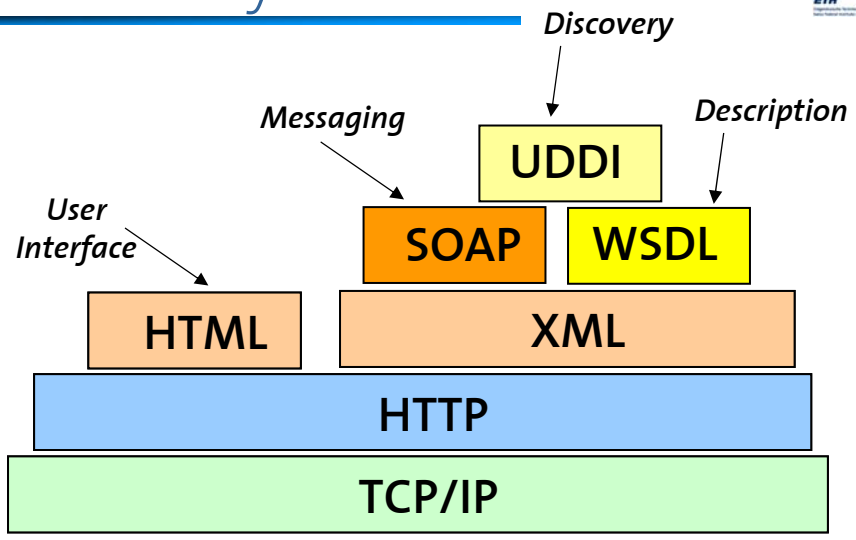
IKS Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Distributed Systems
SOAP, WSDL

Dr. Cesare Pautasso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
pautasso@inf.ethz.ch
http://www.inf.ethz.ch/~pautasso

Standard Layers



The Web services stack

	WSDL-based		Semantic Web	ebXML
Messaging	SOAP			ebXML MSS
Description	WSDL		RDF	ebXML CPP
Nonfunctional description	WSEL		DAML-S	
Conversations	WSCL	WSCI		ebXML BPSS
Choreography	WS-Coordination			
Business processes	BPEL4WS WSFL/XLANG	BPML		BPML
Contracts				ebXML CPA
Discovery	UDDI			ebXML registries
Transactions	WS-Transactions	BTP		BTP
Security	WS-Security			SAML S/MIME

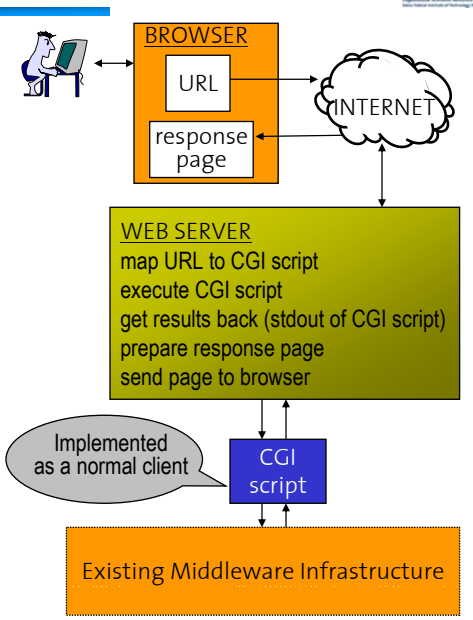
IKS Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

HTTP, HTML and XML

WWW basics

- The earliest implementations were very simple and built directly upon the existing systems (client/server based on RPC, TP-Monitors, or any other form of middleware which allowed interaction through a programmable client)
  - the CGI script (or program) acted as client in the traditional sense (for instance using RPC)
  - The user clicked in a given URL and the server invoked the script corresponding to that URL
  - the script executed, produced the results and passed them back to the server (usually as the address of a web page)
  - the server retrieved the page and send it to the browser



HTTP as a communication protocol

- HTTP was designed for exchanging documents. It is almost like e-mail (in fact, it uses RFC 822 compliant mail headers and MIME types):
- Example of a simplified request (from browser to Web server):

```
GET  /docu2.html HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/gif
User-Agent:  Lynx/2.2  libwww/2.14
    * a blank line *
```

File being requested (docu2.html) and version of the protocol

List of MIME types accepted by the browser

Information about the environment where the browser is running

End of request

- If the "GET" looks familiar, it is not a coincidence. The document transfer protocol used is very similar to ftp

HTTP server side response

- Example of a response from the server (to the request by the browser):

```
HTTP/1.0 200 OK
Date: Wednesday, 02-Feb-94
   23:04:12 GMT
Server: NCSA/1.1
MIME-version: 1.0
Last-modified: Monday, 15-
   Nov-93 23:33:16 GMT
Content-type: text/html
Content-length: 2345
     * a blank line *
<HTML><HEAD><TITLE> . . .
   </TITLE> . . .etc.
</HTML>
```

Protocol version, code indicating request status (200=ok)

Date, server identification (type) and format used in the request

MIME type of the document being sent

Header for the document (document length in bytes)

Document content

- Server is expected to convert the data into a MIME type specified in the request ("Accept:" headers)

Parameter passing

- The introduction of forms for allowing users to provide information to a web server required to modify HTML (and HTTP) but it provided a more advanced interface than just retrieving files:

```
POST /cgi-bin/post-query HTTP/1.0
Accept: www/source
Accept: text/html
Accept: video/mpeg
Accept: image/jpeg
...
Accept: application/postscript
User-Agent:  Lynx/2.2  libwww/2.14
Content-type: application/x-www-
   form-urlencoded
Content-length: 150
     * a blank line *
&name = Cesare+Pautasso
&email= pautasso@inf.ethz.ch
...
```

POST request indicating the CGI script to execute (post-query) GET can be used but requires the parameters to be sent as part of the URL: /cgi-bin/post-query?name=...&email=...

