SOAP / WSDL

INTRODUCTION TO SOAP, WSDL AND UDDI, THE COMBO FOR BIG WEB SERVICES

PETER R. EGLI
peteregli.net

Contents

1. What is a web service?

W3C definition of web service (see W3C glossary under http://www.w3.org/TR/ws-gloss/):

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

The key elements are:

- → **Machine-to-machine communication.**
- → **Machine-processable interface description (WSDL).**
- → **Communication through messages (SOAP) using HTTP as transport protocol.**
- → **XML serialization.**

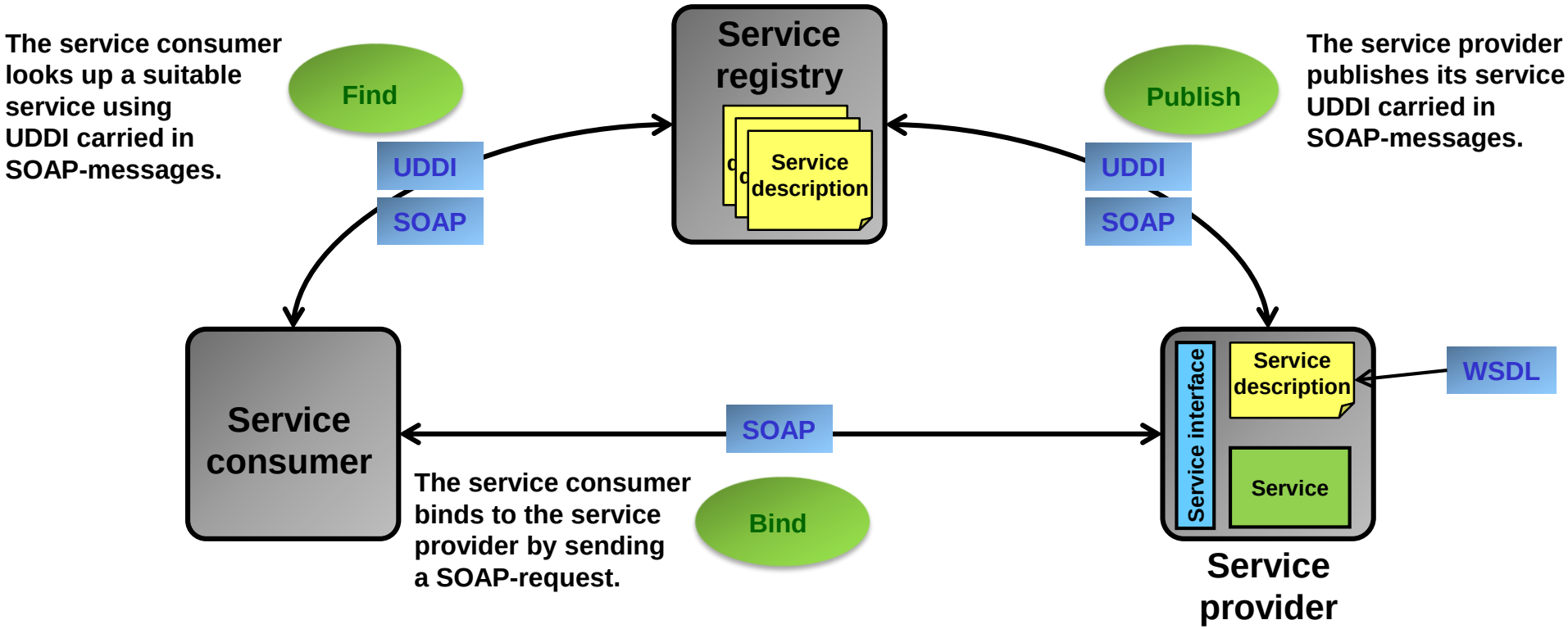
The word 'web' in the term 'web service':

Even though SOAP-WS (sometimes also called 'classical web' services or 'big web services') are transported in the web's core protocol HTTP, they make very little use of the features and functions of HTTP.
In fact, SOAP-WS messages may be carried in protocols other than HTTP.

2. Web service architecture

The combo SOAP+WSDL+UDDI defines a general model for a web service architecture.

SOAP:	**Simple Object Access Protocol**
WSDL:	**Web Service Description Language**
UDDI:	**Universal Description and Discovery Protocol**
Service consumer:	**User of a service**
Service provider:	**Entity that implements a service (=server)**
Service registry:	**Central place where available services are listed and advertised for lookup**



3. Web service versus conventional object middleware (e.g. CORBA)

Feature-wise comparison:

Aspect	CORBA	Web services (SOAP/WSDL/UDDI)
Data model	Object model	SOAP message exchange model
Client-Server coupling	Tight	Loose (decoupling through asynchronous messaging)
Location transparency	Object references	URL
Type system	IDL, static + runtime checks	XML schemas, runtime checks only
Error handling	IDL exception	SOAP fault messages
Serialization	IIOP / GIOP protocol implemented in ORB	User definable formats such as XML, JSON
Parameter passing	By reference, by value (value type)	By value (no notion of objects)
Transfer syntax	CDR used on the wire (binary format)	XML used on the wire (Unicode)
State	Stateful (server object represents state)	Stateless (no SOAP session, but sessions possible on application level)
Request semantics	At-most-once	No guarantees by SOAP (extensions through WS-ReliableMessaging)
Registry	Interface Repository, implementation repository	UDDI / WSDL
Service discovery	CORBA naming / trading service, RMI registry	UDDI
Language support	Any language with an IDL binding	Any language
Security	CORBA security service	HTTP / SSL, XML signature, additional WS-Security standards
Firewall Traversal	Difficult (CORBA uses arbitrary port numbers)	Uses HTTP port 80 (when using HTTP transport binding)
Events	CORBA event service	E.g. through message exchange patterns out-only, Further standards like WS-Eventing

Color coding:

Comparable / similar functionality Considerable conceptual differences

4. SOAP (1/12)

What is SOAP?

