A Research Perspective on Web Services

Gustavo Alonso
Swiss Federal Institute of Technology, ETH

Christoph Bussler
Digital Enterprise Research Institute (DERI)

Tutorial EDBT 2004, Heraklion, Greece

Contents

I.
Introduction to Web Services

Contents

SOAP

What is SOAP?

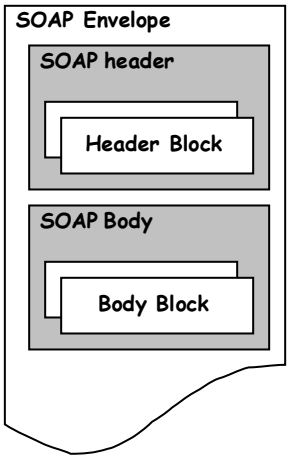
- The W3C started working on SOAP in 1999. The current W3C recommendation is Version 1.2
- SOAP covers the following four main areas:
  - A message format for one-way communication describing how a message can be packed into an XML document
  - A description of how a SOAP message (or the XML document 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

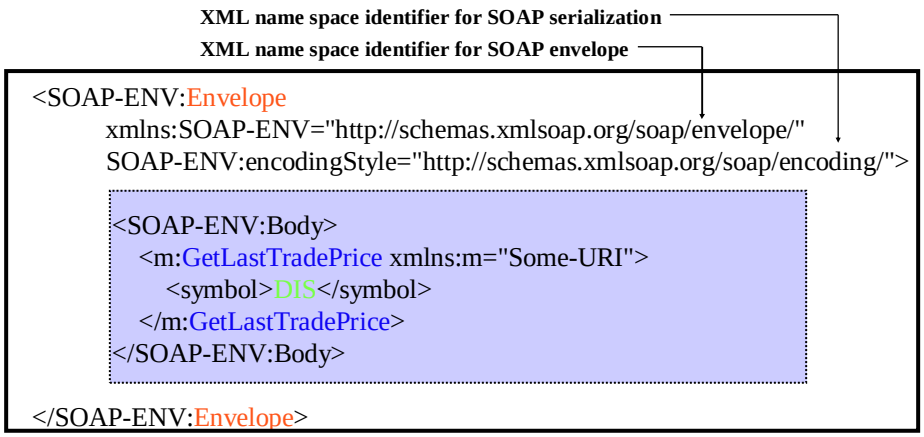
- SOAP was originally conceived as the minimal possible infrastructure necessary to perform RPC through the Internet:
  - use of XML as intermediate representation between systems
  - very simple message structure
  - mapping to HTTP for tunneling through firewalls and using the Web infrastructure
-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

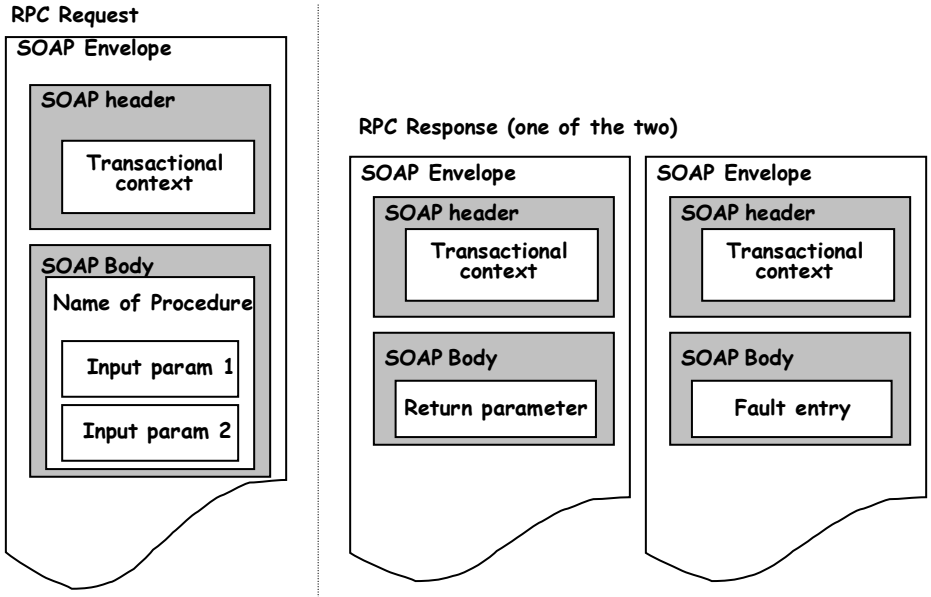
SOAP Fault element (v 1.2)

