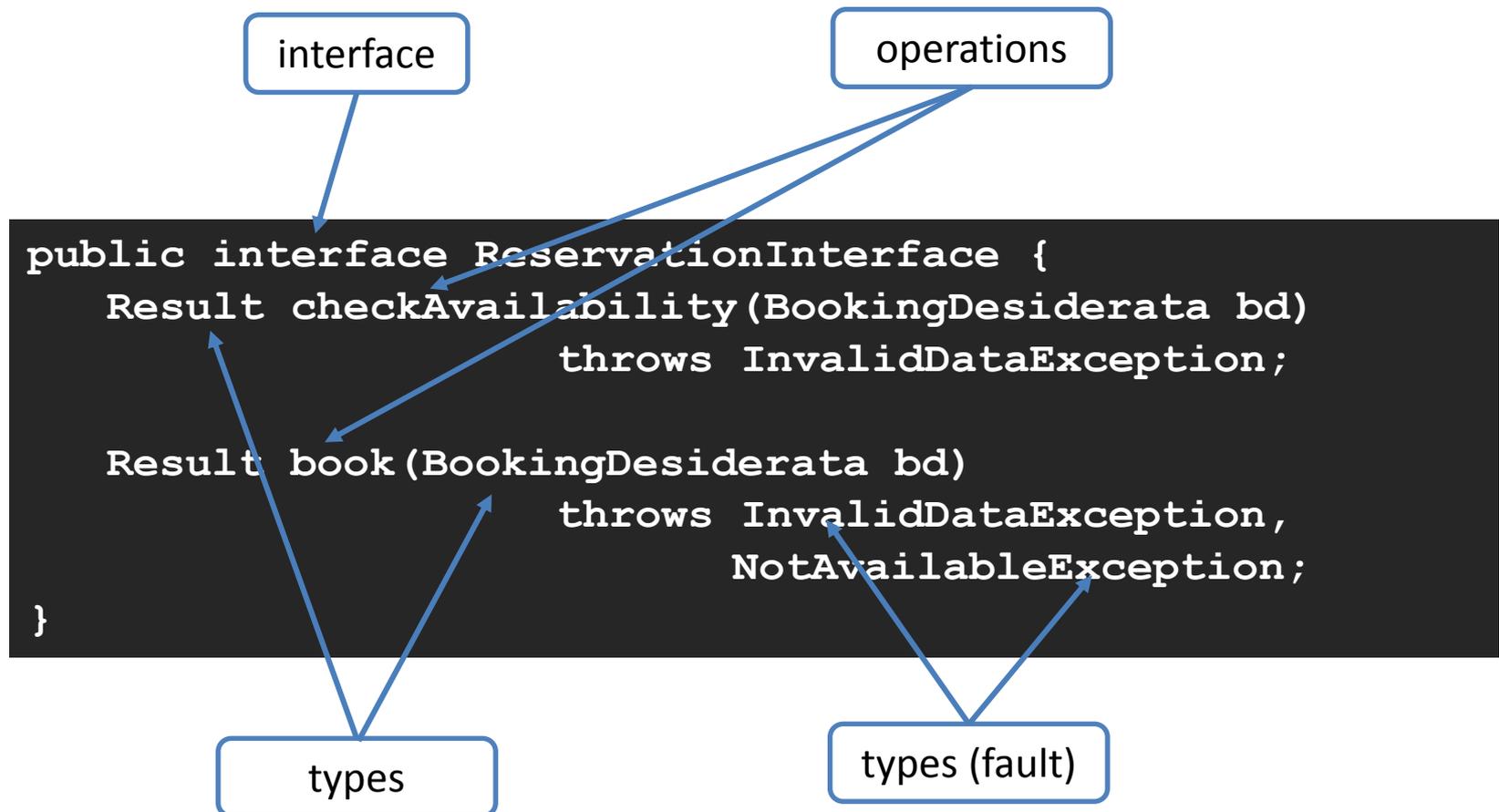
- Evolution of the standard:
 - WSDL 1.0 (first release) – 2001
 - WSDL 1.1. – 2003
 - WSDL 2.0 (current version) – 2007
- Originally, WSDL was primarily meant to describe SOAP web services; with WSDL 2.0 it is also possible to describe REST web services.

CAUTION:

What follows is an overview of WSDL 2.0, which is a significant (non-compatible) change with earlier WSDL versions!

