

# Introduction to Web Services

Alberto Pace  
Information Technology Department  
CERN, Geneva, Switzerland

With input from Andreas Pfeiffer, CERN, Geneva, Switzerland

## Agenda

- ◆ HTTP
- ◆ XML
  - ◆ Syntax, Namespaces, DTD, XSL, XSLT ...
- ◆ Web Services
  - ◆ XMLRPC, SOAP

## Introduction to HTTP

## The HTTP protocol

- ◆ The Hyper Text Transfer Protocol (HTTP) is the client-server network protocol that has been in use by the World-Wide Web since 1990.
- ◆ Whenever you surf the web, your browser will be sending HTTP *request messages* for HTML pages, images, scripts and styles sheets. Web servers handle these requests by returning *response messages* that contain the requested resource.
- ◆ See:  
<http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- ◆ It is based on TCP and, by default it runs on port 80 (or 443 for its secure version)

## HTTP request example

HTTP method (or verb)

relative URL

version of HTTP

```
GET /data/en/catalogue/ HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
Connection: Keep-Alive
```

a set of name/value pairs (*headers*)  
header values control how the request is processed by the server.

5

## HTTP response example

version of HTTP

Status Code

version of HTTP

```
HTTP/1.1 200 OK
Date: Mon, 04 Oct 2004 12:04:43 GMT
Cache-Control: no-cache
Content-Type: text/html; charset=utf-8
Content-Length: 8307
```

Blank  
line

```
<html>
  <head></head>
  <body><p>Hello World</p></body >
</html>
```

HTML document

name/value pairs  
(*headers*)

6

**DEMO**

## HTTP Request Headers (examples)

- ◆ Accept: \*/\*
  - ◆ types of content accepted by the client
- ◆ Accept-Language: en-gb
  - ◆ The client prefers British English content
- ◆ Accept-Encoding: gzip, compress
  - ◆ The client can understand and can decompress the listed formats
- ◆ Connection Keep-Alive
  - ◆ request to use a persistent TCP connection
- ◆ Host: www.cern.ch
  - ◆ HTTP/1.1 requires that the host name is supplied with every request allowing multiple domains can be hosted on a single IP address.
- ◆ User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
  - ◆ Version of client software
- ◆ ...

## HTTP Response Headers (examples)

- ◆ Date: Wed, 4 Oct 2004 12:00:00 GMT
  - ◆ Date and time on the server
- ◆ Content-Length: 2748
  - ◆ Gives the length in bytes of the body that follows the headers.
- ◆ Content-Type: image/gif
  - ◆ Gives the type of the body
- ◆ Cache-Control: no-cache
  - ◆ indicates whether the resource may be cached by the client. The value no-cache disables all caching
- ◆ Expires: -1
  - ◆ The date when the content is out of date. -1 indicates that the content expires immediately
- ◆ X-zzzzzz
  - ◆ Web applications can use custom headers to add comments or annotations to an HTTP message. The convention is to prefix the header name with X- to indicate that it is non-standard.

## HTTP Status Codes

- ◆ 1xx - Informational
  - ◆ intermediate response. For example, the server can reply initially with 100 Continue when it receives a POST request and then with 200 OK once it has been processed
- ◆ 2xx - Successful
  - ◆ Request successfully processed. For example, the value 200 is used when the requested resource is being returned to the HTTP client in the body of the response message.
- ◆ 3xx - Redirection
  - ◆ For example the code 302 returns another URL to which the client should issue request again.
  - ◆ The code 304 indicates that the resource was not modified and the client should read from its local cache instead.
- ◆ 4xx - Client Error
  - ◆ In addition to the well known 404 (404 The requested resource does not exist), worth mentioning the 401 code which is the "access denied" response (Authentication required)
- ◆ 5xx - Server Error
  - ◆ An error occurred on the server while processing the request. The code 500 is typically "An internal error occurred on the server" while the code 503 is "service is currently unavailable"

## HTTP Cookies

- ◆ A cookie is a piece of data that is issued by a server in an HTTP response that the client re-supplies in subsequent requests to the same server.
- ◆ Setting cookies
  - ◆ Cookies allows the server to store user preferences, identity, application state information for individual clients.
  - ◆ Cookies have a Name, a Value, an Expires Date/time attribute, a Path and a Domain
- ◆ Retrieving Cookies
  - ◆ At every requests, the client consults its local cookie store to see if any unexpired cookies match the path and domain it is about to use.
  - ◆ Any matching cookie values are submitted back to the server using the cookie header.

## HTTP Caching

- ◆ Caching avoids retransmitting the same information multiple times
- ◆ Client caching is controlled by the use of the Cache-Control, Last-Modified and Expires response headers.
  - ◆ Servers set the Cache-Control response header to no-cache to indicate that content should not be cached by the client
- ◆ The Cache-Control header can be set to one of the following values to allow caching:
  - ◆ **<absent>** If the Cache-Control header is not set, then any cache may store the content.
  - ◆ **private** The content is intended for use by a single user and should only be cached locally in the browser.
  - ◆ **public** The content may be cached in public caches (e.g. shared proxies) and private browser caches.
- ◆ To make effective use of cached content, the modification and the expiration date of the content must be supplied in the response header:
  - ◆ **Last-Modified: Wed, 15 Sep 2004 12:00:00 GMT**
  - ◆ **Expires: Sun, 17 Jan 2038 19:14:07 GMT**

## HTTP Cache

- ◆ There are at least 4 cache scenarios which you must be aware of
  - ◆ A resource is never cached and is always downloaded; even with back/forward buttons.
  - ◆ A resource can be cached but has no expiration or modification date. In this case it is always downloaded when the page is first visited in a new browser session or if the user refreshes the page.
  - ◆ A resource can be cached and has a modification date but no expiration date. Therefore it is always checked but not downloaded when the page is first visited in a browser session or if the user refreshes the page.
  - ◆ A resource can be cached and has an expiration date. The browser can reuse the image in a new browser session without having to send any request to the server.
- ◆ For example, the Google logo is set to expire in 2038 and will only be downloaded on your first visit to google.com or if you have emptied your browser cache. To change the image they can use a different image file name or path.

