5 From RPC/RMI to Service Oriented Architectures (SOA) SOAP

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
onsono@inf.ethz.ch
http://www.iks.inf.ethz.ch/
The Web as software layer (N-tier)

- N-tier architectures result from connecting several three tier systems to each other and/or by adding an additional layer to allow clients to access the system through a Web server.

- The Web layer was initially external to the system (a true additional layer); today, it is slowly being incorporated into a presentation layer that resides on the server side (part of the middleware infrastructure in a three tier system, or part of the server directly in a two tier system).

- The addition of the Web layer led to the notion of “application servers”, which was used to refer to middleware platforms supporting access through the Web.
The earliest implementations were very simple and built directly upon the existing systems (client/server based on RPC, TP-Monitors, or any other form of middleware which allowed interaction through a programmable client)

- the CGI script (or program) acted as client in the traditional sense (for instance using RPC)
- the user clicked in a given URL and the server invoked the corresponding script
- the script executed, produced the results and passed them back to the server (usually as the address of a web page)
- the server retrieved the page and send it to the browser
Applets and clients

- The problem of using a web browser as a universal client is that it does not do much beyond displaying data (it is a thin client):
  - multiple interactions are needed to complete complex operations
  - the same operations must be done over and over again for all clients
  - the processing power at the client is not used

- By adding a JVM (Java Virtual Machine) to the browser, now it becomes possible to dynamically download the client functionality (an applet) every time it is needed

- The client becomes truly independent of the operating system and is always under the control of the server
Web server as a client of a EAI system

- CGI scripts were initially widely used as there was no other way of connecting the web server with the IT system so that it could do something beyond sending static documents.

- However, CGI scripts have several problems that are not easy to solve:
  - CGI scripts are separate processes, requiring additional context switches when a call is made (and thereby adding to the overall delay).
  - Fast-CGI allows calls to be made to a single running process but it still requires two context switches.
  - CGI is really a quick hack not designed for performance, security, scalability, etc.
Servlets

- Servlets fulfill the same role as CGI scripts: they provide a way to invoke a program in response to an http request.

- However:
  - Servlets run as threads of the Java server process (not necessarily the web server) not as separate OS processes
  - unlike CGI scripts, that can be written in any language, Servlets are always written in Java (and are, therefore, portable)
  - can use all the mechanisms provided by the JVM for security purposes
Servlets and HTML

**HTML request includes**

```html
<SERVLET NAME=MyServlet>
    <PARAM NAME=param1 VALUE=val1>
    <PARAM NAME=param2 VALUE=val2>
    ...
</SERVLET>
```

**Servlet code**

```java
import java.servlet.*;
public class MyServlet extends GenericServlet {
    public void service (ServletRequest request, ServletResponse response) throws ServletException, IOException {
        ...
    }
}
```
Just one more layer ...

SALES POINT CLIENT
IF no_customer_# THEN New_customer
ELSE Lookup_customer
Check_inventory
IF enough_supplies THEN Place_order
ELSE ...

INVENTORY CONTROL
CLIENT
Lookup_product
Check_inventory
IF supplies_low THEN Place_order
Update_inventory
...

Server 1
New_customer
Lookup_customer
Delete_customer
Update_customer

Server 2
New_product
Lookup_product
Delete_product
Update_product

Server 3
Place_order
Cancel_order
Update_inventory
Check_inventory

RPC based system
Customer database

DBMS

DBMS

DBMS

©Gustavo Alonso, D-INFK. ETH Zürich.
... on top of existing systems

![Diagram](image)

- CGI script calls
- Yearly balance
- Monthly average revenue
- Front end
- TP-Monitor environment
- Control (load balancing, cc and rec., replication, distribution, scheduling, priorities, monitoring …)
- Recoverable queue
- App server 3
- Wrappers
- Branch 1
- Branch 2
- Finance Dept.
Business to Business (B2B)
Limitations of the WWW

- HTTP was originally designed as a document exchange protocol (request a document, get the document, display the document). It lacked support for client side parameters.

- Its architecture was originally designed with human users in mind. The document format (HTML) was designed to cope with GUI problems not with semantics. In EAI, the goal is almost always to remove humans from the business processes (mostly to reduce costs and to speed the process up). Strict formatting rules and tagging are key to exchanging messages across heterogeneous systems.

- Interaction through document exchange can be very inefficient when the two sides of the interaction are programs (documents must be created, sent, parsed on arrival, information extracted, etc.). Unfortunately, http does not directly support any other form of interaction.

