ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Information Systems
From Middleware to Web services (Part 2)
VS - WS 2002/2003

Prof. Dr. Gustavo Alonso
Computer Science Department
ETH Zürich
alonso@inf.ethz.ch

What is SOAP?

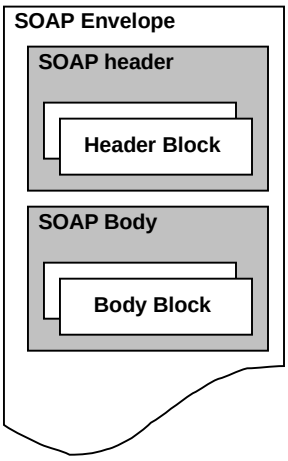
- The W3C started working on SOAP in 1999. SOAP 1.0 was entirely based on HTTP. The current specification is SOAP 1.1 (May 2000) is more generic by including other transport protocols. Version 1.2 is at the working draft stage.
- SOAP 1.1 covers the following four main areas:
  - → A message format for 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

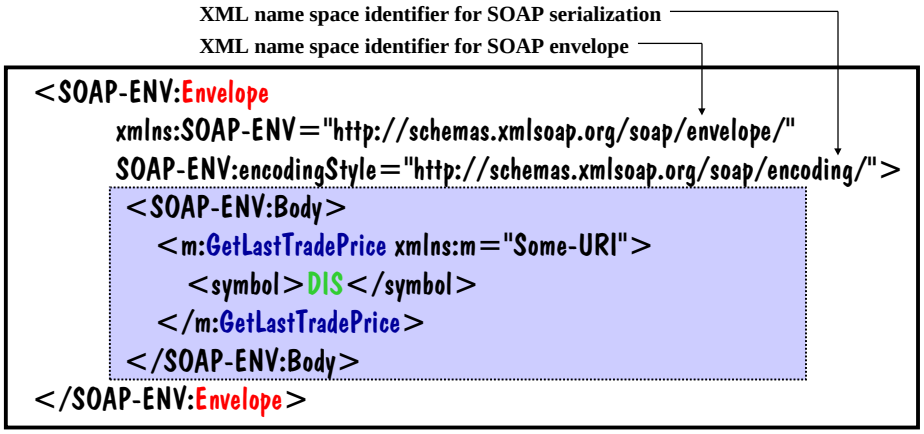
- SOAP was originally conceived as the minimal possible infrastructure necessary to perform RPC through the Internet:
  - → use of XML as intermediate representation between systems
  - → very simple message structure
  - → mapping to HTTP for tunneling through firewalls and using the Web infrastructure
- 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 any way)
- 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 it is typically the 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.

SOAP messages

- SOAP is based on message exchanges
- Messages are seen as envelops 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 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 (for, e.g., transactions), security information (e.g., certificates)
- SOAP provides mechanisms to specify who should deal with headers and what to do with them. For this purpose it includes:
  - → SOAP actor attribute: who should process that particular header entry (or header block). The actor can be either: none, next, ultimateReceiver. 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

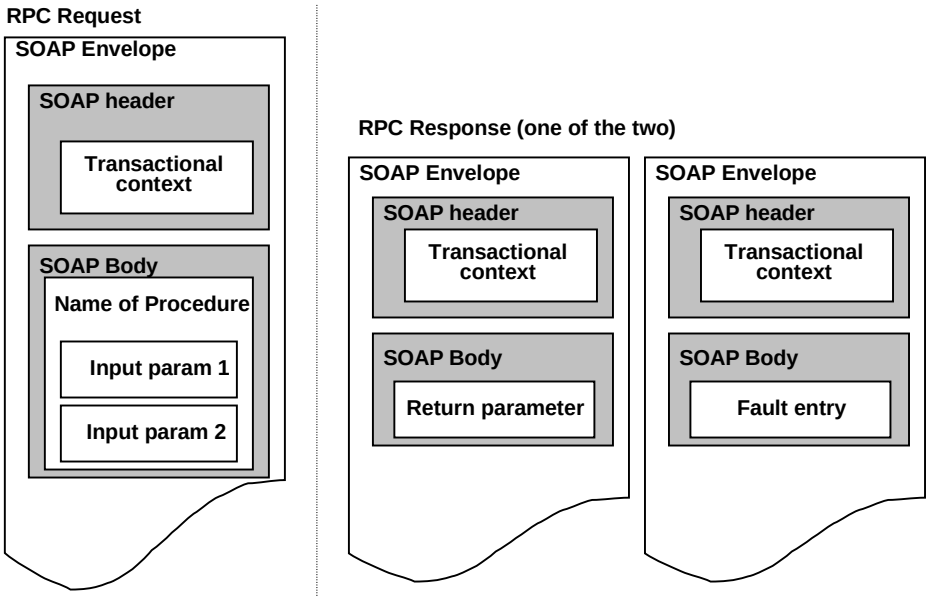
SOAP Fault element (v 1.2)

- 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



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):

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

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

List of MIME types accepted by the browser

Information about the environment where the browser is running

E-mail or identifier of the user (provided by the browser)

End of request

- Request methods: GET (retrieve data), POST (append information), PUT (send information), DELETE (remove information), ...

HTTP server side

- 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.
```

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 sent

- 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
From:  grobe@www.cc.ukans.edu
Content-type: application/x-www-form-urlencoded
Content-length: 150
    * a blank line *
&name = Gustavo
&email= alonso@inf.ethz.ch
...
```

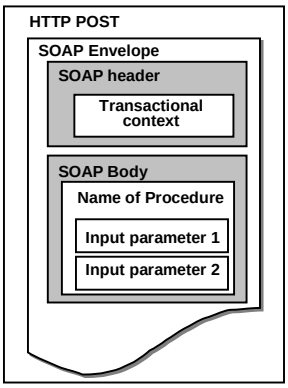
POST request indicating the CGI script to execute (post-query) GET can be used but requires the parameters to be sent as part of the URL: /cgi-bin/post-query?name=...&email=...