## HTTP Cache Example

### Request

```
GET /images/logo.gif HTTP/1.1
Accept: */*
Referer: http://www.google.com/
Accept-Encoding: gzip
If-Modified-Since: Thu, 23 Sep 2004 17:42:04 GMT
Host: www.google.com
```

### Response

```
HTTP/1.1 304 Not Modified
Content-Type: text/html
Server: GWS/2.1
Content-Length: 0
Date: Thu, 04 Oct 2004 12:00:00 GMT
```

## HTTP Methods

- ◆ The HTTP method (sometimes called 'verb') is supplied in the request and specifies the operation that the client has requested.
  - ◆ There are lot of methods defined (GET, POST, HEAD, OPTION, DELETE, PUT, ...) but the most used are GET and POST
- ◆ The GET method is used to retrieve information from a specified URI
  - ◆ It is the typical method used to display a web page in a browser
  - ◆ The GET method has only a header. There is no body.
  - ◆ GET requests can only supply data in the form of parameters encoded in the URI (known as a Query String) or as cookies in the cookie request header.
- ◆ The POST method has a body in addition to the header that can be used to transfer information
  - ◆ The POST content body that is normally used to send parameters and data. Unlike using the request URI or cookies, there is no upper limit on the amount of data that can be sent and POST must be used if files or other variable length data has to be sent to the server.

## GET / POST Examples

### GET

```
GET /data/?User=Joe HTTP/1.1
Accept: */*
Accept-Encoding: gzip
Host: www.cern.ch
```

### POST

```
POST /data/ HTTP/1.1
Accept: */*
Accept-Encoding: gzip
Host: www.cern.ch

User=Joe
```

## HTTP Authentication

- ◆ HTTP supports several authentication mechanisms based around the use of the 401 status code and the WWW-Authenticate response header
- ◆ The standardized HTTP authentication mechanisms are Basic and Digest. Other (NTLM, certificates) are also very common.
- ◆ Basic Authentication
  - ◆ The client sends the username and password encoded in base64
  - ◆ It should only be used with HTTPS, as the password can be easily captured and reused over HTTP.
- ◆ Digest
  - ◆ The client sends a hashed form of the password to the server
  - ◆ Although, the password cannot be captured over HTTP, it is possible to replay requests using the hashed password.





## HTTP Authentication Example

A server forces authentication by rejecting the request with a 401 code and setting the WWW-Authenticate response header

```
GET /protectedfiles/ HTTP/1.1 401 Access Denied
Host: www.cern.ch           WWW-Authenticate: Basic realm="CERN"
                             Content-Length: 0
```

Most web clients will then display a login dialog, allowing the user to enter a username and password. This information is used to retry the request with an Authorization request header

```
GET /protectedfiles/ HTTP/1.1
Host: www.cern.ch
Authorization: Basic <USERNAME>:<PASSWORD>
```

Note that anyone tapping the network can intercept the HTTP request (and its username / password)



## Secure HTTP (HTTPS)

- ◆ It is the HTTP protocol encrypted over a Secure Channel (SSL or TLS).
  - ◆ Prevents eavesdropping, tampering or replaying of messages
  - ◆ Uses certificates to authenticate servers and optionally clients
- ◆ <http://www.faqs.org/rfcs/rfc2818.html>
- ◆ Typically runs on TCP port 443 instead of 80



## More on HTTP

- ◆ See more on “Encoding”, “Redirection” and “Compression in the References
- ◆ Reference:
  - ◆ <http://www.w3.org/Protocols/rfc2616/rfc2616.html>



## Introduction to XML XML

eXtensible Markup Language

## Overview

- ◆ XML – what it (not) is
- ◆ XML syntax
- ◆ DTD – describing XML documents
- ◆ Related technologies

## XML – what it is (not)

- ◆ XML is a markup language and *only* a markup language
  - ◆ **Not a programming language !**
    - ◆ No "compiler" for XML
  - ◆ **Not a network protocol !**
  - ◆ **Not a database either !**

```
<book ISBN="0596000588">  
  <title> XML in a nutshell </title>  
  <author> E.R.Harold </author>  
  <author> W.S.Means </author>  
  <publisher> O'Reilly </publisher>  
</book>
```

## HTML versus XML ?

- ◆ HTML is a particular case of XML
- ◆ All (good) HTML documents are also XML documents
- ◆ Only a particular class of XML documents are HTML

```
<head>
  <title> XML in a nutshell </title>
</head> <body>
  <p>XML in a nutshell</p>
  <p>by E.R.Harold and W.S.Means</p>
</body>
```

## XML – what it is

- ◆ (Meta) Data description language
  - ◆ E.g., config files
  - ◆ Extensible – you can add your own “words”
- ◆ Simple, well-documented data format
  - ◆ Truly cross-platform
  - ◆ Ideal for long-term storage
  - ◆ Text file
    - ◆ Unicode (ASCII or other) encoding
- ◆ XML is derived from the Standard Generalized Markup Language (SGML), an ISO standard for documents.

## Meta Markup Language

- ◆ No fixed set of tags and elements
  - ◆ Just a couple of “special characters” which must be replaced with escape sequences
- ◆ Users can define their own specific “language”
  - ◆ Document Type Definition (DTD), XML Schemas
  - ◆ Huge flexibility, very powerful
  - ◆ Namespaces to avoid conflicts

## XML syntax: Tags

- ◆ Start tags begin with “<”
- ◆ End tags begin with “</”
- ◆ Tags are closed by “>”
  - ◆ Empty elements are closed by “/>”

Equivalent syntax

```
<book ISBN="0596000588"></book>
<book ISBN="0596000588" />
<title> XML in a nutshell </title>
```

## XML syntax: Tags

- ◆ Case sensitive
  - ◆ Unless the application decides to ignore case
- ◆ Comments like in HTML
  - ◆ `<!-- this is a comment -->`
  - ◆ The double hyphen -- should not appear anywhere else in the comment

## XML syntax

- ◆ Names of tags reflect the type of content, not formatting information
- ◆ Tags, *elements*, **attributes**, **values**

```
<?xml version="1.0"?>
<book ISBN="0596000588">
  <title> XML in a nutshell </title>
  <author> E.R.Harold </author>
  <author> W.S.Means </author>
  <publisher> O'Reilly </publisher>
</book>
```

## XML syntax: Elements

- ◆ Elements consist of  
`<Tag> content </Tag>`
- ◆ Typically elements contain other elements
  - ◆ XML documents are trees
  - ◆ Parent-child relation for elements in a tree
    - ◆ Each child has exactly one parent (with the exception of the first element in the doc ('root'))