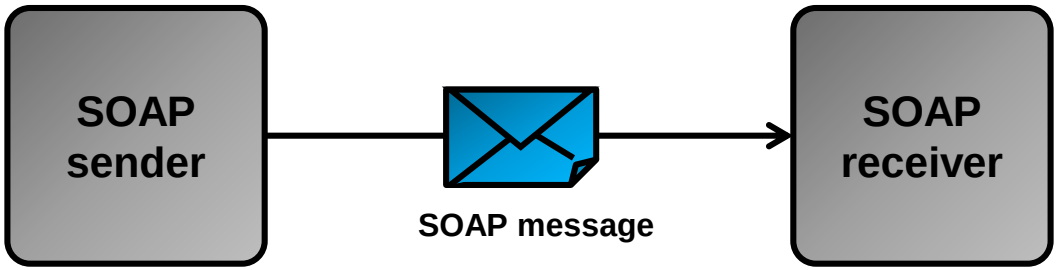
SOAP defines some key functions needed in a distributed computing environment, namely:

- **One-way stateless message exchange mechanism**
- **Message processing model (roles, must-understand, intermediary) for SOAP nodes**
- **Abstract, structured message definition able to run on different serializations**
- **Definition of bindings to transport protocols (HTTP for firewall traversal, SMTP)**
- **Extension mechanism through header elements enabling functionality defined in different XML namespaces such as WS-addressing**
- **Fault handling model**

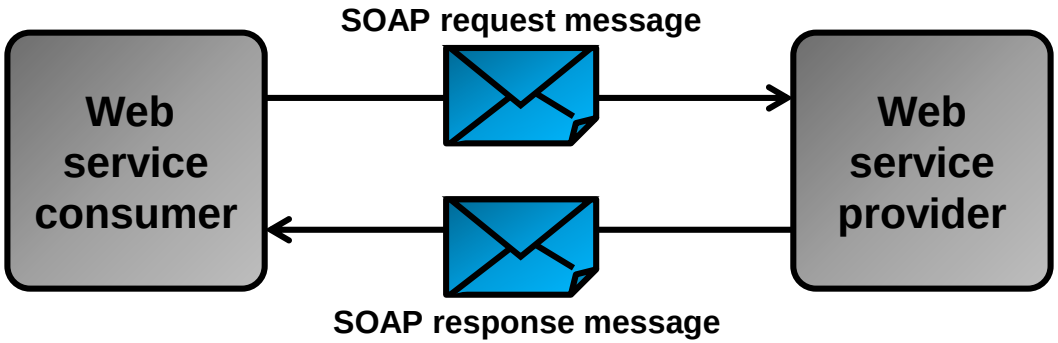
4. SOAP (2/12)

SOAP message exchange mechanism (1/5):

SOAP defines the message structure for the message exchange between a web service provider and consumer.
SOAP defines the roles SOAP sender and receiver. The basic message model of SOAP is one-way and stateless, i.e. a sender sends a message to a receiver without retaining state about the message exchange.



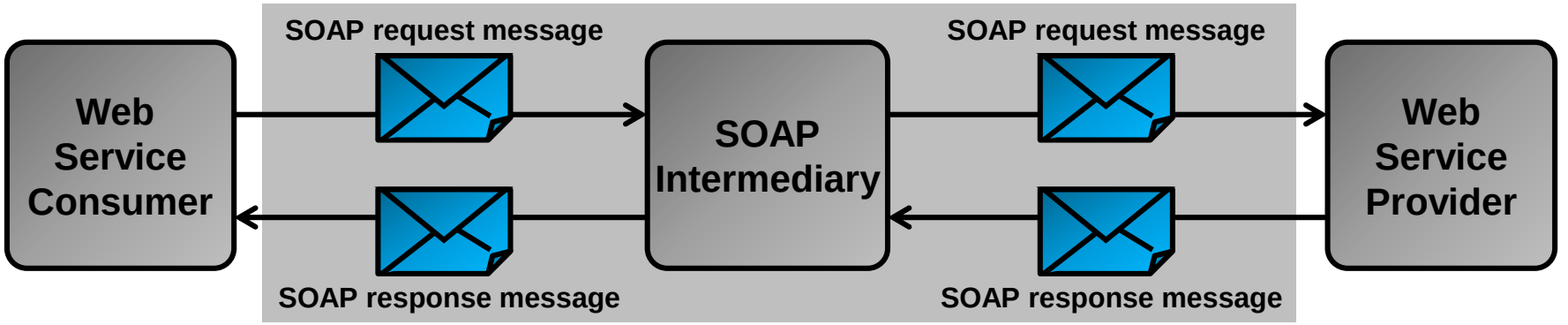
The most common message exchange pattern is request-response between a web service consumer and provider.
Web service provider and consumer are then both SOAP sender and receiver.