As before

Data provided through the form and sent back to the server

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)

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

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

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
     <m:GetLastTradePrice xmlns:m="Some-URI">
        <symbol>DIS</symbol>
     </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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

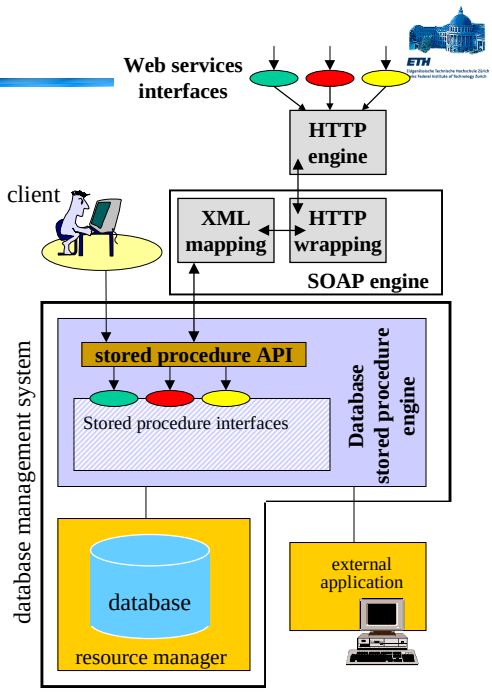
- SOAP, in its current form, provides a basic mechanism for:
  - → encapsulating messages into an XML document
  - → mapping the XML document with the SOAP message into an HTTP request
  - → transforming RPC calls into SOAP messages
  - → simple rules on how to process a SOAP message (rules becoming more precise and comprehensive in v1.2 of the specification)
- 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)
- 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 are very tied to RPC and HTTP.

SOAP and the client server model

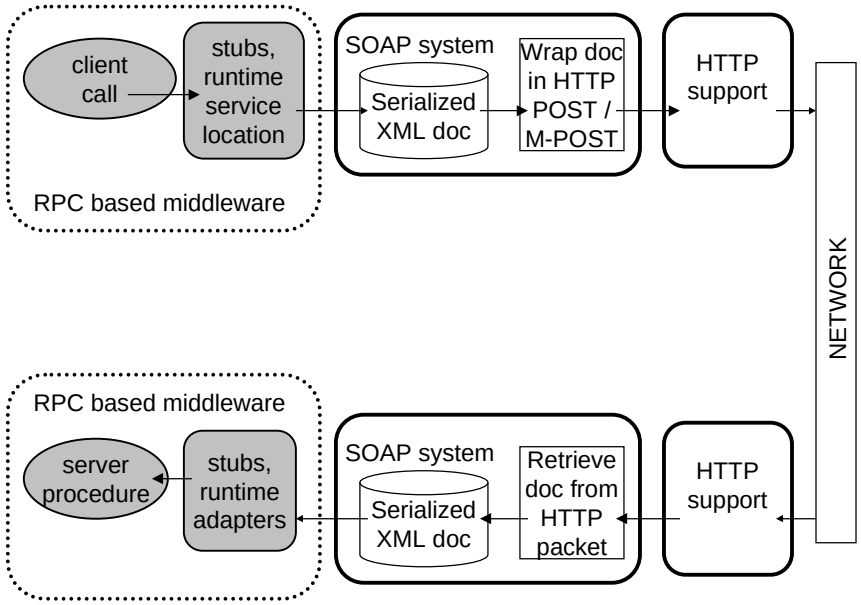
- The close relation between SOAP, RPC and HTTP has two main reasons:
  - → SOAP has been initially designed for client server type of interaction which is typically implemented as RPC or variations thereof
  - → RPC, SOAP and HTTP follow very similar models of interaction that can be very easily mapped into each other (and this is what SOAP has done)
- The advantages of SOAP arise from its ability to provide a universal vehicle for conveying information across heterogeneous middleware platforms and applications. In this regard, SOAP will play a crucial role in enterprise application integration efforts in the future as it provides the standard that has been missing all these years
- The limitations of SOAP arise from its adherence to the client server model:
  - → data exchanges as parameters in method invocations
  - → rigid interaction patterns that are highly synchronous
- and from its simplicity:
  - → SOAP is not enough in a real application, many aspects are missing

A first use of SOAP

- Some of the first systems to incorporate SOAP as an access method have been databases. The process is extremely simple:
  - → a stored procedure is essentially an RPC interface
  - → Web service = stored procedure
  - → IDL for stored procedure = translated into WSDL
  - → call to Web service = use SOAP engine to map to call to stored procedure
- This use demonstrates how well SOAP fits with conventional middleware architectures and interfaces. It is just a natural extension to them



Automatic conversion RPC - SOAP



SOAP exchange patterns (v 1.2)

SOAP response message exchange

- It involves a request which is not a SOAP message (implemented as an HTTP GET request method which eventually includes the necessary information as part of the requested URL) and a response that is a SOAP message
- This pattern excludes the use of any header information (as the request has no headers)

