Lesson 3 – SOAP message structure

Service Oriented Architectures Security

Module 1 - Basic technologies

Unit 2 – SOAP

Ernesto Damiani

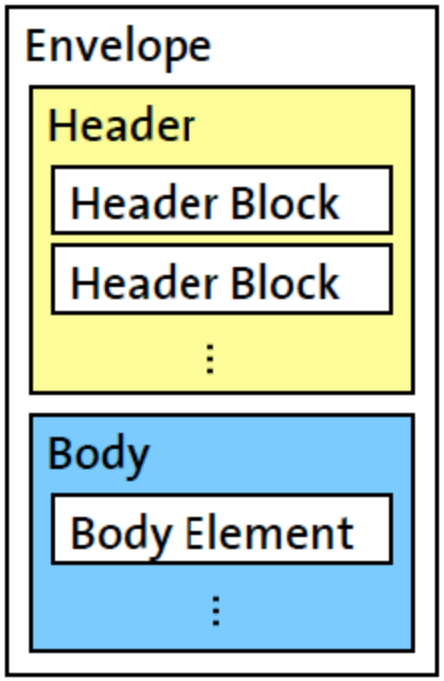
Università di Milano

SOAP structure (1)

- 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 structure (2)

- SOAP does not define the semantics of the header nor the body, but only the structure of the message



SOAP message structure

- <?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:Fault>
- ...
- ...
- </soap:Fault>
- </soap:Body>
- </soap:Envelope>

SOAP header (1)

- 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)

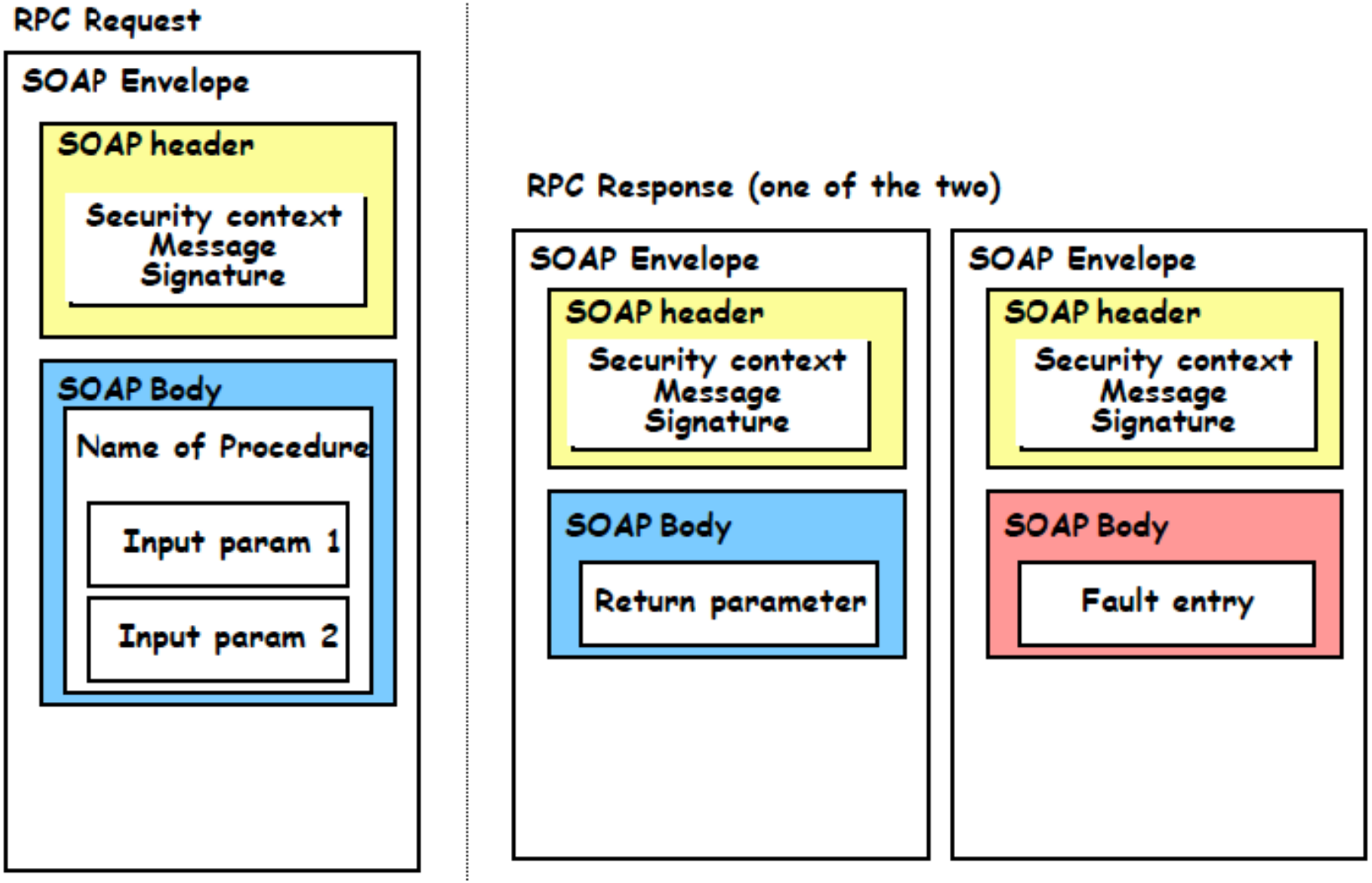
SOAP header (2)

