Prof.M.Sudha , VIT

THE SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

Introduction

- SOAP was designed by Dave Winner, Don Box,.. In 1998 from Microsoft as an object-access protocol.
- Is a lightweight protocol for exchanging structured information in the implementation of web services.
- SOAP messages could be send to a web service enabled website with parameters needed for a search.
- The site would return xml-formatted document with the resulting data.
- SOAP is xml, i.e., relied heavily on xml schema and namespaces for its definition and function.

What is SOAP?

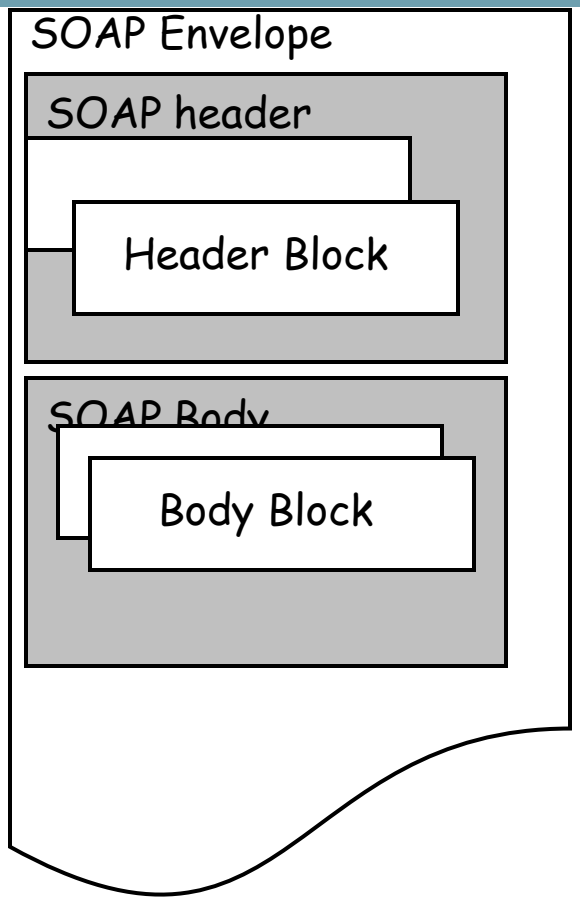
- The W3C started working on SOAP in 1999. The current W3C recommendation is Version 1.2
- SOAP is XML. That is, SOAP is an application of the XML specification. It relies heavily on XML standards like XML Schema and XML Namespaces for its definition and function.
- SOAP covers the following four main areas:
  - A message format for Communication
  - A description of how a SOAP message should be transported using HTTP
  - A set of rules that must be followed when processing a SOAP message
  - 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

Background - 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

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



SOAP,

XML name space identifier for SOAP serialization

XML name space identifier for SOAP envelope

```
<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

Serialization

Serialization - The process of converting an object into a stream of bytes. This stream of bytes can be persisted. Deserialization is an opposite process, which involves converting a stream of bytes into an object. Serialization is used usually during remoting (while transporting objects) and to persist file objects & database objects.

XML serialization converts (serializes) the public fields and properties of an object, or the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document

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>
```