SOAP request-response message exchange

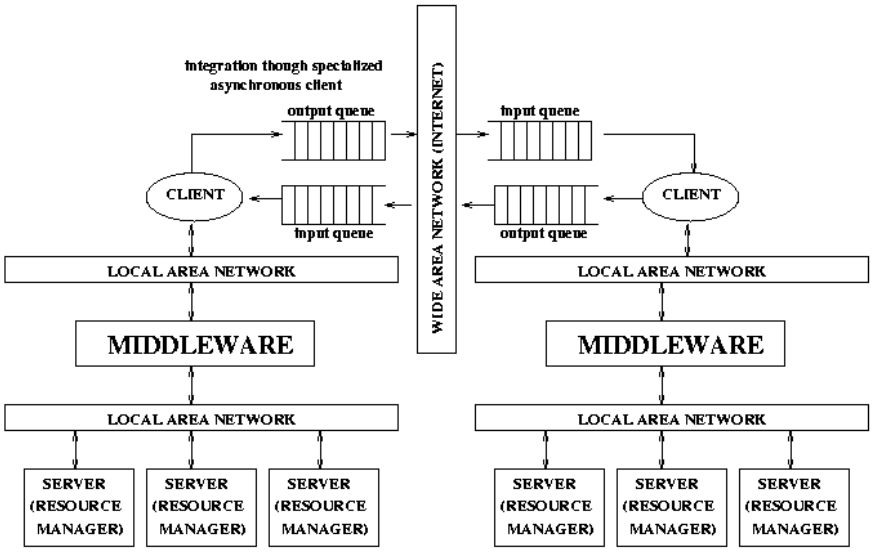
- It involves sending a request as a SOAP message and getting a second SOAP message with the response to the request
- This is the typical mode of operation for most Web services and the one used for mapping RPC to SOAP.
- This exchange pattern is also the one that implicitly takes advantage of the binding to HTTP and the way HTTP works

The crucial aspect in both cases is the pattern being implemented: it is a conventional client server pattern with a client making a request and the server sending a response in return. The only difference is whether the request is a SOAP message or not (which is only a minor point to accommodate the way many web browser and web pages work today)

Mapping SOAP to e-mail

- Currently, the SOAP specifications (including 1.2) do not contain an e-mail (SMTP) binding, they just show an example of how to send a SOAP message in an e-mail (in 1.2). Two possible options are:
  - →as normal e-mail text
  - →as an attachement
- In both cases, the SOAP message is not different from what has been discussed so far
- E-mail, however, changes the interaction patterns considered in SOAP (which are very tied to HTTP)
  - →SMTP does implement a mechanism whereby an e-mail message is automatically responded to with a delivery notification
  - →SOAP cannot use the delivery notification message to return the response to the request since the delivery notification message happens at the level of SMTP not at the level of the SOAP protocol
  - →the current 1.2 draft warns about the limitations of e-mail binding for SOAP reflecting once more the implicit client server model that inspires the design and development of SOAP

How to implement this with SOAP?



Implementing message queues

- In principle, it is not impossible to implement asynchronous queues with SOAP:
  - →SOLUTION A:
    - use SOAP to encode the messages
    - create an HTTP based interface for the queues
    - use an RPC/SOAP based engine to transfer data back and forth between the queues
  - →SOLUTION B:
    - use SOAP to encode the messages
    - create appropriate e-mail addresses for each queue
    - use an e-mail (SMTP) binding for transferring messages
- Both options have their advantages and disadvantages but the main problem is that none is standardize. Hence, there is no guarantee that different queuing systems with a SOAP will be able to talk to each other: all the advantages of SOAP are lost
- The fact that SOAP is so simple also makes it difficult to implement these solutions: a lot additional functionality is needed to implement reliable, practical queue systems

The need for attachments

- SOAP is based on XML and relies on XML for representing data types
- The original idea in SOAP was to make all data exchanged explicit in the form of an XML document much like what happens with IDLs in conventional middleware platforms
- This approach reflects the implicit assumption that what is being exchanged is similar to input and output parameters of program invocations
- This approach makes it very difficult to use SOAP for exchanging complex data types that cannot be easily translated to XML (and there is no reason to do so): images, binary files, documents, proprietary representation formats, embedded SOAP messages, etc.

```
<env:Body>
  <p:itinerary
    xmlns:p="http://.../reservation/travel">
     <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
    </p:departure>
    <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference/>
     </p:return>
  </p:itinerary>
</env:Body>
```

From SOAP Version 1.2 Part 0: Primer. December 2002

A possible solution

- There is a"SOAP messages with attachments note" proposed in 11.12.02 that addresses this problem
- It uses MIME types (like e-mails) and it is based in including the SOAP message into a MIME element that contains both the SOAP message and the attachment (see next page)
- The solution is simple and it follows the same approach as that taken in e-mail messages: include a reference and have the actual attachment at the end of the message
- The MIME document can be embedded into an HTTP request in the same way as the SOAP message
- The Apache SOAP 2.2 toolkit supports this approach
- Problems with this approach:
  - →handling the message implies dragging the attachment along, which can have performance implications for large messages
  - →scalability can be seriously affected as the attachment is sent in one go (no streaming)
  - →not all SOAP implementations support attachments
  - →SOAP engines must be extended to deal with MIME types (not too complex but it adds overhead)
- There are alternative proposals like DIME of Microsoft (Direct Internet Message Encapsulation) and WS-attachments

Attachments in SOAP

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
      start="<claim061400a.xml@claiming-it.com>"
Content-Description: This is the optional message description.
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
..
<theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
..
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.tiff@claiming-it.com>