- The initial WWW model was heavily biased towards the server side: the client (the browser) does not do much beyond displaying the document. For complex applications that meant:
  - much more traffic between client and server
  - high loads at the server as the number of users increases.
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 *
  ```

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

- 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
Challenges of B2B

- The basic idea behind B2B is simple and follows the client/server model. A service provided by one company can be directly invoked by a client running in another company. That way, the interactions between the companies are automated and their IT systems can directly interact with each other, thereby speeding up all transactions between both companies.

- There are many examples of B2B interactions. The most basic one is a “purchase order” whereby a company directly places an order with another company. If done correctly, even this basic interaction can become a very powerful advantage for a company.

- The problem is how to implement such a system:
  - the client is no longer near the server
  - joint development of client and server makes no sense
  - the server and client are likely to be hidden behind firewalls
  - the interaction takes place among existing systems, it is not possible to homogenize the supporting platforms
  - the Internet is cheap but open to everybody (unlike leased lines that are expensive but private)

- Existing systems/protocols are not really designed for such type of interactions
Contents and presentation

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

- HTML is based on SGML, that is, the tags and the grammar used in HTML documents have been defined using SGML.

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

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

<!ELEMENT trees (tree+)>
<!ELEMENT tree (branch,branch,branch?,length?)>
<!ELEMENT branch (node,length?)>
<!ELEMENT node ((branch,branch)|specie)>
<!ELEMENT length (#PCDATA)>
<!ELEMENT specie (#PCDATA)>

<?xml version="1.0" ?>
<!DOCTYPE trees SYSTEM "treefile.dtd">
<trees>
  <tree>
    <branch>
      <node>
        <specie>
          'Mouse'
        </specie>
        <length>0.792449</length>
      </node>
      <branch>
        <node>
          <branch>
            <node>
              <branch>
                <node>
                  <specie>
                    'Human'
                  </specie>
                  <length>0.105614</length>
                </node>
                <node>
                  <specie>
                    'Chimp'
                  </specie>
                  <length>0.171597</length>
                </node>
                <length>0.074558</length>
              </branch>
              <branch>
                <node>
                  <specie>
                    'Gorilla'
                  </specie>
                  <length>0.152701</length>
                </node>
                <length>0.048980</length>
              </branch>
              <branch>
                <node>
                  <specie>
                    'Orang'
                  </specie>
                  <length>0.303652</length>
                </node>
                <length>0.121196</length>
              </branch>
            </node>
            <branch>
              <node>
                <specie>
                  'Gibbon'
                </specie>
                <length>0.336296</length>
              </node>
            </branch>
          </branch>
          <length>0.485445</length>
        </node>
      </branch>
      <branch>
        <node>
          <specie>
            'Bovine'
          </specie>
          <length>0.902183</length>
        </node>
      </branch>
    </node>
  </branch>
</tree>
</trees>

Data to send

DTD File

XML File
DTDs and documents

- The goal of XML is to provide a standardized way to specify data structures so that when data is exchanged, it is possible to understand what has been sent.

- The Document Type Definition (DTD) specifies how the data structure is described: processing instructions, declarations, comments, and elements.

- Using the DTD, the XML document can be correctly interpreted by a program by simply parsing the document using the grammar provided by the DTD.

- The idea is similar to IDL except that instead of defining parameters as combinations of standard types, a DTD describes arbitrary documents as semi-structured data.

- Using XML is possible to exchange data through HTTP and Web servers and process the data automatically.

- Note that the use of XML reduces the universality of the browser since now a browser needs additional programs to deal with specific markup languages developed using XML (somewhat similar to plug-ins but more encompassing in terms of functionality).

- However, this is not much of a problem since the browser is for humans while XML is for automated processing.

- XML can be used as the intermediate language for marshalling/serializing arguments when invoking services across the Internet.
Web services

This could be RPC, CORBA, DCOM, using SOAP as protocol

CLIENT

SERVER

call

service

stubs, runtime
location

stubs, runtime
adapters

INTERNET

HTTP support (web client)

HTTP support (web server)

SOAP system

Serialized XML doc

Wrap doc in HTTP POST request

Retrieve doc from HTTP response

Serialized XML doc

Web services

HTTP support (web client)

HTTP support (web server)

This could be RPC, CORBA, DCOM, using SOAP as protocol
Web Services Architecture

- A popular interpretation of Web services is based on IBM’s *Web service architecture* based on three elements:

  1. Service **requester**: The potential user of a service (the client)
  2. Service **provider**: The entity that implements the service and offers to carry it out on behalf of the requester (the server)
