Chapter 6: Simple Object Access Protocol (SOAP)

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

- The W3C started working on SOAP in 1999. The current W3C specification is Version 1.2 which was announced in 2007.
- 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)
  IIOP: Internet Inter-ORB Protocol; GIOP: General Inter-ORB Protocol
  ORB: Object Request Protocol

- 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 than HTTP. This process, however, is only in its very early stages.
The background for HTTP and MIME

- The HyperText Transfer Protocol (HTTP) is a network (application layer) protocol for distributed information systems.
  - HTTP is a client/server type protocol.
  - Typically, web browsers or spiders act as clients, and an application running on the computer hosting the web site acts as a server.
  - An HTTP session is a sequence of request-response transactions. An HTTP client initiates a request. Upon receiving the request, the server sends back a status line and a message containing the resource requested or an error message.

- Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of e-mail.
  - It supports character sets other than ASCII.
  - It supports non-text attachments such as image/jpeg or audio/mp3.
SOAP messages

- SOAP is based on message exchanges
- Messages are seen as envelopes 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

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

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

- The header is intended as a generic place holder for information that is not necessarily application dependent (the application may not even be aware that a header was attached to the message).
- Typical uses of the header are: coordination information, identifiers (e.g., 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 of 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

RPC Request

SOAP Envelope

SOAP header

Transactional context

SOAP Body

Name of Procedure

Input param 1

Input param 2

RPC Response (one of the two)

SOAP Envelope

SOAP header

Transactional context

SOAP Body

Return parameter

SOAP Envelope

SOAP header

Transactional context

SOAP Body

Fault entry
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"

```xml
<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

```xml
<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

HTTP POST

SOAP Envelope

SOAP header

Transactional context

SOAP Body

Name of Procedure

Input parameter 1

Input parameter 2

HTTP Acknowledgement

SOAP Envelope

SOAP header

Transactional context

SOAP Body

Return parameter

SERVICE REQUESTER

RPC call

SOAP engine

HTTP engine

SERVICE PROVIDER

Procedure

SOAP engine

HTTP engine

HTTP engine

RPC call

SOAP engine

HTTP engine
Using SOAP to extend existing services
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

RPC based middleware

client call

stubs, runtime service location

SOAP system

Serialized XML doc

Wrap doc in HTTP POST / M-POST

HTTP support

RPC based middleware

server procedure

stubs, runtime adapters

SOAP system

Serialized XML doc

Retrieve doc from HTTP packet

HTTP support

NETWORK
Limitations of SOAP
- asynchronous messaging
SOAP exchange patterns (v 1.2)

- **SOAP response**
  - 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**
  - 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 in 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 attachment
- 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 implements 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 standard. 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.
Limitations of SOAP
- attachments and binary data
SOAP Encoding

- Simple type, derived type
  - Integer
  - Floating point number
  - String

- Compound type, array
  - Constructed from simple or derived types
  - E.g., part information containing name, id, weight, and so on
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: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: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 on 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--
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 the size of the 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 Web services (discussed later on).
Limitations of SOAP
- additional functionality
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
Alternative architectures for messaging
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 ...

Abstract ebXML Messaging Service

- Messaging service layer
  (maps the abstract interface to the transport service)

- Transport service(s)

- but extended to incorporate additional features (next page)
ebXML messaging service

MESSAGING SERVICE INTERFACE

AUTHENTICATION, AUTHORIZATION AND REPUDIATION SERVICES

HEADER PROCESSING

ENCRYPTION, DIGITAL SIGNATURE

MESSAGE PACKAGING MODULE

DELIVERY MODULE
SEND/RECEIVE
TRANSPORT MAPPING AND BINDING

TRANSPORT SERVICES

FTP HTTP IIOP SMTP
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 other 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.

- These are all typical features of a communication protocol that are needed anyway (including practical SOAP implementations).
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.