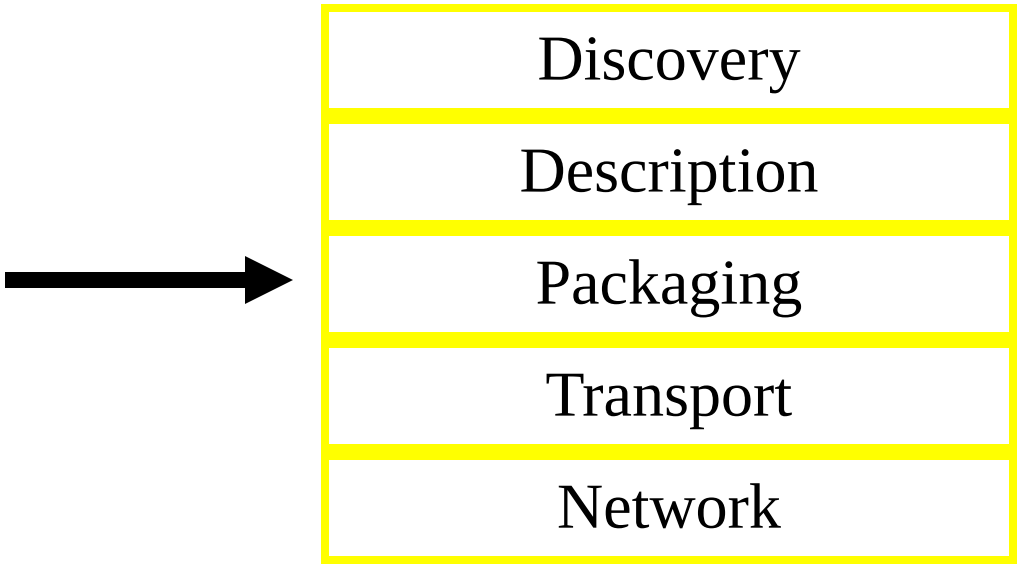
```
    <SOAP-ENV:Body>
            <m:GetLastTradePrice xmlns:m="Some-URI">
                <symbol>DEF</symbol>
            </m:GetLastTradePrice>
        </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

SOAP

- An application of the XML specification
- Relies on XML Schema, XML Namespaces
  - *www.w3c.org*
- Platform independent
- Provides a standard way to structure XML Messages

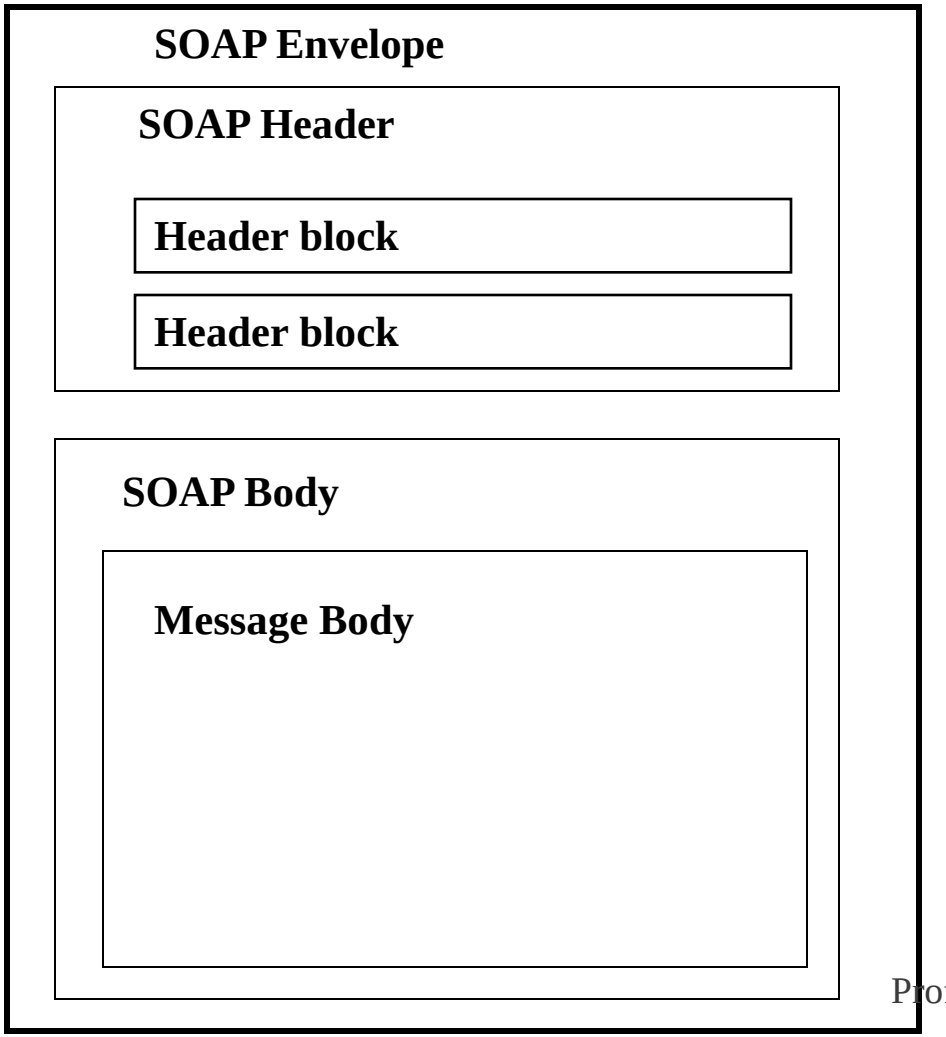

Application
XML Message
Application


Discovery
Description
Packaging
Transport
Network

It is necessary to define:

- The types of information to be exchanged
- How to express the information as XML
- How to send the information

SOAP Messages



Header contains blocks of information regarding how to process the message:

- Routing and delivery settings
- Authentication/authorization assertions
- Transaction contexts

Body contains actual message to be delivered and processed

Soap- Messaging Style

- SOAP for EDI (known as "document-style" SOAP), then the XML will be a purchase order, tax refund, or similar document.
- If you use SOAP for RPC (known, unsurprisingly, as "RPC-style" SOAP) then the XML will be a representation of parameter or return values.

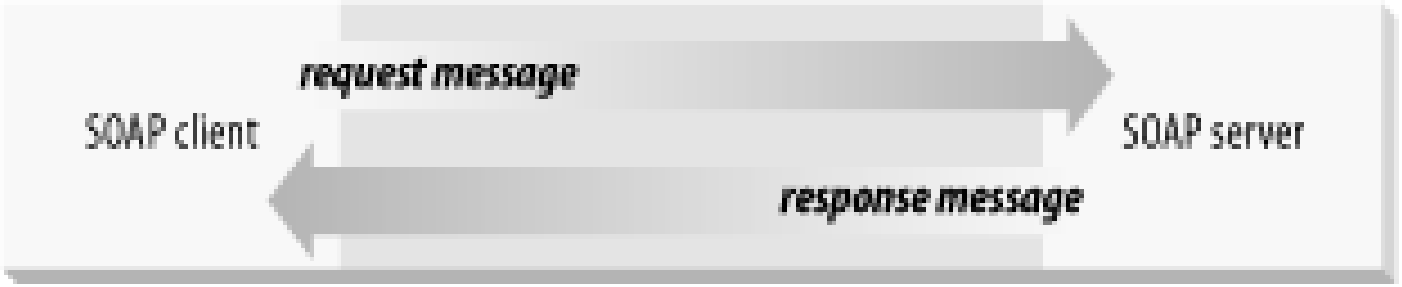
A purchase order in Document-style SOAP