- 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 added a relay attribute (forward header if not processed)

SOAP header example

```
<?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>
```

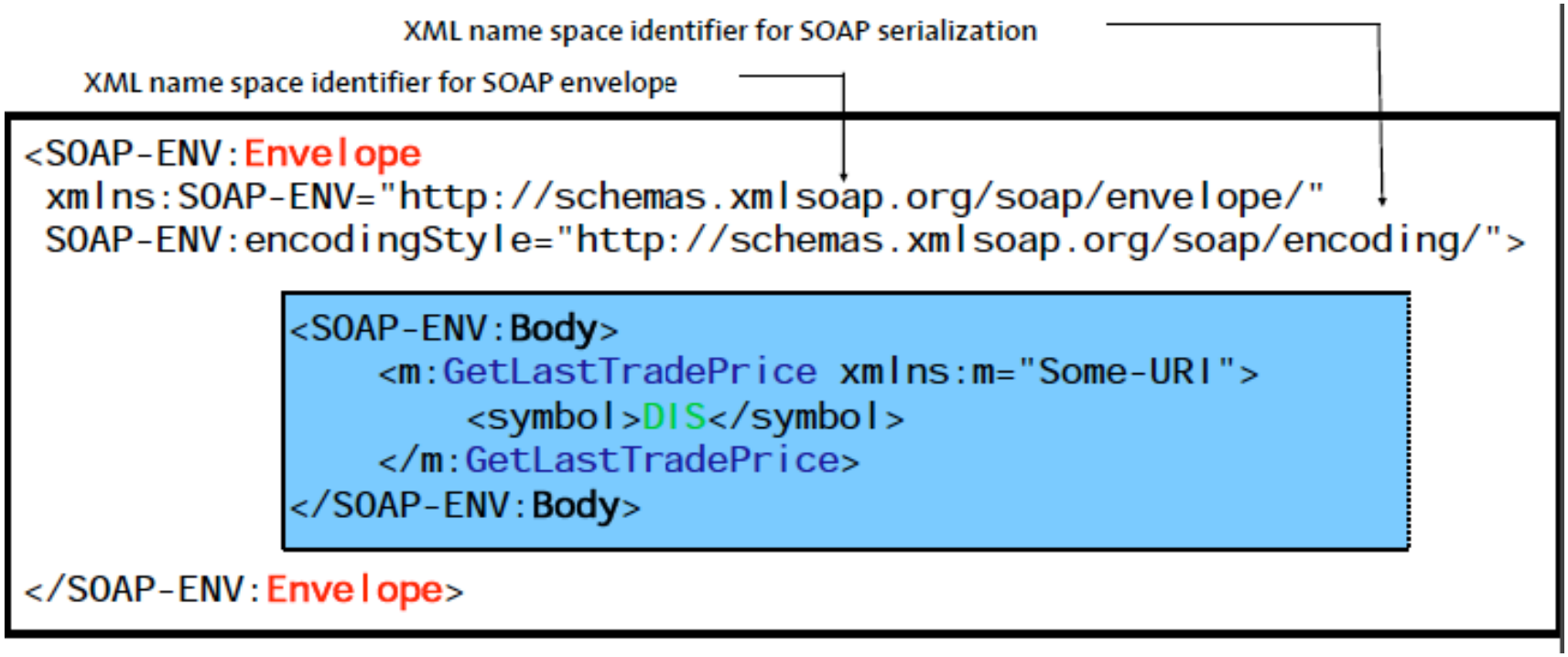
Example: security headers



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: e.g., it provides a mapping of RPC to a SOAP body element (RPC conventions)
  - The Fault entry (for reporting errors in processing a SOAP message)

Sample SOAP body



Putting it together

```
<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

Fault management (1)

- 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

Fault management (2)

- The fault codes include:
  - Version Mismatch: invalid namespace in SOAP envelope
  - Must Understand: a header element with "must understand" 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 (1)