## XML syntax: Attributes

- ◆ XML elements can have attributes
  - ◆ In the start tag
- ◆ Attributes are name/value pairs
  - ◆ name = "value" (or name = 'value')
  - ◆ Not sensitive to whitespace
- ◆ Usually used for meta-data
  - ◆ E.g., ID in a database

```
<book title="XML in a nutshell" ISBN="0596000588" />
```

## Attributes vs. elements

- ◆ In general, any attribute can be expressed as an element

```
<book title="XML in a nutshell" ISBN="0596000588" />
```

```
<book ISBN="0596000588">  
<title>XML in a nutshell</title>  
</book>
```

```
<book>  
<title>XML in a nutshell</title>  
<ISBN>0596000588</ISBN>  
</book>
```

- ◆ The opposite is not true:
  - ◆ attributes cannot contain multiple values (child elements can)
  - ◆ attributes cannot describe structures (child elements can)
  - ◆ attributes are not easily expandable (for future changes)
  - ◆ attributes are more difficult to manipulate by program code
  - ◆ attribute values are not easy to test against a DTD

## Attributes vs. elements (II)

- ◆ The general recommendation is to use attributes for metadata (data about data, e.g. an ID)
- ◆ Avoid using attributes as containers for data to avoid ending up with documents that are difficult to read and maintain.



## XML Names

- ◆ Names in XML may contain alphanumeric (also non-english) characters plus :
  - ◆ **\_ Underscore,**
  - ◆ **- Hyphen, and**
  - ◆ **. Period**
- ◆ All other punctuation chars are not allowed
  - ◆ **No whitespace allowed in names**
  - ◆ **Colon is reserved for namespaces**
- ◆ Names may start with letters, ideograms, or \_

## XML special characters

- ◆ For normal text (not markup), there are no special characters: just make sure your document refers to the correct encoding scheme for the language and/or writing system you want to use, and that your computer correctly stores the file using that encoding scheme.
- ◆ In all cases you can use a symbolic notation called 'entity referencing'. Entity references can either be numeric, using the decimal or hexadecimal Unicode **code point** for the character
  - ◆ Example: if your keyboard has no Euro symbol (€) you can type `&#8364;` Discussed later
  - ◆ You can also have "short names" which you declare in your DTD (eg `<ENTITY euro "&#8364;">`) and then use as `&euro;` in your document.
  - ◆ If you are using a Schema, you must use the numeric form for all except the five below because Schemas have no way to make character entity declarations.
- ◆ If you use XML with no DTD, then these five character entities are assumed to be predeclared, and you can use them without declaring them:

<code>&amp;lt;</code>	<code>&lt;</code>	Less than
<code>&amp;gt;</code>	<code>&gt;</code>	greater than
<code>&amp;amp;</code>	<code>&amp;</code>	ampersand
<code>&amp;apos;</code>	<code>'</code>	apostrophe
<code>&amp;quot;</code>	<code>"</code>	quotation

Entity references always start with the "&" character and end with the ";" character.  
 Note: Only the characters "<" and "&" are strictly illegal in XML. Apostrophes, quotation marks and greater than signs are legal, but it is a good habit to replace them.

## Escape Characters

- ◆ Illegal XML characters have to be replaced by entity references.
  - ◆ If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
    - ◆ You cannot write something like this:

```
<message>if salary < 1000 then</message>
```
    - ◆ To avoid this, you have to replace the "<" character with an entity reference, like this:

```
<message>if salary &lt; 1000 then</message>
```

## Escaping using CDATA sections

- ◆ Tells parser to interpret following data literally ("raw" character data, not interpreting '<' and '&' as special characters)
  - ◆ E.g., embedded HTML or other XML sources
- ◆ Section starts with "<![CDATA["
- ◆ Section ends with "]]>"
  - ◆ Which is of course forbidden in the data

```
<message>
  <![CDATA[if salary < 1000 then]]>
</message>
```

## Who Defines the XML Tags?

- ◆ YOU
- ◆ XML offers a universal way to describe and work with data.
- ◆ XML allows developers to create their own XML vocabularies that are customized for describing their particular data structures.
- ◆ Once developers harness the power of XML to describe their data, they can easily interoperate with any other homogenous or heterogeneous system that also understands XML.
- ◆ Developers can consume data from any other system as long as it's also described using XML. A developer who leverages XML no longer needs to worry about platform, operating system, language, or data store differences when interoperating with other systems.
- ◆ XML becomes the least common denominator for system interoperability.

## XML Namespaces

- ◆ Because XML is truly about interoperability and everyone is free to create their own XML vocabularies, everything would start to break down rather quickly if different developers chose identical element names to represent conceptually distinct entities.
- ◆ To safeguard against these potential conflicts, the W3C introduced namespaces into the XML language.
- ◆ XML namespaces provide a context for your XML document elements. XML namespaces allow developers to resolve elements to a particular implementation semantic.

```
<Customer>
  <Name>John Smith</Name>
  <Address>Geneva, Switzerland</Address>
</Customer>
```

```
<Computer>
  <Name>PCWEB23</Name>
  <Address>137.138.12.34</Address>
</Computer>
```

## XML Namespaces

- ◆ Distinguish between elements and attributes from different domains
  - ◆ “Name” and “Address” can have different meaning
- ◆ Group all related elements and attributes from a given domain (or XML application)
- ◆ Allows to be able to interpret the data
- ◆ Syntax:
  - ◆ The xmlns keywords define the name space
  - ◆ Using a prefix to refer (bind) to the URI
    - ◆ `<MyNS:RDF xmlns:MyNS=http://www.cern.ch/MyNamespace>`
  - ◆ Bindings have a scope with the element (and it's children)

## Validating XML documents

- ◆ Is this XML Document Valid ?
  - ◆ In term of XML Syntax ? YES
  - ◆ Namespace ? YES
  - ◆ But it may fail at the application level

```
<Customer>
  <Name>Bob White</Name>
  <Address>Helsinki, Finland</Address>
  <Customer>
    <Name>Alice Reading</Name>
    <Name>John Smith</Name>
    <Address>Geneva, Switzerland</Address>
  </Customer>
</Customer>
```