WSDL document structure

- The components of a WSDL file parallel those of an interface in a programming language



WSDL document structure

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl">
  <wsdl:types> ... </wsdl:types>
  <wsdl:interface> ... </wsdl:interface>
  <wsdl:binding> ... </wsdl:binding>
  <wsdl:service> ... </wsdl:service>
</wsdl:description>
```

- The `<wsdl:types>` element defines the « types » of the messages that are to be exchanged.
- By default, WSDL assumes that the messages will be in XML, and it uses XML Schema to describe the exact structure of the XML that needs to be exchanged.

Note

In principle, WSDL allows other schema languages or type systems (one could imagine, an other language to describe the structure of JSON content to be exchanged), but this seems to be not widely implemented.

Defining message types

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsdl"
  targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
  ... >
...
<types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://greath.example.com/2004/schemas/resSvc">

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

    <xs:element name="checkAvailabilityResult" type="xs:double"/>
    <xs:element name="invalidDataError" type="xs:string"/>
  </xs:schema>
</types>
...
</description>
```

WSDL document structure

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl">
  <wsdl:types> ... </wsdl:types>
  <wsdl:interface> ... </wsdl:interface>
  <wsdl:binding> ... </wsdl:binding>
  <wsdl:service> ... </wsdl:service>
</wsdl:description>
```

- The `<wsdl:interface>` element defines the « interfaces » provided by the web service (as collections of operations).
- For each operation, it specifies the kind of message exchange pattern (MEP). Popular MEPS are: request + response, request-only, response-only, ...

Defining interfaces

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsdl"
  targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
  xmlns:wsdlix="http://www.w3.org/ns/wsdl-extensions">

<types> ... </types>

<interface name = "reservationInterface" >
  <fault name = "invalidDataFault"

  <operation name="opCheckAvailabi
    pattern="http://www.w3.org/ns/wsdl/in-out"
    style="http://www.w3.org/ns/wsdl/style/iri"
    wsdlx:safe = "true">
    <input messageLabel="In" element="ghns:bookingDesiderata " />
    <output messageLabel="Out" element="ghns:checkAvailabilityResult" />
    <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
  </operation>

  <...
  </description>
```

Request+response MEP

Message type to send for operation call
(element defined in types)

Message type to send for operation call
is "safe": does not induce client
obligation (like buying something)

WSDL document structure

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl">
  <wsdl:types> ... </wsdl:types>
  <wsdl:interface> ... </wsdl:interface>
  <wsdl:binding> ... </wsdl:binding>
  <wsdl:service> ... </wsdl:service>
</wsdl:description>
```

- The `<wsdl:binding>` element defines **how** the messages should be transmitted, i.e., what transport protocol should be used to access a given interface. (E.g. SOAP, plain HTTP, ...)
- It is possible to give multiple bindings for a single interface. (E.g. support both SOAP and some other binding.)

Defining bindings

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsdl"
  targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
  xmlns:wsoap= "http://www.w3.org/2003/05/soap"
  xmlns:soap="http://www.w3.org/2003/05/soap/bindings/HTTP/"
  ...
  <types> ... </types>

  <interface name="reservationInterface"> ... </interface>

  <binding name="reservationSOAPBinding"
    interface="tns:reservationInterface"
    soap:binding="http://www.w3.org/2003/05/soap/bindings/HTTP/"
    <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

      <fault ref="tns:invalidDataFault"
        wsoap:code="soap:Sender"/>
    </binding>
  ...
</description>
```

Expose the tns:reservationInterface defined earlier over SOAP, using the SOAP-over-HTTP transport protocol

If a fault element is sent, specify that the SOAP error code should be set to "sender"

Specify that the opCheckAvailability in this interface should be called using HTTP GET

WSDL document structure

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl">
  <wsdl:types> ... </wsdl:types>
  <wsdl:interface> ... </wsdl:interface>
  <wsdl:binding> ... </wsdl:binding>
  <wsdl:service> ... </wsdl:service>
</wsdl:description>
```

- The `<wsdl:service>` element defines **where** the service can be accessed (its endpoint).
- Again, the same binding can be made accessible at multiple places (through multiple endpoint elements)

Defining services

```
<?xml version="1.0" encoding="utf-8" ?>
<description
  xmlns="http://www.w3.org/ns/wsdl"
  targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
  xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
...
<types> ... </types>

<interface name="reservationInterface" > ... </interface>

<binding name="reservationSOAPBinding"
  interface="tns:reser
...
</binding>

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

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

</service>
</description>
```

Expose the tns:reservationSOAPBinding binding at this URI

RESTful WSDL

- With WSDL 2.0 it is also possible to describe RESTful web services.