4. SOAP (3/12)

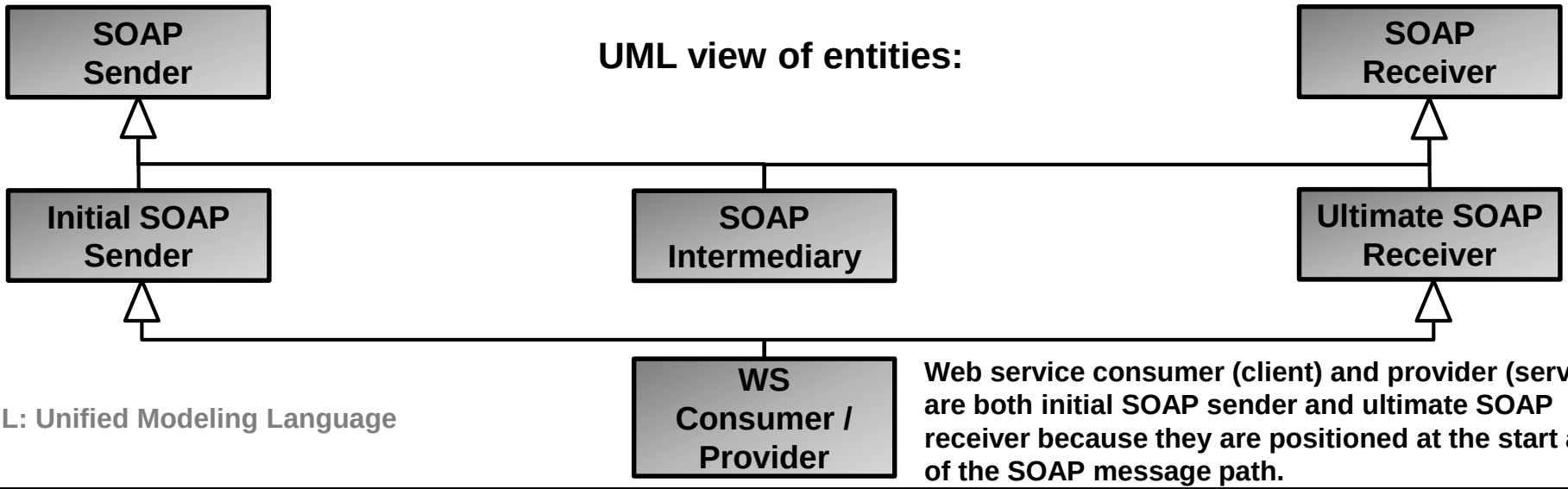
SOAP message exchange mechanism (2/5):

The SOAP model defines another entity called intermediary, a node that can be addressed to perform some particular function such as caching or message filtering.



SOAP message path

UML view of entities:



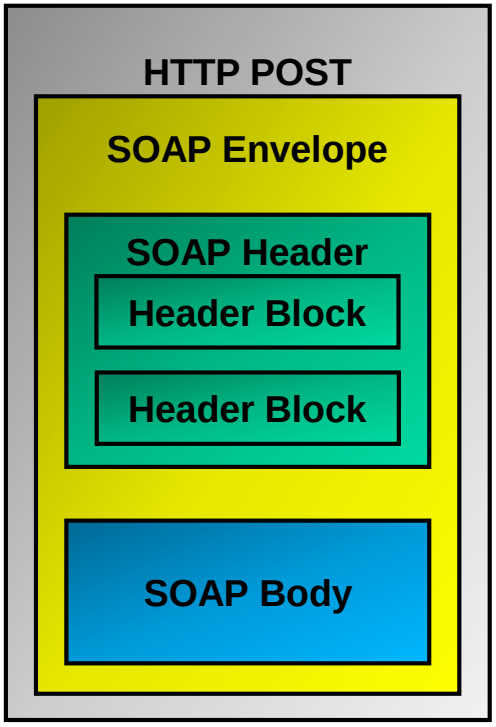
UML: Unified Modeling Language

Web service consumer (client) and provider (server) are both initial SOAP sender and ultimate SOAP receiver because they are positioned at the start and end of the SOAP message path.

4. SOAP (4/12)

SOAP message exchange mechanism (3/5):

SOAP defines a generic structure of an abstract message.
Elements of a SOAP message:



SOAP Envelope:
The envelope (=root element of SOAP message) is a container for the optional SOAP header and the mandatory SOAP body element.

SOAP Header:
The SOAP header may carry control information which is not application payload.
Such information is organized in header blocks, each with its individual XML namespace defining the schema.
The SOAP header is extensible, i.e. arbitrary namespaces with a particular way of message processing can be "woven" into the header.

SOAP Body:
The SOAP body carries the actual application information, encoded as an XML document. The schema of the body is defined by a WSDL document.

4. SOAP (5/12)

SOAP message exchange mechanism (4/5):