- ◆ To catch problems like this, we need a standard mechanism for validating an XML document against some set of predefined rules

## Class of XML Documents

- ◆ To solve the problem of validating XML documents, there are two technologies available to define classes of documents
  - ◆ **DTD (Document Type Definitions)**
    - ◆ Not written in XML
    - ◆ Can be referenced from XML
  - ◆ **XML-Data schema definitions (XSD)**
    - ◆ Less used
    - ◆ written in XML – you can reuse the normal XML technology
    - ◆ more verbose
    - ◆ Can be referenced in the namespace
  - ◆ **RelaxNG**
    - ◆ Less used
    - ◆ written in XML – you can reuse the normal XML technology

## XML documents validation

- ◆ “Well formed”
  - ◆ Correct XML syntax (tags closed, ...)
- ◆ “Valid” documents
  - ◆ Validation against an existing DTD/Schema
  - ◆ No unknown elements/attributes
  - ◆ No invalid values for elements/attributes

## XML Schema definition, RelaxNG

- ◆ Formal syntax to describe exactly what is allowed in an XML document
- ◆ Not discussed further.
- ◆ For XML Schema definition (XSD)
  - ◆ See <http://www.w3.org/TR/xmlschema-0/>
- ◆ For RelaxNG
  - ◆ See <http://www.relaxng.org/>

## Document Type Definitions (DTDs)

- ◆ Formal syntax to describe exactly what is allowed in an XML document
  - ◆ HTML: “ul” may only contain “li”s ...
- ◆ Used for validation of XML documents
  - ◆ If required by the user
- ◆ Declares elements `<!ELEMENT ... >`
- ◆ Declares attributes `<!ATTLIST ... >`
- ◆ A DTD is not an XML document !
  - ◆ Different from XML Schema, RelaxNG

## Example of a DTD

```
<!ELEMENT person (name, profession*)>
<!ELEMENT name (firstName, lastName)>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

*Ordering !*

*Parsed  
Character  
DATA*

```
<!ELEMENT response (data | fault)*>
```

*? Zero or one  
+ One or more  
\* Zero or more*

## Usage of persons DTD

- ◆ The example from last slide is in a file called "person.dtd"

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE person SYSTEM "person.dtd">
<person>
  <name>
    <firstName> Andreas </firstName>
    <lastName> Pfeiffer </lastName>
    <profession> physicist </profession>
  </name>
</person>
```

*Prologue*

**Where is the error ?**

## DTD inside the document

```
<?xml version="1.0"?> ← Standalone version !
<!DOCTYPE person [
  <!ELEMENT person (name, profession*)>
  <!ELEMENT name (firstName, lastName)>
  <!ELEMENT firstName (#PCDATA)>
  <!ELEMENT lastName (#PCDATA)>
  <!ELEMENT profession (#PCDATA)>
]>
<person>
  <name>
    <firstName> Andreas </firstName>
    <lastName> Pfeiffer </lastName>
  </name>
  <profession> physicist </profession>
</person>
```

## Attribute Declarations

### ◆ Syntax

◆ `<!ATTLIST elementName attName attType attDefault>`

### ◆ Example

```
<!ATTLIST image source CDATA #REQUIRED
                width CDATA #REQUIRED
                height CDATA #REQUIRED
                alt CDATA #IMPLIED ← optional
>
```



## Attribute types

- ◆ **CDATA** – text string
  - ◆ Most generic
- ◆ **NMTOKEN**
  - ◆ Same rules as for any XML name
- ◆ **NMTOKENS**
  - ◆ One or more NMTOKEN separated by whitespace
- ◆ **Enumeration**
  - ◆ List of possible choices
- ◆ **ID**
  - ◆ Must contain a name unique within the document
- ◆ **IDREF**
  - ◆ Reference to an ID type attribute of an element in the doc.
- ◆ **IDREFS**
  - ◆ Separated by whitespace
- ◆ **ENTITY**
  - ◆ Name of an unparsed entity
- ◆ **ENTITIES**
  - ◆ Separated by whitespace
- ◆ **NOTATION**
  - ◆ Rarely used

## Attribute defaults

- ◆ **#IMPLIED**
  - ◆ Optional, no default provided
- ◆ **#REQUIRED**
  - ◆ Each instance of the element must provide a value for this attribute, no default.
- ◆ **#FIXED**
  - ◆ Attribute value is constant and immutable, cannot be overwritten
- ◆ **Default values can be provided**
  - ◆ `<!ATTLIST webPage protocol NMTOKEN "http">`

## Entity references

- ◆ “Shorthand” notation for DTDs
  - ◆ Five pre-defined
    - ◆ &lt; &gt; &amp; &quot; &apos;
  - ◆ You can define your own
    - ◆ Eases readability and re-use

```
<!ENTITY coord "( (x,y) | (y,x) | (th,r) | (r,th) )" >  
<!-- a polygon has at least three points -->  
<!ELEMENT polygon (&coord;, &coord;, &coord;+)>
```

## Parsing XML

- ◆ Solutions to access programmatically XML data
  - ◆ Software capable of reading, writing, modifying XML documents and provide access to their structure
- ◆ Often referred as an XML processor or an XML API
- ◆ Two main API specifications have gained popularity:
  - ◆ Document Object Model (DOM)
  - ◆ Simple API for XML (SAX).

## Programming models for XML

- ◆ “Event driven” (SAX) vs. “object based” (DOM)

### Parsers

SAX (simple API for XML)	DOM (Document Object Model)
Doesn't store data while parsing	Constructs an in memory copy of the document
No support for writing/modifying	In-memory tree can be modified
Document data becomes available as it's parsed	Entire document must be parsed before tree is available

## More on DOM

- ◆ In-memory tree representation of the XML document
- ◆ When loaded, the processor builds an in-memory tree that correctly represents the document
- ◆ DOM exposes programmatic interfaces that allow traversing the XML tree and manipulate the elements, values, and attributes

Example: DOM enumeration of child elements using MSXML

```
Set xmlDoc = CreateObject("MSXML.DOMDocument")
bSuccess = xmlDoc.load("customers.xml")
If bSuccess Then
  For Each node in xmlDoc.documentElement.childNodes
    val = node.text
  Next
End If
```