- 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

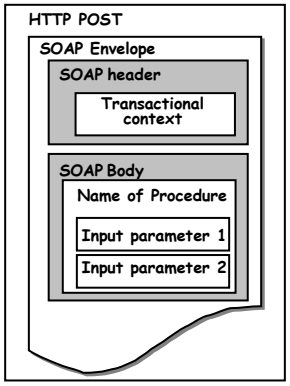
- SOAP specifies in detail how messages must be processed (in particular, how header entries must be processed)
  - Each SOAP node along the message path looks at the role associated with each part of the message
  - There are three standard roles: none, next, or ultimateReceiver
  - Applications can 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



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: "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>
```

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

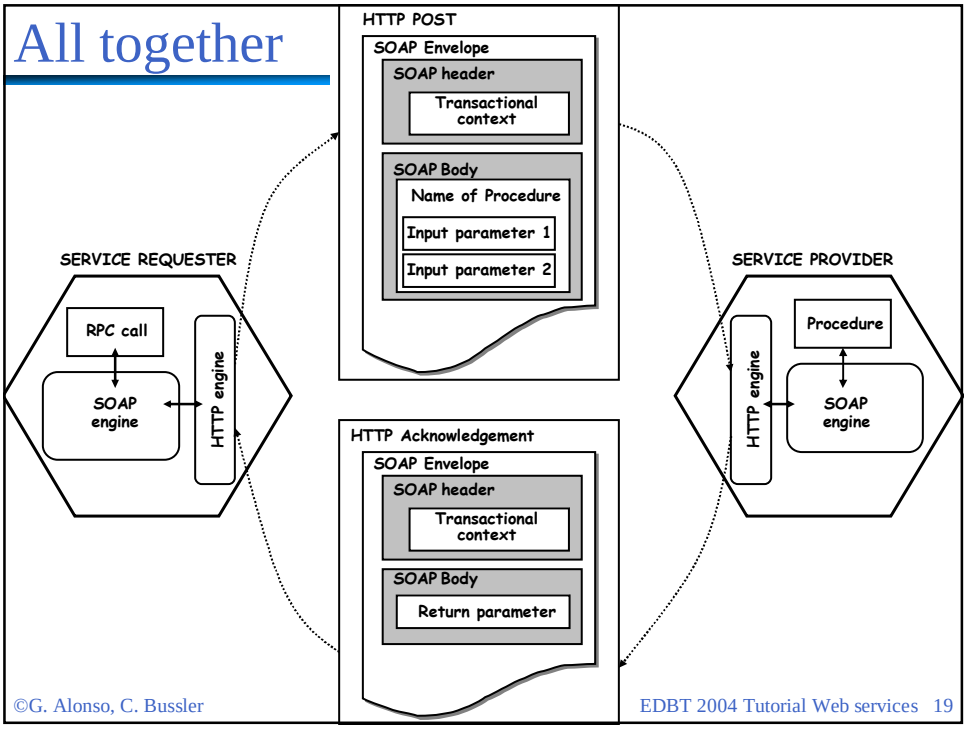
In XML (the response)

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>
```