...binary TIFF image...
--MIME_boundary--
```

SOAP MESSAGE

ATTACHMENT

From SOAP Messages with Attachments. W3C Note 11 December 2000

The problems with attachments

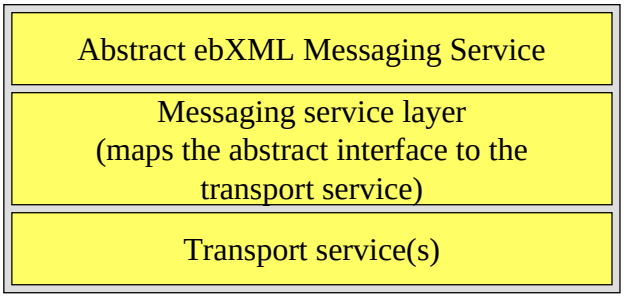
- Attachments are relatively easy to include in a message and all proposals (MIME or DIME based) are similar in spirit
- The differences are in the way data is streamed from the sender to the receiver and how these differences affect efficiency
  - → MIME is optimized for the sender but the receiver has no idea of how big a message it is receiving as MIME does not include message length for the parts it contains
  - → this may create problems with buffers and memory allocation
  - → it also forces the receiver to parse the entire message in search for the MIME boundaries between the different parts (DIME explicitly specifies the length of each part which can be use to skip what is not relevant)
- All these problems can be solved with MIME as it provides mechanisms for adding part lengths and it could conceivably be extended to support some basic form of streaming
- Technically, these are not very relevant issues and have more to do with marketing and control of the standards
- The real impact of attachments lies on the specification of Web services (discussed later on)

SOAP as simple protocol

- SOAP does not include anything about:
  - → reliability
  - → complex message exchanges
  - → transactions
  - → security
  - → ...
- As such, it is not adequate by itself to implement industrial strength applications that incorporate typical middleware features such as transactions or reliable delivery of messages
- SOAP does not prevent such features from being implemented but they need to be standardized to be useful in practice:
  - → WS-security
  - → WS-Coordination
  - → WS-Transactions
  - → ...
- A wealth of additional standards are being proposed to add the missing functionality

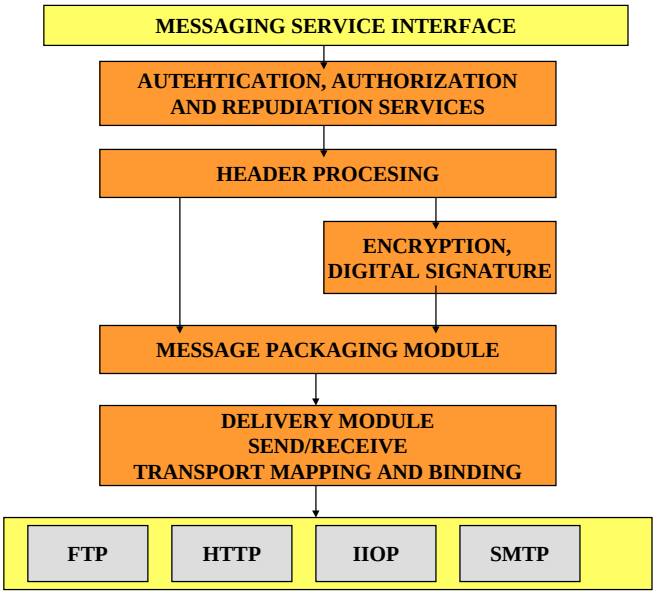
Beyond SOAP

- Not everybody agrees to the procedure of SOAP + WS-"extensions", some organizations insist that a complete protocol specification for Web services needs to address much more than just getting data across
- ebXML, as an example, proposes its own messaging service that incorporates many of the additional features missing in SOAP. This messaging service can be built using SOAP as a lower level protocol but it considers the messaging problem as a whole
- The idea is not different from SOAP ...



- but extended to incorporate additional features (next page)

ebXML messaging service



ebXML and SOAP

- The ebXML Messaging specification clarifies in great detail how to use SOAP and how to add modules implementing additional functionality:
  - → ebXML message = MIME/Multipart message envelope according to "SOAP with attachments" specification
  - → ebXML specified standard headers:
    - MessageHeader: id, version, mustUnderstand flag to 1, from, to, conversation id, duplicate elimination, etc.
  - → ebXML recommends to use the SOAP body to declare (manifest) the data being transferred rather than to carry the data (the data would go in pther parts of the MIME message)
  - → ebXML defines a number of core modules and how information relevant to these modules is to be exchanged:
    - security (for encryption and signature handling)
    - error handling (above the SOAP error handling level)
    - sync/reply (to maintain connections open across intermediaries)

Additional features of ebXML messages

- Reliable messaging module
  - → a protocol that guarantees reliable delivery between two message handlers. It includes persistent storage of the messages and can be used to implement a wide variety of delivery guarantees
- Message status service
  - → a service that allows to ask for the status of a message previously sent
- Message ping service
  - → to determine if there is anybody listening at the other end of the line
- Message order module
  - → to deliver messages to the receiver in a particular order. It is based on sequence numbers
- Multi-hop messaging module
  - → for sending messages through a chain of intermediaries and still achieve reliability
- This are all typical features of a communication protocol that are needed anyway (including practical SOAP implementations)

What is WSDL?

- The Web Services Description Language specification is in version 1.1 (March 2001) and currently under revision (v1.2 is in the working draft stage)
- WSDL 1.1 discusses how to describe the different parts that comprise a Web service:
  - → the type system used to describe the interfaces (based on XML)
  - → the messages involved in invoking the service
  - → the individual operations that make up the service
  - → the sets of operations that constitute a service
  - → the mapping to a transport protocol for the messages
  - → the location where the service resides
  - → groups of locations that can be used to access the same service
- It also includes specification indicating how to bind WSDL to SOAP, HTTP and MIME

WSDL vs 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

IDL Example in SUN's XDR

```
const MAX;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;

struct Data {
 int length;
 char buffer[MAX];
}

struct writeargs {
  FileIdentifier f;
  FilePointer position;
  Length length;
}

struct readargs {
  FileIdentifier f;
  FilePointer position;
  Length length;
}
```