## XML related technologies (I)

- ◆ XLinks
  - ◆ Attribute-based syntax for hyperlinks between XML and non-XML docs

```
<customer xml:link="simple" href="http://cern.ch/customer.htm"> </customer>
```

- ◆ XPointers
  - ◆ More generic than Xlinks, goes beyond URI
  - ◆ Syntax to identify parts of an XML doc
  - ◆ Used often in conjunction with an XLink
  - ◆ consists of a series of location terms. Each location term has a keyword (such as id, child, ancestor, and so on) and can have arguments, such as an instance number, element type, or attribute.
  - ◆ For example, the XPointer:

```
child(2, customer)
```

refers to the second child element whose type is *customer*

## Transforming XML

- ◆ Even when using DOM or SAX API it is tedious to extract specific pieces of data from large documents or to transform certain parts of XML into another format (such as HTML)
- ◆ These tasks can be accomplished using Extensible Stylesheet Language (XSL)
  - ◆ Combined with a query language (XSL Patterns) to extract data
  - ◆ And rules to transform XSLT (XSL transformations)

## Extensible Stylesheet Language (XSL)

- ◆ XSL helps transforming nodes from an XML format into another format
  - ◆ The need is originated on the Web as developers wanted to take their XML data and transform it into HTML for the user to view
- ◆ XSL can also define transformations from a given XML format to another distinct XML format
  - ◆ Ease of interoperability
  - ◆ No need to agree on a universal vocabulary for describing a type of data
- ◆ An XSL file is a list of declarative templates that define the transformation rules
- ◆ Each template defines exactly how you wish to transform a given node from the source document
- ◆ You use XSL Patterns within a template to define which portions of the document the template applies to

## XSLT Example

```
<?xml version="1.0"?>
<hamburgers>
  <hamburger lowfat="dream on">
    <name>CowBurger</name>
    <description>Greasy and good.</description>
    <price>2.99</price>
  </hamburger>
</hamburgers>
```

**XSLT processor**  
 Internet Explorer, Cocoon (Apache)  
 Stand-alone: Saxon, Xalan


```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=" http://www.w3.org/TR/WD-xsl ">
  <xsl:template match="/">
    <html>
      <body>
        <h1>hamburgers</h1>
        <xsl:for-each select="hamburgers[@lowfat='dream on']">
          <li><xsl:value-of select="name"/>, <xsl:value-of select="
            <xsl:value-of select="description"/></li>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
<html>
<body>
<h1>hamburgers</h1>
<ol>
  <li>CowBurger, $2.99, Greasy and good.</li>
</ol>
</body>
</html>
```

## Summary

- ◆ XML
- ◆ Namespaces, Schemas
- ◆ DTD, XSD
- ◆ XLink, XPointer
- ◆ XML parsers, SAX, DOM, XPath
- ◆ XML verification
- ◆ XSL, XSLT, transformation
- ◆ Formatting Objects (XSL-FO)

## Other applications of XML

- ◆ More specific to concrete domains 
  - ◆ W3C endorsed standards
    - ◆ Scalable Vector Graphics – SVG
    - ◆ Mathematical ML – MathML
    - ◆ Resource Description Framework – RDF
    - ◆ Synchronized Multimedia Integration Language (SMIL)
    - ◆ Vector Markup Language (VML)
    - ◆ XHTML
  - ◆ Other
    - ◆ Chemical ML – CML
      - ◆ One of the first XML applications
    - ◆ Channel Definition Framework – CDF
      - ◆ MS specific (publish web sites to IE)
    - ◆ XML Instant messaging

## XML in HEP

- ◆ Configuration files
- ◆ Detector geometry description
  - ◆ “Standard” is evolving
- ◆ Schema for introspection and persistency
  - ◆ LCG Dictionary through gcc-xml
- ◆ Data interchange
  - ◆ AIDA XML standards for Data Analysis related items
    - ◆ Histograms (binned and unbinned),
    - ◆ Vectors of data,
    - ◆ Ntuples,
    - ◆ Functions and Fits

## Links

- ◆ WWW consortium
  - ◆ <http://www.w3.org/>
  - ◆ with lots of further links !
- ◆ XML – development
  - ◆ <http://www.xml.org/>

## Web Services

Definition, Architecture, XML-RPC, SOAP, WSDL, ...

## Web services

- ◆ Will use “Web services” as generic term
  - ◆ Although there is a more specialized definition from W3C
    - ◆ Requires SOAP and WSDL
- ◆ Allow for cross platform interoperability
  - ◆ “The Internet is the platform”
- ◆ Although HTTP is the protocol used for web service, these can be offered on any transport
  - ◆ SMTP (electronic mail) is a good alternative



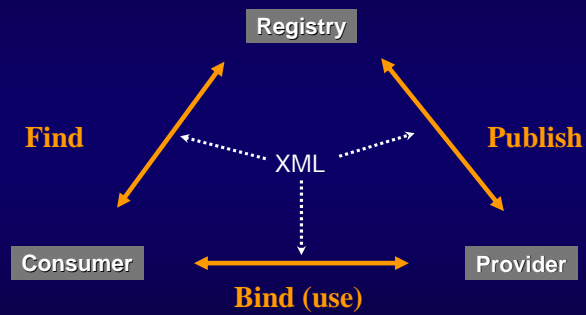
## Web Services

- ◆ Web/network interface to application
  - ◆ Independent of language of implementation
- ◆ Using XML for information exchange
  - ◆ For both: methods and data
- ◆ Kind of “Remote Procedure Call” using XML
- ◆ Two possibilities:
  - ◆ SOAP needs a rather complex “infrastructure”, offers where, what and how to find
  - ◆ XML-RPC is more simple, less heavy

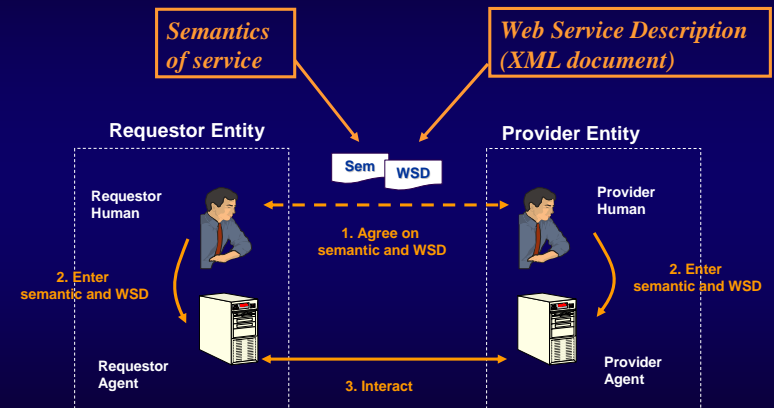
## W3C on Web Services

- ◆ “Definition: A Web service is a software system identified by a URI [\[RFC 2396\]](#), whose **public interfaces and bindings are defined and described using XML**. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, **using XML based messages conveyed by Internet protocols.**”