As before

Data provided through the form and sent back to the server

Contents and presentation

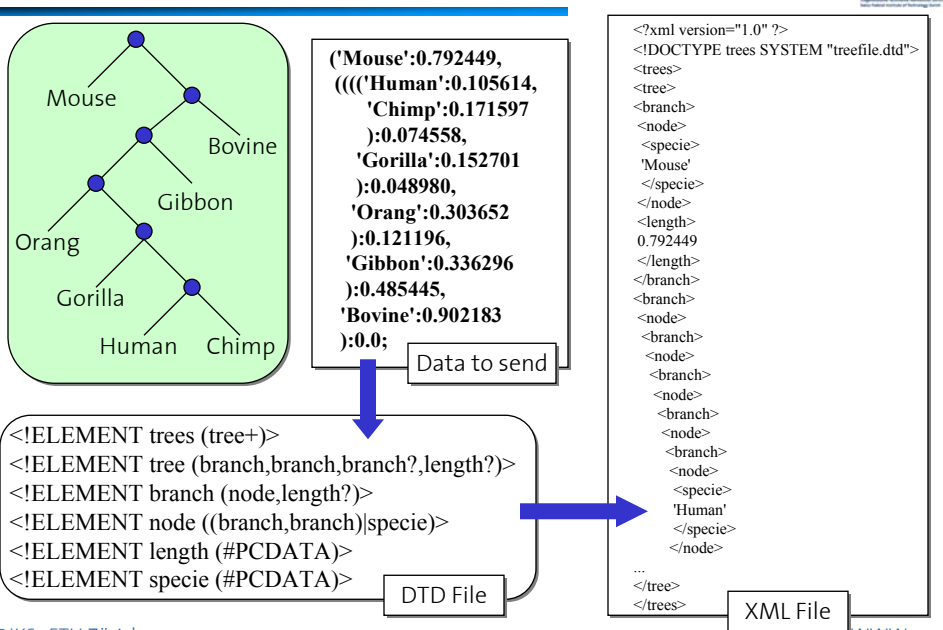
- HTML is a markup language designed to describe how a document should be displayed (the visual format of the document).
- HTML is one of the many markup languages that exist, some of them having being in use before HTML even existed
- Markup languages have been developed and are used in many industries (aircraft manufacturing, semiconductors, computer manuals). Markup languages provide a standardized grammar defining the meaning of **tags** and their use
- Markup languages use SGML, an international text processing standard from the 80's, to define tag sets and grammars
- HTML is based on SGML, that is, the tags and the grammar used in HTML documents have been defined using SGML.

```
<h2>Table of contents</h2><a name=TOC></a>
<ul>
<li><a href="SG.htm">1 A Gentle Introduction
    to SGML</a></li>
<li><a href="SG11.htm">2 What's Special
    about SGML?  </a></li>
<ul>
<li><a href="SG11.htm#SG111">2.1 Descriptive
    Markup</a></li>
<li><a href="SG11.htm#SG112">2.2 Types of
    Document</a></li>
<li><a href="SG11.htm#SG113">2.3 Data
    Independence  </a></li>
</ul>
<li><a href="SG12.htm">3 Textual
    Structure</a></li>
<li><a href="SG13.htm">4 SGML
    Structures</a></li>
<ul>
<li><a href="SG13.htm#SG131">4.1
    Elements</a></li>
<li><a href="SG13.htm#SG132">4.2 Content
    Models:  An Example</a></li>
</ul>
```


HTML and XML

- HTML only provides primitives for formatting a document with a human user in mind
- Using HTML there is no way to indicate what are the contents of a document (its semantics)
- For instance, a query to Amazon.com returns a book and its price as an HTML document
  - a human has no problem interpreting this information once the browser displays it
  - to parse the document to automatically identify the price of the book is much more complicated and an ad-hoc procedure (different for every bookstore)
- B2B applications require documents that are much more structured so that they can be easily parsed and the information they contain extracted
- To cope with this requirement, the XML standard was proposed
- Important aspects of XML:
  - XML is not an extension to HTML
  - XML is a simplified version of SGML that can be implemented in a Web browser
  - XML is not a language but a "meta-language" used to define markup languages
  - XML tags have no standard meaning that can be interpreted by the browser. The meaning must be supplied as an addition in the form of a style sheet or program

Data structures in XML



DTDs and documents

- The goal of XML is to provide a standardized way to specify data structures so that when data is exchanged, it is possible to understand what has been sent
- The Document Type Definition (DTD) specifies how the data structure is described: processing instructions, declarations, comments, and elements
- Using the DTD, the XML document can be correctly interpreted by a program by simply parsing the document using the grammar provided by the DTD
- The idea is similar to IDL except that instead of defining parameters as combinations of standard types, a DTD describes arbitrary documents as semi-structured data
- Using XML is possible to exchange data through HTTP and Web servers and process the data automatically
- Note that the use of XML reduces the universality of the browser since now a browser needs additional programs to deal with specific markup languages developed using XML (somewhat similar to plug-ins but more encompassing in terms of functionality)
- However, this is not much of a problem since the browser is for humans while XML is for automated processing
- XML can be used as the intermediate language for marshalling/serializing arguments when invoking services across the Internet

XML Schema

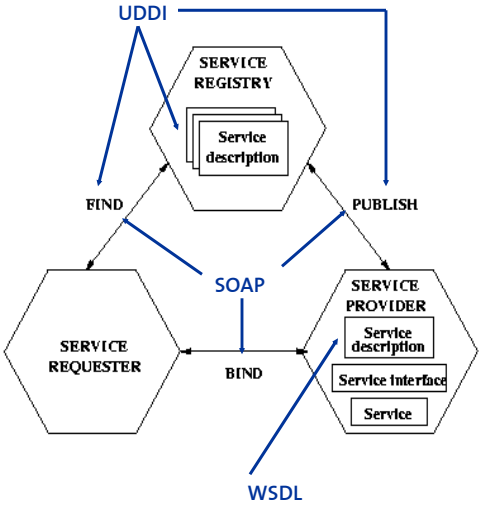
- A different problem related to accessing EAI systems through a web interface is the representation of relational data.
- If HTML is used, the data is formatted for presentation, not for processing
- If XML and DTDs are used, then the structured is better suited for processing but the processing is ad-hoc (one can define any DTD one wants)
- XML Schema has been proposed to allow database like query processing over XML documents.
- XML Schema is a data definition language for XML documents that allows to treat them as relational data in a standardized manner
- What is different between XML Schema and DTDs?. XML Schema:
  - uses the same syntax as XML (DTDs have a different syntax)
  - provides a wider set of types (similar to those in SQL)
  - allows to define complex types from the basic types
  - supports key and referential integrity constraints
  - can be used by query languages (XQuery, for instance) to parse XML documents and treat them as relational data
  - Can be used to specify the data model used by a Web service interface