- For each message received, every SOAP node on the message path must process the message as follows:
  1. 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

Message processing (2)

2. 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
3. Process the mandatory header blocks and, in case of the ultimate receiver, the body. Other header blocks targeted at the node maybe processed. The order of processing is not significant

- SOAP intermediaries will finally forward the message

Message processing (3)

- 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 relay attribute is set to true
- Active SOAP intermediaries may also change a message in other ways (e.g., encrypt the message)

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:

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

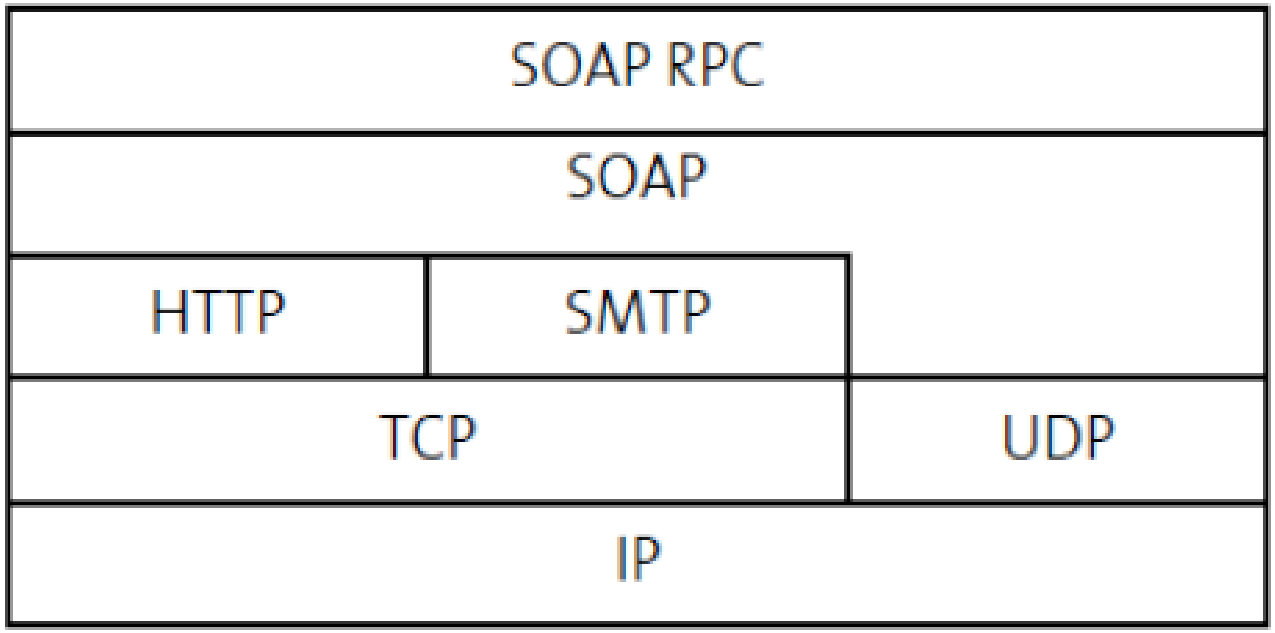
Response:

```
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
<Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
```

SOAP HTTP binding (1)

- 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
- Binding specifies how response and request messages are correlated
- The SOAP binding framework expresses guidelines for specifying a binding to a particular protocol

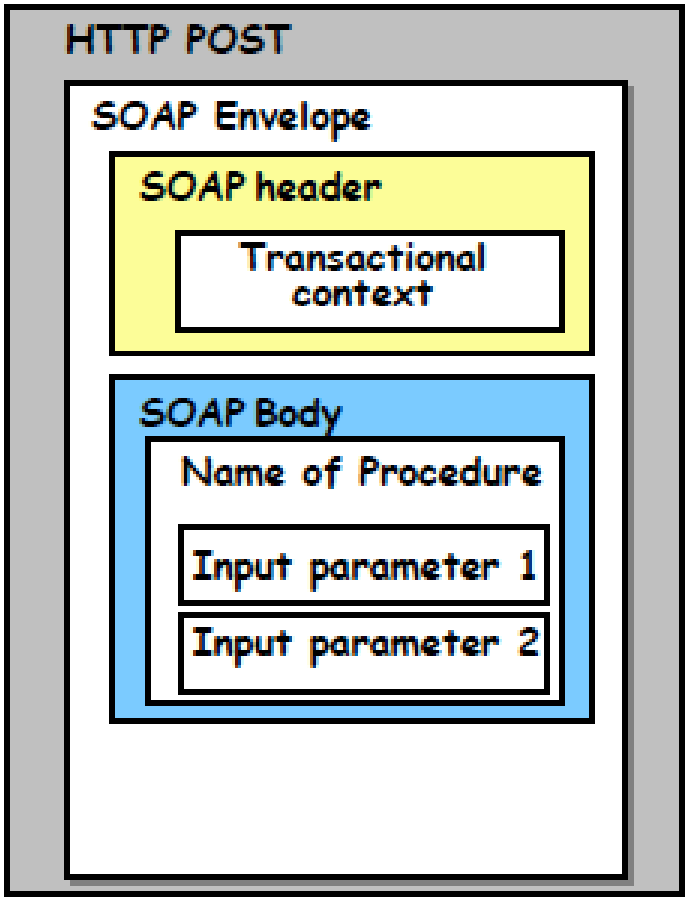
SOAP HTTP binding (2)



SOAP HTTP binding (3)

- 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)

SOAP HTTP binding (4)



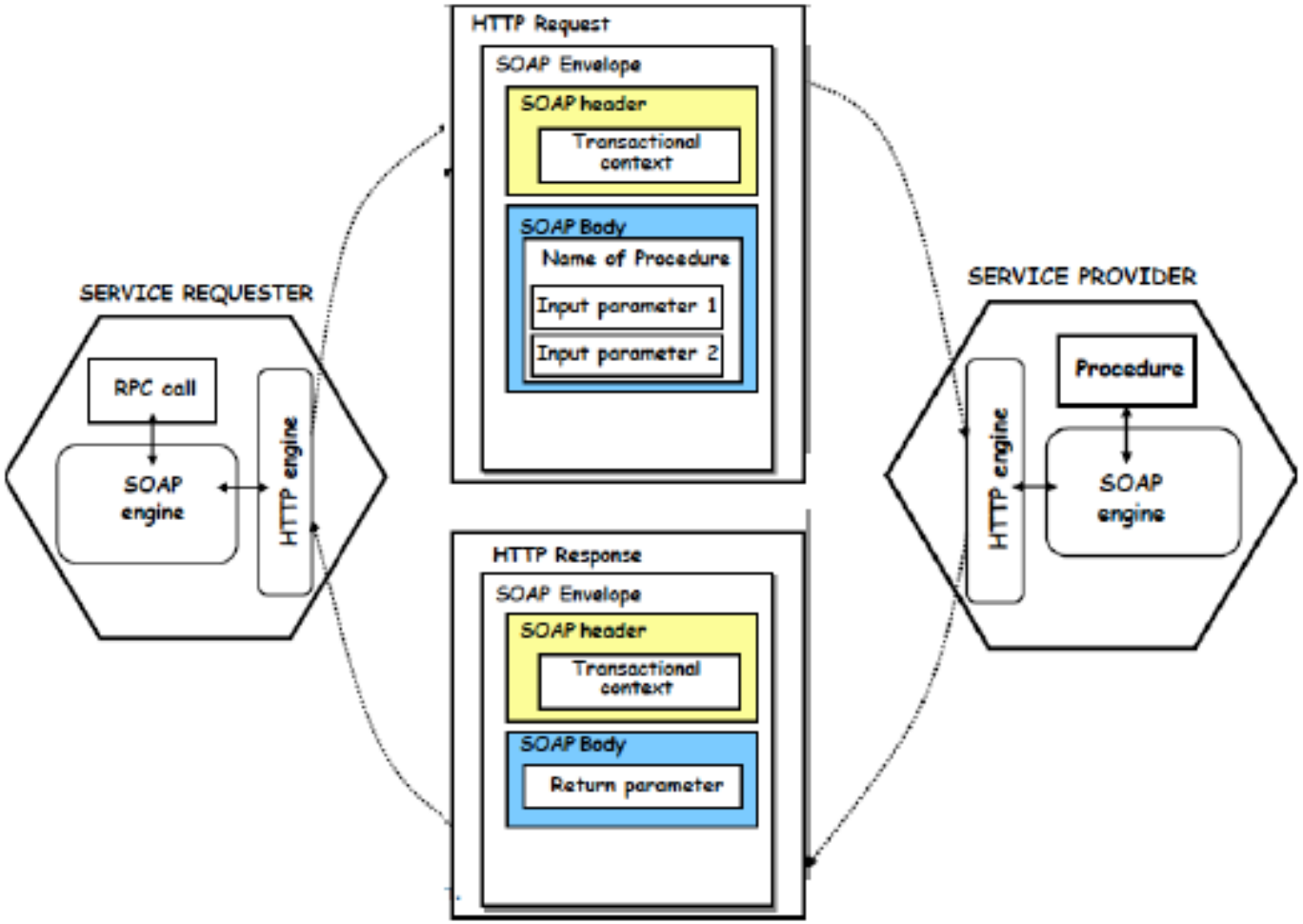
POST request example

