Web Services - Overview

Recuperación de Información 2007
Lecture 7.

Contents

- Motivation
- Concepts:
  - Web Services, Service-Oriented Architectures
- Roots: RPC, Distributed Computing
- Basic Architecture
- Standards:
  - SOAP
  - WSDL
- Conceptual Example
- Real Example
  - Build/deploy/call Web services with Apache AXIS

Motivation

From static to dynamic

http://www.renfe.es

http://amazon.com

Current Web pages offer not only static data but also dynamic services, e.g. bying books, booking hotels, bying train tickets, etc.

- Question: Can we automatize service usage in a similar way as aggregation/querying of static data?
- Just like data integration, making applications and software components interoperable/combinable is not a new issue in the IT landscape... keyword: "Middleware"!

Services on the Web: What's next?

- Applications
  - E-marketplaces.
  - Open, automated B2B e-commerce.
  - Business process integration on the Web.
  - Resource sharing, distributed computing.
- Current approach is *ad-hoc* on top of existing standards.
  - e.g., application-to-application interactions with HTML forms, screen scraping/wrapper technologies.
- Goal: **enabling systematic+standardized application-to-application interaction on the Web.**

Concepts: Web Services

"Web services" are an effort to build a distributed computing platform for the Web.

Yet another one?! (CORBA, RMI, etc.)

NO! Provide a *family of standards* for communication and dynamic interactions of applications just as XML provides for static data exchange.

Designing Web Services I

- Goals
  - Enable universal interoperability.
  - Widespread adoption, ubiquity: fast!
  - Enable (Internet scale) dynamic binding.
    - Support a service oriented architecture (SOA).
  - Efficiently support both open (Web) and more constrained environments (within enterprise boundaries).

Designing Web Services II

- Requirements
  - Based on *standards*. Pervasive support is essential (critical mass!), neither CORBA nor any of its competitors has made it to an ubiquitous standard
  - Minimal amount of required infrastructure is assumed.
    - Only a minimal set of standards must be implemented.
    - web server, application server,internet access should do…
  - Very low level of application integration is expected.
    - But may be increased in a flexible way.
  - Focus on the messages and documents, not on specific programming languages or APIs.

Concepts

Web Services: Definitions

1) "Loosely coupled, reusable software components that encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols", *The Stencil Group*

2) Web service applications are encapsulated, loosely coupled Web "components" that can bind dynamically to each other, *F. Curbera*

3) "Web Services are a new breed of application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web Services perform functions, which can be anything from simple request to complicated business processes", *The IBM Web Services tutorial*

Common to all definitions:

- Components providing functionality
- Distributed
- Accessible over the Web

Scope

- Applications are (ideally) to be assembled from a set of appropriate Web Services and no longer to be written manually
- Web Services are often viewed like Remote Procedure Calls (RPCs) over the Web…
- … but have potential beyond this:
  - Open, modular, extensible standards
  - Combine e.g. with Semantic Web technologies (SAWSDL)
  - Data and control flow to be defined separately (BPEL)

Roots – RPC, Distributed applications

Inter Process Communication (IPC)

- Provide communication between processes
  - To transfer information
  - To invoke an action in a remote process (optionally running in a remote machine)

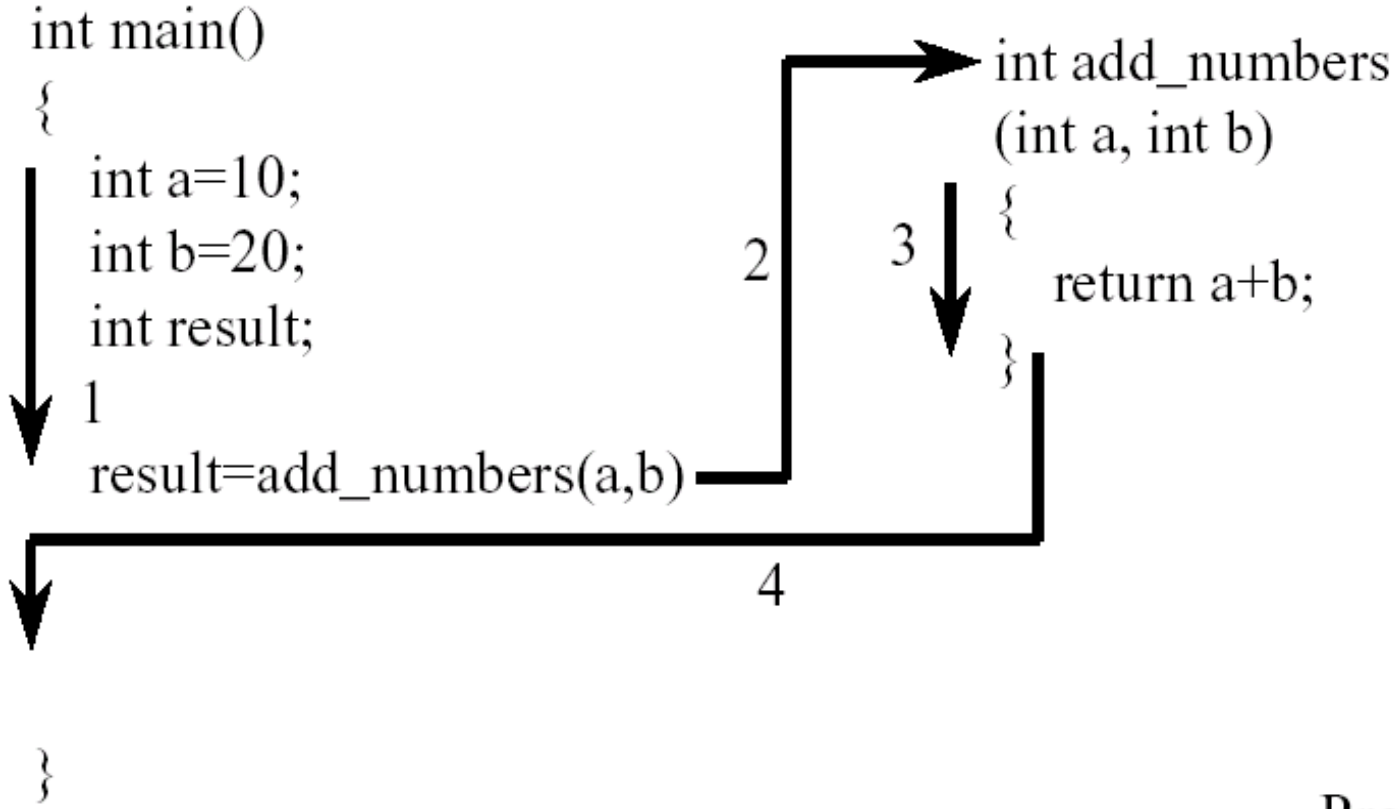
Taken from Chris Greenhalgh

Remote Procedure Call (RPC)

- High level representation of IPC remotely
- Uses the normal procedure call metaphor and programming style
- Calls a procedure in another process
- Hides/abstracts from underlying communication