```
program FILEREADWRITE {
  version VERSION {
     void WRITE(writeargs)=1; /*proc number*/
     Data READ(readargs)=2; /*proc number*/
     }=2; /* version number */
}=2001; /* Program number*/
```


Example (XDR in SUN RPC)

- Marshalling or serializing can be done by hand (although this is not desirable) using (in C) *sprintf* and *sscanf*.

Message= "Alonso" "ETHZ" "2001"

```
char *name="Alonso", place="ETHZ";
int year=2001;

sprintf(message, "%d %s %s %d %d",
  strlen(name), name, strlen(place), place,
  year);
```

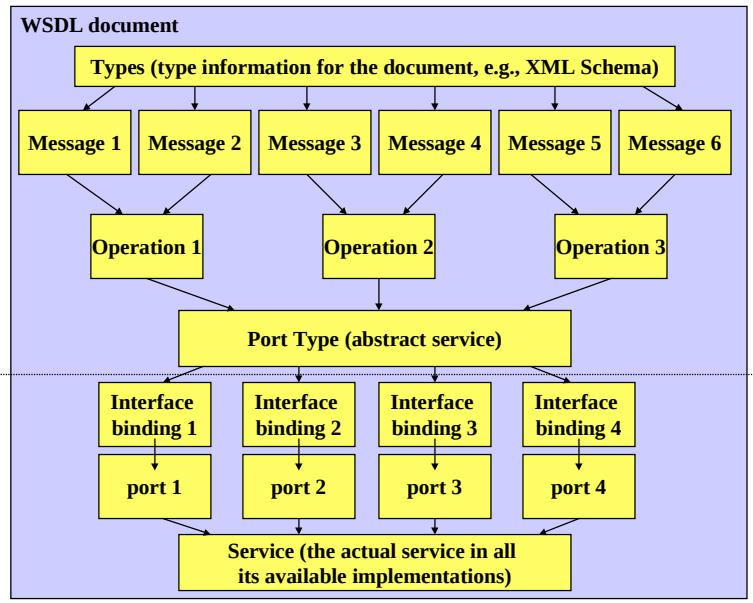
Message after marshalling =
"6 Alonso 4 ETHZ 2001"

- Remember that the type and number of parameters is know, we only need to agree on the syntax ...

- SUN XDR follows a similar approach:
  - → messages are transformed into a sequence of 4 byte objects, each byte being in ASCII code
  - → it defines how to pack different data types into these objects, which end of an object is the most significant, and which byte of an object comes first
  - → the idea is to simplify computation at the expense of bandwidth

6	String length
A l o n	
s o	
4	String length
E T H Z	
2 0 0 1	cardinal

Elements of WSDL



Types in WSDL

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

- 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

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

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


Operations

- Operations provide the first level of context for the messages. In WSDL, there are four types of operations (WSDL does not talk about client/server but about endpoints):
  - → 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 with a response
  - → Notification: the server sends a message
- WSDL only defines bindings for the first two

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

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>
```

Port Types

- A Port Type corresponds to the abstract definition of a Web service (abstract because it does not specify location or access protocol)
- 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

```
<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)

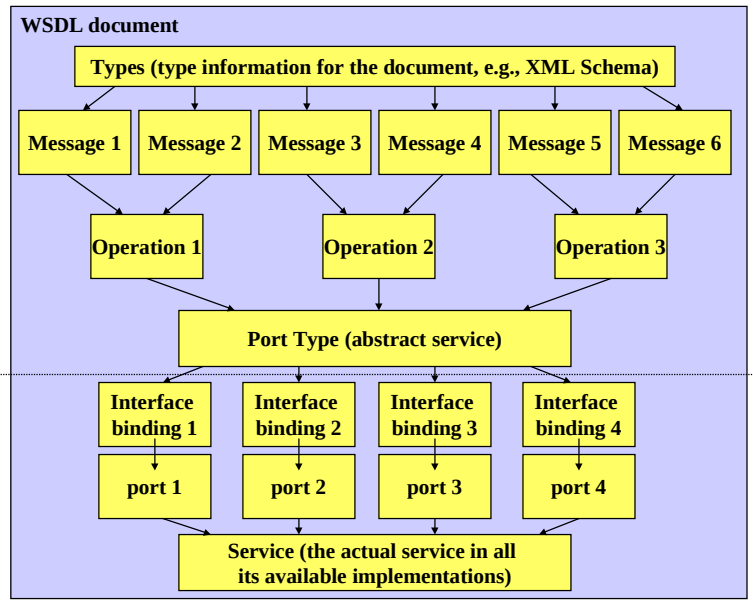
From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

```
<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 bindings (SOAP) 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 binding (SOAP) 2

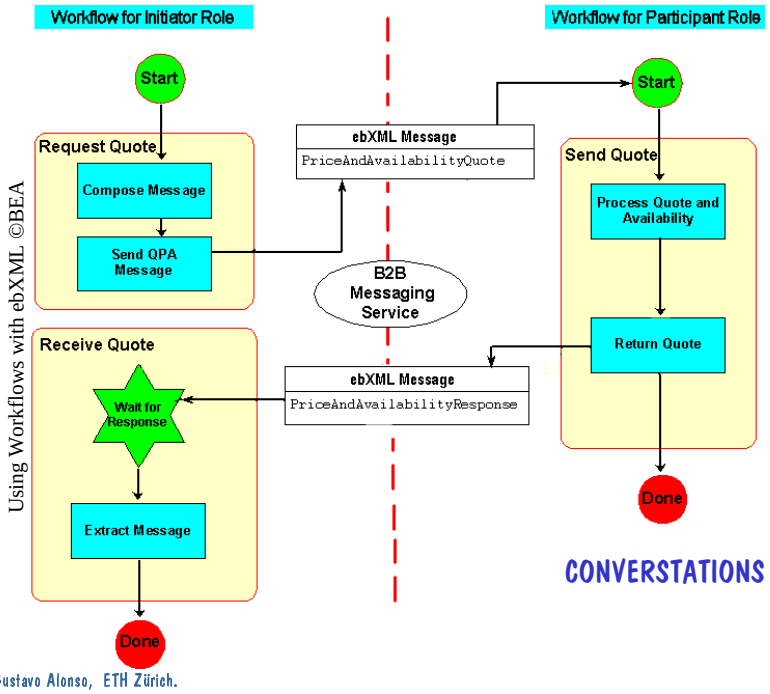
From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

```
<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>
```


