# 4. Distributed Information Systems II: Web Services

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Business Economics and Information Systems
& Institute for Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

## Web Service Protocol Stack

<table>
<thead>
<tr>
<th>layer</th>
<th>task</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Service) Transport Protocol</td>
<td>transport messages</td>
<td>HTTP, SMTP, FTP</td>
</tr>
<tr>
<td>(XML) Messaging Protocol</td>
<td>encode messages</td>
<td>XML-RPC, WS-Addressing, SOAP</td>
</tr>
<tr>
<td>(Service) Description Protocol</td>
<td>describe public interface</td>
<td>WSDL</td>
</tr>
<tr>
<td>(Service) Discovery Protocol</td>
<td>discover services</td>
<td>UDDI</td>
</tr>
</tbody>
</table>
1. Message Transport: HTTP

2. Message Encoding: SOAP

3. Implementing Web Services: Axis2 engine

Request and Response
Open Systems Interconnection Basic Reference Model (OSI Model)

Communication is structured in so-called **network layers**:

### Layers

- **Anwendungs-Schicht**
  - HTTP, FTP, ...
- **Transport-Schicht**
  - TCP, UDP
- **Netzwerkschicht**
  - IP
- **Physikalische Schicht**
  - Ethernet, X.25, Tokenring

The full OSI model contains 7 layers:

- application, presentation, session, transport, network, data Link, and physical layer

some of which often are lumped together in 5 or 4 layers

**as above.**

---

**Hypertext Transfer Protocol (HTTP)**

- HTTP is a protocol for the exchange of information via a **request/response paradigm**.

- HTTP is coordinated by
  - W3C and
  - the Internet Engineering Task Force (IETF).

- Different versions of HTTP are described in a series of **Request for Comments (RFCs)**, most actually
  - HTTP 1.1 in RFC 2616 from June, 1999.

- HTTP messages consist of
  - a **response/request line**,  
  - optional **header lines**  
  - an **entity body**  
    (delimited by an empty line from the headers)
Hypertext Transfer Protocol (HTTP) / Syntax

\[ \langle \text{request} \rangle := \langle \text{request line} \rangle \]  
\[ (\langle \text{general header} \rangle | \langle \text{request header} \rangle | \langle \text{entity header} \rangle)^* \]  
\[ \langle \text{CRLF} \rangle \]  
\[ \langle \text{entity body} \rangle \]  

\[ \langle \text{response} \rangle := \langle \text{response line} \rangle \]  
\[ (\langle \text{general header} \rangle | \langle \text{response header} \rangle | \langle \text{entity header} \rangle)^* \]  
\[ \langle \text{CRLF} \rangle \]  
\[ \langle \text{entity body} \rangle \]  

\[ \langle \text{request line} \rangle := \langle \text{method} \rangle \ SP \langle \text{request uri} \rangle \ SP \langle \text{http version} \rangle \ CRLF \]  

\[ \langle \text{response line} \rangle := \langle \text{http version} \rangle \ SP \langle \text{status code} \rangle \ SP \langle \text{reason phrase} \rangle \ CRLF \]  

where \( \langle SP \rangle \) denotes a space  
and \( \langle CRLF \rangle \) a newline.

Most common Request Headers

**Host**  
server request is sent to.

**From**  
client response originated from.

**User-Agent**  
browser used on the client.

**Accept, Accept-Charset, Accept-Encoding, Accept-Language**  
charset, encoding and language preferred by the client.

**Referer**  
URI of resource containing the link to the request URI.

**Authorization**  
login and password information.

**If-modified-since**  
conditional request.
Most common Entity and General Headers

Entity headers:

**Content-Encoding, Content-Length, Content-Type, Content-Language**

encoding, length, type and language of content entity returned.

**Last-modified**

timestamp entity last has been modified.

**Expires**

timestamp until entity is valid.

General headers:

**Date**

date and time of request / response.

Request Methods

**GET**

– Requests the entity identified by the request URI.
– Signals that the resource should not be altered by the operations.

**POST**

– Submits data to the specified resource and requests a result entity in return.
– The data is sent in the entity body of the request.

**PUT**

– Uploads an entity for storage under the request URI.

**DELETE**

– Deletes the entity identified by the request URI.

as well as the more specialized methods HEAD, TRACE, OPTIONS and CONNECT.
The success of the request is signaled by a status code:

<table>
<thead>
<tr>
<th>code</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
</tr>
<tr>
<td>402</td>
<td>Payment Required</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
</tr>
</tbody>
</table>

Example HTTP Headers

GET /index.html HTTP/1.1
Host: localhost:8090
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.1.14) Gecko/20080410 SUSE/2.0.0.14-0.1 Firefox/2.0.0.14
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.5
Keep-Alive: 300
Connection: keep-alive

Figure 3: Request by Firefox

GET /index.html HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.7 (like Gecko) SUSE
Accept: text/html, image/jpeg, image/png, text/*, image/*, */*
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *,q=0.5
Accept-Language: en, de
Host: localhost:8090
Connection: Keep-Alive