  3. Service **registry**: A place where available services are listed and that allows providers to advertise their services and requesters to lookup and query for services
Main Web Services Standards

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

- The service bus can be seen as a refactoring of the basic Web service architecture, where a higher degree of loose coupling has been added.
Benefits of Web services

- One important difference with conventional middleware is related to the standardization efforts at the W3C that should guarantee:
  - Platform independence (Hardware, Operating System)
  - Reuse of existing networking infrastructure (HTTP has become ubiquitous)
  - Programming language neutrality (.NET talks with Java, and vice versa)
  - Portability across Middleware tools of different Vendors
  - Web services are “loosely coupled” components that foster software reuse
  - WS technologies should be composable so that they can be adopted incrementally
<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport</td>
<td>HTTP, IIOP, SMTP, JMS</td>
</tr>
<tr>
<td>Messaging</td>
<td>XML, <strong>SOAP</strong></td>
</tr>
<tr>
<td>Description</td>
<td>XML Schema, <strong>WSDL</strong></td>
</tr>
<tr>
<td>Discovery</td>
<td><strong>UDDI</strong></td>
</tr>
<tr>
<td>Choreography</td>
<td><strong>WSCL</strong></td>
</tr>
<tr>
<td>Business Processes</td>
<td><strong>WS-BPEL</strong></td>
</tr>
<tr>
<td>Stateful Resources</td>
<td><strong>WS-Resource Framework</strong></td>
</tr>
<tr>
<td>Transactions</td>
<td><strong>WS-CAF</strong></td>
</tr>
<tr>
<td>Reliable Messaging</td>
<td><strong>WS-Reliability</strong></td>
</tr>
<tr>
<td>Security</td>
<td><strong>WS-Security</strong></td>
</tr>
<tr>
<td>Event Notification</td>
<td><strong>WS-Notification</strong></td>
</tr>
<tr>
<td>Management</td>
<td><strong>WSDM</strong></td>
</tr>
<tr>
<td>Data Access</td>
<td><strong>OGSA-DAI</strong></td>
</tr>
</tbody>
</table>

**WS-Standards and Specifications**

©Gustavo Alonso, D-INFK. ETH Zürich.
What is SOA

- SOA = Services Oriented Architecture
  - **Services** = another name for large scale components wrapped behind a standard interface (Web services although not only)
  - **Architecture** = SOA is intended as a way to build applications and follows on previous ideas such as software bus, IT backbone, or enterprise bus

- The part that it is not in the name
  - **Loosely-coupled** = the services are independent of each other, heterogeneous, distributed
  - **Message based** = interaction is through message exchanges rather than through direct calls (unlike Web services, CORBA, RPC, etc.)
The novelty behind SOA

- The concept of SOA is not new:
  - Message oriented middleware
  - Message brokers
  - Event based architectures

- The current context is different
  - Emergence of standard interfaces (Web services)
  - Emphasis on simplifying development (automatic)
  - Use of complex underlying infrastructure (containers, middleware stacks, etc.)

- Interest in SOA arises from a number of reasons:
  - Basic technology in place
  - More clear understanding of distributed applications
  - The key problem is integration not programming
The need for SOA

- Most companies today have a large, heterogeneous IT infrastructure that:
  - Keeps changing
  - Needs to evolve to adopt new technology
  - Needs to be connected to that of commercial partners
  - Needs to support an increasing amount of purposes and goals

- This was the field of Enterprise Application Integration using systems like CORBA or DCOM. However, solutions until now suffered from:
  - Tightly integrated systems
  - Vendor lock-in (e.g., vendor stacks)
  - Technology lock-in (e.g., CORBA)
  - Lack of flexibility and limitations when new technology arises (e.g., Internet)

- SOA is an attempt to build on standards (web services) to reduce the cost of integration

- It introduces very interesting possibilities:
  - Development by composition
  - Large scale reuse
  - Frees developers from “lock-in” effects of various kinds
SOA vs. Web services

- Web services are about
  - Interoperability
  - Standardization
  - Integration across heterogeneous, distributed systems

- Service Oriented Architectures are about:
  - Large scale software design
  - Software Engineering
  - Architecture of distributed systems

- SOA is possible but more difficult without Web services
- SOA introduces some radical changes to software:
  - Language independence (what matters is the interface)
  - Event based interaction (no longer synchronous models)
  - Message based exchanges (no RPC)
  - Composition and orchestration
SOA and web services

- WS Invocation Framework
  - Use WSDL to describe a service
  - Use WSIF to let the system decide what to do when the service is invoked:
    - If the call is to a local EJB then do nothing
    - If the call is to a remote EJB then use RMI
    - If the call is to a queue then use JMS
    - If the call is to a remote Web service then use SOAP and XML
  - There is a single interface description, the system decides on the binding
  - This type of functionality is at the core of the notion of Service Oriented Architecture

There is no problem in system design that cannot be solved by adding a level of indirection.

There is no performance problem that cannot be solved by removing a level of indirection.

Take advantage of Middleware but let the system decide what to use
2 SOAP

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/
Contents – SOAP

- Background
- SOAP overview
- Structure of a SOAP Message
- Processing SOAP Messages
- Mapping SOAP to a transport protocol
Background and historical perspective
Basic Problems to solve

1. How to make the service invocation part of the language in a more or less transparent manner.
   - Don’t forget this important aspect: whatever you design, others will have to program and use

2. How to exchange data between machines that might use different representations for different data types. This involves two aspects:
   - data type formats (e.g., byte orders in different architectures)
   - data structures (need to be flattened and the reconstructed)

3. How to find the service one actually wants among a potentially large collection of services and servers.
   - The goal is that the client does not necessarily need to know where the server resides or even which server provides the service.

4. How to deal with errors in the service invocation in a more or less elegant manner:
   - server is down,
   - communication is down,
   - server busy,
   - duplicated requests ...
Remote calls in RPC/DCE

Diagram showing the process of remote calls in RPC/DCE, including application client procedure, IDL source, IDL compiler, interface header, application server procedure, language-specific call interface, client stubs, server stubs, RPC run-time service library, RPC protocols, DCE directory services, and DCE security services.
Remote calls in CORBA

Identifying and locating services

Marshalling and serializing arguments

Client stub
CORBA runtime

Interface repository

Implementation repository

Skeleton Object adapter

Client

Service (server)

Client stub
CORBA runtime

ORB

ORB

TCP/IP socket

TCP/IP socket

Local Area Network

MIDDLEWARE

ORB

ORB

MIDDLEWARE
Remote calls in DCOM

Identifying and locating services

Marshalling and serializing arguments

CM = Service Control Manager

©Gustavo Alonso, D-INFK. ETH Zürich.
Wire-protocols, XML and SOAP

- RPC, CORBA, DCOM, even Java, use different mechanisms and protocols for communicating. All of them map to TCP or UDP one way or another but use different syntax for marshalling, serializing and packaging messages.

- The problem is that these mechanisms are a legacy from the time when communications were mostly within LANs and within homogeneous systems.

- Building a B2B environment combining the systems of different companies becomes difficult because the protocols available in RPC, CORBA, or DCOM are too low level and certainly not compatible among each other (gateways are needed, etc.).

- To address this problem, XML was used to define SOAP.

- SOAP is conceptually quite simple:
  - RPC using HTTP
  - (at the client) turn an RPC call into an XML document
  - (at the server) turn the XML document into a procedure call
  - (at the server) turn the procedure’s response into an XML document
  - (at the client) turn the XML document into the response to the RPC
  - use XML to serialize the arguments following the SOAP specification.
The background for SOAP

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

- The idea was to avoid the problems associated with CORBA’s IIOP/GIOP (which fulfilled a similar role but using a non-standard intermediate representation and had to be tunneled through HTTP anyway)

- The goal was to have an extension that could be easily plugged on top of existing middleware platforms to allow them to interact through the Internet rather than through a LAN as in the original case. Hence the emphasis on RPC from the very beginning (essentially all forms of middleware use RPC at one level or another)

- Eventually SOAP started to be presented as a generic vehicle for computer driven message exchanges through the Internet and then it was opened to support interactions other than RPC and protocols other than HTTP.
SOAP as RPC mechanism

This could be RPC, CORBA, DCOM, using SOAP as protocol

CLIENT

call

stubs, runtime service location

SOAP system

Serialized XML doc

Wrap doc in HTTP POST request

HTTP support (web client)

INTERNET

SERVER

service

stubs, runtime adapters

SOAP system

Serialized XML doc

Retrieve doc from HTTP response

HTTP support (web server)
SOAP
What is SOAP?

- The W3C started working on SOAP in 1999. The current W3C recommendation is Version 1.2
- Originally: Simple Object Access Protocol
- SOAP covers the following main areas:
  - **Message construct**: A message format for one-way communication describing how a message can be packed into an XML document
  - **Processing model**: rules for processing a SOAP message and a simple classification of the entities involved in processing a SOAP message. Which parts of the messages should be read by whom and how to react in case the content is not understood
  - **Extensibility Model**: How the basic message construct can be extended with application specific constructs
  - **Protocol binding framework**: Allows SOAP messages to be transported using different protocols (HTTP, SMTP, ...)
    - A concrete **binding for HTTP**
  - 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
SOAP: a messaging framework

- SOAP ≠ RPC: Since version 1.1, SOAP abstracts from the RPC programming model
- SOAP is “a lightweight protocol intended for exchanging structured information [...]”, “a stateless, one-way message exchange paradigm”
- Defines the general format of a message and how to process it
- More complex interaction patterns can be created by applications
- RPC is implemented on top of the core specification following conventions of the “SOAP RPC representation”
- SOAP ≠ HTTP: Since version 1.1, SOAP abstracts from the protocol used to transport the messages
- HTTP is one of many possible transports
A SOAP message can pass through multiple hops on the way from the initial sender to the ultimate receiver.

The entities involved in transporting the message are called SOAP nodes.

SOAP intermediaries forward the message and may manipulate it.

Every SOAP node assumes a certain role which influences the message processing at the node.
Structure of a SOAP Message
SOAP messages

- SOAP message = SOAP envelope
- Envelope contains two parts:
  - Header (optional): independent header blocks with meta data (security, transactions, session,...)
  - Body: several blocks of application data
- SOAP does not define the semantics of the header nor the body, only the structure of the message.
Skeleton SOAP message