Conversations

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



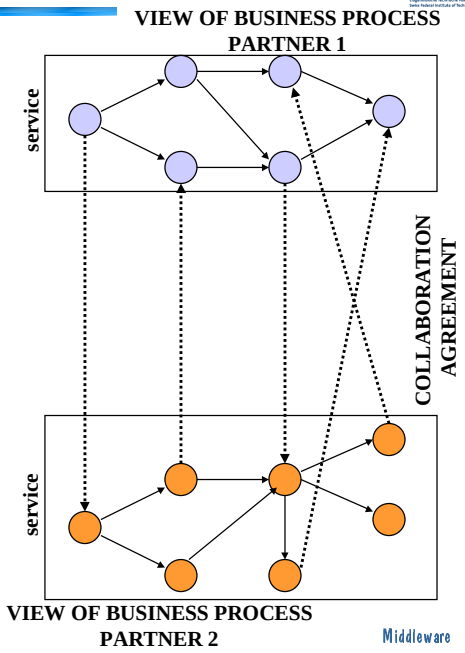
Conversations (example)

"The xCBL 3.5 ChangeOrder document is a buyer-initiated document that can be used to change an existing Order already received and responded to by a seller. The document can be used to make changes to header level information, change line items, cancel line items, add line items, etc. Note that if an OrderResponse has not been received for a given Order, a ChangeOrder is not necessary (an Order with a purpose of "Replace" should be used). Similarly, if an entire order is to be cancelled (regardless of whether a response has been received or not) an Order with a purpose of "Cancellation" should be used."

xCBL 3.5 Order Management Recommended Use, Version 1.0 November 19, 2001

Other standards

- ebXML shows here again what could be a possible evolution path for WSDL (or the type of technology that is being built on top of WSDL)
- ebXML does not consider a client/server model but an interaction between partners (peer-to-peer)
- Consequently, the service description model for ebXML is the description of how two business processes interact with each other:
  - → partners publish their processes (an external view over them)
  - → a collaboration agreement is drawn based on those processes
  - → the collaboration agreement describes the business protocol between those partners



Automatic development

- The ultimate goal of WSDL is to provide support for automating as much as possible for the development process for Web services:
  - → given the WSDL description, a WSDL compiler generates the stubs or skeletons necessary to develop clients that can interact with the service
  - → for that purpose, WSDL must rely on a standard protocol so that generic stubs can be created, this is where SOAP comes into the picture
  - → WSDL is meant as a bridge between internal services and external (Web) services)
- Similarly, the ultimate goal in ebXML is to automate the process of developing a collaboration agreement, deploying it and enforcing its rules:
  - → given a collaboration agreement (possibly a standard one), one should be able to automatically generate a stub or skeleton for the individual business processes at the ends of the agreement
  - → partners need only to extend the stub process with their own internal logic
  - → this is why ebXML needs more than SOAP as the agreement is used to control and direct the flow of messages between partners at the platform level

What is UDDI?

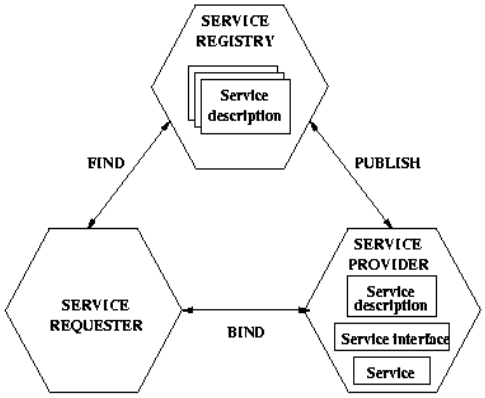
- The UDDI specification is probably the one that has evolved the most from all specifications we have seen so far. The latest version is version 3 (July 2002):
  - →version 1 defined the basis for a business service registry
  - →version 2 adapted the working of the registry to SOAP and WSDL
  - →version 3 redefines the role and purpose of UDDI registries, emphasizes the role of private implementations, and deals with the problem of interaction across private and public UDDI registries
- Originally, UDDI was conceived as an "Universal Business Registry" similar to search engines (e.g., Google) which will be used as the main mechanism to find electronic services provided by companies worldwide. This triggered a significant amount of activity around very advanced and complex scenarios (Semantic Web, dynamic binsing to partners, runtime/automatic partner selection, etc.)
- Nowadays UDDI is far more pragmatic and recognizes the realities of B2B interactions: it presents itself as the "infrastructure for Web services", meaning the same role as a name and directory service (i.e., binder in RPC) but applied to Web services and mostly used in constrained environments (internally within a company or among a predefined set of business partners)