IKS Information and Communication Systems Research Group

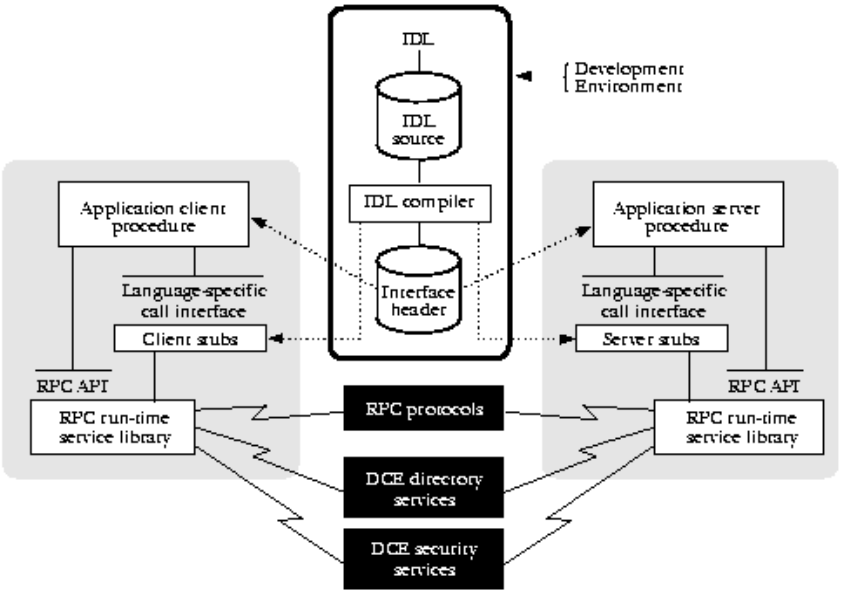
RPC across the Web

Web Services and Client/Server

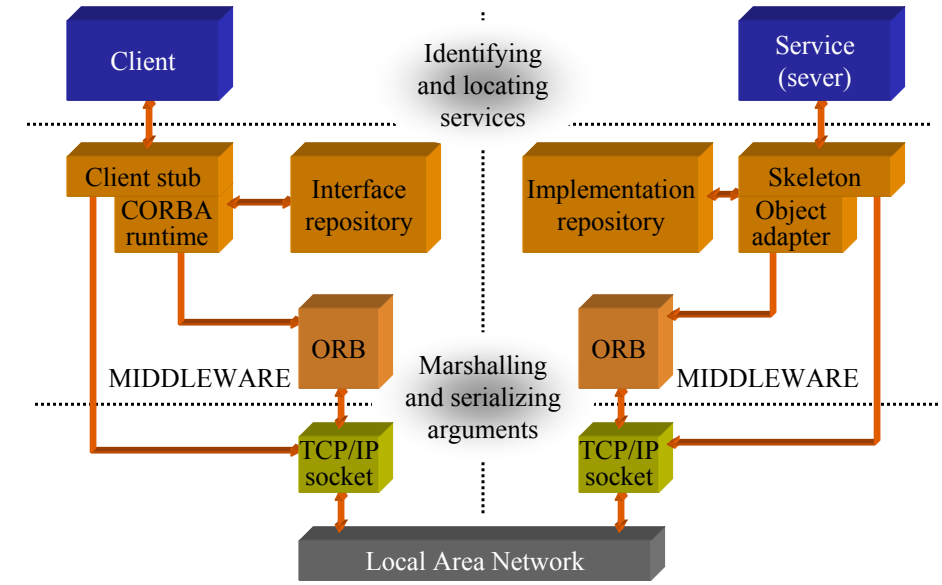
- The Web service architecture proposed by IBM is based on two key concepts:
  - architecture of existing synchronous middleware platforms
  - current specifications of SOAP, UDDI and WSDL
- The architecture has a remarkable client/server flavor
- It reflects only what can be done with
  - SOAP (Simple Object Access Protocol)
  - UDDI (Universal Description and Discovery Protocol)
  - WSDL (Web Services Description Language)