```
<?xml version="1.0" encoding="utf-8"
<description xmlns="http://www.w3.org/2003/XMLSchema"
xmlns:whttp="http://www.w3.org/2004/08/wsdl/http"
...

```

Support the operations of reservationInterface by the HTTP transport protocol, using GET as the method

```
<binding name="reservationHTTPBinding"
interface="tns:reservationInterface"
type="http://www.w3.org/ns/wsdl/http"
whttp:methodDefault="GET">

```

```
<operation ref="tns:opCheckAvailability"
whttp:location="{bookingDesiderata}" />
</binding>

```

The check availability operation can be called by appending the bookingDesiderata in application/x-www-form-urlencoded form to the endpoint URL

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

```

```
<!-- HTTP 1.1 GET End Point
<endpoint name="reservationEndpoint"
binding="tns:reservationHTTPBinding"
address="http://greath.example.com/2004/checkAvailability/" />
</service>

```

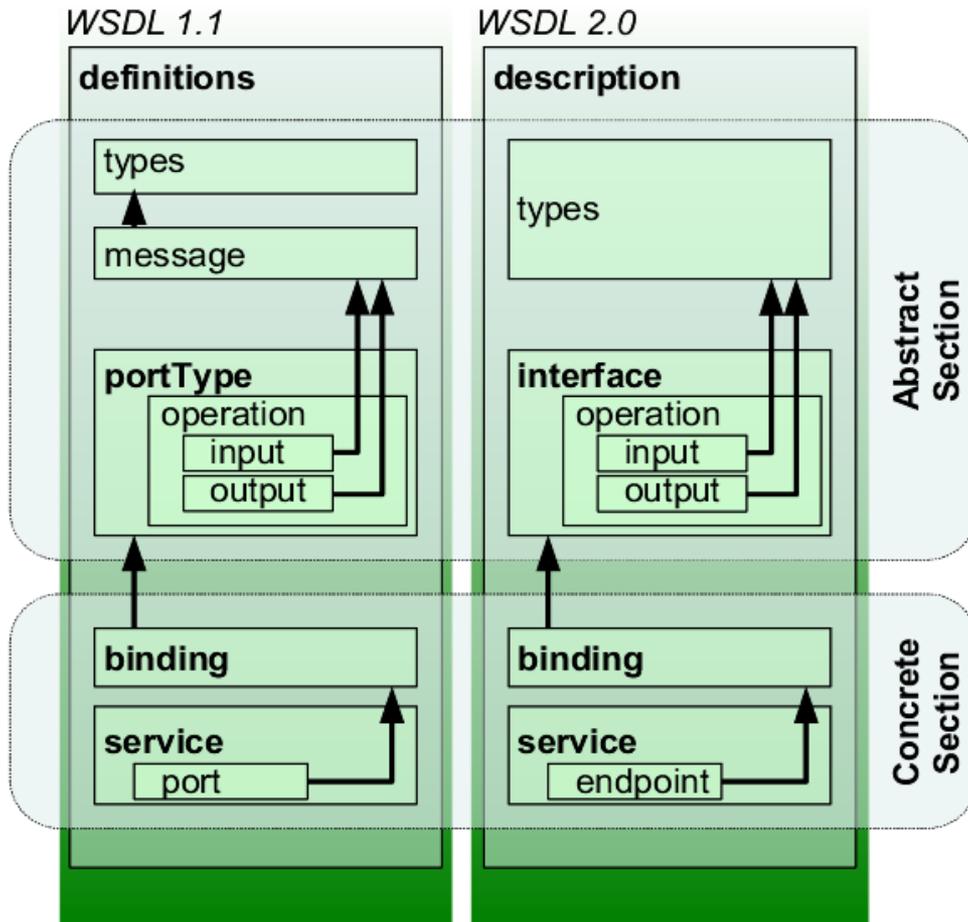
This gives, e.g.,
<http://greath.example.com/2004/checkAvailability/5-5-5?checkOutDate=6-6-5&roomType=foo>.

Phew ...

- That seems complicated and a lot of work. Why bother?
- Automated tooling support! (E.g. Java -> WSDL, WSDL -> Java)



WSDL 1.1 versus 2.0



- WSDL 1.1 is still used a lot in practice (and most Java tools do not support 2.0)
- WSDL 1.1 has an extra message element. And it uses portType instead of interface and port instead of endpoint.

Image source: wikipedia

http://en.wikipedia.org/wiki/File:WSDL_11vs20

References

- M. P. Papazoglou, Web Services: Principles and Technology 2nd edition, Prentice Hall
- World Wide Web Consortium, SOAP Version 1.2 Part 0: Primer (Second Edition) <http://www.w3.org/TR/soap12-part0/>
- World Wide Web Consortium, Web Services Description Language (WSDL) Version 2.0 Part 0: Primer <http://www.w3.org/TR/wsdl20-primer>