Hype and reality

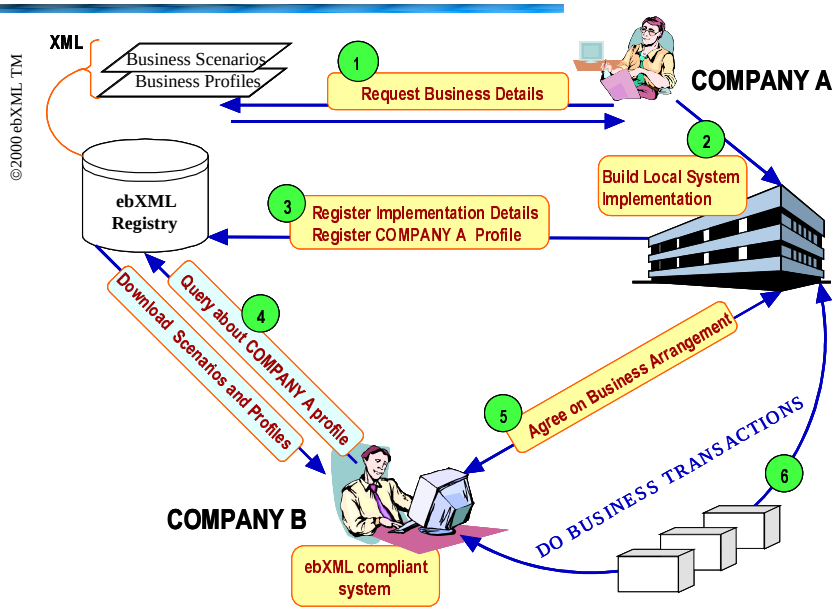
- There are a few universal UDDI registries in operation (maintained by IBM, Microsoft, SAP, etc)
- These registries are very visible and often the first thing one sees of Web services
- Unfortunately, these registries are still very small and most of the entries in them do not work or do not correspond to any real service
- This has been a source of criticism to We services in general. The criticism has not been entirely undeserved but it is often misguided: what was there to criticize was not UDDI itself but the use that was been made of it and the hype around dynamic Web services
- UDDI is rather useful if seen as supporting infrastructure for Web services in well defined and constrained environments (i.e., without public access and where there is a context that provides the missing information)
- Most of the UDDI registries in place today are private registries operating inside companies (recall that the widest use of Web services today is for conventional EAI) or maintained by a set of companies in a private manner
- UDDI has now become the accepted way to document Web services and supply the information missing in WSDL descriptions

Role of UDDI

- Services offered through the Internet to other companies require much more information that a typical middleware service
- In many middleware and EAI efforts, the same people develop the service and the application using the service
- This is obviously no longer the case and, therefore, using a service requires much more information that it is typically available for internal company services
- This documentation has three aspects to it:
  - → basic information
  - → categorization
  - → technical data



More detailed (ebXML architecture)



UDDI data

- An entry in an UDDI registry is an XML document composed of different elements (labeled as such in XML), the most important ones being:
  - → *businessEntity* : is a description of the organization that provides the service.
  - → *businessService*: a list of all the Web services offered by the business entity.
  - → bindingTemplate: describes the technical aspects of the service being offered.
  - → *tModel*: ("technical model")is a generic element that can be used to store addotional information about the service, typically additional technical information on how to use the service, conditions for use, guarantees, etc.
- Together, these elements are used to provide:
  - → white pages information: data about the service provider (name, address, contact person, etc.)
  - → yellow pages information: what type of services are offered and a list of the different services offered
  - → green pages information: technical information on how to use each one of the services offered, including pointers to WSDL descriptions of the services (which do not reside in the UDDI registry)

Business entity

- The generic white and yellow pages information about a service provider is stored in the businessEntity, which contains the following data:
  - → each businessEntity has a businessKey
  - → discoveryURLs: a list of URLs that point to alternate, file based service discovery mechanisms.
  - → Name: (textual information)
  - → Business description: (textual information)
  - → Contacts: (textual information)
  - → businessServices: a list of services provided by the businessEntity
  - → identifierBag: a list of external identifiers
  - → categoryBag: a list of business categories (e.g., industry, product category, geographic region)
- The businessEntity does not need to be the company. It is meant to represent any entity that provides services: it can be a department, a group of people, a server, a set of servers, etc

Business service

- The services provided by a business entity re described in business terms using businessService elements. A businessService element can describe a single Web service or a group of related Web services (all of them offered by the same businessEntity)
- A businessEntity can have several businessServices but a businessService belongs to one businessEntity
- The businessService can actually by provided by a different businessEntity that the one where the element is found. This is called projection and allows to include services provided by other organizations as part of the own services
- It contains:
  - → a serviceKey that uniquely identifies the service and the businessEntity (not necessarily the same as where the businessService is found)
  - → name: as before
  - → description: as before
  - → categoryBag: as before
  - → bindingTemplates: a list to all the bindingTemplates for the service with the technical information on how to access and use the service