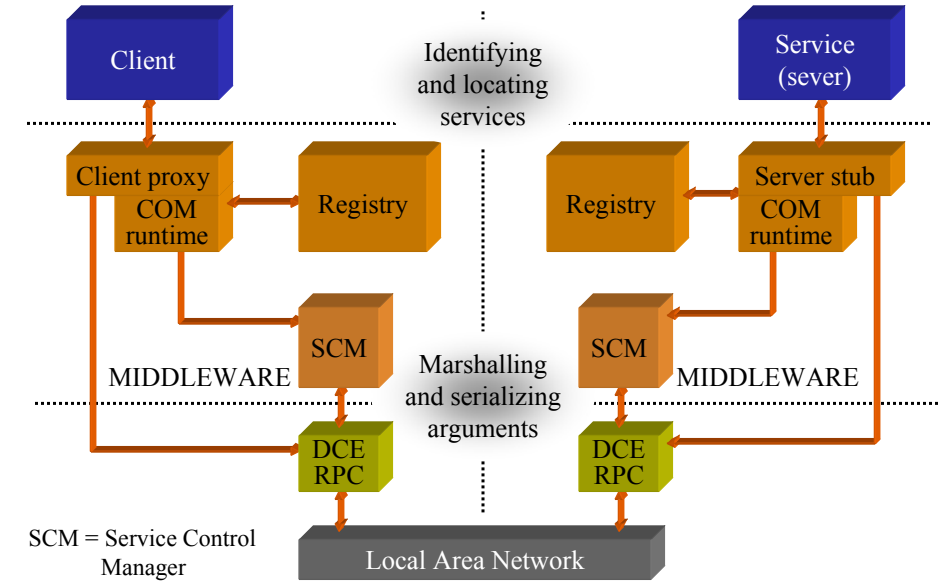
Remote calls in RPC/DCE



Remote calls in CORBA

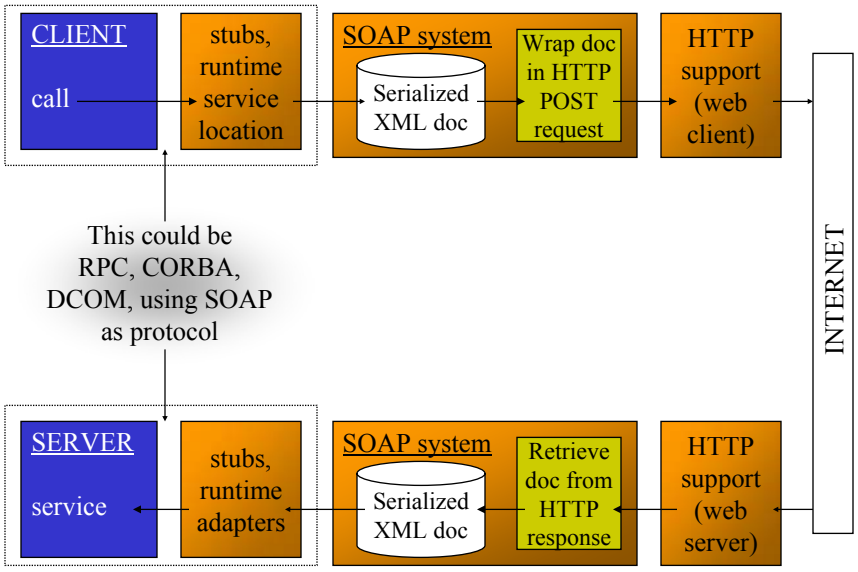


Remote calls in DCOM



Simple Object Access Protocol (SOAP)

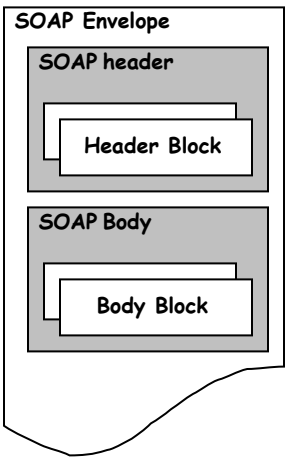
SOAP as RPC mechanism



Wire-protocols, XML and SOAP

- RPC, CORBA, DCOM, even Java, use different mechanisms and protocols for communicating. All of them map to TCP or UDP one way or another but use different syntax for marshalling, serializing and packaging messages
- The problem is that these mechanisms are a legacy from the time when communications were mostly within LANs and within homogeneous systems
- Building a B2B environment combining the systems of different companies becomes difficult because the protocols available in RPC, CORBA, or DCOM are too low level and certainly not compatible among each other (gateways are needed, etc.)
- To address this problem, XML was used to define SOAP (Simple Object Access Protocol)
- SOAP is conceptually quite simple:
  - RPC using HTTP
  - (at the client) turn an RPC call into an XML document
  - (at the server) turn the XML document into a procedure call
  - (at the server) turn the procedure's response into an XML document
  - (at the client) turn the XML document into the response to the RPC
  - use XML to serialize the arguments following the SOAP specification

SOAP as an RPC wrapper



- SOAP has been conceived as a way of wrapping different protocols into an XML document sent using HTTP (or other mechanisms)
- The structure of a SOAP message is very simple: it contains headers and body. The header and the body can be divided into blocks so that the information sent in them can be structured.
- SOAP as a communication protocol is also extremely simple: it only has two types of interaction (request response or notification). It also allows for client initiated or server initiated exchanges (mechanism is identical, the only difference is who starts)

What is SOAP?

- The W3C started working on SOAP in 1999. The current W3C recommendation is Version 1.2
- 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 HTTP (for Web based interaction) or SMTP(for e-mail based interaction)
  - 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. It also specifies what parts of the messages should be read by whom and how to react in case the content is not understood
  - 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 (how the client makes an RPC call, this is translated into a SOAP message, forwarded, turned into an RPC call at the server, the reply of the server converted into a SOAP message, sent to the client, and passed on to the client as the return of the RPC call)

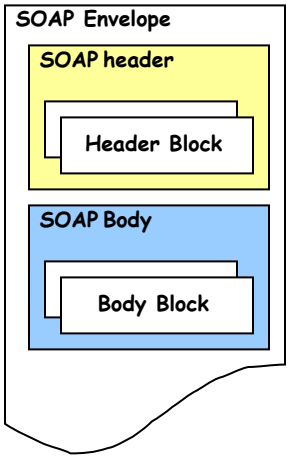
The background for SOAP

- SOAP was originally conceived as the minimal possible infrastructure necessary to perform RPC through the Internet:
  - use of XML as intermediate representation between systems
  - very simple message structure
  - mapping to HTTP for tunneling through firewalls and using the Web infrastructure
- The idea was to avoid the problems associated with CORBA's IIOP/GIOP (which fulfilled a similar role but using a non-standard intermediate representation and had to be tunneled through HTTP anyway)
- The goal was to have an extension that could be easily plugged on top of existing middleware platforms to allow them to interact through the Internet rather than through a LAN as in the original case. Hence the emphasis on RPC from the very beginning (essentially all forms of middleware use RPC at one level or another)
- Eventually SOAP started to be presented as a generic vehicle for computer driven message exchanges through the Internet and then it was open to support interactions other than RPC and protocols other then HTTP. This process, however, is only in its very early stages.