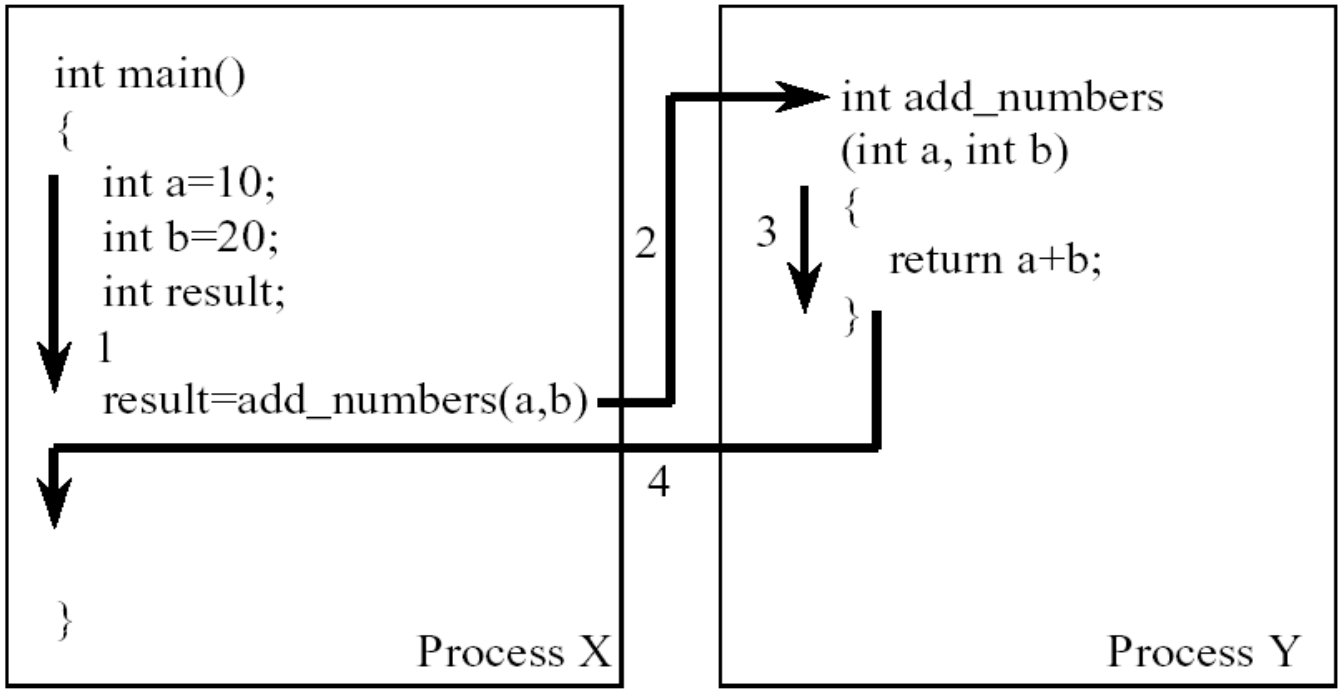
Taken from Chris Greenhalgh

A local procedure call



Taken from Chris Greenhalgh

A Remote procedure call

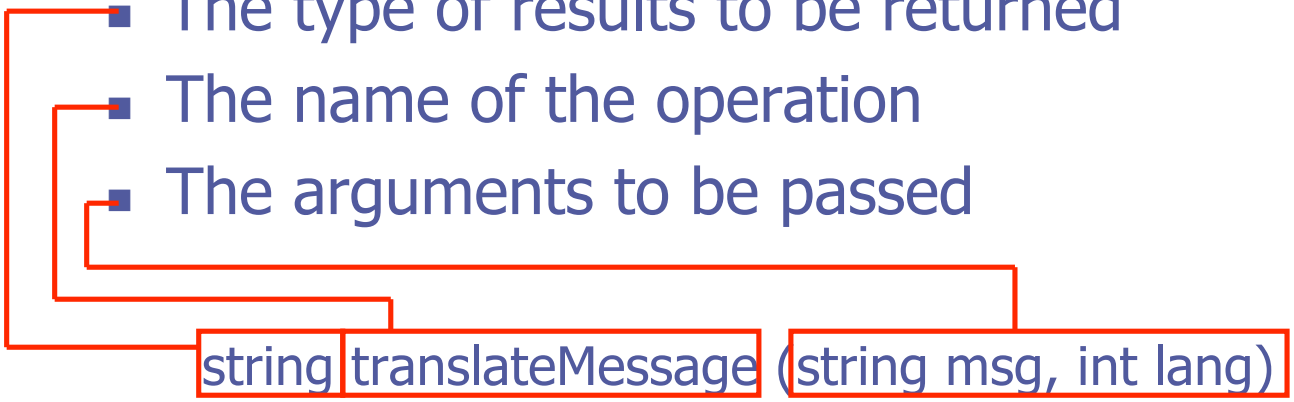


- Flow 1 and 3 are identical in both cases
  - but each must be a different thread/process
- Flow 2: call jumps from process X to Y
- Flow 4: return jumps from process Y to X

Taken from Chris Greenhalgh

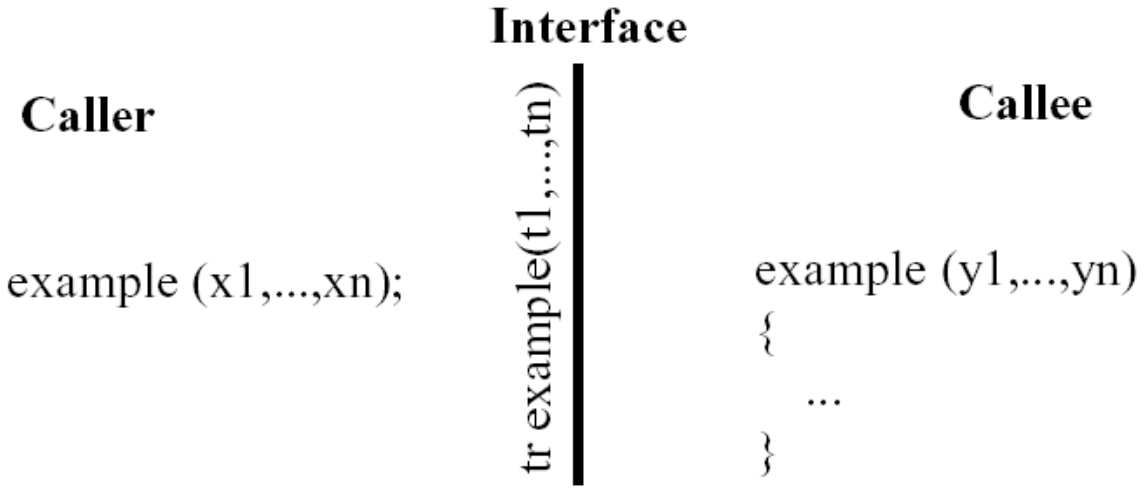
What is a procedure call ??

- A general abstraction mechanism in imperative programming languages
- Parameterized "language extension"
- Defined by an interface which specifies:
  - The type of results to be returned
  - The name of the operation
  - The arguments to be passed



Taken from Chris Greenhalgh

The procedure call interface



A procedure call must specify

- procedure name (***operation***)
- the ***arguments*** and
- ***return types***.
- Also it must specify ***where to locate*** *the procedure*

Taken from Chris Greenhalgh

Requirements:

- A layer of abstraction must hide the **communication protocol** to the programmer
- A **common interface definition language** must abstract from underlying programming language, computer (IDL, WSDL)
- "Look-and-feel" shall be similar to a local procedure call!

Implementation Principles

- Transparency
- Heterogeneity
- Concurrency
- Binding

Taken from Chris Greenhalgh

Transparency

- Hiding from the programmer whether a particular procedure call is local or remote, or which machine it runs on.
  - the programmer shouldn't need to "care" whether a call was local or remote
  - or on which machine it runs

Taken from Chris Greenhalgh

Heterogeneity

- Allowing RPCs between processes on different machines
  - Different operating systems
  - Different data representations
    - Integer and other type sizes and representations
    - Character sets
  - Different underlying programming languages, etc.

Taken from Chris Greenhalgh

Concurrency

- An RPC has two threads (flows of control)
  - Caller thread
  - Callee thread
- During RPC the caller thread could either:
  - Block, awaiting the result of the RPC (**synchronous**)
  - Continue with other operations until the result is returned (**asynchronous**)
    - Need some way to "rejoin" results

Taken from Chris Greenhalgh

Binding

- A local procedure call is executed in the local process:
  - Linker matches procedure names
  - Fills in process-local address of procedure
- But: An RPC allows us to call a procedure in another process:
  - How is an appropriate remote procedure located
  - How do we specify (i.e. name / context) it?

  We distinguish: ***static*** and ***dynamic*** binding

Taken from Chris Greenhalgh

Making a local procedure call

- A local procedure call
  - Preserves the return address
  - Copies the arguments onto the stack
  - Allocates space for the return results
  - Moves the program counter to the start of the called function
    - Which executes to completion and copies a return value into registers or onto the stack
  - Tidies up afterwards (pop old address from stack & continue)

Taken from Chris Greenhalgh

Making a remote procedure call - Overview

- Concepts
- The client & server stubs
- Results
- Flow of control
- Distributed objects