```
<s:Envelope
xmlns:s="http://www.w3.org/2001/06/soap-envelope">
<s:Header>
<m:transaction xmlns:m="soap-transaction”  s:mustUnderstand="true">
<transactionID>1234</transactionID>
</m:transaction>
</s:Header>
<s:Body>
<n:purchaseOrder xmlns:n="urn:OrderService">
<from><person>Christopher Robin</person>
<dept>Accounting</dept></from>
<to><person>Pooh Bear</person>
<dept>Honey</dept></to>
<order><quantity>1</quantity>
<item>Pooh Stick</item></order>
</n:purchaseOrder>
</s:Body>
</s:Envelope>
```

RPC Style

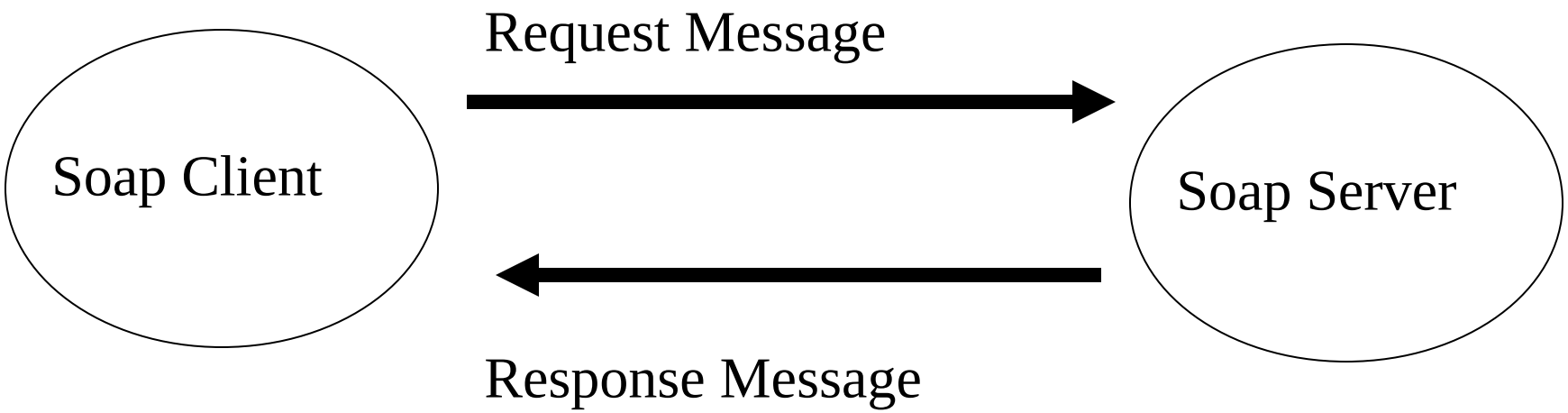
- Now let's see an RPC-style message. Typically messages come in pairs,
- the request (the client sends function call information to the server) and the response (the server sends return value(s) back to the client).
- SOAP doesn't require every request to have a response, or vice versa, but it is common to see the request-response pairing.

Basic RPC messaging architecture



Next., illustrates a simple RPC-style SOAP message that represents a request for IBM's current stock price.
Again, we show a header block that indicates a transaction ID of "1234".

RPC Messages



RPC-style SOAP message