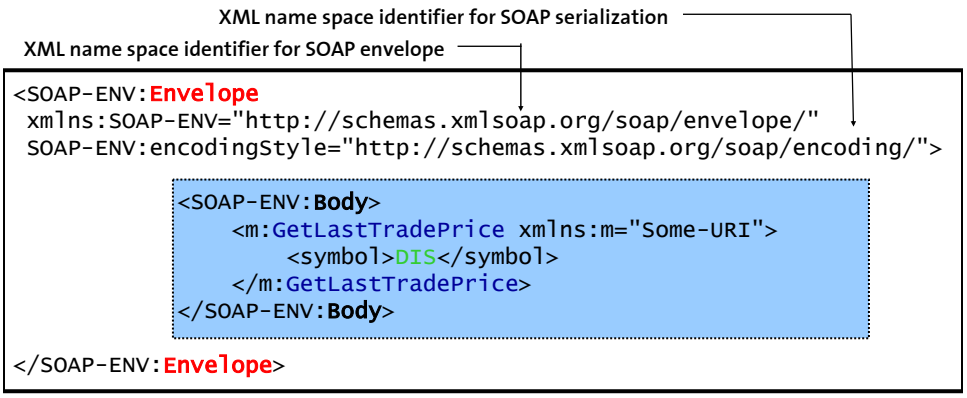
Structure of a SOAP Message

SOAP Messages

- SOAP is based on message exchanges
- Messages are structured with an envelope where the application encloses the data to be sent
- A message has two main parts:
  - **header**: which can be divided into blocks
  - **body**: which can also be divided into blocks
- SOAP does not say what to do with the header and the body, it only states that the header is optional and the body is mandatory
- Use of header and body, however, is implicit. The body is for application level data. The header is for infrastructure level data



For the XML fans (SOAP, body only)



From the: Simple Object Access Protocol (SOAP) 1.1. ©W3C Note 08 May 2000

SOAP example, header and body

```
<SOAP-ENV:Envelope
     xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
     SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"/>
          <SOAP-ENV:Header>
              <t:Transaction
                  xmlns:t="some-URI"
                  SOAP-ENV:mustUnderstand="1">
                      5
              </t:Transaction>
          </SOAP-ENV:Header>
          <SOAP-ENV:Body>
              <m:GetLastTradePrice xmlns:m="Some-URI">
                  <symbol>DEF</symbol>
              </m:GetLastTradePrice>
          </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

From the: Simple Object Access Protocol (SOAP) 1.1. © W3C Note 08 May 2000

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:
  - SOAP actor attribute: who should process that particular header entry (or header block). The actor can be either: none, next, ultimateReceiver. **None** is used to propagate information that does not need to be processed. **Next** indicates that a node receiving the message can process that block. **ultimateReceiver** indicates the header is intended for the final recipient of the message
  - 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 body is intended for the application specific data contained in the message
- A body entry (or a body block) is syntactically equivalent to a header entry with attributes actor= ultimateReceiver and mustUnderstand = 1
- Unlike for headers, SOAP does specify the contents of some body entries:
  - mapping of RPC to a collection of SOAP body entries
  - the Fault entry (for reporting errors in processing a SOAP message)
- 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

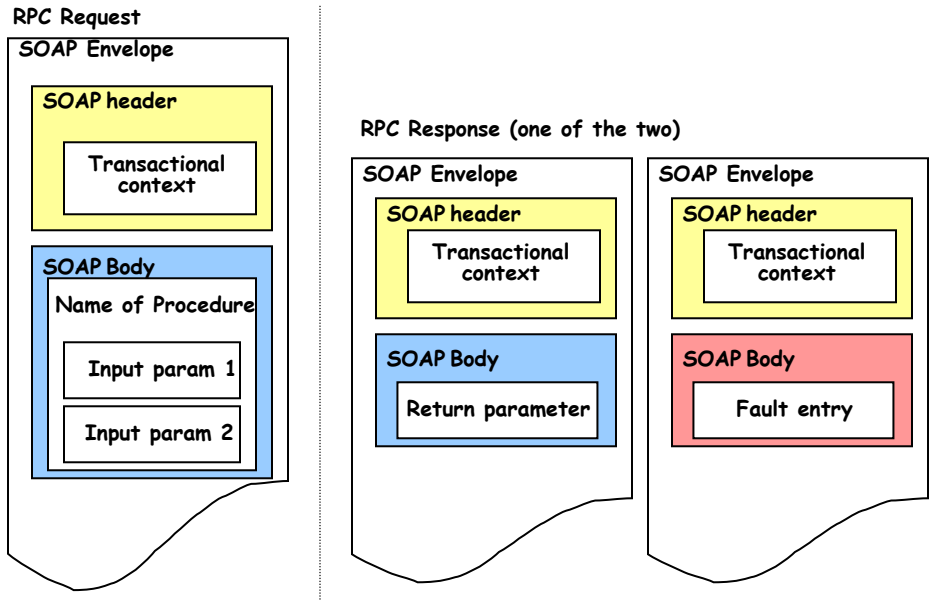
SOAP Fault element (v 1.2)

- 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 o f the original headers was not understood.

Message processing

- SOAP specifies in detail how messages must be processed (in particular, how header entries must be processed)
  - Each SOAP node along the message path looks at the role associated with each part of the message
  - There are three standard roles: none, next, or ultimateReceiver
  - Applications can define their own roles and use them in the message
  - The role determines who is responsible for each part of a message
- If a block does not have a role associated to it, it defaults to ultimateReceiver
- If a mustUnderstand flag is included, a node that matches the role specified must process that part of the message, otherwise it must generate a fault and do not forward the message any further
- SOAP 1.2 includes a relay attribute. If present, a node that does not process that part of the message must forward it (i.e., it cannot remove the part)
- The use of the relay attribute, combined with the role next, is useful for establishing persistence information along the message path (like session information)

From TRPC to SOAP messages



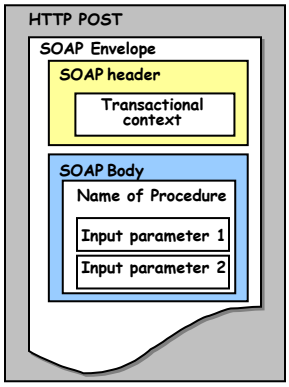
IKS Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Mapping SOAP to a transport protocol

SOAP and HTTP

- A binding of SOAP to a transport protocol is a description of how a SOAP message is to be sent using that transport protocol
- The typical binding for SOAP is HTTP
- 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



In XML (a request)

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

<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>
```