```
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>
```

POST response example

```
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>
```

Global view


HTTP Request
SOAP Envelope
SOAP header
Transactional context
SOAP Body
Name of Procedure
Input parameter 1
Input parameter 2
SERVICE REQUESTER
RPC call
SOAP engine
HTTP engine
SERVICE PROVIDER
Procedure
HTTP engine
SOAP engine
HTTP Response
SOAP Envelope
SOAP header
Transactional context
SOAP Body
Return parameter

Other bindings

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/e
    ncoding/">
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>
```

WS Invocation Framework

- 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 (1)

- 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

SOAP attachments (2)

```
<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:seatPreference/>
   </p:return>
  </p:itinerary>
</env:Body>
```

SOAP attachment problem (1)

- This approach reflects the implicit assumption that what is being exchanged is similar to input and output parameters of program invocations
- It 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.

A preliminary solution (1)

- There is a “SOAP message with attachments note” proposed in 2002 that addressed 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)

A preliminary solution (2)

- The solution is simple and it follows the same approach as that taken in e-mail messages: it includes a reference and has 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

Other solutions

- Problems with this technique: 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)
- Alternative proposals include DIME of Microsoft (Direct Internet Message Encapsulation) and WS-attachments

Example

```
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 Message
<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>                          Reference
 </SOAP-ENV:Envelope>
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.tiff@claiming-it.com>          ATTACHMENT

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

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

SOAP attachment problem (2)

- Attachments are relatively easy to include in a message and all proposals (MIME or DIME based) are similar in spirit

SOAP attachment problem (3)

- 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)

SOAP attachment problem (4)

- 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 we'll see later on (how to model attachments in WSDL?)

SOAP and client-server model

- The close relation among 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)

SOAP SWOT analysis (1)

- 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

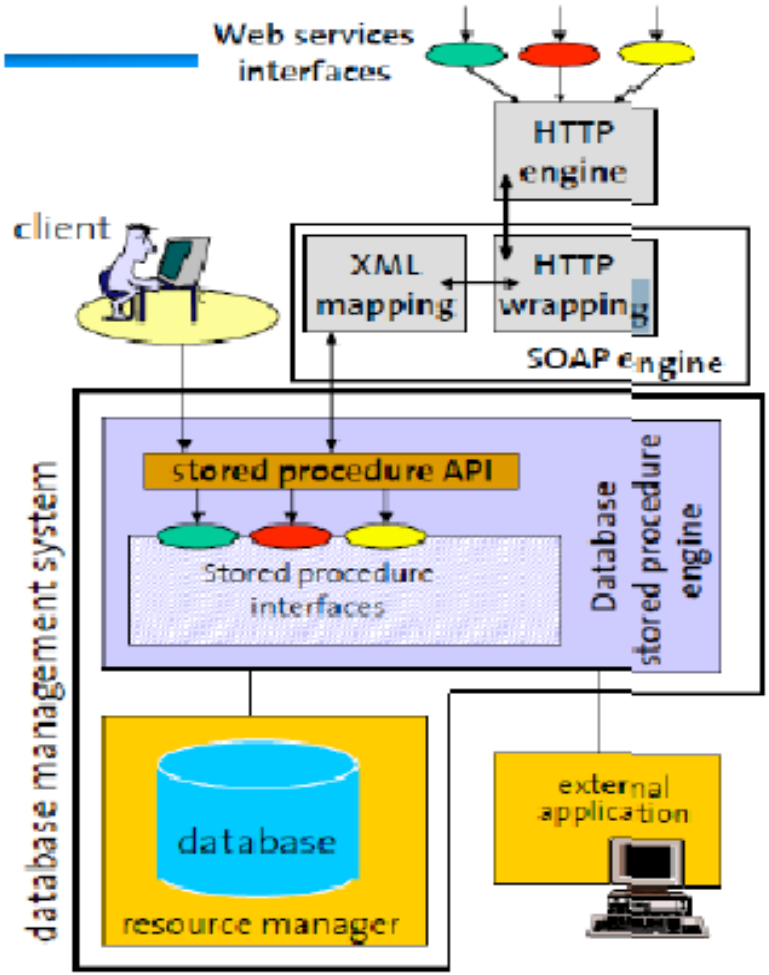
SOAP SWOT analysis (2)

- 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
  - its simplicity: SOAP is not enough in a real application, many aspects are missing

SOAP and databases (1)

- 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 databases (2)



SOAP summary (1)

- 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 summary (2)

- 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)

SOAP summary (3)

- 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)