```
<s:Envelope
xmlns:s="http://www.w3.org/2001/06/soap-envelope">
<s:Header>
<m:transaction xmlns:m="soap-transaction"
                        s:mustUnderstand="true">
<transactionID>1234</transactionID>
</m:transaction>
</s:Header>
<s:Body>
<n:getQuote xmlns:n="urn:QuoteService">
<symbol xsi:type="xsd:string"> IBM </symbol>
</n:getQuote>
</s:Body>
</s:Envelope>
```

Response Msg.,

```
<s:Envelope
xmlns:s="http://www.w3.org/2001/06/soap-envelope">
<s:Body>
<n:getQuoteRespone
xmlns:n="urn:QuoteService">
<value xsi:type="xsd:float">
98.06
</value>
</n:getQuoteResponse>
</s:Body>
</s:Envelope>
```

RPC-style SOAP Message

public Float getQuote(String symbol);

```
<s:Envelope xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Header>
    <m:transaction xmlns:m="soap-transaction" s:mustUnderstand="true">
      <transactionID>1234</transactionID>
    </m:transaction>
  </s:Header>
  <s:Body>
    <n:getQuote xmlns:n="urn:QuoteService">
      <symbol xsi:type="xsd:string">
        IBM
      </symbol>
    </n:getQuote>
  </s:Body>
</s:Envelope>
```

SOAP response

```
<s:Envelope xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Body>
    <n:getQuoteResponse xmnls:n="urn:QuoteService">
      <value xsi:type="xsd:float">
        98.06
      </value>
    </n:getQuoteResponse>
  </s:Body>
</s:Envelope>
```

SOAP Faults

```
<s:Envelope xmlns:s="…">
  <s:Body>
    <s:Fault>
    <faultcode>Client.Authentication</faultcode>
    <faultstring>Invalid credentials</faultstring>
    <faultactor>http://acme.com/</faultactor>
    <details> <!-- application specific details></details>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