From the: Simple Object Access Protocol (SOAP) 1.1. © W3C Note 08 May 2000

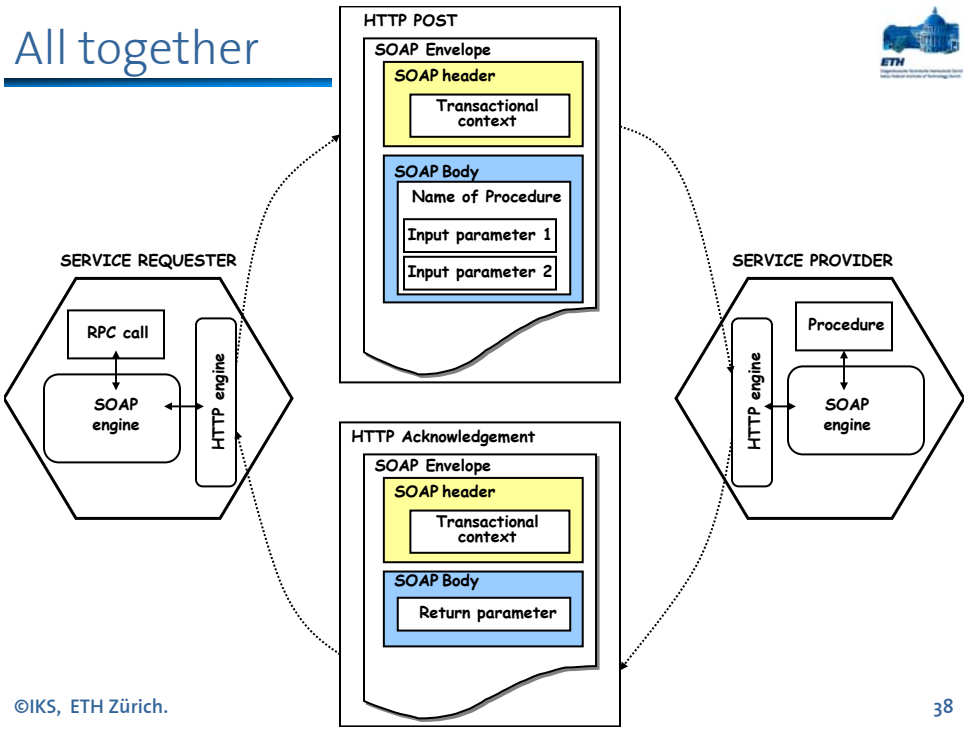
In XML (the response)

From the: Simple Object Access Protocol (SOAP) 1.1. © W3C Note 08 May 2000

```
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>
```


All together



SOAP Summary

- SOAP, in its current form, provides a basic mechanism for:
  - encapsulating messages into an XML document
  - mapping the XML document with the SOAP message into an HTTP request
  - transforming RPC calls into SOAP messages
  - simple rules on how to process a SOAP message (rules became more precise and comprehensive in v1.2 of the specification)
- SOAP is a very simple protocol intended for transferring data from one middleware platform to another. In spite of its claims to be open (which are true), current specifications and are very tied to RPC and HTTP.
- SOAP takes advantage of the standardization of XML to resolve problems of data representation and serialization (it uses XML Schema to represent data and data structures, and it also relies on XML for serializing the data for transmission). As XML becomes more powerful and additional standards around XML appear, SOAP can take advantage of them by simply indicating what schema and encoding is used as part of the SOAP message. Current schema and encoding are generic but soon there will be vertical standards implementing schemas and encoding tailored to a particular application area (e.g., the efforts around EDI)

IKS Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Web Services Description Language (WSDL)

What is WSDL?

- The Web Services Description Language specification is in version 1.1 (March 2001) and currently under revision (v2.0 is in the working draft stage, August 2004)
- WSDL 1.1 discusses how to describe the different parts that comprise a Web service interface
  - the type system used to describe the service data model (XML Schema)
  - the messages involved in the interaction with the service
  - the individual operations composed of 4 possible message exchange patterns
  - the sets of operations that constitute a service
  - the mapping to a transport protocol for the messages
  - the location where the service provider resides
  - groups of locations that can be used to access the same service
- It also includes specification indicating how to bind WSDL to the SOAP, HTTP (POST/GET) and MIME protocols