```xml
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    ...
  </soap:Body>

  <soap:Fault>
    ...
  </soap:Fault>
</soap:Envelope>
```

From http://www.w3schools.com
The SOAP header

- The header is intended as a generic place holder for information that is not necessarily application dependent (the application may not even be aware that a header was attached to the message).

- Typical uses of the header are: coordination information, identifiers (e.g., for transactions), security information (e.g., certificates)

- SOAP provides mechanisms to specify who should deal with headers and what to do with them. For this purpose it includes:
  - **Actor** attribute: who should process that particular header block.
  - Boolean **mustUnderstand** attribute: indicates whether it is mandatory to process the header. If a header is directed at a node (as indicated by the actor attribute), the mustUnderstand attribute determines whether it is mandatory to do so.
  - SOAP 1.2 adds a **relay** attribute (forward header if not processed)
SOAP Header Example

```xml
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
 <m:Trans
 xmlns:m="http://www.w3schools.com/transaction/
 soap:mustUnderstand="1">234</m:Trans>
</soap:Header>

... ...

</soap:Envelope>
```

From http://www.w3schools.com
Example: SOAP Headers for Security

RPC Request

SOAP Envelope

SOAP header

Security context
Message
Signature

SOAP Body

Name of Procedure

Input param 1

Input param 2

RPC Response (one of the two)

SOAP Envelope

SOAP header

Security context
Message
Signature

SOAP Body

Return parameter

SOAP Envelope

SOAP header

Security context
Message
Signature

SOAP Body

Fault entry
The SOAP body

- The body is intended for the application specific data contained in the message
- A body element is equivalent to a header block with attributes actor=ultimateReceiver and mustUnderstand=1
- Unlike for header blocks, SOAP does specify the contents of some body elements:
  - mapping of RPC to a SOAP body element (RPC conventions)
  - the Fault entry (for reporting errors in processing a SOAP message)
SOAP body example

From the: Simple Object Access Protocol (SOAP) 1.1 ©W3C Note 08 May 2000
SOAP example, header and body

```xml
<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 fault

- When a SOAP message could not be processed, a SOAP fault is returned
- A fault must carry the following information:
  - Fault Code: indicating the class of error and possibly a subcode (for application specific information)
  - Fault String: human readable explanation of the fault (not intended for automated processing)
  - Fault actor: who caused the fault to happen
  - Detail: Application specific data related to the fault
- The fault codes include:
  - Version mismatch: invalid namespace in SOAP envelope
  - Must Understand: a header element with must understan set to “true” was not understood
  - Client: message was incorrect (format or content)
  - Server: problem with the server, message could not be processed
- Errors in understanding a mandatory header block are responded using a fault element but also include a special header indicating which one of the original header blocks was not understood.
Message Processing Model