## Architecture of Web Services (I)



## Architecture of Web Services (II)



## Roles of the agents

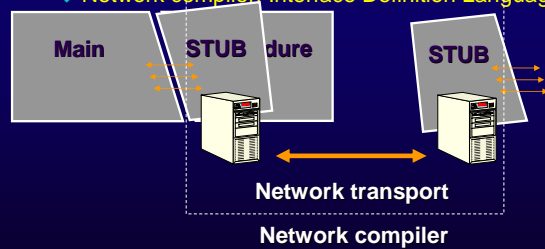
- ◆ Service requestor
- ◆ Service provider
- ◆ Discovery agency
- ◆ Are not fixed, a given agent can “play” several roles

## Calling a procedure on a remote system

- ◆ Needs
  - ◆ A procedure (with agreed semantics)
  - ◆ Arguments to the procedure
  - ◆ Return values from the procedure
  - ◆ Remote system where the procedure is implemented/running
  - ◆ An agreement on how to communicate

## Remote procedure calls (I)

- ◆ **RPC**
  - ◆ Since early 1980's
  - ◆ eXternal Data Representation (XDR) to communicate values
  - ◆ Specific server/client models
  - ◆ **CORBA and DCOM**
    - ◆ Network compiler-Interface-Definition-Languages (IDL)



## Remote procedure calls (II)

- ◆ **XML based**
  - ◆ XML-RPC
  - ◆ SOAP
  - ◆ Late 1990's (parallel development)

## XML-RPC

- ◆ <http://www.xmlrpc.org/>
- ◆ “It's remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.”

## XML-RPC

- ◆ Is a Remote Procedure Call protocol
  - ◆ Working over the Internet
- ◆ Using HTTP as the transport layer
  - ◆ An XML-RPC message is an HTTP-POST request
- ◆ And XML as the encoding
  - ◆ The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML.
  - ◆ Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

## XML-RPC goals

### ◆ Discoverability

- ◆ *"We wanted a clean, extensible format that's very simple. It should be possible for an HTML coder to be able to look at a file containing an XML-RPC procedure call, understand what it's doing, and be able to modify it and have it work on the first or second try."*

### ◆ Easy to implement

- ◆ *"We also wanted it to be an easy to implement protocol that could quickly be adapted to run in other environments or on other operating systems."*

*From: <http://www.xmlrpc.org/spec>*

## XML-RPC example

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181
```

*HTTP POST request*

*Content-length must be correct*

```
<?xml version="1.0"?>
<methodCall>
  <methodName> examples.getStateName </methodName>
  <params>
    <param> <value> <i4> 41 </i4> </value> </param>
  </params>
</methodCall>
```

*Body of the request*

## XML-RPC Basic Types

Tag	Type	Example
<i4> or <int>	Four-byte signed integer	42
<boolean>	0(false) or 1(true)	1
<string>	string	Hello world
<double>	Double-precision signed	-3.1415926
<dateTime.iso8601>	Date/time	20030716T09:53:42
<base64>	Base64-encoded binary	eW91IGNhbid0IHJlYWQgdGhpcyE=

## XML-RPC <struct>

```

<struct>
  <member>
    <name> lowerBound </name>
    <value> <i4> 18 </i4> </value>
  </member>
  <member>
    <name> upperBound </name>
    <value> <i4> 139 </i4> </value>
  </member>
</struct>

```

*structs contain members,  
members have name and value*

- ◆ **<struct>s can be recursive, any <value> may contain a <struct> (or <array>)**

## XML-RPC <array>

```
<array>
  <data>
    <value> <i4> 42 </i4> </value>
    <value> <string> Egypt </string> </value>
    <value> <boolean> 0 </boolean> </value>
    <value> <i4> -31 </i4> </value>
  </data>
</array>
```

*arrays contain data,  
data contains value(s),  
array elements have no names*

- ◆ **<array>s** can be recursive, any **<value>** may contain an **<array>** (or **<struct>**)

## Response example

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
  <methodResponse>
    <params>
      <param>
        <value> <string>South Dakota</string> </value>
      </param>
    </params>
  </methodResponse>
```



## Fault-Response example

```
[HTTP header ...]
<?xml version="1.0"?>
  <methodResponse>
    <fault>
      <value>
        <struct>
          <member>
            <name>faultCode</name>
            <value> <int>4</int></value>
          </member>
          <member>
            <name>faultString</name>
            <value><string>Too many parameters.</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

*fault contains a value, which is a struct with two elements:*  
- one int member named faultCode and  
- one string member named faultString

## XML-RPC extensions

- ◆ **Multicall**
  - ◆ Problem with HTTP round-trip times (latency)
  - ◆ Solution: group requests/responses in arrays and use only one call ("boxcarring")
- ◆ **Server side introspection**
  - ◆ `system.listMethods`
  - ◆ `system.methodSignature`
  - ◆ `system.methodHelp`

## SOAP

- ◆ Simple Object Access Protocol
- ◆ Developed in parallel to XML-RPC
  - ◆ Started by UserLand and Microsoft developers (1998)
  - ◆ Now mainly Microsoft and IBM, endorsed by w3c
  - ◆ Specifications: <http://www.w3.org/TR/soap/>
- ◆ SOAP vs. XML-RPC
  - ◆ User defined data types
  - ◆ Able to specify the recipient
  - ◆ Message specific processing control

## Soap namespaces

- ◆ Extensive use of namespaces and attribute specification tags in almost every element of a message
- ◆ env (Envelope)
  - ◆ <http://www.w3.org/2003/05/soap-envelope>
- ◆ enc (Encoding)
  - ◆ <http://www.w3.org/2003/05/soap-encoding>
- ◆ rpc (Remote Procedure Call)
  - ◆ <http://www.w3.org/2003/05/soap-rp>
- ◆ xs (XML Schema specifications)
  - ◆ <http://www.w3.org/2001/XMLSchema>
- ◆ xsi (XML Schema instances specifications)
  - ◆ <http://www.w3.org/2001/XMLSchema-instance>

## SOAP envelope

- ◆ **Header**
  - ◆ **Optional**
  - ◆ **Information on how the message is to be processed**
- ◆ **Body**
  - ◆ **Required**
  - ◆ **Contains actual message to be delivered / processed**



## SOAP message example

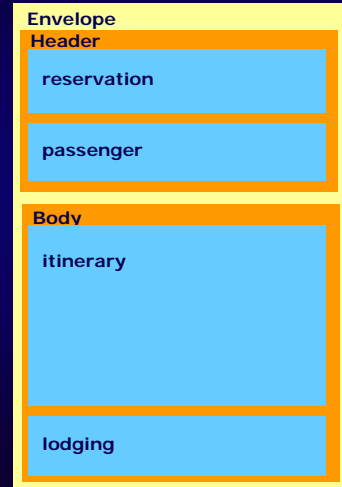
```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