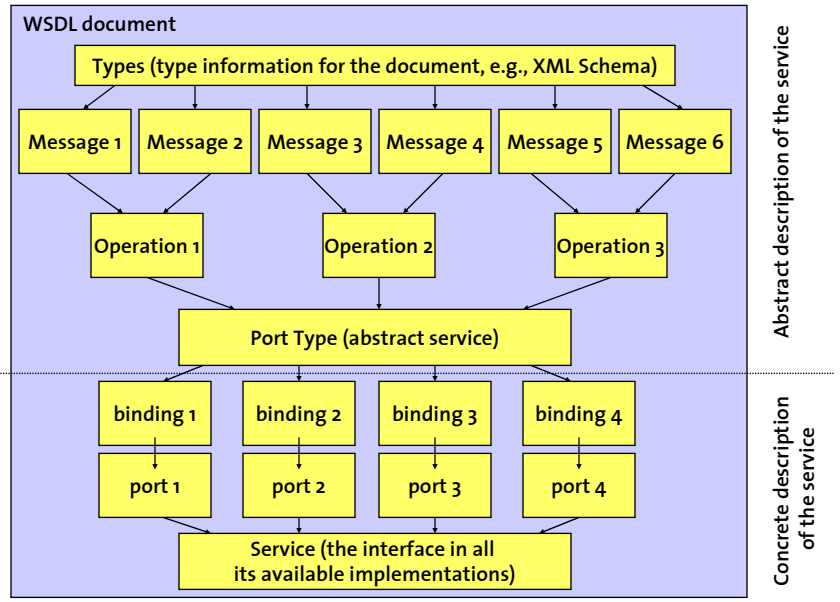
WSDL as an IDL

- WSDL can be best understood when we approach it as an XML version of an IDL that also covers the aspects related to integration through the Internet and the added complexity of Web services
- An IDL in conventional middleware and enterprise application integration platforms has several purposes:
  - description of the interfaces of the services provided (e.g., RPC)
  - serve as an intermediate representation for bridging heterogeneity by providing a mapping of the native data types to the intermediate representation associated to the IDL in question
  - serve as the basis for development through an IDL compiler that produces stubs and libraries that can be use to develop the application
- A conventional IDL does not include information such as:
  - location of the service (implicit in the platform and found through static or dynamic binding)
  - different bindings (typically an IDL is bound to a transport protocol)
  - sets of operations (since an interface defines a single access point and there is no such a thing as a sequence of operations involved in the same service)

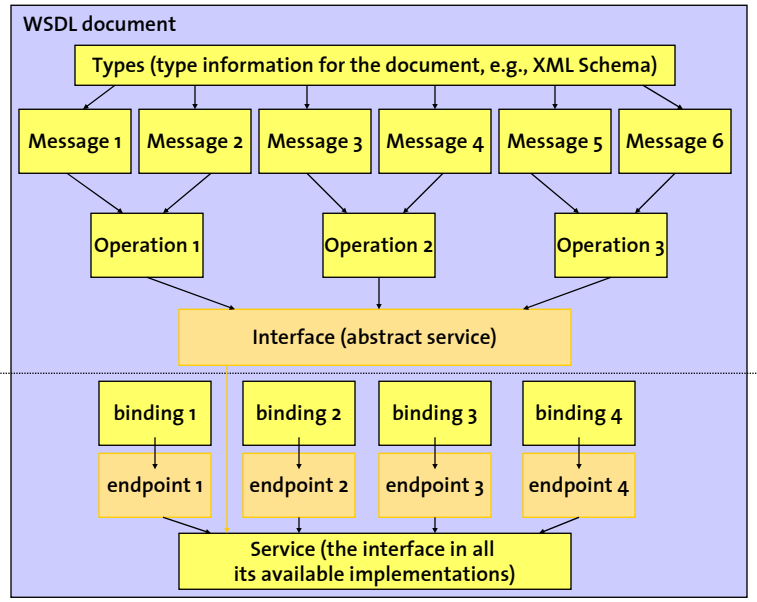
IDL (Interface Definition Language)

- All RPC systems have a language that allows to describe services in an abstract manner (independent of the programming language used). This language has the generic name of IDL (e.g., the IDL of SUN RPC is called XDR)
- The IDL allows to define each service in terms of their names, and input and output parameters (plus maybe other relevant aspects).
- An interface compiler is then used to generate the stubs for clients and servers (*rpcgen* in SUN RPC). It might also generate procedure headings that the programmer can then used to fill out the details of the implementation.
- Given an IDL specification, the interface compiler performs a variety of tasks:
- generates the client stub procedure for each procedure signature in the interface. The stub will be then compiled and linked with the client code
- Generates a server stub. It can also create a server *main*, with the stub and the dispatcher compiled and linked into it. This code can then be extended by the designer by writing the implementation of the procedures
- It might generate a *.h file for importing the interface and all the necessary constants and types

Elements of WSDL 1.1



Elements of WSDL 2.0



Types in WSDL

- The types in WSDL are used to specify the contents of the messages (normal messages and fault messages) that will be exchanged as part of the interactions with the Web service
- The type system is typically based on XML Schema (structures and data types) - support is mandatory for all WSDL processors
- An extensibility element can be used to define a schema other than XML Schema

Types in WSDL (Example)

PURCHASE ORDER TYPE

```
<element name="PO" type="tns:POType"/>
<complexType name="POType">
   <all>
      <element name="id" type="string"/>
      <element name="name" type="string"/>
      <element name="items">
         <complexType>
            <all>
               <element name="item" type="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
            </all>
         </complexType>
      </element>
   </all>
</complexType>
```

ITEM TYPE

```
<complexType name="Item">
   <all>
      <element name="quantity" type="int"/>
      <element name="product" type="string"/>
   </all>
</complexType>
```

INVOICE TYPE

```
<element name="Invoice" type="tns:InvoiceType"/>
<complexType name="InvoiceType">
   <all>
      <element name="id" type="string"/>
   </all>
</complexType>
```

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

Messages and Faults

- Messages have a name that identifies them throughout the XML document. Messages are divided into parts, each of them being a data structure represented in XML. Each part must have a type (basic or complex types, previously declared in the WSDL document).
- A WSDL message element matches the contents of the body of a SOAP message. By looking at the types and looking at the message, it is possible to build a SOAP message that matches the WSDL description (and this can be done automatically since the description is XML based and the types also supported by SOAP)
- A message does not define any form of interaction, it is just a message
- In WSDL 1.0, the structure of a "message" is explicitly defined, listing all of its parts.
- In WSDL 2.0, a "message reference component" is defined as part of an operation and contains three elements :
  - Message label (indicating the message pattern used for the message)
  - Direction (whether it is an inbound or outbound message)
  - Message element (the actual contents of the message expressed in terms of the types previously defined)
- Faults are a special kind of message used to report errors

1.0

```
<message name="PO">
   <part name="po" element="tns:PO"/>
   <part name="invoice" element="tns:Invoice"/>
</message>
```

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

Operations

- Operations provide the first level of context for the messages. In WSDL 1.0, there are four types of operations:
  - one-way: the client send a message to the server
  - request-response: the client sends a request, the server replies with a response
  - Solicit-response: the server sends a message and the client replies
  - Notification: the server sends a message
- In WSDL 2.0, an operation is a set of messages and faults. The sequencing and number of messages in the operation is determined by the message exchange pattern
- The style of an operation distinguishes between RPC-like behavior, document oriented message exchange or (in 2.0) set- and get- of attributes
- Operations can be annotated with features and properties (e.g., reliability, security, routing)