- For each message received, every SOAP node on the message path must process the message as follows
  - Decide in which roles to act (standard roles: next or ultimateReceiver, or other application-defined roles). These roles may also depend on the contents of the message.
  - Identify the mandatory header blocks targeted at the node (matching role, mustUnderstand=true)
  - If a mandatory header block is not understood by the node, a fault must be generated. The message must not be processed further.
  - Process the mandatory header blocks and, in case of the ultimate receiver, the body. Other header blocks targeted at the node may be processed. The order of processing is not significant.

- SOAP intermediaries will finally forward the message
  - Processed header blocks may be removed depending on the specification for the block.
  - Header blocks which were targeted at the intermediary but not processed are relayed only if the the relay attribute is set to true.

- Active SOAP intermediaries may also change a message in ways not described here (e.g., encrypt the message).
RPC with SOAP
SOAP RPC representation

- SOAP specifies a uniform representation for RPC requests and responses which is platform independent. It does not define mappings to programming languages.
- SOAP RPC does not support advanced RPC/RMI features such as object references or distributed garbage collection. This can be added by applications or additional standards (see WSRF).
- Formally, RPC is not part of the core SOAP specification. Its use is optional.
RPC Example

- Request:
  ```xml
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
  ```

- Response:
  ```xml
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
  ```
Mapping SOAP to a transport protocol
SOAP protocol binding framework

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

<table>
<thead>
<tr>
<th>SOAP RPC</th>
<th>SOAP</th>
<th>HTTP</th>
<th>SMTP</th>
<th>TCP</th>
<th>UDP</th>
<th>IP</th>
</tr>
</thead>
</table>

©Gustavo Alonso, D-INFK. ETH Zürich.
SOAP and HTTP

- SOAP messages are typically transferred using HTTP
- The binding to HTTP defined in the SOAP specification
- SOAP can use GET or POST. With GET, the request is not a SOAP message but the response is a SOAP message, with POST both request and response are SOAP messages (in version 1.2, version 1.1 mainly considers the use of POST).
In XML (a request)

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

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

HTTP Request

SOAP Envelope

SOAP header

Transactional context

SOAP Body

Name of Procedure

Input parameter 1

Input parameter 2

HTTP Response

SOAP Envelope

SOAP header

Transactional context

SOAP Body

Return parameter

SERVICE REQUESTER

RPC call

HTTP engine

SERVICE PROVIDER

Procedure

HTTP engine

SOAP engine

HTTP engine

SOAP engine
Additional bindings (example)

- SOAP over Java Message Service 1.0 RC1:

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
3 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5   <soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
6     <postMessage><ngName xsi:type="xsd:string">news.current.events</ngName>
7     <msg xsi:type="xsd:string">This is a sample news item.</msg>
8     </postMessage>
9   </soapenv:Body>
10 </soapenv:Envelope>
```
Additional bindings

- WS Invocation Framework
  - Use WSDL to describe a service
  - Use WSIF to let the system decide what to do when the service is invoked:
    - If the call is to a local EJB then do nothing
    - If the call is to a remote EJB then use RMI
    - If the call is to a queue then use JMS
    - If the call is to a remote Web service then use SOAP and XML
  - There is a single interface description, the system decides on the binding
  - This type of functionality is at the core of the notion of Service Oriented Architecture
SOAP Attachments
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.

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

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
<SOAP-ENV:Body>

..

<theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
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 used 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 the interface of Web services (how to model attachments in WSDL?)
Practical uses of SOAP
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
**SOAP Summary**

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

- SOAP is a very simple protocol intended for transferring data from one middleware platform to another. In spite of its claims to be open (which are true), current specifications and implementations are very tied to RPC and HTTP.

- SOAP takes advantage of the standardization of XML to resolve problems of data representation and serialization (it uses XML Schema to represent data and data structures, and it also relies on XML for serializing the data for transmission). As XML becomes more powerful and additional standards around XML appear, SOAP can take advantage of them by simply indicating what schema and encoding is used as part of the SOAP message. Current schema and encoding are generic but soon there will be vertical standards implementing schemas and encoding tailored to a particular application area (e.g., the efforts around EDI).