Figure 4: Request by Konqueror
Example HTTP Headers

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
ETag: W/"10852-1213607922000"
Last-Modified: Mon, 16 Jun 2008 09:18:42 GMT
Content-Type: text/html
Content-Length: 10852
Date: Mon, 16 Jun 2008 20:59:45 GMT
Connection: keep-alive

<html lang="de">
<head>
...
</head>
...</html>
```

Figure 5: Response by Tomcat

1. Message Transport: HTTP
2. Message Encoding: SOAP
3. Implementing Web Services: Axis2 engine
- SOAP defines a format for exchanging structured and typed information between peers in a decentralized, distributed environment, consisting of:


  - **Adjuncts**: SOAP Data Model, SOAP Encoding, SOAP RPC Representation, a Convention for Describing Features and Bindings, Message Exchange Patterns and Features, SOAP HTTP Binding.

- SOAP is an XML application. Its namespace is http://www.w3.org/2003/05/soap-envelope

- SOAP can use the XML Schema type system.

- SOAP is managed by the W3C, its actual version is SOAP 1.2 (April 27, 2007).

- SOAP originally was the acronym for *Simple Object Access Protocol*, but this name is no longer used.

---

### Core SOAP Components

1. **SOAP Message Format**:  
   - provides overall structure (*envelope*) of request/response messages.

2. **SOAP Data Model**:  
   - conceptual type system for arguments and return values of procedures.

3. **SOAP Encoding**:  
   - XML representation of the SOAP Data Model.

4. **SOAP Remote Procedure Calls (RPCs)**:  
   - how to specify method calls.

5. **SOAP HTTP Binding**:  
   - how to transport SOAP messages via HTTP.

Only the SOAP Envelope is mandatory, all other components can be replaced by other specifications independently.
The root element of a SOAP message:

```xml
<Envelope>
  Content: ⟨Header⟩ ?
  ⟨Body⟩
</Envelope>
```

The optional header carries information about the processing of the message by intermediary SOAP nodes. (not handeled here)

The mandatory body element:
- contains any number of children elements.
- which are web service-specific (i.e., not described by SOAP).

The SOAP data model models data items as **directed labeled graphs**.

It contains the following data items:

- **simple value**: a simple lexical value. a node with a **lexical value** and an optional **XML schema type**.

- **struct**: a compound of named parts. a node with **labeled outgoing edges**.

- **array**: a compound of indexed parts. a node with **numbered outgoing edges**.
The target nodes of structs and arrays can be any valid SOAP data items, i.e., simple values as well as themselves structs or arrays.

The SOAP encoding provides a representation for SOAP data instances as XML:

- Each edge of a SOAP data instance is represented as element. Its name is the label of the edge (structs) or arbitrary (arrays).
- The SOAP type of the target node (optionally) can be expressed by the attribute nodeType as: “simple”, “struct” or “array”.
- Simple values of target nodes are expressed as character content of the edge element, their type by the attribute xsi:type.

The names for the SOAP encoding primitives belong to the namespace

http://www.w3.org/2003/05/soap-encoding
– **Structs** as target nodes are expressed as *sequence of outgoing named edges*.

```
<XXX enc:nodeType="struct" xmlns:app="http://www.ismll.de/examples/soap/encoding1">
  <app:forename enc:nodeType="simple" xsi:type="xs:string">Anna</app:forename>
  <app:surname enc:nodeType="simple" xsi:type="xs:string">Miller</app:surname>
  <app:age enc:nodeType="simple" xsi:type="xs:int">17</app:age>
</XXX>
```

– **Arrays** as target nodes are expressed as *sequence of unnamed edges*.

– The element name is arbitrary, the position denotes the index.
– The type of the array element and the size of the array can be specified by the attributes

```xml
<XXX enc:nodeType="array" enc:itemType="xs:string" enc:arraySize="3">
  <Y>Anna</Y>
  <Y>Berta</Y>
  <Y>Clara</Y>
</XXX>
```
SOAP Encoding / References

- Instead of provided as element content, target nodes can also be pointed to by the attribute ref.
- For each ref, there must an element with attribute id having the same value in the same envelope.

```
<XXX enc:nodeType="struct" xmlns:app="http://www.ismll.de/examples/soap/encoding1">
  <app:forename enc:ref="Annas forename"/>
  <app:sirname enc:ref="Anna/sirname"/>
  <app:age enc:ref="v13"/>
</XXX>
```

```
<Y enc:id="Annas forename" enc:nodeType="simple" xsi:type="xs:string">Anna</Y>
<Y enc:id="Anna/sirname" enc:nodeType="simple" xsi:type="xs:string">Miller</Y>
<Y enc:id="v13" enc:nodeType="simple" xsi:type="xs:int">17</Y>
```

SOAP Encoding / env:encoding-style

To signal that a data item has been encoded using the SOAP encoding, one has to set the attribute `env:encoding-style` to the namespace URI