Taken from Chris Greenhalgh

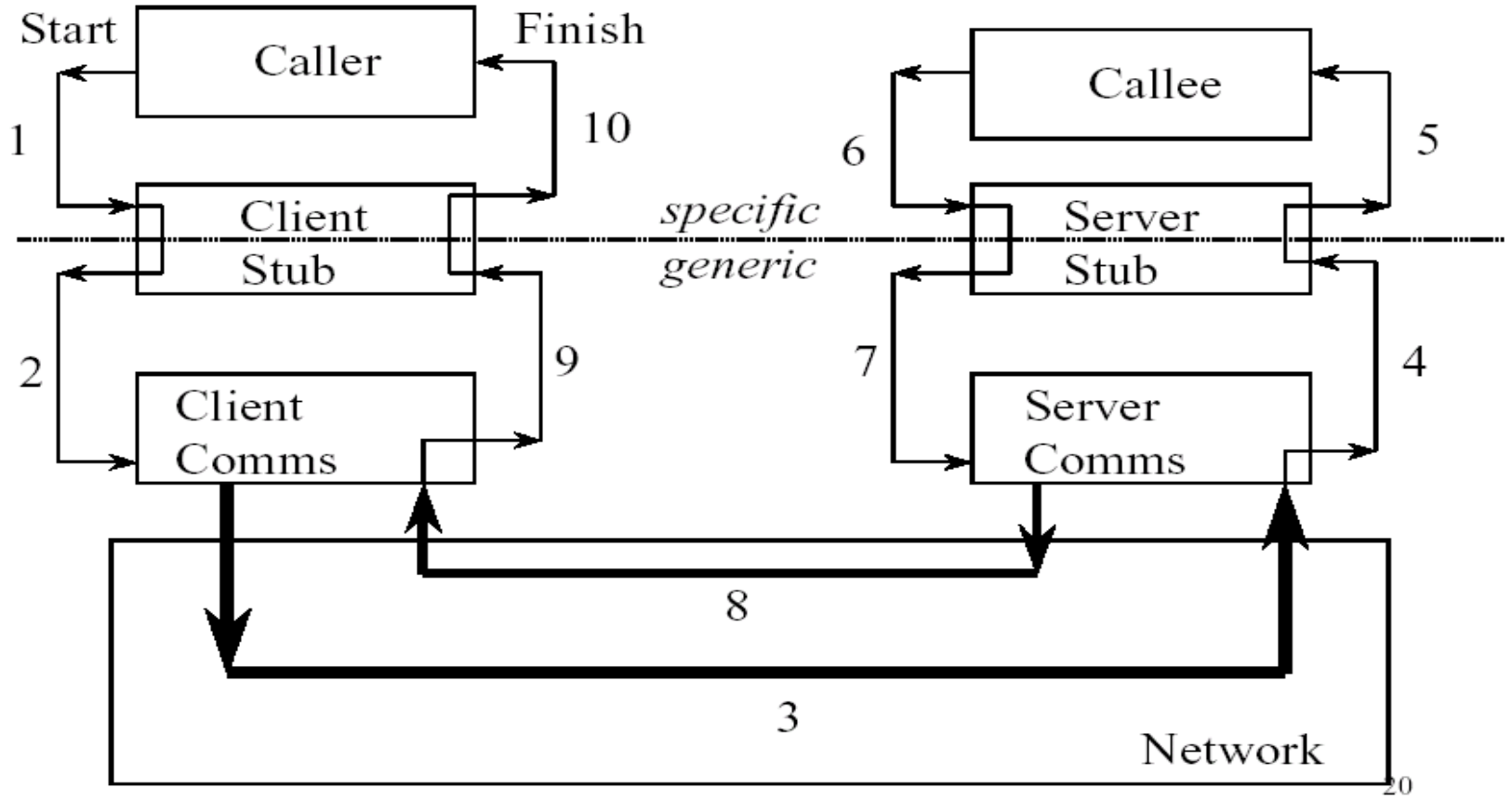
Concepts

- **Stub** is a local object which acts as a proxy for the remote service. When the stub is created before runtime, it is usually called a *static stub* (static binding)
- **Dynamic Invocation Interface.** Allows a client to call a remote procedure even if the signature of the remote procedure or the name of the service are unknown until runtime (dynamic binding)

Remark: *Dynamic binding can be combined with* ***name and directory services*** *which find appropriate bindings at runtime, e.g. useful for redundancy or load-balancing. There are respective extensions for RPC, as we'll see, UDDI provides the web service counterpart for this.*

Taken from Chris Greenhalgh

Stub-based Remote procedure call



Taken from Chris Greenhalgh

The client stub

- The caller's auxiliary operations are gathered into a **client stub** (1)
- The stub makes the required preparations and the passes the data to the processes' communications function (2)
- The communication function passes a stream of bytes to the remote process, which is listening for them (3)

Taken from Chris Greenhalgh

The server stub

- Server communication function listens for incoming communications (3)
- Collects the incoming data, derives the procedure being called, and passes the data to the stub for that procedure (4)
- The stub then calls the actual procedure code with the received arguments (5)

Taken from Chris Greenhalgh

Results

- Procedure calls results on the server side are returned to server stub (6)
- The server stub the passes the results to the server comm (7) which sends them to the client comm as a byte stream (8)
- Client comm returns the values to the waiting client stub (9) which returns the value to the calling function (10)

Taken from Chris Greenhalgh

Extensions of RPC

- Distributed Objects
- Asynchronous RPC
- Message queues
- Transaction processing monitors (transaction management)
- etc.

Distributed Objects

- Remote procedure call becomes

  Remote method invocation (RMI)
  - i.e. methods can be invoked on objects which are in another process
- Access and location transparency:
  - Local objects references are ultimately obtained by instantiating an object (or loading a class) in the local process.
  - Remote object references are obtained from the distributed object system itself.

Taken from Chris Greenhalgh

Distributed object system

- Often called “middleware” or “run-time infrastructure”:
  - Provide the glue to link objects in different process/on different machines
  - May provide additional services and support
- Examples:
  - CORBA
  - DCOM
  - Java RMI

Taken from Chris Greenhalgh

Implementing Distributed objects

- Like RPCs…
- Relies on interface definitions
  - e.g. CORBA IDL, Java RMI interface bytecode
    (Web Services: WSDL!)
- Defines what a client can ask
  - Client-middleware interface (methods signature)
- Defines what a server can do
  - middleware-server interface
- Middleware handles requests in generic, portable form

Taken from Chris Greenhalgh

Requirements & History…

- Having a service-oriented architecture and redefining middleware protocols is not sufficient to address application integration problems in a general way, unless these languages and protocols become standardized and widely adopted.
- Standards e.g. set by OASIS or W3C
- „WEB services“ -> Web = promises high degree of standardization.

- Earlier "Standardization" attempts before lack simplicity or generality:

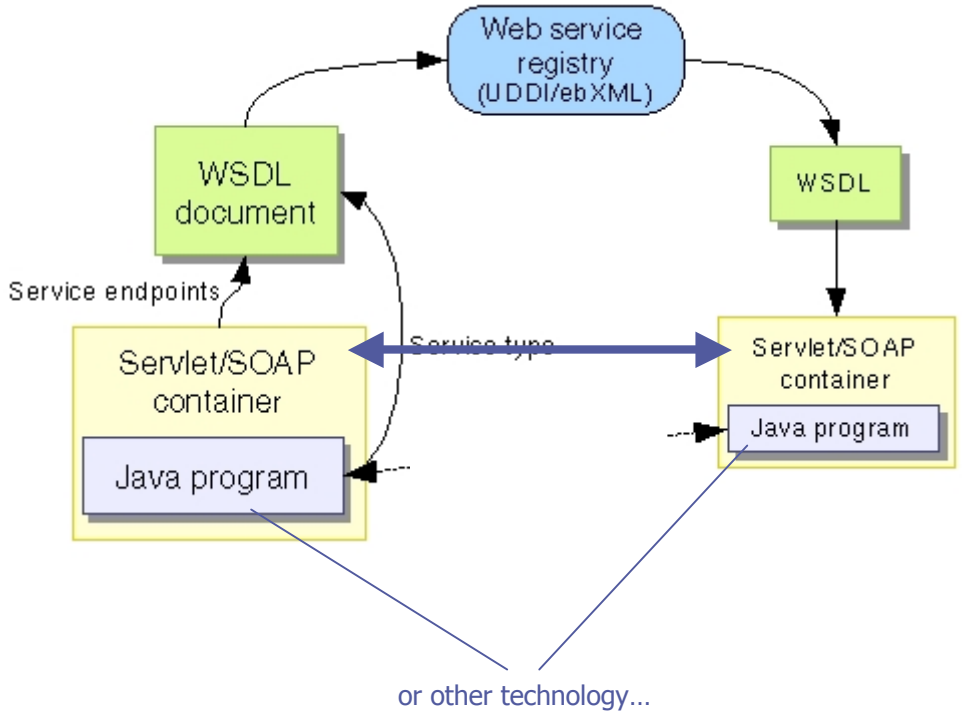
  - CORBA is platform- and language independent, but not simple.
  - RMI is simple and platform independent, but not language independent

Basic Architecture

Down to the basics: What infrastructure do Web Services need?

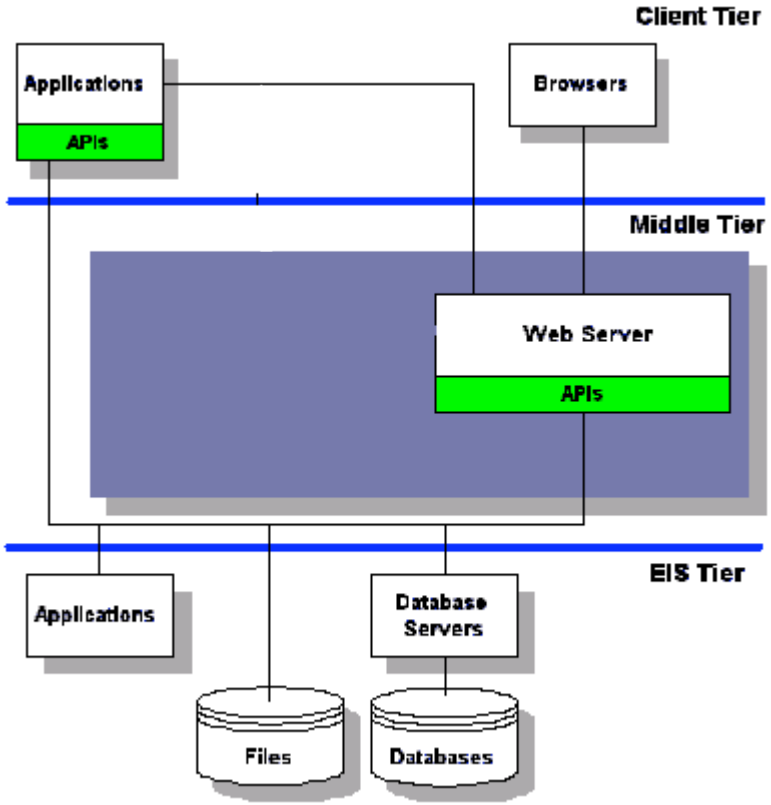
- Standard Protocols
  - usually SOAP messages or some other binding (e.g. HTTP get).
- An interface definition language (WSDL)
- Implementations:
  - Can be normal Java classes/programs
  - usually deployed in an in the Web container of the Web Server(application server, e.g. Tomcat)
  - special APIs development kits (e.g. AXIS)
  - reusing existing Web server infrastructure
  - Sometimes counted into the technology stack: directory infrastructure (e.g. UDDI)

Web Service Description



Taken from Chris Greenhalgh

General Structure of Web applications



Web Servers/Application Servers let you access
Services and applications via different interfaces (human-readable HTML, machine-reradable SOAP, etc.)