A few SOAP header attributes define the processing behavior on SOAP nodes:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true"
      env:relay="false"
      env:encodingStyle="http://indigoo.com/exampleEncoding">
    <m:reference>uuid:xxx</m:reference>
    <m:dateAndTime>2012</m:dateAndTime>
```

Attribute	Description	Possible values

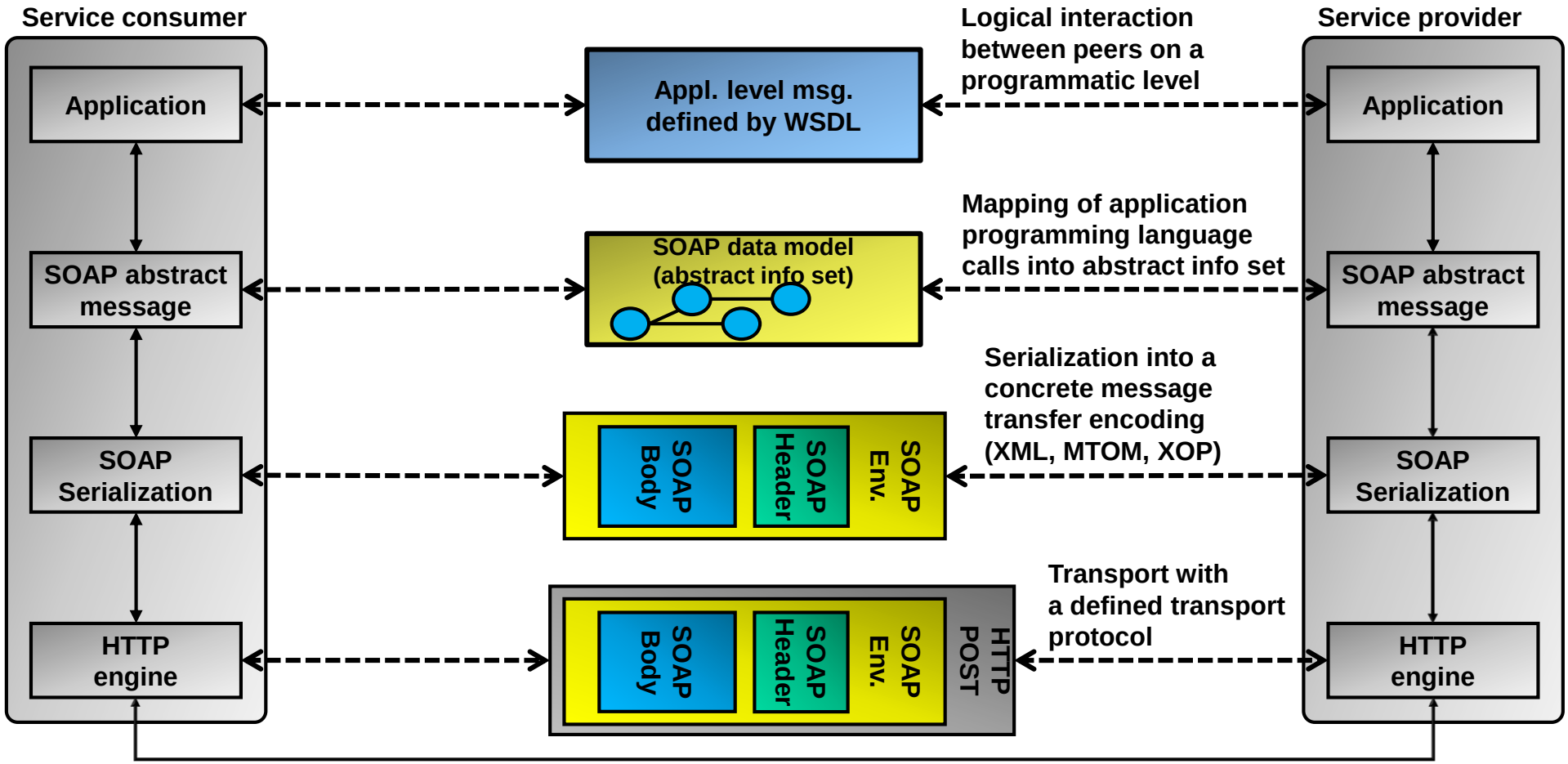
`env:role`	Defines which node along the message path is to process the header block.	`next`
		`none`
		`ultimateReceiver`

`env:mustUnderstand`	Tells if header block must be processed by the targeted SOAP node. The processing is header block specific (what the SOAP node is supposed to do depends on the semantics of the header block).	`true` `false`
`env:relay`	If set to `true` and `env:mustUnderstand` is absent or false, an intermediary SOAP node must forward (relay) the header block if it does not process it. Otherwise the header block is consumed by the SOAP node and not forwarded to the next SOAP node.	`true` `false`
`env:encodingStyle`	Defines the encoding or serialization scheme of the block where it is present. Default: `soap-encoding` (XML-encoding)	Any encoding scheme defined by a namespace URI.

4. SOAP (6/12)

SOAP message exchange mechanism (5/5):

SOAP provides the mapping between application messages defined by a WSDL schema and physical messages transported over the network.



4. SOAP (7/12)

SOAP & RPC (1/3):

Initially SOAP was modelled as an XML-based variant of RPC (Remote Procedure Call).
The basic pattern of RPC is request-response.

SOAP now is a generalized message exchange mechanism not mandating a specific message exchange pattern.
Multiple patterns are possible (in-out = request-response, in-only, out-in etc.).

SOAP V1.2 defines some conventions for modelling the programmatic concept of RPC with SOAP messages as follows:

Transport binding:
If the SOAP transport binding is HTTP, SOAP RPC maps to HTTP request and response where the HTTP URI is the address of a SOAP processor.

Message exchange pattern:
SOAP-RPC uses the SOAP-Response message exchange pattern (MEP).
The SOAP specification recommends to map the resource address to the request URI.

4. SOAP (8/12)

SOAP & RPC (2/3):

Identification of resource:

SOAP-RPC recommends to identify the resource by both an URI and the name of an operation plus arguments.

Example:

URI = `http://services.indigoo.com/weatherUpdate`

```
getWeatherUpdate(lat="123", long="456")
```

➔ The SOAP RPC recommendation proposes to duplicate the resouce identification (URI + name of operation plus arguments) when using HTTP transport binding for the sake of compatibility with WWW principles (every resource has its own individual address).

➔ Here, common programming language constructs are in conflict with web paradigms (every resource has its own individual address).

➔ REST is much clearer in this respect and cleanly separates resource identification (URI) from operation to be performed on the resource (HTTP method).

4. SOAP (9/12)

SOAP & RPC (3/3):

Encoding of request:

The request is modeled as a struct, e.g. an XML fragment whose outermost node is the name of the RPC method or operation and the contained nodes are arguments as exemplified below:

```
<w:getWeatherUpdate xmlns:m=http://examples.indigoo.com/weatherService
 env:encodingStyle=http://www.w3.org/2003/05/soap-encoding
  xmlns:w="http://weatherService.indigoo.com/">
  <w:location>
    <w:latitude>47.359169</w:latitude>
    <w:longitude>8.563843</w:longitude>
  </w:location>
</s:getWeatherUpdate>
```

Encoding of response:

The response data is encoded as an XML struct again.
The return value that is distinguished from other output parameters may be enclosed in a <result> element:

```
<w:getWeatherResponse
   env:encodingStyle=http://www.w3.org/2003/05/soap-encoding>
  <rpc:result>w:temperature</rpc:result>
  <w:temperature>12.5°C</w:temperature>
</w:getWeatherResponse>
```

4. SOAP (10/12)

SOAP transport binding (1/2):

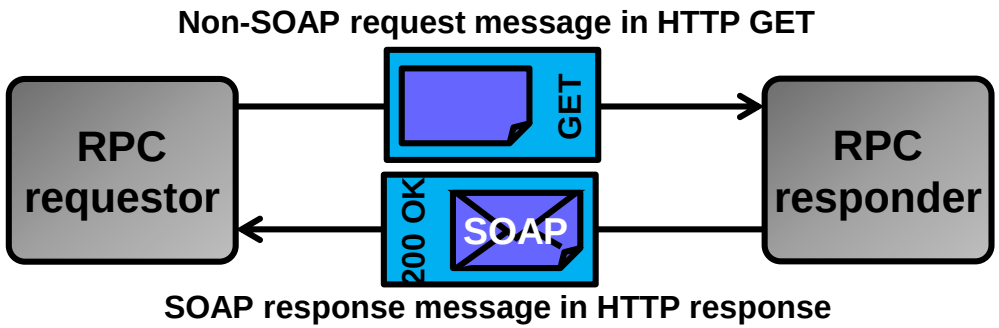
The transport binding defines the transport protocol to be used.
The only standardized binding is HTTP. However, other transports such as SMTP (email) are possible by virtue of the extensible architecture of SOAP.

The HTTP transport binding of SOAP defines 2 possible message exhange patterns (MEP).

a. Simple and safe information retrieval by RPC call:

If a SOAP node simply retrieves information from another SOAP node without altering data on the queried node (safe method as per HTTP RFC2616 – no side-effects), SOAP recommends the SOAP-response message exchange pattern.

- **→ Web method = HTTP GET**
- **→ SOAP-Response MEP (non-SOAP request, SOAP response)**
- **→ No header blocks**



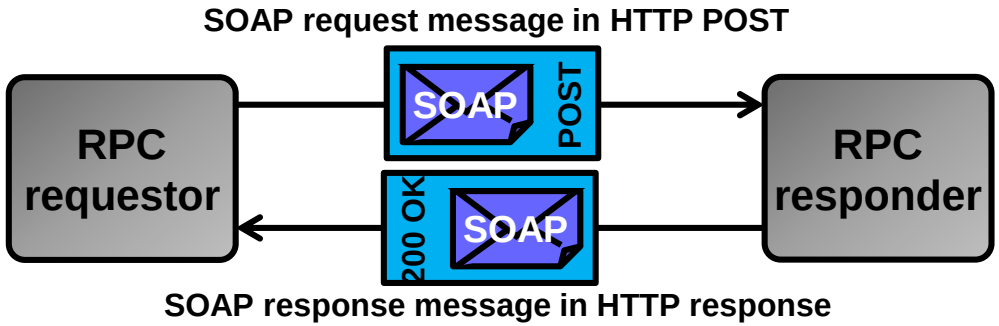
4. SOAP (11/12)

SOAP transport binding (2/2):

b. Non-safe RPC call (modification of data on responder/server):

Here, data is updated on the queried node with data carried in a SOAP-request message.

- → **Web method = HTTP POST (addressing the SOAP processor)**
- → **SOAP Request-Response-MEP (both request and response are SOAP messages)**



4. SOAP (12/12)

SOAP fault element:

SOAP uses the XML fault element as part of the SOAP body to indicate SOAP errors.
The fault element is used to indicate SOAP-level errors (wrong formatting) or application level errors (e.g. inexistent RPC method called).

Example SOAP message with fault element (invalid method called):

```
<?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 (getWeatherUpdte) in class
          (WeatherUpdate) at /usr/local/weatherUpdate.py line 143.
        </faultstring>
      </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5. WSDL 2.0 (1/12)

A WSDL (Web Service Description Language) document has basically 3 purposes with regard to a web service:

1. Describe the "What"

→ XML-based abstract definition of a web service comprising:

a. Type system used to describe the service meta model

b. Messages / data types involved in the interaction with the web service

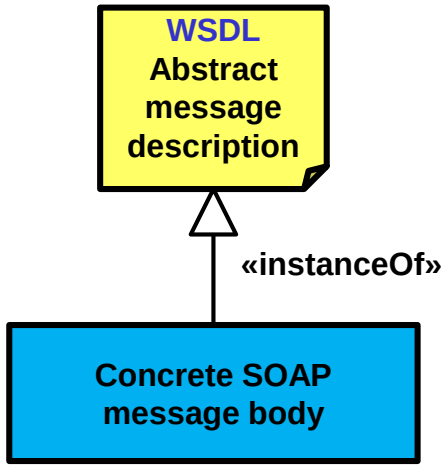
c. Message exchange pattern(s) used in the interaction with the web service

2. Describe the "How"

→ Define „how“ to access the abstract web service through a transport binding

3. Describe the "Where"

→ Definition of location(s) where the abstract web service can be accessed.