## SOAP message details

- ◆ **Header**
  - ◆ a way to pass information that is not application payload
    - ◆ directives or contextual information related to the processing of the message
  - ◆ **Control of routing**
    - ◆ if multiple nodes or intermediaries process the message
    - ◆ "role"s in headers, "mustUnderstand" attributes
  - ◆ **Nodes may modify the header blocks (or add new ones)**
- ◆ **Body**
  - ◆ is mandatory and is where the main end-to-end information must be carried
  - ◆ Only the ultimate SOAP receiver should alter the body
- ◆ The choice of what goes in the header or the body are decisions taken at application design time

## SOAP conversation Example (I)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.org/reservation" >
      <m:reference>uuid:093a2dal-q345-739r-ba5d-pqff98fe8j7dc</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <m:passenger xmlns:n="http://mycompany.com/employees"
      env:mustUnderstand="true">
      <n:name>Åke Jøgvæn Øyvind</n:name>
    </m:passenger>
  </env:Header>
  <env:Body>
    <p:itinerary
      xmlns:p="http://travelcompany.org/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:seatPreference/>
      </p:return>
    </p:itinerary>
    <q:lodging
      xmlns:q="http://travelcompany.org/reservation/hotels">
      <q:preference>none</q:preference>
    </q:lodging>
  </env:Body>
</env:Envelope>
```



## SOAP conversation Example (II)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.org/reservation">
      <m:reference>uuid:093a2dal-q345-739r-ba5d-pqff98fe8j7dc/m:reference</m:reference>
      <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
    </m:reservation>
  </env:Header>
  <m:passenger xmlns:m="http://mycompany.com/employees"
    env:mustUnderstand="true">
    <n:name>Åke Jøgvann Øyvind</n:name>
  </m:passenger>
</env:Header>
<env:Body>
  <p:itineraryClarification
    xmlns:p="http://travel.org/reservation/travel">
    <p:departure>
      <p:departing>
        <p:airportChoices>
          JFK LGA EWR
        </p:airportChoices>
      </p:departing>
    </p:departure>
    <p:returns>
      <p:arriving>
        <p:airportChoices>
          JFK LGA EWR
        </p:airportChoices>
      </p:arriving>
    </p:returns>
  </p:itineraryClarification>
</env:Body>
</env:Envelope>
```



## SOAP conversation Example (III)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.org/reservation">
      <m:reference>uuid:093a2dal-q345-739r-ba5d-pqff98fe8j7dc/m:reference</m:reference>
      <m:dateAndTime>2001-11-29T13:36:50.000-05:00</m:dateAndTime>
    </m:reservation>
  <m:passenger xmlns:m="http://mycompany.com/employees"
    env:mustUnderstand="true">
    <n:name>Åke Jøgvann Øyvind</n:name>
  </m:passenger>
</env:Header>
<env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.org/reservation/travel">
    <p:departure>
      <p:departing>LGA</p:departing>
    </p:departure>
    <p:returns>
      <p:arriving>EWR</p:arriving>
    </p:returns>
  </p:itinerary>
</env:Body>
</env:Envelope>
```



## SOAP protocol binding

- ◆ HTTP GET method
  - ◆ SOAP Response message exchange pattern
  - ◆ Used when an application is assured that the message exchange is for the purposes of information retrieval, where the information resource is "untouched" as a result of the interaction.
  - ◆ Interactions are referred to as safe and idempotent in the HTTP specification
- ◆ HTTP POST method
  - ◆ SOAP Request / Response message exchange pattern
  - ◆ HTTP Content-type header "application/soap+xml"
- ◆ E-mail
  - ◆ Identical to HTTP POST method
  - ◆ Asynchronous
  - ◆ SMTP protocol RFC2822

## Using SOAP for RPC

- ◆ SOAP defines a representation for RPC invocations
- ◆ Detailed definition of
  - ◆ Target Node, Procedure and method names
  - ◆ identities, types and argument values
  - ◆ Data / context / status to be carried in header blocks

## SOAP data types (I)

- ◆ Same basic types as for XML-RPC
  - ◆ int, boolean, double, string, date/time, base64
- ◆ References (to the same object in memory)

```
<value xsi:type="xsd:int" id="v1"> 42 </value>  
<value href="#v1" />
```

- ◆ **Structs**
  - ◆ SOAP structs define a set of name value pairs. Structs can be named.

## SOAP Arrays

- ◆ SOAP arrays define a grouping of elements with no limitation mixing data types like integers and strings within the same array. Arrays can be named.
  - ◆ Access by ordinal position in the group (structs by name)
  - ◆ ArrayType attribute to specify which types occur where in the array
  - ◆ Multidimensional arrays possible
  - ◆ Handling of sparse arrays

## SOAP Array Examples

*1-dim, 3 entries*

```
<someArray xsi:type="SOAP-ENC:Array"
           SOAP-ENC:arrayType="se:string[3]">
  <se:string> Joe </se:string>
  <se:string> John </se:string>
  <se:string> Louis </se:string>
</someArray>
```

*2-dim, sparse: 2 entries*

```
<names xsi:type="SOAP-ENC:Array"
       SOAP-ENC:arrayType="xsd:string[10,10]">
  <name SOAP-ENC:position="[2,5]"> Guido </name>
  <name SOAP-ENC:position="[4,2]"> Jim </name>
</names>
```

## SOAP data types (II)

### ◆ Array of Bytes

- ◆ Rules for an array of bytes are similar to those for a string.
- ◆ Containing element of the array of bytes value MAY have an "id" attribute. Additional accessor elements MAY then have matching "href" attributes."

### ◆ Enumerations

- ◆ A list of distinct values appropriate to the base type
- ◆ All simple types except boolean.
- ◆ XML Schema Part 2: Datatypes  
<http://www.w3.org/TR/xmlschema-2/>



## SOAP data types (III)

### ◆ Polymorphic Accessors

- ◆ An accessor "...that can polymorphically access values of several types, each type being available at run time. A polymorphic accessor instance **MUST** contain an "xsi:type" attribute that describes the type of the actual value."