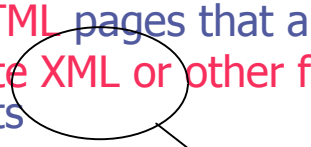
Databases, File systems, etc are accessible through APIs located in the middle tier. JDBC, SQLJ, or JDO API

Web server or container

- A Web application runs within a Web container of a Web server. The Web container provides the runtime environment through components that provide naming context and life cycle management.

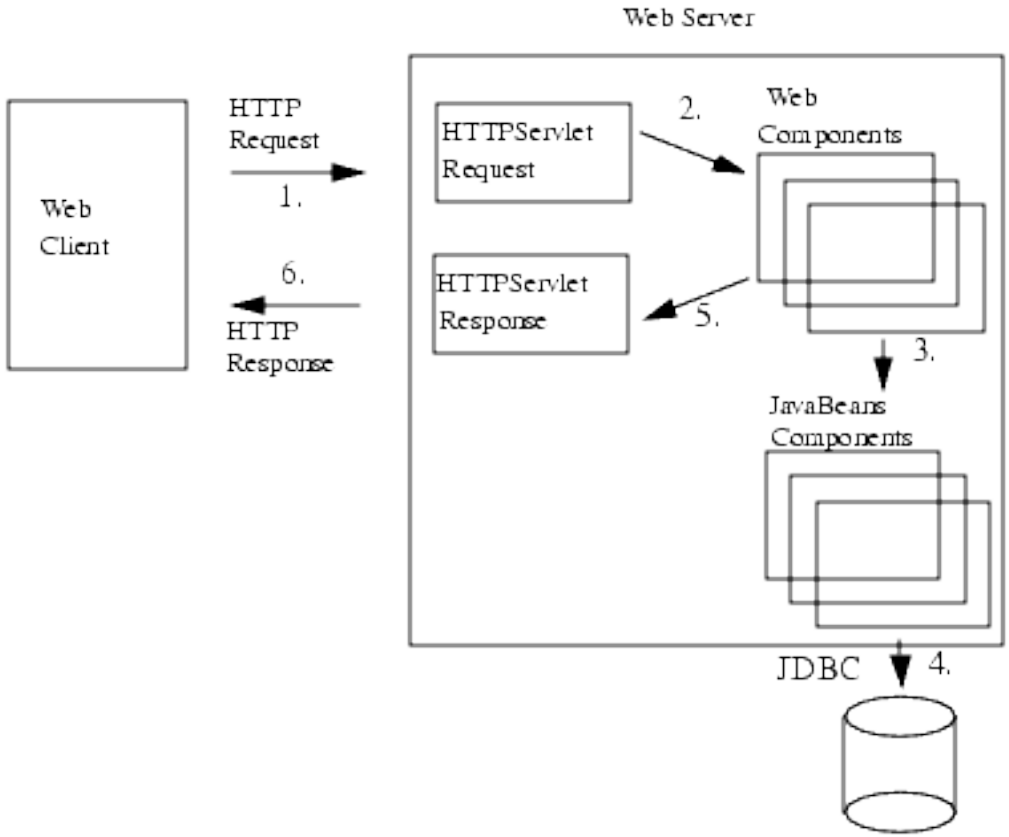
- General Web applications are not only web services but can be composed of arbitrary web components and other data such as HTML pages.
- Web components can be servlets, JSP pages created with the JavaServer Pages™ technology, web filters, web event listeners, etc. These components typically execute in a web server and may respond to HTTP requests from web clients.

- Servlets, JSP pages, and filters
  - may be used to generate HTML pages that are an application's user interface.
  - may also be used to generate XML or other format data that is consumed by other application components



We'll see, this is exaclty what Web Service technology is providing: XML based Messaging standards → SOAP

General Structure of the Web Server



General Structure of the Web Server

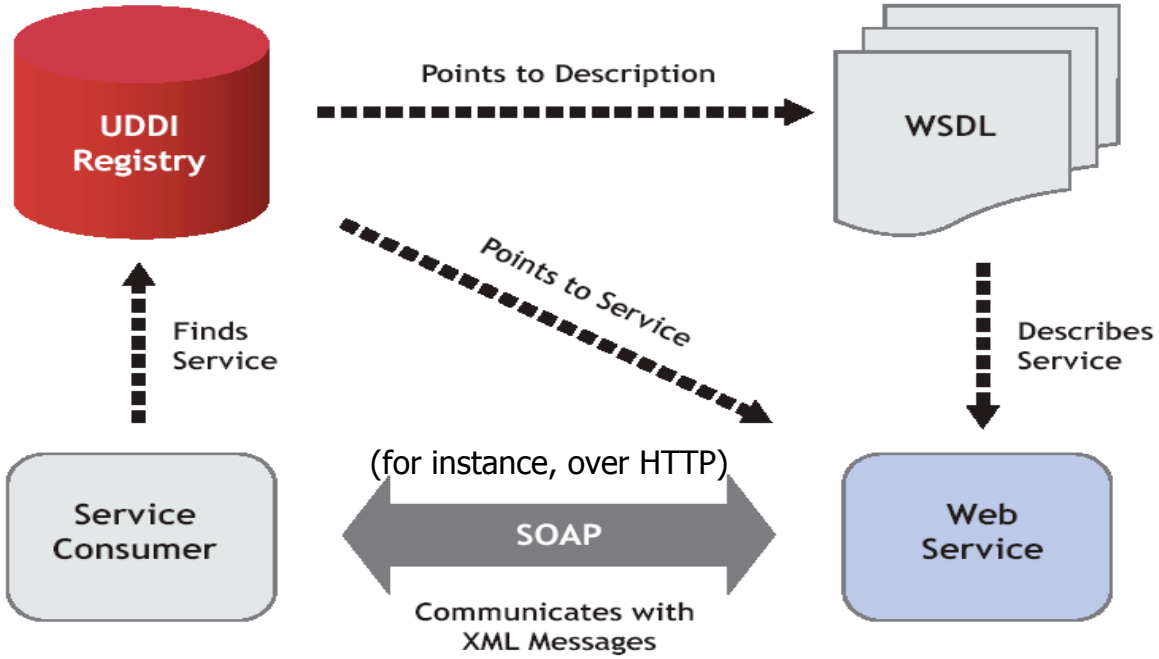
- Sample Scenario (without web services):
  - The client sends an HTTP request to the Web server. A Web server that implements Java Servlet and JavaServer Pages technology converts the request into an HTTPServletRequest object.
  - This object is delivered to a Web component which may interact with JavaBeans components or a database to generate dynamic content.
  - The Web component may then generate an HTTPServletResponse or it may pass the request to another Web component.
  - Eventually, the response is generated, and Web server returns it to the client.

Web Services Technologies…

… instantiate this basic infrastructure for direct app-to-app communication.

Sample scenario:

1. service consumer searches UDDI directory for appropriate service.
2. Gets WSDL description of service returned
3. WSDL contains all necessary info to bind (dynamic or static) to particular service/operation.
4. Invocation over agreed protocol (e.g. SOAP over HTTP)



Standards Requirments:

- Transport Standards
  (common protocol)
- Message Standards
  (common message syntax)
- Description Standards
  (common interface descriptions)
- Repository/Discovery Standards
  (common registry to find other services)
- Process description standards:
  (common standards to describe interaction protocols, control and data flow, etc.) BPEL4WS, WSCI, etc.
- Further topics standards: Security and Trust, …

Transport Standards

- **HTTP (HyperText Transfer Protocol).**
  Specifies the ASCII messages clients may send to serves and what they get in return. Supports persistent connections. Port 80.
- **SMTP (Simple Mail Transfer Protocol).**
  Simple ASCII based protocol to deliver mail messages. Error reporting if error occurs. Port 25.
- **FTP (File Transfer Protocol).**
  File sharing protocol, based on data transferring in a reliably and efficiently way, Port 21.

Message Standards

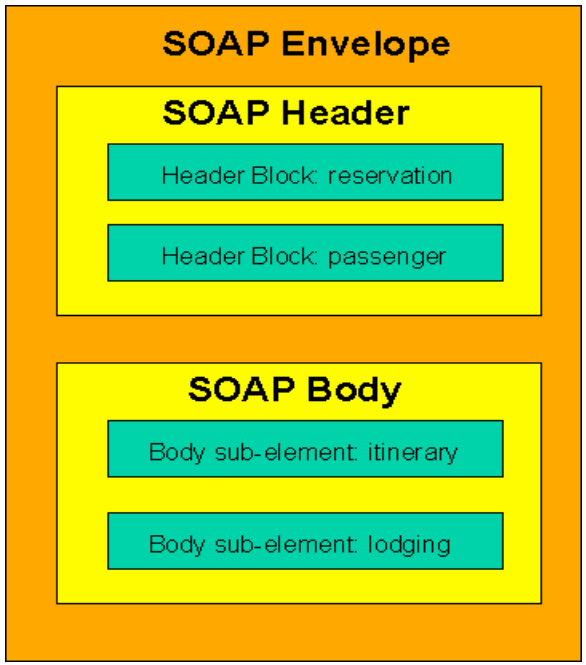
- Permits to get data from one place to another over the network.
- Web service need to publish interfaces and represent data in an abstract way.