```
http://www.w3.org/2003/05/soap-encoding
```

Do not confuse this setting with the namespace setting for the names of the SOAP encoding namespace.
To invoke an SOAP RPC, the following information is needed:

– The address of the target SOAP node.
– A procedure or method name.
– Arguments passed to the procedure as identity/value pairs.
– Property values of the binding.
– Header data (optional).

The namespace for SOAP rpc primitives is
http://www.w3.org/2003/05/soap-rpc

An **RPC invocation** is encoded as single struct with the in or in/out arguments as parts, i.e.:

– Encoded as element in the SOAP body.
  The name of the element is the name of the procedure called.

– Each in or in/out argument as outgoing edge named by the argument name, i.e., as nested element.

An **RPC response** is encoded as single struct with the out or in/out arguments and the result as parts:

– Encoded as element in the SOAP body.
  The name of the element is arbitrary.

– Each out or in/out argument as edge named by the argument name.

– If the result type is not void, an outgoing edge named rpc:result
Example / Request

Assume there is a webservice at the address “http://localhost:8080/axis2/services/CalculatorService” offering a procedure “add” that takes two integer arguments “i1” and “i2” and returns the sum of both values.

To invoke this service, we could sent the following SOAP message:

```xml
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmns:xs="http://www.w3.org/2001/XMLSchema"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmns:enc="http://www.w3.org/2003/05/soap-encoding"
xmns:calc="http://ismll.de/examples/soap/Calculator">
  <env:Body>
    <calc:add env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
      <calc:i1 enc:nodeType="simple" xsi:type="xs:int">7</calc:i1>
      <calc:i2 enc:nodeType="simple" xsi:type="xs:int">8</calc:i2>
    </calc:add>
  </env:Body>
</env:Envelope>
```

Figure 14: A simple SOAP request.

Example / Response

The service could respond with the following SOAP message:

```xml
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmns:xs="http://www.w3.org/2001/XMLSchema"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmns:enc="http://www.w3.org/2003/05/soap-encoding"
xmns:calc="http://ismll.de/examples/soap/Calculator"
xmns:rpc="http://www.w3.org/2003/05/soap-rpc">
  <env:Body>
    <calc:response env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
      <rpc:result enc:nodeType="simple" xsi:type="xs:int">15</rpc:result>
    </calc:response>
  </env:Body>
</env:Envelope>
```

Figure 15: A simple SOAP response.
SOAP HTTP Binding

SOAP allows different underlying protocols for transporting the message.

The most common one is HTTP via the POST method:

- Usually with media-type `application/soap+xml` (specified in the HTTP header field `Content-Type`).

Practically, to send our SOAP request message, we could use a download tool such as `wget`:

```
wget --post-file=request-manual-all.xml
   --header='Content-Type: application/soap+xml'
   http://localhost:8080/axis2/services/CalculatorService
```

(The HTTP header field “SOAPAction” from SOAP 1.1 is obsolete in SOAP 1.2. The action optionally can be encoded as the action feature of the MIME type.)

A remark about SOAP 1.1

In older examples you will also find the old SOAP v1.1 namespace identifiers

```
http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/encoding/
```

that should no longer be used.
1. Message Transport: HTTP

2. Message Encoding: SOAP

3. Implementing Web Services: Axis2 engine

Implementing a Web Service

SOAP tells you how to use a web service technically, when you know,

– where it is,
– which methods it offers and
– with which signatures.

SOAP does not tell you at all how to implement a web service.

How to implement a web service depends on the web service engine, e.g.,

  (v1.5.1, Oct. 2009)
Installing Apache Axis2

unzip axis2-1.4-bin.zip
cd axis2-1.4/
chmod a+x bin/axis2server.sh
./bin/axis2server.sh

Now the Axis2 engine is running on port 8080.

You can get a list of deployed web services by visiting http://localhost:8080/ with a web browser.

Deploying a Web Service in Axis2 (1/2)

A minimal web service implementation is made from just two files:

1. The implementation Calculator.java:

```
public class Calculator {
    public int add(int i1, int i2) {
        return i1 + i2;
    }

    public int subtract(int i1, int i2) {
        return i1 - i2;
    }
}
```

2. An Axis-specific webservice descriptor services.xml:

```
<service name="CalculatorService" scope="application" targetNamespace="http://ismll.de/examples/soap/Calculator">
    <description>Calculator</description>
    <messageReceivers>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
            class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
            class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
    <schema schemaNamespace="http://ismll.de/examples/soap/Calculator"/>
    <parameter name="ServiceClass">Calculator</parameter>
</service>
```
Web services can be archived in **Axis archives** (.aar; jar-archives):

- containing the classes in the root and
- the webservice descriptor services.xml in the subdirectory META-INF.

```
> jar tf Calculator.aar
META-INF/
META-INF/MANIFEST.MF
Calculator.class
META-INF/services.xml
```

Axis archives can be deployed by simply copying them to the **Axis2 services repository**:

```
> cp Calculator.aar ~/ws/axis2-1.4/repository/services/
```