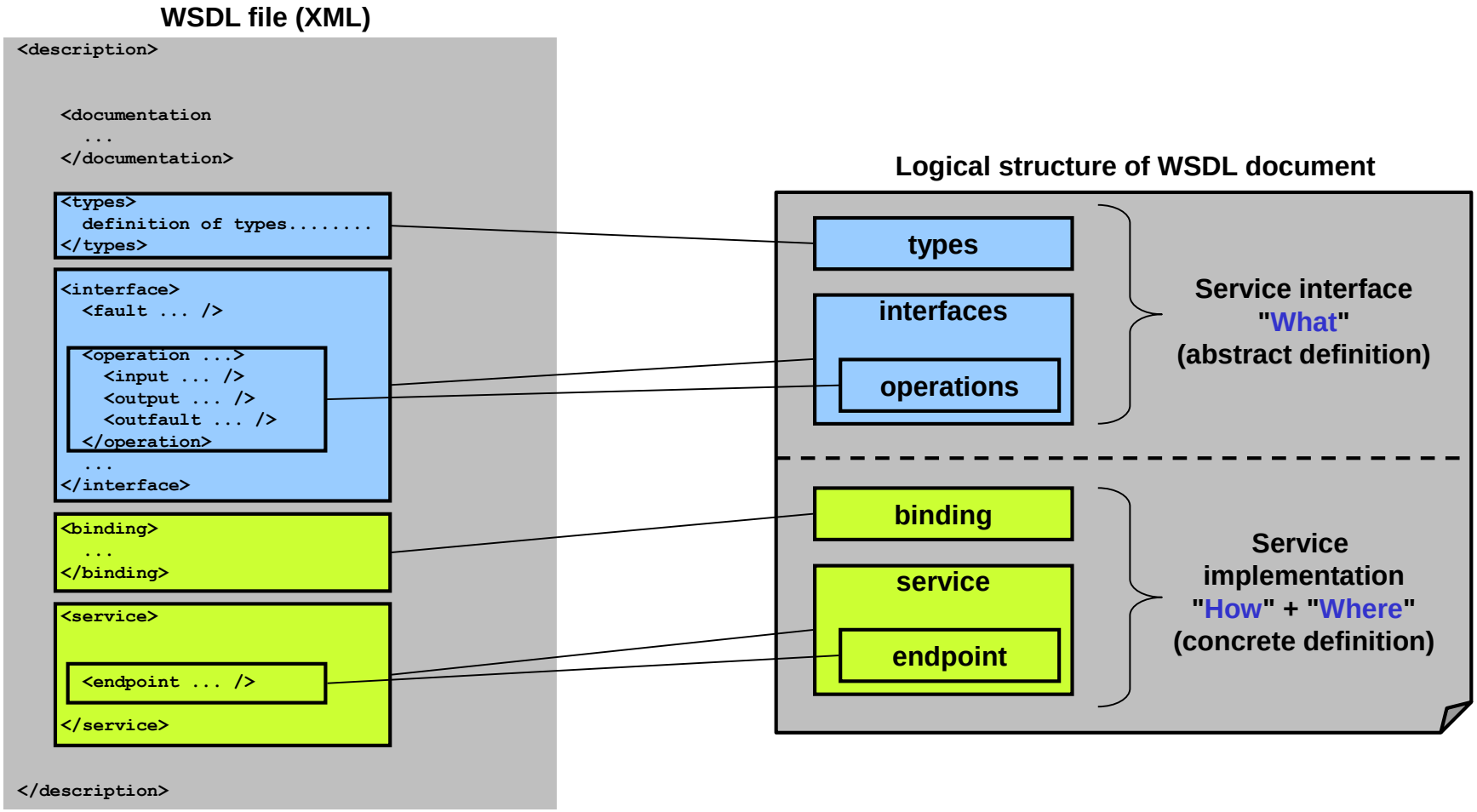


WSDL defines the schema while a SOAP message is an instance of the WSDL schema.

5. WSDL 2.0 (2/12)

Structure / elements of a WSDL document:

A WSDL 2.0 document is partitioned into an abstract / logical interface description and a concrete interface implementation part.



5. WSDL 2.0 (3/12)

Elements of WSDL 2.0 (1/6):

Description:

`Description` is the top-level element and contains all the other elements plus namespace declarations.

The `targetNamespace` attribute denotes the namespace of the web service defined in this WSDL file.

Example:

```
<?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"
    . . . >
  . . .
</description>
```

5. WSDL 2.0 (4/12)

Elements of WSDL 2.0 (2/6):

Types:

The `types` element contains user defined data types, e.g. complex data structures. WSDL uses *XML Schema* to define types (XSD syntax).

Type elements are basically „messages“ that are sent between service client and server (in WSDL 1.1 they are called messages).

Example:

```
<types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://greath.example.com/2004/schemas/resSvc"
        xmlns="http://greath.example.com/2004/schemas/resSvc">
      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate" type="xs:date"/>
          <xs:element  name="checkOutDate" type="xs:date"/>
          <xs:element  name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      ...
```

5. WSDL 2.0 (5/12)

Elements of WSDL 2.0 (3/6):

Interface:

An `interface` is an abstract definition of the web service without details where and how the service is provided.
An interface describes the allowed operations of the web service and possible errors (faults).
An abstract MEP (Message Exchange Pattern) defines the abstract message exchange flow without specifying any realization details such as protocols to use.

Operation:

An `operation` defines input and output data structures (formerly called message in WSDL 1.x) that constitute a basic service operation (e.g. add an address entry operation to an address book).

Example:

```
<interface name="reservationInterface" >
   <fault name="invalidDataFault" element="igns:invalidDataError"/>
   <operation name="opCheckAvailability"
            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="igns:checkAvailability"/>
      <output messageLabel="Out" element="igns:checkAvailabilityResponse"/>
      <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
   </operation>
 </interface>
```

Operation name

Abstract MEP

Input and output arguments

5. WSDL 2.0 (6/12)

Elements of WSDL 2.0 (4/6):

Fault:

The fault element as part of an (abstract) interface defines possible errors that may be returned to the client.

Example:

```
<fault name = "invalidDataFault" element = "igns:invalidDataError"/>
...
<operation ...>
  ...
  <output ...>
  <outfault ref="invalidDataFault" messageLabel="Out"/>
  ...
```

Definition of fault

refers to

Declaration of possible fault of the operation

Input / Output:

`Input` **and** `output` **define input and output messages, respectively.**

In the simple case of an in-out message exchange pattern (see MEP), the messageLabel attribute has the values „In“ and „Out“, respectively. In more complicated MEPs with multiple input and output messages, additional attributes allow defining the proper sequence of the messages.

5. WSDL 2.0 (7/12)

Elements of WSDL 2.0 (5/6):

Binding:

For an abstract service interface, the `binding` defines the concrete message format and transport protocol (how the service can be accessed).
The binding must be defined for every operation, thus every operation defined as part of an interface is referenced in the binding element.
Example:

```
<binding name="reservationSOAPBinding"
         interface="tns:reservationInterface" type="http://www.w3.org/ns/wsdl/soap"
          wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
  <operation ref="igns:opCheckAvailability"
    wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
  <fault ref="tns:invalidDataFault" wsoap:code="soap:Sender"/>
</binding>
```

SOAP over HTTP binding

Concrete MEP to be used

N.B.: The concrete MEP …`/mep/soap-response` is concrete from the WSDL point of view because abstract messages defined by WSDL are mapped to a concrete SOAP MEP.
From the SOAP point of view, however, the MEP is abstract and in turn is mapped to a concrete transport protocol, most probably the HTTP binding (see above).