ONE-WAY:

```
<wsdl:operation name="Purchase">
  <wsdl:input name="Order" message="PO"/>
</wsdl:operation>
```

REQUEST-RESPONSE:

```
<wsdl:operation name="Purchase">
 <wsdl:input name="Order" message="PO"/>
  <wsdl:output name="Confirm" message="Conf"/>
  <wsdl:fault name="Error" message="POError"/>
</wsdl:operation>
```

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

Port Types

- A Port Type corresponds to the abstract definition of a Web service (abstract because it does not specify any information about where the service resides or what protocols are used to invoke the Web service)
- The Port Type is simply a list of operations that can be used in that Web service
- Operations are not defined by themselves but only as part of a PortType
- In WSDL 2.0 Port Types have been renamed to Interfaces (which also support inheritance)

```
<message name="m1">
  <part name="body" element="tns:GetCompanyInfo"/>
</message>

<message name="m2">
 <part name="body" element="tns:GetCompanyInfoResult"/>
 <part name="docs" type="xsd:string"/>
 <part name="logo" type="tns:ArrayOfBinary"/>
</message>

<portType name="pt1">
  <operation name="GetCompanyInfo">
      <input message="m1"/>
      <output message="m2"/>
  </operation>
</portType>
```

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

Bindings and ports

- A binding defines message formats and protocol details for the operations and messages of a given Port Type
- A binding corresponds to a single Port Type (obvious since it needs to refer to the operations and messages of the Port Type)
- A Port Type can have several bindings (thereby providing several access channels to the same abstract service)
- The binding is extensible with elements that allow to specify mappings of the messages and operations to any format or transport protocol. In this way WSDL is not protocol specific.
- A port specifies the address of a binding, i.e., how to access the service using a particular protocol and format
- Ports can only specify one address and they should not contain any binding information
- The port is often specified as part of a service rather than on its own

Bindings and Ports (example)

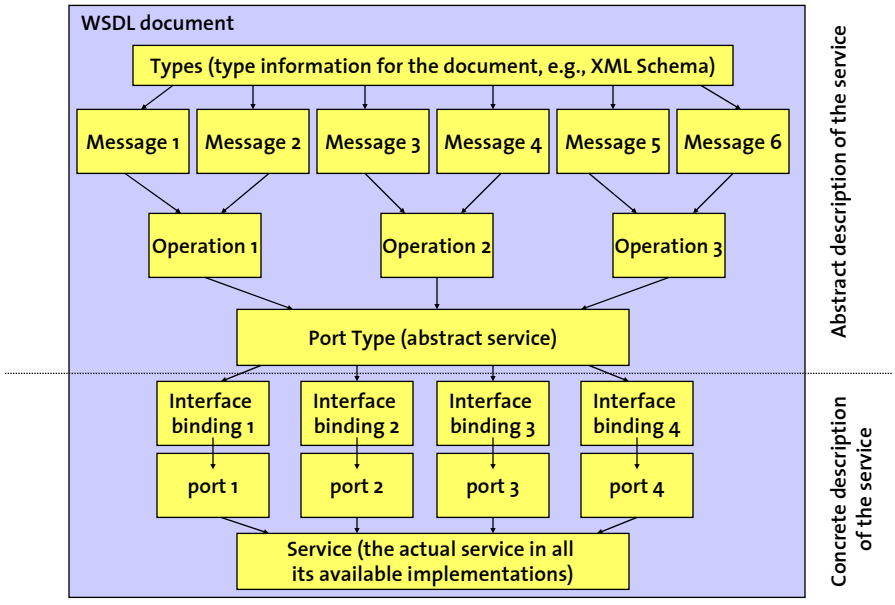
From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

```
<binding name="b1" type="tns:pt1">
    <operation name="GetCompanyInfo">
        <soap:operation soapAction="http://example.com/GetCompanyInfo"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <mime:multipartRelated>
                    <mime:part>
                        <soap:body parts="body" use="literal"/>
                    </mime:part>
                    <mime:part>
                        <mime:content part="docs" type="text/html"/>
                    </mime:part>
                    <mime:part>
                        <mime:content part="logo" type="image/gif"/>
                        <mime:content part="logo" type="image/jpeg"/>
                    </mime:part>
                </mime:multipartRelated>
            </output>
    </operation>
</binding>
<service name="CompanyInfoService">
        <port name="CompanyInfoPort"binding="tns:b1">
                <soap:address location="http://example.com/companyinfo"/>
        </port>
</service>
```

Services

- Services group a collections of ports together and therefore become the complete definition of the service as seen by the outside:
  - a service supports several protocols (it has several bindings)
  - access to the service under a given protocol is through a particular address (specified in the ports of each binding)
  - the operations and messages to exchange are defined in the Port Type
- Ports that are part of the same service may not communicate with each other
- Ports that are part of the same service are considered as alternatives all of them with the same behavior (determined by the Port Type) but reachable through different protocols

Elements of WSDL



WSDL example (1)

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

```
<?xml version="1.0"?>
    <definitions name="StockQuote“ targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl“
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:xsd1="http://example.com/stockquote.xsd“
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="GetTradePriceInput">
    <part name="tickerSymbol" element="xsd:string"/>
    <part name="time" element="xsd:timeInstant"/>
  </message>

  <message name="GetTradePriceOutput">
    <part name="result" type="xsd:float"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetTradePrice">
      <input message="tns:GetTradePriceInput"/>
      <output message="tns:GetTradePriceOutput"/>
    </operation>
  </portType>
```


WSDL example (2)

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

```
  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetTradePrice">
        <soap:operation soapAction="http://example.com/GetTradePrice"/>
        <input>
          <soap:body use="encoded" namespace="http://example.com/stockquote"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output>
          <soap:body use="encoded" namespace="http://example.com/stockquote"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>>
   </binding>

  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>
</definitions>
```