```
<cost xsi:type="xsd:float">29.95</cost>
```

- ◆ Similar to COM/DCOM "Variant"

### ◆ User Defined Data-Types

- ◆ Developers can define their own simple, or complex, data types.

## SOAP Faults

- ◆ model for handling faults in the processing of a message

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode><env:Value>rpc:BadArguments</env:Value></env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing error</env:Text>
      </env:Reason>
      <env:Detail>
        <e:myFaultDetails xmlns:e="http://travelcompany.org/faults">
          <e:message>Name does not match card number</e:message>
          <e:errorCode>999</e:errorCode>
        </e:myFaultDetails>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

## Web Service Description Language

- ◆ WSDL specification
  - ◆ <http://www.w3.org/TR/wsdl>
- ◆ Describes the abstract interface of a web service and the details how a specific web service has implemented it
  - ◆ “WSDL defines an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.”

## WSDL Service (I)

- ◆ Services are defined using six elements:
  - ◆ **Service**: used to aggregate a set of related ports
  - ◆ **Binding**: specifies protocol and data format specifications for the operations and messages defined by a particular portType
  - ◆ **Port**: specifies an address for a binding, thus defining a single communication endpoint.
  - ◆ **PortType**: set of abstract operations. Each operation refers to an input message and output messages.
  - ◆ **Message**: definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
  - ◆ **Types**: which provides data type definitions used to describe the messages exchanged.

## WSDL Example

- ◆ One function

```
int SumNumbers (int a, int b);
```

- ◆ Published at

```
http://myserver.org/myservice
```


- ◆ Gives

```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions>
  <wsdl:types> ... </wsdl:types>
  <wsdl:message name="SumNumbersSoapIn"> ... </wsdl:message>
  <wsdl:message name="SumNumbersSoapOut"> ... </wsdl:message>
  <wsdl:portType name="ServiceSoap"> list of messages </wsdl:portType>
  <wsdl:binding name="ServiceSoap"> protocol </wsdl:binding>
  ...
  <wsdl:service name="Service">
    <wsdl:port name="ServiceSoap">
      <soap:address location="http://myserver.org/myservice" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

## WSDL Type information

- ◆ Defines exact parameters for the call

```
<wsdl:types>
  <s:element name="SumNumbers">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="a" type="s:long" />
        <s:element minOccurs="1" maxOccurs="1" name="b" type="s:long" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="SumNumbersResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="SumNumbersResult"
          type="s:long" />
      </s:sequence>
    </s:complexType>
  </s:element>
</wsdl:types>
```



## Using a Web Service

```

Shell - Konsole <Z>
Session Edit View Settings Help
pcitap113:pfeiffer >
pcitap113:pfeiffer > python2.2
Python 2.2.2 (#1, Jan 30 2003, 21:26:22)
[GCC 2.96 20000731 (Red Hat Linux 7.3 2.96-112)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
...
.pythonrc executed
>>>
>>> import WebService
>>> AirportWeather = WebService.ServiceProxy("http://live.cernscience.com/wsdl/GlobalWeather.wsdl")
>>>
>>>
>>> for key in AirportWeather.methods.keys():
...     print key
...
searchByCountry
searchByRegion
isValidCode
getStation
listCountries
searchByCode
searchByName
getWeatherReport
>>> nodes=AirportWeather.getWeatherReport("GVA")
>>> len(nodes)
53
>>>

```


*Start Python*

*Create a proxy and connect to service*

*List the methods available from this service*

*Get the weather for Geneva airport (GVA)*

104 CERN School of Computing 2006 Information Technology Division  
Internet Services Group



## Limits of WSDL

- ◆ WSDL provides all the info on how to interact with a service to the consumer
- ◆ How to find what services are there ?
  - ◆ UDDI, Universal Description, Discovery and Integration project

105 CERN School of Computing 2006 CERN  
Information Technology Division  
Internet Services Group

## UDDI

- ◆ Universal Description, Discovery and Integration
  - ◆ <http://www.uddi.org/>
- ◆ Global network of linked registries
- ◆ List of provider of web services
  - ◆ `<businessEntity>`
    - ◆ Information on the company
    - ◆ List of services provided
- ◆ List of web services provided by `<businessEntity>`
  - ◆ `<businessService>`
    - ◆ Info and templates on how to bind to the service
- ◆ Info provided using WSDL

## Web services in HEP

- ◆ Distributed analysis (reconstruction)
  - ◆ E.g. Clarens <http://clarens.sourceforge.net/>
    - ◆ CMS distributed data server for remote analysis
    - ◆ Python with XML-RPC (and SOAP)
    - ◆ Interfacing to Grid services
  - ◆ Similar activities at SLAC
    - ◆ Using Java and Agents
- ◆ Just starting ...
  - ◆ Web services is the "standard" technology retained for all grid development

## Summary

- ◆ Web/network interface to application
  - ◆ Independent of language of implementation
  - ◆ "The Internet is the platform"
- ◆ Using XML for information exchange
  - ◆ Methods and data
- ◆ SOAP needs a rather complex "infrastructure"
  - ◆ WDSL, UDDI
- ◆ XML-RPC is more simple, less heavy
  - ◆ But follows development of SOAP

## Links

- ◆ WWW consortium
  - ◆ <http://www.w3.org/>
- ◆ XML-RPC
  - ◆ <http://www.xmlrpc.org/>
- ◆ SOAP
  - ◆ <http://www.w3.org/TR/soap/>