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

All together



SOAP summary

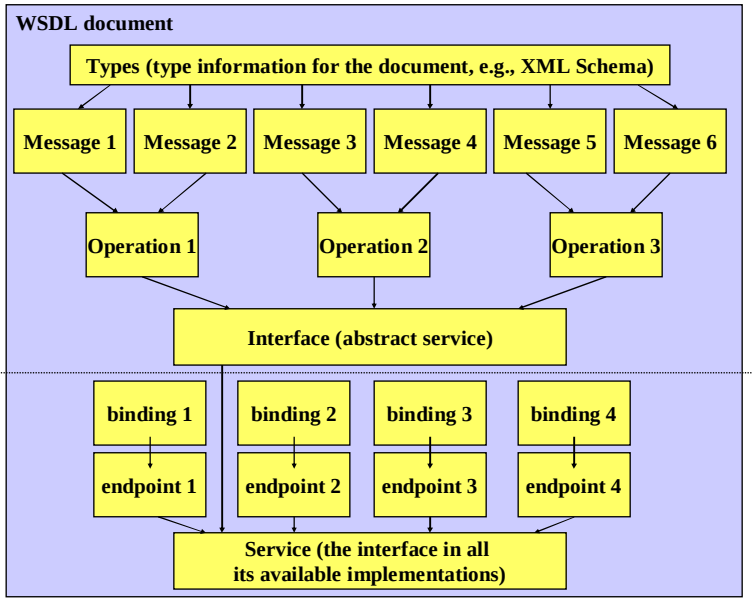
- SOAP, in its current form, provides a basic mechanism for:
  - encapsulating messages into an XML document
  - mapping the XML document with the SOAP message into an HTTP request
  - transforming RPC calls into SOAP messages
  - simple rules on how to process a SOAP message (rules became more precise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

WSDL

What is WSDL?

- The Web Services Description Language specification is in working draft 2.0 (November 2003)
- WSDL 1.1 discusses how to describe the different parts that comprise a Web service:
  - Abstract description
    - the type system used to describe the messages (based on XML Schema)
    - the messages involved in invoking the service
    - the individual operations composed of different message exchange patterns
    - an interface that groups the operations that constitute an abstract service
  - Concrete description
    - binding the interface to a transport protocol
    - the endpoint or network address of the binding
    - a service as a collection of all bindings of the same interface

Elements of WSDL



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

Messages and Faults

- Called “message reference component”, it contains three elements:
  - message reference: indicating the message pattern used for this message
  - direction: whether it is an inbound or outbound message
  - message: the actual contents of the message expressed in terms of the types previously defined
- 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).
- If a SOAP binding is used, a WSDL message element is meant to match 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)
- Called the “fault reference component”, it contains:
  - a name
  - message reference: the message to which the fault refers to
  - direction: whether the fault is inbound or outbound
  - message: the actual contents

Operations

- 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
- An operation has:
  - name
  - message exchange pattern
  - message references: the messages involved
  - fault references: the faults involved
  - style: RPC, set-attribute or get-attribute
  - features and properties
- Style:
  - RPC = implies interactions mirroring the behavior of RPC
  - set- and get- attribute = implies interactions of the type commonly found in object oriented languages
- Features and properties:
  - are used to specified characteristics of the message exchange implied by an operation. Examples include reliability, security, routing, etc

Message exchange patterns

- IN-ONLY
  - a single incoming message (A) with no faults
- ROBUST IN-ONLY
  - an inbound message (A) that might trigger a fault message
- IN-OUT
  - An incoming message (A) received from node N
  - An outgoing message (B) sent to node N
  - Faults, if any, replace message B
- IN-MULTI-OUT
  - Like IN-OUT but with zero or more outbound messages and “fault replaces message” behavior
- OUT-ONLY
  - An outbound message (A) that expects no faults
- ROBUST OUT-ONLY
  - An outbound message (A) that might trigger a fault
- OUT-IN
  - An outbound message (A) to node N
  - An inbound message (B) from node N
  - Faults, if any, replace message B
- ASYNCHRONOUS OUT-IN
  - Like OUT-IN but with trigger behavior for messages
- OUT-MULTI-IN
  - reverse of IN-MULTI-OUT

Interfaces

- An interface defines the messages a service sends or receives by grouping the messages into operations
- An interface can extend the operations of other interfaces (inheritance)
- An interface has:
  - name
  - extended interfaces: other interfaces that this one extends
  - style default: default style for operations
  - operations
  - features and properties
- An interface corresponds to the abstract description of the Web service, it does not contain any information about where the service resides or what protocols are used to invoke the Web service