5. WSDL 2.0 (8/12)

Elements of WSDL 2.0 (6/6):

Service:

The `service` element defines where a service can be accessed.

A service element contains a single interface attribute that defines which abstract interface the service implements. Furthermore, a service contains 1 or multiple endpoint elements that define where this interface can be accessed. N.B.: The „how“ is defined in the binding.

Example:

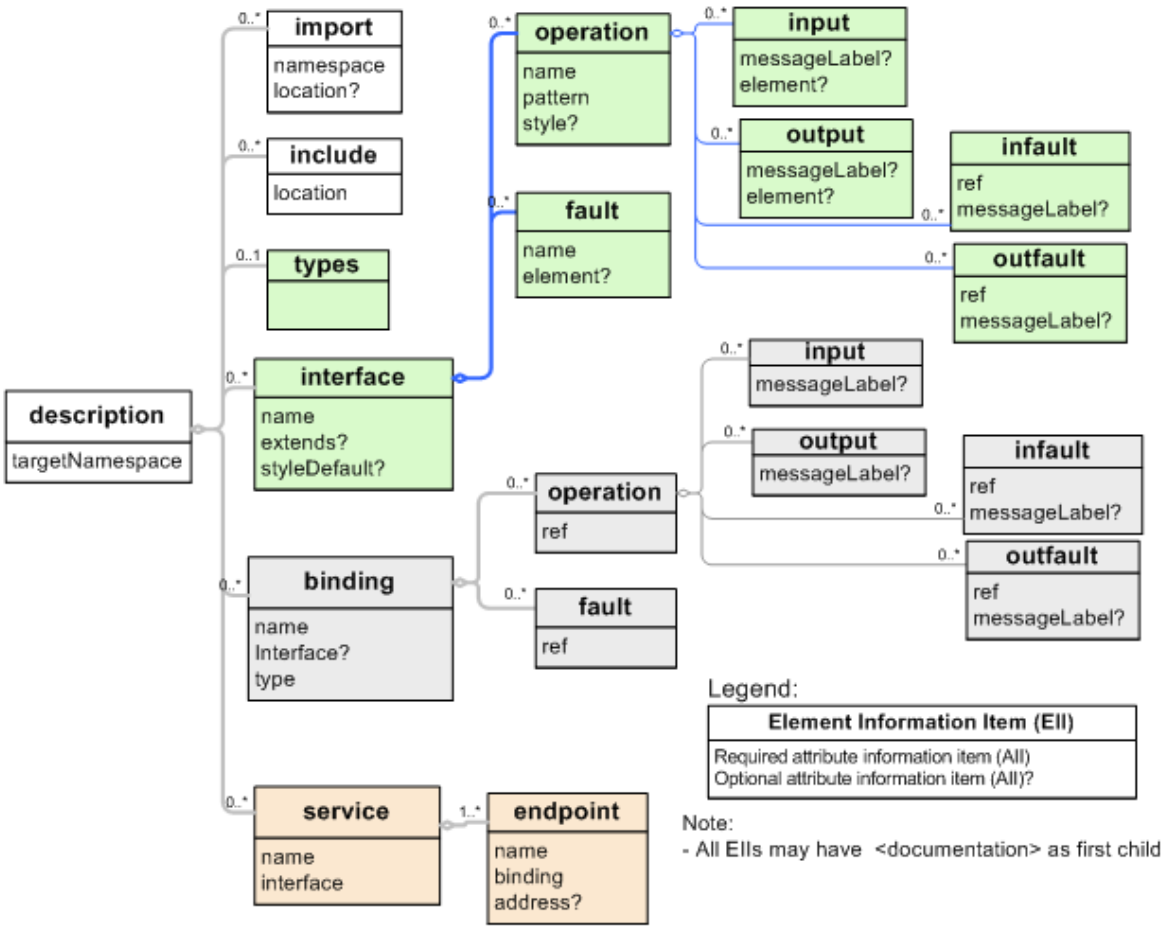
```
<service name="reservationService" interface="tns:reservationInterface">
  <endpoint name="reservationEndpoint"
            binding="tns:reservationSOAPBinding"
            address ="http://greath.example.com/2004/reservation"/>
</service>
```

Endpoint:

An `endpoint` defines a „Service Access Point“ under which the web service can be accessed. It defines a name, a binding and an address (=URL).

5. WSDL 2.0 (9/12)

The WSDL 2.0 infoset defines the elements and their relationship including cardinality:



„What“

„How“

„Where“

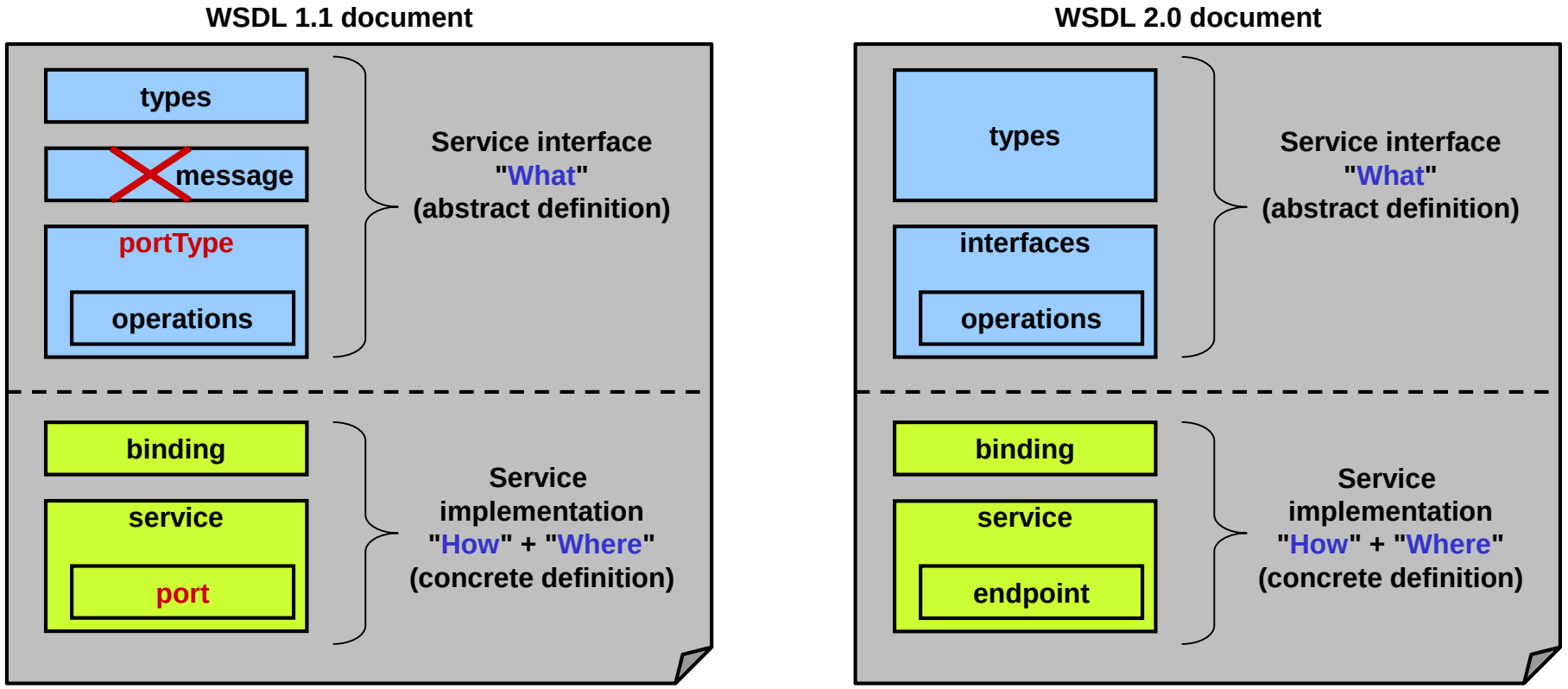
Source: http://www.w3.org/TR/wsdl20-primer/#WSDL-PART2

5. WSDL 2.0 (10/12)

WSDL 1.1 versus 2.0:

WSDL 2.0 integrates the `message` element in the `types` element and renames `portType` to `interface` and `port` to `endpoint`, respectively.

Additionally, the top-level element was changed from `definitions` (WSDL 1.1) to `description` (WSDL 2.0).



5. WSDL 2.0 (11/12)

WSDL message exchange patterns (MEP):

MEPs define how and in which cardinality messages are exchanged.

WSDL 2.0 defines 8 MEPs that cover the most common use cases including Server→Client, but additional MEPs may be defined and used by a web server where the need arises.

In-only:
The consumer only sends an outbound message, but does not receive a response message.

Robust In-Only:
Same as In-Only, but service may trigger a fault and thus send back a message with a fault.

In-Out:
This is equivalent to *request-response*. A standard two-way message exchange where the consumer sends a message, the provider responds with a message or fault and the consumer responds with a status.

In Optional-Out:
A standard two-way message exchange where the provider's response is optional.

Out-Only:
The service operation produces only an outbound message and cannot trigger a fault.

Robust Out-Only:
Similar to Out-Only, but the service may trigger a fault.

Out-In:
The service produces an outbound message first which is followed by an inbound message.

Out-Optional-In:
Same as Out-In, but inbound message is optional.

5. WSDL 2.0 (12/12)

Comparison of WSDL with conventional middleware IDL file:
WSDL is similar to an IDL file (e.g. CORBA) in that it describes the operations and parameters with data types that are part of the interface.

There are, however, some notable differences between WSDL and IDL:

1. IDL files do not specify location of the service
The location (e.g. URL) must be hardcoded in the client or passed through some other means to the client (e.g. command line arguments).

2. IDL files have a fixed binding to a transport protocol
Usually an IDL is bound to a specific transport protocol, namely TCP.

3. IDLs do not specify sequences of method calls
IDLs usually only define interface and operations (=methods in IDL-speak) that can be called on these interfaces or classes. Sets of operations cannot be specified (IDL file only specifies individual operations, but does not allow to specifiy sequences of operations).
WSDL defines request – response pairs of messages.

6. UDDI (1/2)

The idea of UDDI:

UDDI was conceived as a universal business registry similar to search engines (Google et. al.) where services can be located based on different criteria.
Servers that provide public UDDI registry and lookup service are called nodes.
An UDDI business registration provides 3 distinct sets of information:

White Pages	Address, contact, and known identifiers.
Yellow Pages	Industrial categorizations based on standard taxonomies.
Green Pages	Technical information about services exposed by the business.

The vision was that service consumers would be linked to service providers through a public brokerage system.

The reality of UDDI:

UDDI did not gain widespread use as yet even though it had the backing of large companies like IBM and Microsoft.
UDDI is mostly used in limited environments (inside companies). For that purpose, UDDI is too complicated and most of the data provided by UDDI is not needed.
Microsoft, IBM and SAP shut down their public UDDI nodes (servers) in 2006.

6. UDDI (2/2)

UDDI interfaces:

UDDI defines a set of interfaces for accessing the UDDI registry.

UDDI Inquiry	**→ Lookup services**
UDDI Publication	**→ Publish and modify published services**
UDDI Security	**→ Define access control of published services**
UDDI Custody Transfer	**→ Change the ownership of information in the registry and move a publication to a different node**
UDDI Subscription	**→ Subscribe to changes of information in the UDDI registry**
UDDI Replication	**→ Functions for replicating registry entries between UDDI nodes**