- **SOAP** is the most used protocol to access Web services and interchange information in a Web environment.

  → Idea: a standard XML base format transferred over HTTP.

SOAP (Simple Object Access Protocol)

- **XML Messaging Protocol** that allows software running on disparate operating systems, and environments to make RPC. Stateless, one-way message exchange paradigm.



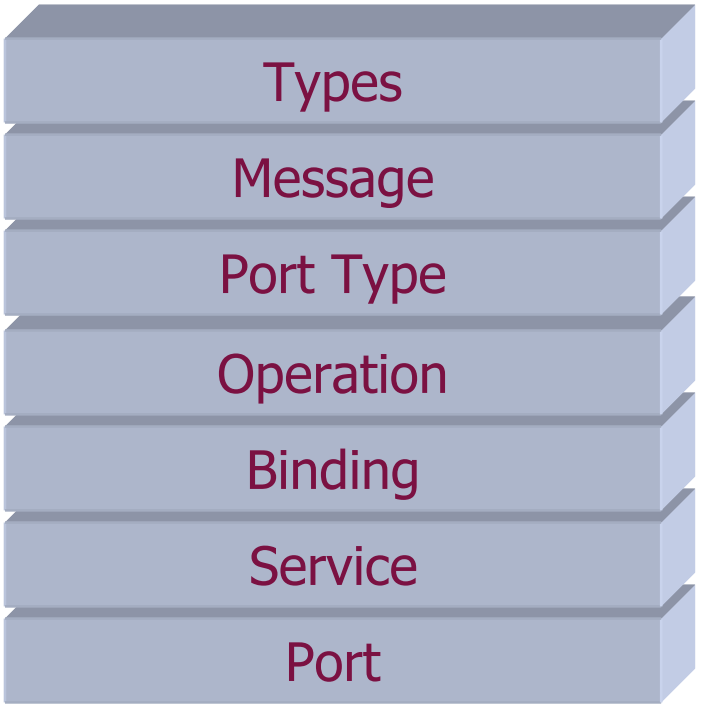
- **Envelope**: main end-to-end information
- **Header**: pass information in SOAP messages that is not application payload.
- **Body**: main end-to-end information, rely on XSD

Description Standards

- Service requestor checks services to choose the one that fulfills its requirements
- Web service must facilitate information regarding its operational capabilities (functional and non-functional requirements (e.g. transport and messaging protocols, network endpoint, price, time-cost, etc)

WSDL (Web Service Description Language)

- A WSDL document defines **services** as collections of network endpoints (**ports**)



Interface

Note that:
-This doesn't say anything about what services can do, nor how they do it.
- Just defines messages and endpoints in a machine understandable and processable way, i.e. how to access the service

endpoint

Repository/Discovery Standards

- **UDDI** provides a service registry and API to locate and search web services. A UDDI registry can be thought of as a DNS for business applications.

Authenticated set of operations that allow organizations to **publish** businesses, services, service type specifications

Non authenticated public set of operations that allows users to **extract** information out of the UDDI registry.

- **Classification** of businesses and services according to standard taxonomies
- Can *link to* **WSDL** (or any other metadata)

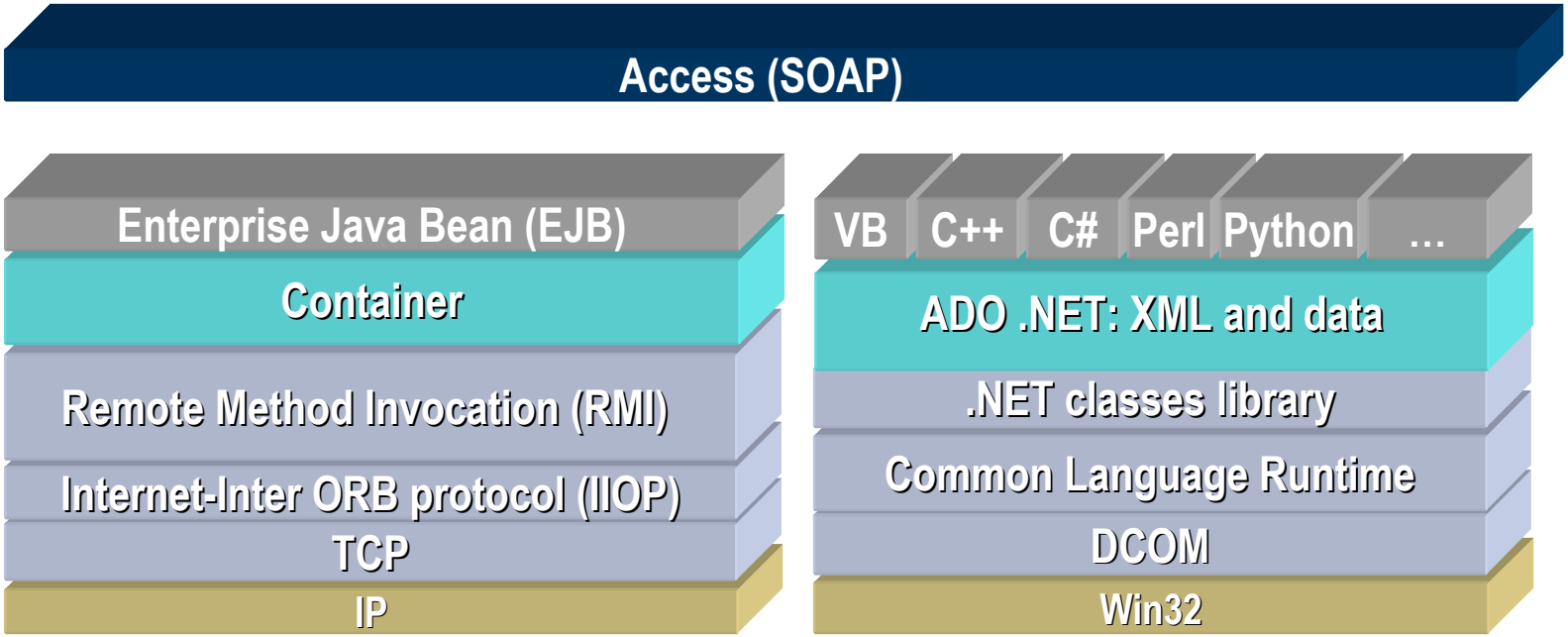
SOAP

SOAP
(Simple Object Access Protocol)

- SOAP is a simple XML based protocol to let applications exchange information over HTTP.
- Or more simple: SOAP is a protocol for accessing a Web Service.
- Messaging framework for exchanging XML formatted data over the network
- One-way and asynchronous protocol (only a message format)
- W3C recommendation: Version 1.2, June 2003

The Web Services Architecture: SOAP

Exactly speaking, web services build up ON TOP of all traditional distributed architectures, abstracting from and linking them together:



SOAP Building blocks

- A SOAP message is an ordinary XML document containing the following elements:

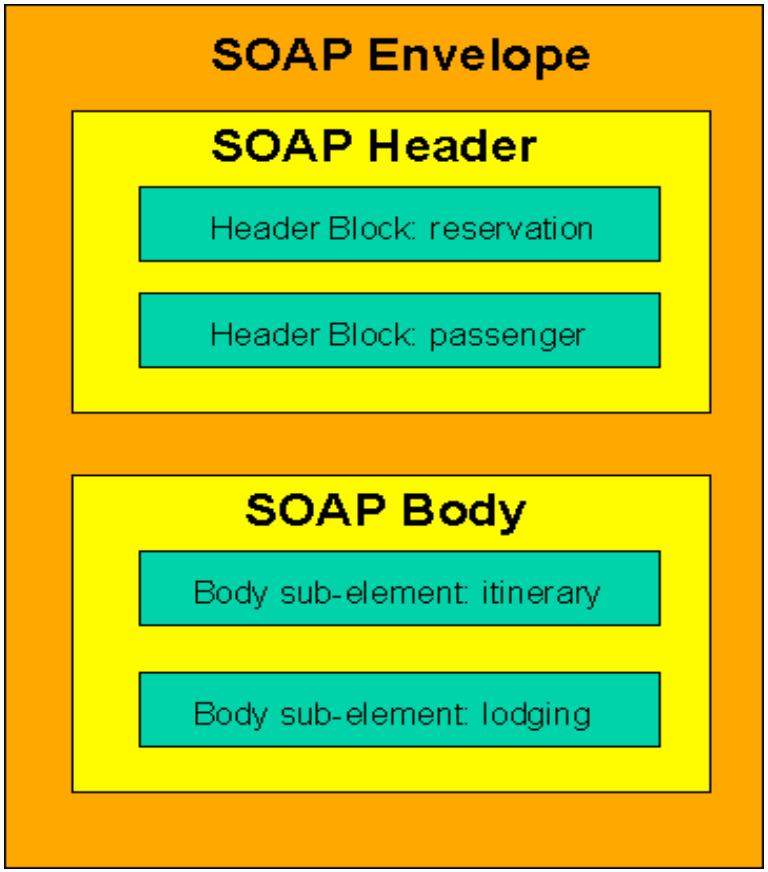
  - A *required* **Envelope** element that identifies the XML document as a SOAP message. Defines the start and the end of the message
  - An *optional* **Header** element that contains any optional attributes of the message, either at an intermediary point or at the ultimate end point
  - A required **body** element that contains call and response information. Contains the XML data comprising the message being sent
  - An optional **fault** element that provides information about errors that occurred while processing the message

SOAP Building blocks

- All the elements above are declared in the default namespace for the SOAP envelope:
  - http://www.w3.org/2001/12/soap-envelope
- and the default namespace for SOAP encoding and data types is:
  - http://www.w3.org/2001/12/soap-encoding

SOAP (Simple Object Access Protocol)

A travel booking message example…



SOAP Envelope

The **envelope**

- indicates the start and the end of the message so that the receiver knows when an entire message has been received,
- is the root element of a SOAP message. It defines the XML document as a SOAP message.