Upgrade header

The Upgrade header

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
<s:Header>
<V:Upgrade xmlns:V="http://www.w3.org/2001/06/soap-upgrade">
<envelope qname="ns1:Envelope"
xmlns:ns1="http://www.w3.org/2001/06/soap-envelope"/>
</V:Upgrade>
</s:Header>
<s:Body>
<s:Fault>
<faultcode>s:VersionMismatch</faultcode>
<faultstring>Version Mismatch</faultstring>
</s:Fault>
</s:Body>
</s:Envelope>
```

SOAP Faults

- A SOAP fault is a special type of message specifically targeted at communicating information about errors that may have occurred during the processing of a SOAP message.

Misunderstood header

- SOAP fault structure is not allowed to express any information about which headers were not understood.
- To solve this, SOAP 1.2 defines a standard header block “Misunderstood”. (optional)

```
<s:Envelope xmlns:s="...">
   <s:Header>
      <f:Misunderstood qname="abc:transaction"
                       xmlns:="soap-transactions" />
   </s:Header>
   <s:Body>
      <s:Fault>
       <faultcode>MustUnderstand</faultcode>
       <faultstring>
            Header(s) not understood
```

SOAP fault

```
<s:Envelope xmlns:s="...">
<s:Body>
<s:Fault>
<faultcode>Client.Authentication</faultcode>
<faultstring>
Invalid credentials
</faultstring>
<faultactor>http://acme.com</faultactor>
<details>
<!-- application specific details -->
</details>
</s:Fault>
</s:Body>
</s:Envelope>
```

Fault Msg.,

- *The fault code*
- An algorithmically generated value for identifying the type of error that occurred.
- The value must be an XML Qualified Name, meaning that the name of the code only has
- meaning within a defined XML namespace.
- *The fault string*
- A human-readable explanation of the error.

Ctd.,

- *The fault actor*
- The unique identifier of the message processing node at which the error occurred (actors will be discussed later).
- *The fault details*
- Used to express application-specific details about the error that occurred. This must be present if the error that occurred is directly related to some problem with the body of the message.
- It must not be used, however, to express information about errors that occur in relation to any other aspect of the message process.

SOAP Faults

- Fault info. is coded in to <Body> element of SOAP msg.
- The name of this body element is called FAULT
- This support the following 4 child elements
- 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

Standard SOAP Fault Codes

- SOAP defines four standard types of faults that belong to the http://www.w3.org/2001/06/soap-envelope namespace. These are described here:
- Version Mismatch

  The SOAP envelope is using an invalid namespace for the SOAP Envelope element.
- MustUnderstand

  A Header block contained a mustUnderstand="true" flag that was not understood by the message recipient.

- Server
- An error occurred that can't be directly linked to the processing of the message.
- Client
- There is a problem in the message. For example, the message contains invalid authentication credentials, or there is an improper application of encoding rules

Ctd..

- A header block contained within a SOAP message may indicate through the mustUnderstand="true" flag that the recipient of the message must understand how to process the contents of the header block.

- If it cannot, then the recipient must return a MustUnderstand fault back to the sender of the message.

Misunderstood Header block

- To solve this problem, the SOAP Version 1.2 specification defines a standard Misunderstood header block that can be added to the SOAP fault message to indicate which header blocks in the received message were not understood.

Misunderstood header

The Misunderstood header

<s:Envelope xmlns:s="...">

<s:Header>

<f:Misunderstood qname="abc:transaction"

xmlns:="soap-transactions" />

</s:Header>

<s:Body>

<s:Fault>

<faultcode>MustUnderstand</faultcode>

<faultstring>

Header(s) not understood

</faultstring>

<faultactor>http://acme.com</faultactor>

</s:Fault>

</s:Body>

</s:Envelope>

The Misunderstood header block is optional

Custom Faults

- A web service may define its own custom fault codes that do not derive from the ones defined by SOAP. The only requirement is that these custom faults be namespace qualified
- **A custom fault**

```
<s:Envelope xmlns:s="...">
<s:Body>
<s:Fault xmlns:xyz="urn:myCustomFaults">
<faultcode>xyz:CustomFault</faultcode>
<faultstring>
My custom fault!
</faultstring>
</s:Fault>
</s:Body>
</s:Envelope>
```