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, endpoints, and services

- A binding describes a concrete message format and transmission protocol for a given endpoint
- A binding can be generic or refer to a concrete interface
- A binding can be defined for an entire interface or on an operation basis
- A binding has:
  - name
  - interface: the interface to which this binding applies
  - operations: a set of binding operation components
  - features and properties
- A binding operation component specifies the binding for a given operation:
  - name: the operation for which the binding applies
  - message references
  - fault references
- The binding operation component contains message and fault bindings for all messages and faults of an operation
- An endpoint associates an address to a given binding
- A service groups together all the endpoints for a given interface
- The specification includes bindings for HHTP, SOAP and MIME (the latter may eventually be dropped)

WSDL summary

- WSDL 2.0 provides a mechanism to define the interface to Web services in terms of messages exchanged with that Web service
  - it allows for several forms of interaction (single message, request-response)
  - it allows for several bindings (several implementations of the same interface)
- WSDL plays a similar role as Interface Definition Languages in conventional middleware platforms:
  - describe a service
  - can be used to automatically generate code to invoke the service
  - can be used by the infrastructure to enforce well formed interactions
- Like other IDLs, WSDL does not contain information about
  - semantics
  - business protocols and conversations

UDDI

What is UDDI?

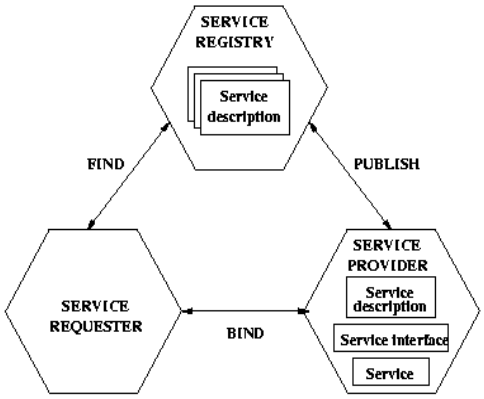
- The UDDI specification is probably the one that has evolved the most from all specifications we have seen so far. The latest version is version 3 (July 2002):
  - version 1 defined the basis for a business service registry
  - version 2 adapted the working of the registry to SOAP and WSDL
  - version 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

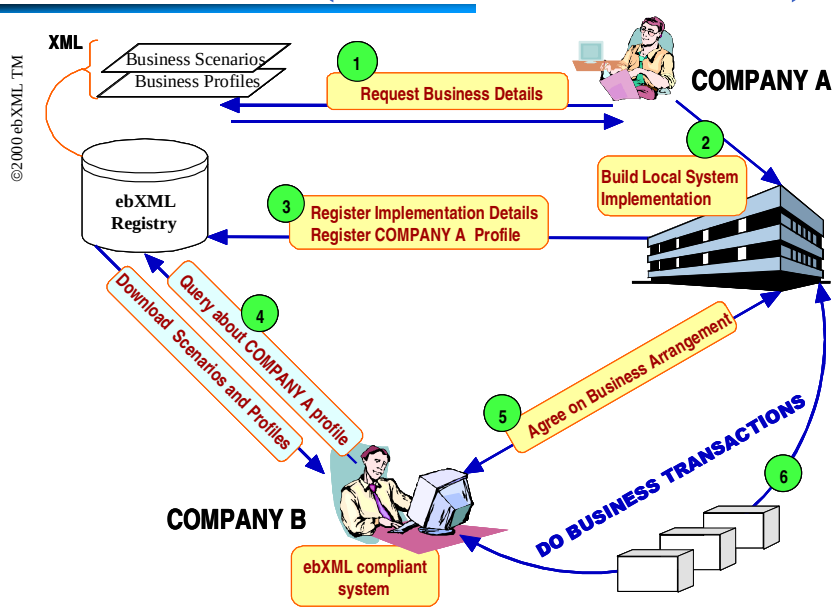
- There are a few universal UDDI registries in operation (maintained by IBM, Microsoft, SAP, etc)
- These registries are very visible and often the first thing one sees of Web services
- Unfortunately, these registries are still very small and most of the entries in them do not work or do not correspond 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