SOAP Envelope : xmlns:soap Namespace

- A SOAP message must always have an Envelope element associated with the "http://www.w3.org/2001/12/soap-envelope" namespace.
- If a different namespace is used, the application must generate an error and discard the message.

SOAP Envelope: Example

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=„http://www.w3.org/2001/12/soap-envelope
   SOAP-ENV:encodingstyle="http://www.w3.org/2001/12/soap-
       encoding /">
       ...
       Message information goes here
       ...
</SOAP-ENV:Envelope>
```

encodingstyle attribute … see below

The SOAP Header Element

- The optional SOAP Header element contains application specific information (like authentication, payment, etc) about the SOAP message.
- If the Header element is present, it must be the first child element of the Envelope element.
- It may include information about additional attributes.
- **Note:** All immediate child elements of the Header element must be namespace-qualified.

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>
```

SOAP attributes:

- `soap:encodingStyle="`*`URI`*`"` used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements.
- `soap:mustUnderstand="0|1"` indicates whether the recipient must process some message part or whether it can be ignored
- `soap:actor="`*`URI`*`"` Not all parts of the SOAP message may be intended for the ultimate endpoint of the SOAP message but, instead, may be intended for one or more of the endpoints on the message path. The SOAP actor attribute may be used to address the Header element to a particular endpoint. Kind of routing…

The SOAP Body Element

- The required SOAP Body element contains the actual application-defined XML intended for the ultimate endpoint of the message.
- So, elements within the body element are usually not part of the soap standard.
- Immediate child elements of the SOAP Body element may be namespace-qualified.
- SOAP defines one element inside the Body element in the default namespace ("http://www.w3.org/2001/12/soap-envelope"):
  - the SOAP Fault element, which is used to indicate error messages.

SOAP Body: 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:Body>
      <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
                  <m:Item>Apples</m:Item>
      </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

SOAP Body: Response 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:Body>
        <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
                <m:Price>1.90</m:Price>
        </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

The SOAP Fault Element

- optional
- An error message from a SOAP message is carried inside a **Fault** element.
- If a Fault element is present, it must appear as a **child element** of the Body element. A Fault element can only appear once in a SOAP message.

The SOAP Fault Element

- The SOAP Fault element has the following sub elements:

`<faultcode>`	A code for identifying the fault
`<faultstring>`	A human readable explanation of the fault
`<faultactor>`	Information about who caused the fault to happen
`<detail>`	Holds application specific error information related to the Body element

The SOAP Fault Codes

- The faultcode values defined below must be used in the faultcode element when describing faults:

`VersionMismatchFound:`
an invalid namespace for the SOAP Envelope.

`elementMustUnderstand:`
An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood

`Client:`
The message was incorrectly formed or contained incorrect information

`Server:`
There was a problem with the server so the message could not proceed

How SOAP binds to HTTP:

- A SOAP method is an HTTP request/response that complies with the SOAP encoding rules.
- **HTTP + XML = SOAP**
- A SOAP request could be an HTTP POST or an HTTP GET request.
- The HTTP POST request specifies at least two HTTP headers: Content-Type and Content-Length.

HTTP headers. Content Type

- The Content-Type header for a SOAP request and response defines the MIME type for the message and the character encoding (optional) used for the XML body of the request or response.

```
Content-Type: MIMEType; charset=character-encoding
```

- Example

```
POST /item HTTP/1.1 Content-Type: application/soap+xml;
charset=utf-8
```

HTTP headers. Content Length

- The Content-Length header for a SOAP request and response specifies the number of bytes in the body of the request or response.

```
Content-Length: bytes
```

- Example

```
POST /item HTTP/1.1 Content-Type: application/soap+xml;
charset=utf-8 Content-Length: 250
```

SOAP Example

- In the example below, a GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter will be returned in the response. The namespace for the function is defined in "http://www.stock.org/stock" address.

SOAP Example. Request

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml; charset=utf-8 Content-
Length: nnn

<?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:Body xmlns:m="http://www.stock.org/stock">
        <m:GetStockPrice>
                <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

SOAP Example. Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?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:Body xmlns:m="http://www.stock.org/stock">
<m:GetStockPriceResponse>
                <m:Price>34.5</m:Price>
</m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

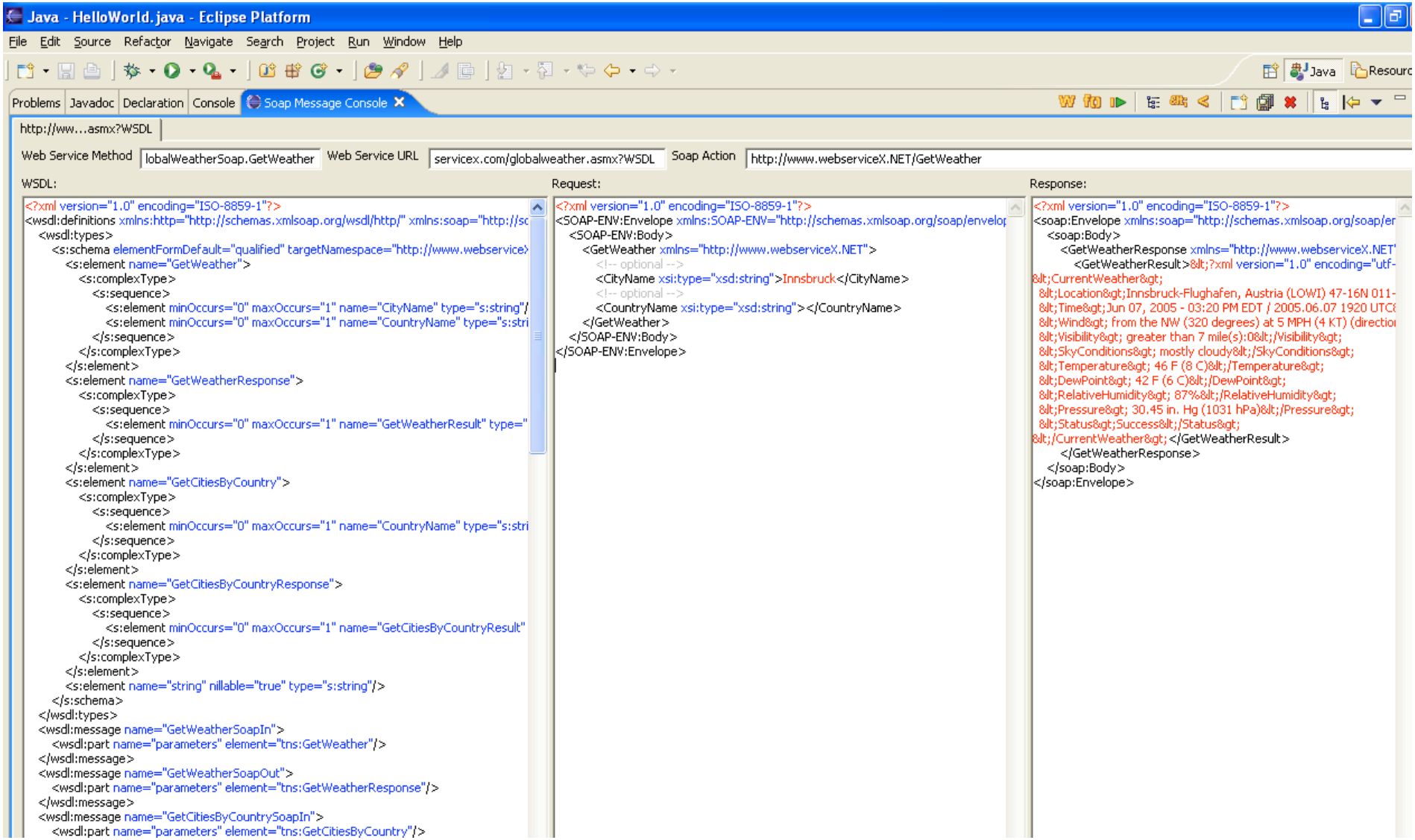
SOAP in one slide, summary:

- A SOAP message MUST be encoded using XML
- A SOAP message MUST use the SOAP Envelope namespace
- A SOAP message MUST use the SOAP Encoding namespace
- A SOAP message must NOT contain a DTD reference
- A SOAP message must NOT contain XML Processing Instructions
- A SOAP message contains the following elements:
  - A required Envelope element that identifies the XML document as a SOAP message
  - An optional Header element that contains header information
  - A required Body element that contains call and response information
  - An optional Fault element that provides information about errors that occurred while processing the message

Wanna watch a SOAP?