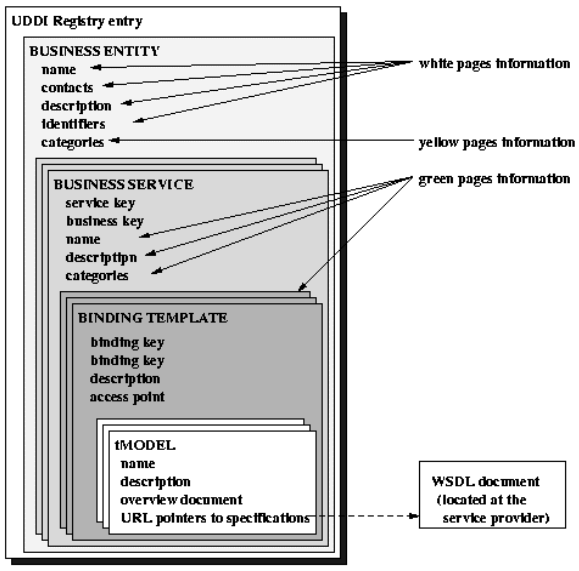
Binding template

- A binding template contains the technical information associated to a particular service. It contains the following information:
  - → bindingKey
  - → serviceKey
  - → description
  - → accessPoint: the network address of the service being provided (typically an URL but it can be anything as this field is a string: e.g., an e-mail address or even a phone)
  - → tModels: a list of entries corresponding to tModels associated with this particular binding. The list includes references to the tModels, documents describing these tModles, short descriptions, etc.
  - → categoryBag: additional information about the service and its binding (e.g., whether it is a test binding, it is on production, etc)
- A businessService can have several bindingTemplates but a binding Tenplate has only one businessService
- The binding template can be best seen as a folder where all the technical information of a service is put together

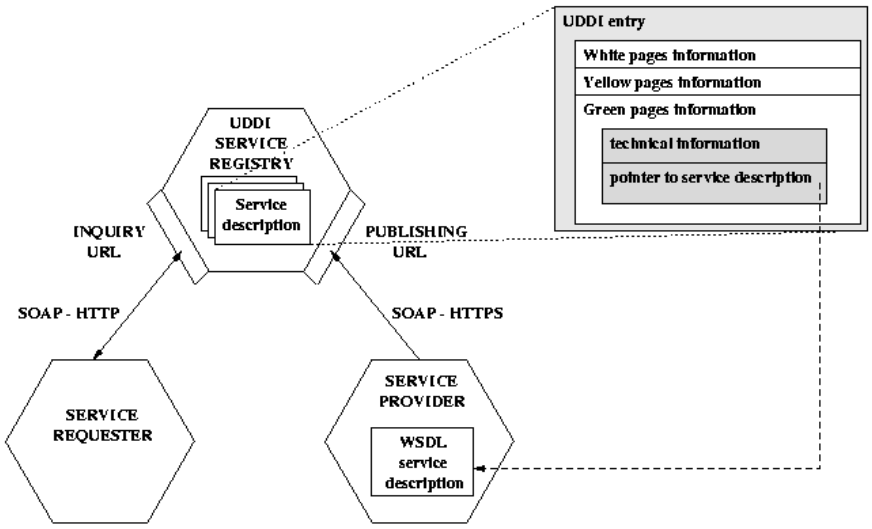
tModel

- A tModel is a generic container of information where designers can write any techical information associated to the use of a Web service:
  - → the actual interface and protocol used, including a pointer to the WSDL description
  - → description of the business protocol and conversations supported by the service
- A tModelis a document with a short description of the technical information and a pointer to the actual information. It contains:
  - → tModelKey
  - → name
  - → description
  - → overviewDoc: (with an overviewURL and useType that indicate where to find the information and its format, e.g., "text" or "wsdldescription")
  - → identifierBag
  - → categoryBag
- A tModel can point to other tModels and eventually different forms of tModels will be standardized (tModel for WSDL services, tModels for EDI based services, etc.)

Summary of the data in UDDI



UDDI and WSDL



UDDI interfaces

- The UDDI specification provides a number of Application Program Interfaces (APIs) that provide access to an UDDI system:
  - → UDDI Inquiry: to locate and find details about entries in an UDDI registry. Support a number of patterns (browsing, drill-down, invocation)
  - → UDDI Publication: to publish and modify information in an UDDI registry. All operations in this API are atomic in the transactional sense
  - → UDDI Security: for access control to the UDDI registry (token based)
  - → UDDI Subscription: allows clients to subscribe to changes to information in the UDDI registry (the changes can be scoped in the subscription request)
  - → UDDI Replication: how to perform replication of information across nodes in an UDDI registry
  - → UDDI Custody and Ownership transfer: to change the owner (publisher) of information and ship custody from one node to another within an UDI registry
- UDDI also provides a set of APIs for clients of an UDDI system:
  - → UDDI Subscription Listener: the client side of the subscription API
  - → UDDI Value Set: used to validate the information provided to an UDDI registry

SOAP and UDDI

- Access to an UDDI registry typically takes place through SOAP messages that are used to invoke the corresponding API
- The implicit assumption is that the APIs behave like RPC and SOAP is used accordingly
- UDDI registries ignore headers, if a message arrives with a mustUnderstand header set to 1, a SOAP fault is generated
- UDDI registries also ignore actor and use a generic SOAP fault message

```
POST /someVerbHere HTTP/1.1
    Host: www.somenode.org
    Content-Type: text/xml; charset="utf-8"
    Content-Length: nnnn
    SOAPAction: "get_bindingDetail"

    <?xml version="1.0" encoding="UTF-8" ?>
    <Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Body>
      <get_bindingDetail xmlns="urn:uddi-org:api_v3">
    ...
```

UDDI Version 3.0 Specification, 19 July 2002

Summary UDDI

- The UDDI specification is rather complete and encompasses many aspects of an UDDI registry from its use to its distribution across several nodes and the consistency of the data in a distributed registry
- Most UDDI registries are private and typically serve as the source of documentation for integration efforts based on Web services
- UDDI registries are not necessarily intended as the final repository of the information pertaining Web services. Even in the "universal" version of the repository, the idea is to standardize basic functions and then built proprietary tools that exploit the basic repository. That way it is possible to both tailor the design and maintain the necessary compatibility across repositories
- While being the most visible part of the efforts around Web services, UDDI is perhaps the least critical due to the complexities of B2B interactions (establishing trust, contracts, legal constrains and procedures, etc.) . The ultimate goal is, of course, full automation, but until that happens a long list of problems need to be resolved and much more standardization is necessary.