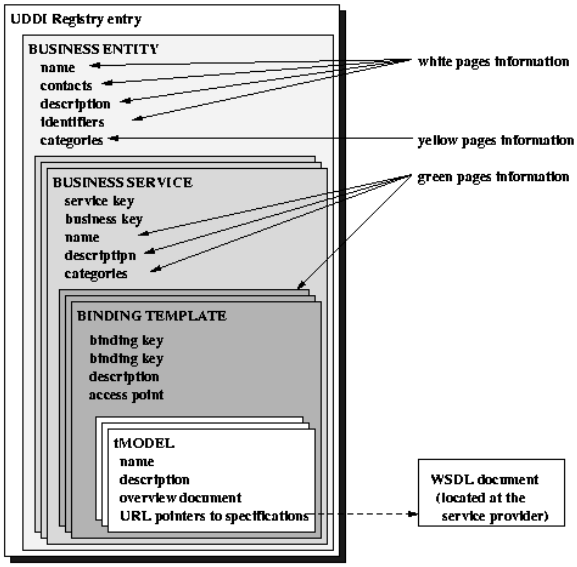
Binding template

- 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 Template 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

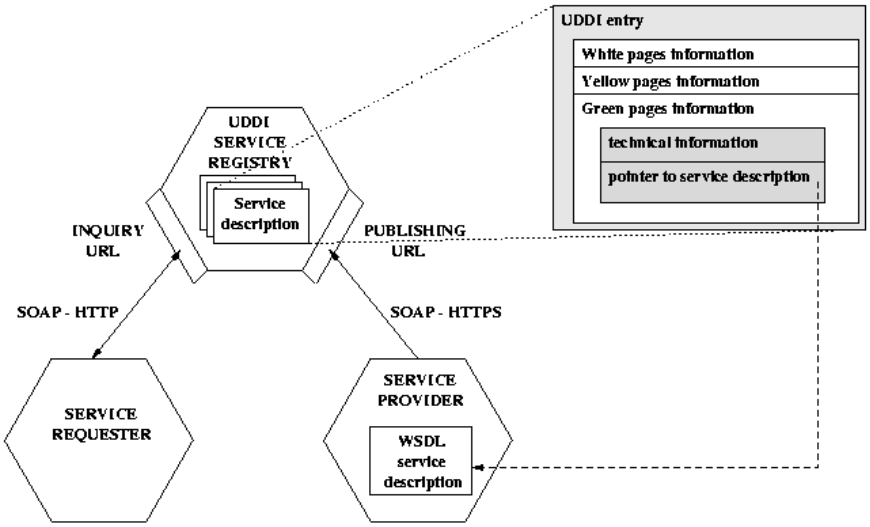
tModel

- 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.

Extensions to SOAP

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.

From SOAP Version 1.2 Part 0: Primer.


```
<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>
```

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

From SOAP Messages with Attachments. © W3C Note 11 December 2000

```
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

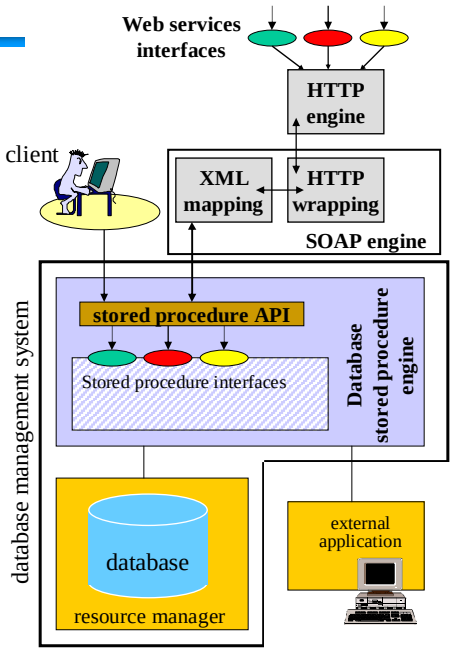
The problems with attachments

- 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)

Common usage patterns

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



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

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

Automatic conversion RPC - SOAP