- You can use, e.g. a nice eclipse plugin:

  Web Service Console plugin: http://wscep.sourceforge.net/



WSDL

Web Services Description Language (WSDL)

- WSDL is an XML-based language for ***describing Web services interfaces*** and how to access them.
- What is WSDL?
  - WSDL stands for Web Services Description Language
  - WSDL is written in XML, ie. a WSDL description is an XML document
  - WSDL is used to
    - describe Web services
    - locate Web services
    - Automatically bind/invoke Web services
  - WSDL 1.1 (a W3C Note only!) is current defacto standard
  - WSDL 2.0 is a W3C working draft (last version 26 March 2007)

Web Services Description Language (WSDL)

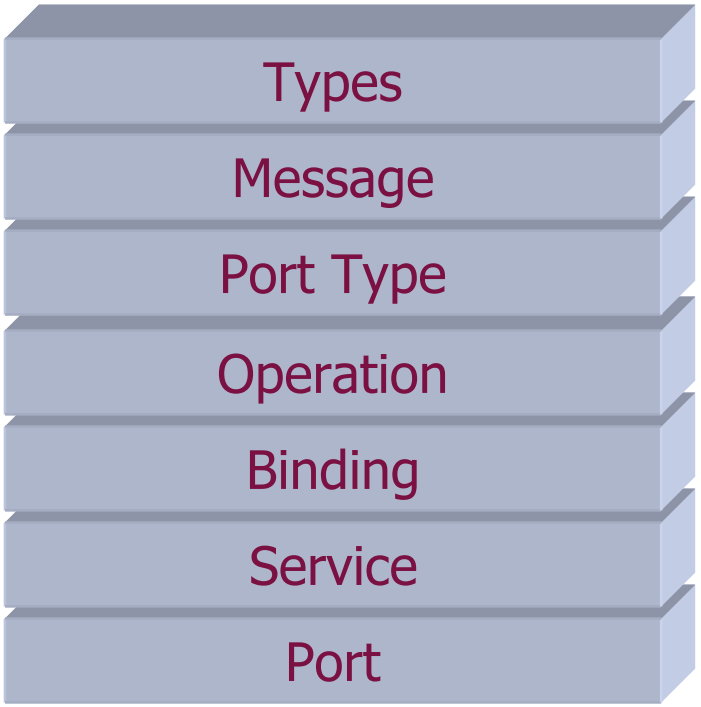
- Web services expose a software-oriented view of a business or consumer function with which applications may interact over the network
- To enable such an interaction, a web service must be described and advertised to its potential users. Description means:
  - What operations it provides
  - What data it expects to receive (input)
  - Which results should deliver (output)
  - What communication protocols or transport it supports (binding)

Web Services Description Language (WSDL)

- WSDL elements contain a **description of the messages**, typically using one or more **XML Schema**s, to be passed to the web service so that both sender and receiver understand the data being exchanged
- WSDL elements also contain a **description of the operations** that the service can perform on that data, so that the receiver of the message knows how to process it and a binding to a protocol or transport, so that the sender knows how to send it.
- Typically, WSDL is used with SOAP, and the WSDL specification includes a SOAP **binding**, but other bindings are allowed as well

WSDL (Web Service Description Language)

- A WSDL document defines **services** as collections of network endpoints (**ports**)



Interface

endpoint

Note however, that WSDL doesn't say anything about what services can do, nor how they do it.

WSDL Ports and Services

- Ports:
  - Ports are used to expose a set of operations or port types.
  - The port types can be grouped for one or more bindings
- Services:
  - A service encloses one or more portTypes
  - It includes some ports

WSDL (Web Service Description Language)

Overall structure of a typical WSDL file:

```
<definitions>
        <types>
        definition of types........
        </types>

        <message>
        definition of a message....
        </message>

        <portType>
        definition of a port types (operations).......
        </portType>

        <binding>
        definition of a binding....
        </binding>

        <service>
        relating ports (individual endpoint) and their
        bindings....
        </service>
</definitions>
```

WSDL Data types

- You can either use XSD simple types or optionally define new types using XML schema syntax in the types element:

```
<types>
   <schema … >
   <complexType name = "PurchaseOrder">
      <element name="NameofProduct" type="xsd:string"/>
      <element name="Price" type="xsd:integer"/>
   </complexType>
   …
   </schema>
</types>
```

WSDL message

- Individual Data Types are mapped into Messages
- Messages can be sent and received by the described web service
- Messages consist of one or more parts.

```
<message name =„PurchaseOrderRequest“ >
       <part name=„MyChristmasOrder“
             type=„xsd1:PurchaseOrder“>
</message>
```

Defined in `<types>`
Side remark: You can alternatively also refer to elements in the schema defined in types using the element=URI attribute

WSDL operation

- Operations perform the actions on the data
- Operations are defined so that the web service knows how to interpret the data and what, if any, data is to be returned on the reply
- Several types:
  - One-way: A message is sent without a requirement to return a reply (only input, or only output)
  - Two-way: A message is sent and the receiver must send a corresponding reply (request-response or solicit-response)

```
<operation name =„ProcessPurchaseOrder“ >
       <input message=„PurchaseOrder“ name=„MyOrder“>
       <output message=„PurchaseOrderResult“
              name=„ResultofMyOrder“>
</operation>
```

WSDL portType

- A logical grouping of operations
- The **<portType>** element is the most important WSDL element.
- It defines a web service, the operations that can be performed, and the messages that are involved.
- The **<portType>** element can be compared to a function library (or a module, or a class) in a traditional programming language.

Operation types

- The **request-response** type is the most common operation type, but WSDL defines four types:

One-way:
The operation receives a message but will not return a response
Request-response:
The operation can receive a request and will return a response
Solicit-response:
The operation can send a request and will wait for a response
Notification:
The operation can send a message but will not wait for a response

(not explicit, but implicit by the fact whether input/output are defined)

One-way Operation

- A one-way operation example:

```
<message name="newTermValue">
       <part name="term" type="xsd:string"/>
       <part name="value" type="xsd:string"/>
</message>

<portType name="glossaryTerms">
       <operation name="setTerm">
            <input name="newTerm" message="newTermValue"/>
       </operation>
</portType >
```

Request-Response Operation

- A request-response operation example:

```
<message name="HelloRequest">
        <part name="term" type="xsd:string"/>
</message>

<message name="HelloResponse">
        <part name="value" type="xsd:string"/> </message>

<portType name="glossaryTerms">
        <operation name="getTerm">
                <input  message="getTermRequest"/>
                <output  message="getTermResponse"/>
        </operation>
</portType>
```

Remark: WSDL provides some default values based on the operation name. If the name attribute is not specified on a one-way or notification message, it defaults to the name of the operation. If the name attribute is not specified on the input or output messages of a request-response or solicit-response operation, the name defaults to the name of the operation with "Request"/"Solicit" or "Response" appended, respectively.

WSDL portType

- A portType groups a set of operations:

```
<portType name=„PurchaseOrderPortType“>
      <operation name =„ProcessPurchaseOrder“ >
        <input message=„PurchaseOrder“ name=„MyOrder“>
        <output message=„PurchaseOrderResult“
                    name=„ResultofMyOrder“>
      </operation>
      <operation name =„CancelPurchaseOrder“ >
        <input message=„CancelPurchaseOrder“
            name=„CancelMyOrder“>
        <output message=„CancelPurchaseOrderResult“
            name=„CancelResultofMyOrder“>
      </operation>
</portType>
```

WSDL Binding

- A binding defines message format and protocol details for operations and messages defined by a particular PortType.
- The **<binding>** element defines the message format and protocol details for each port.
- The **soap:binding** element has two attributes - the style attribute and the transport attribute.
- The style attribute can be "rpc" or "document". In this case we use rpc. The transport attribute defines the SOAP protocol to use. In this case we use HTTP.

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
         <operation name="GetTradePrices">
            <soap:operation soapAction="http://example.com/GetTradePrices"/>
               <input>
                 <soap:body use="encoded" namespace="http://example.com/stockquote"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
               </input>
              <output>
               <soap:body use="encoded" namespace="http://example.com/stockquote"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
              </output>
        </operation>
</binding>
```

Service and Port

- A port defines an individual endpoint by specifying a single address for a binding (*Remark: WSDL then not re-usable for several services*)
- A service groups a set of related ports together

```
<wsdl:definitions .... >
   <wsdl:service .... >
      <wsdl:port name="nmtoken"  binding="qname">
      <-- extensibility element (1) -->
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

A complete example I

```
<?xml version="1.0"?><definitions
name="StockQuote"targetNamespace="http://bankTyrol.com/balance.wsdl"
xmlns:tns="http://bankTyrol.com/balance"
          xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
          xmlns:xsd1="http://bankTryol.com/schema"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns="http://schemas.xmlsoap.org/wsdl/">

   <message name="GetBalanceInput">
      <part name="creditCardNumber" element="xsd:string"/>
   </message>
   <message name="GetBalanceOutput">
      <part name="result" type="xsd:String"/>
   </message>

   <portType name="GetBalancePortType">
      <operation name="GetCurrentBalance" parameterOrder="creditCardNumber">
         <input message="tns:GetBalanceInput"/>
         <output message="tns:GetBalanceOutput"/>
      </operation>
   </portType>
```

A complete example II

```
   <binding name="GetBalanceSoapBinding" type="tns:GetBalancePortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
         <operation name="GetCurrentBalance">
            <soap:operation soapAction="http://bankTyrol.com/GetBalance"/>
               <input>
                  <soap:body use="encoded" namespace="http://bankTyrol.com/Balance "
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                   <soap:body use="encoded" namespace="http://bankTyrol.com/Balance "
                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
          </operation>
    </binding>

    <service name="GetBalanceService">
          <documentation>Service used by bankTyrol.com Inc.</documentation>
          <port name="GetBalancePort" binding="tns:StockQuoteBinding">
             <soap:address location="http://bankTyrol.com/balance"/>
          </port>
    </service>
</definitions>
```

WSDL 2.0: work in progress… still not ready ☺

- We covered WSDL 1.1 so far

  http://www.w3.org/TR/wsdl
  Namespace: http://schemas.xmlsoap.org/wsdl/

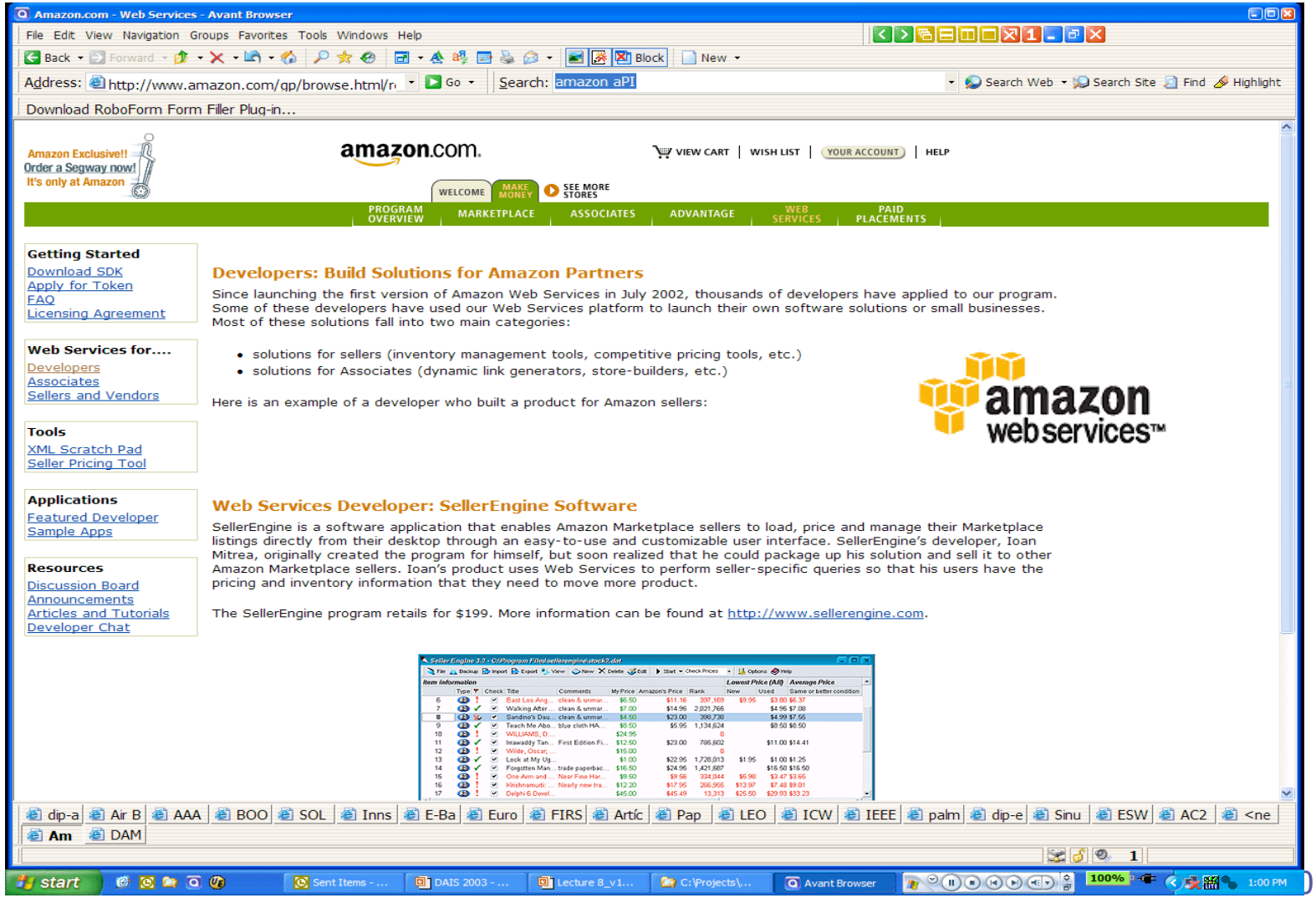
- Most recent version at W3C, WSDL 2.0:
  http://www.w3.org/TR/wsdl20/
  Is a ***proposed recommendation*** (last step before "official" W3C Recommendation) *since* ****yesterday**!**

- Some changes:
  - **portTypes** have been renamed to **interfaces**
  - **port**s have been renamed to **endpoint**s
  - WSDL 2.0 removes operation overloading
  - import other schema languages (DTD, etc.)
  - etc.
- See also: http://jacek.cz/presentations/2006-05-25-wsdl2.html

Real Examples

How where do I find available WS?

Amazon.com http://aws.amazon.com/



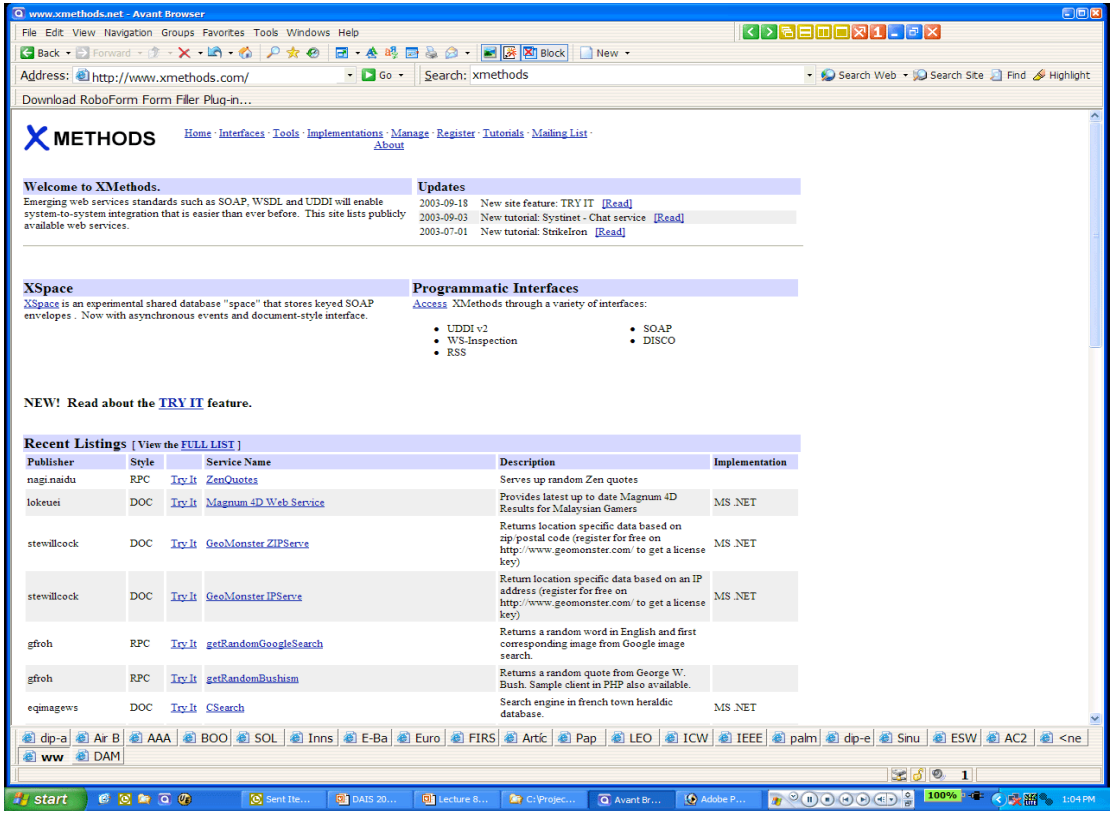
Google

- Just removed their Web Service API from http://www.google.com/apis/ :-(
- What does this show us:
  - Not everybody wants to have his/her services public!
  - … especially if they are commercially successful ;-)
- You can still try to fins some Web services using google…

  search for `filetype:wsdl ...`

  admittedly, maybe not the best way ;-)

Xmethods http://www.xmethods.net/



Remarkably, Web Service Technology is currently still rather used WITHIN companies than that there are really Many publically available Web Services !

References

- **[Aolonso et al. 2003]** G. Alonso, F. Casati, H. Kuno, V. Machiraju: Web Services: Concepts, Architectures and Applications
- **[SOAP 2003]** SOAP version 1.2 Part 0: Primer, W3C Recommendation 24 June 2003.
- **[UDDI 2002]** UDDI Version 3.0, published specification 19 July 2002, available at http://www.uddi.org/specification.html.72-85, January/February 2003.
- **[WSDL 2003]** Web Services Description Language (WSDL) Version 1.2. W3C Working Draft 3 March 2003.
- **[W3C 2002]** W3C: Web Service description requirements, http://www.w3.org/TR/ws-desc-reqs/, W3C working draft 28 October 2002.