

Web Services Technology



Paulo F. Pires & Marta Mattoso
UFRJ

paulopires@nce.ufrj.br marta@cos.ufrj.br

<http://genesis.nce.ufrj.br/dataware/hp/pires>

<http://www.cos.ufrj.br/~marta>

Summary

- What are Web services?
- Web services technologies:
 - SOAP
 - WSDL
 - UDDI
- Examples
 - Java
 - Delphi
- Architecting Web Services
- Applying Web Services Technology
- Web Services & Semantic Web



What are Web Services?



Today's Web

- Web designed for application to human interactions
- Served very well its purpose:
 - Information sharing: a distributed content library.
 - Enabled B2C e-commerce.
 - Non-automated B2B interactions.
- Automated interactions?
 - If you wanted to create an application to process a page, you could do it, but you would have to perform *HTML scraping*.

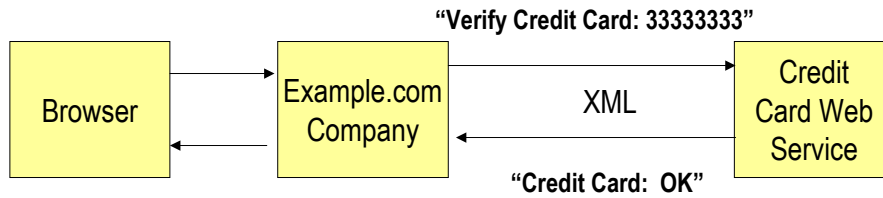
What is a Web Service?

- "...The Web can grow significantly in power and scope if it is extended to support communication between applications, from one program to another."
 - From the W3C XML Protocol Working Group Charter

What is a Web Service?

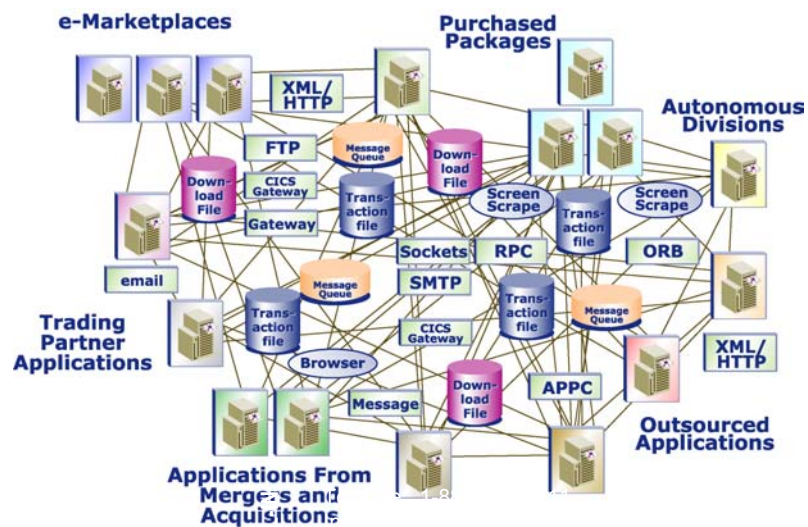
- any service that is made available over the web.
- any service that enables two computer applications to exchange data.
- primarily, but not exclusively based on:
 - XML for data encoding
 - HTTP for data transport

Example: Credit Card Verification



This functionality already exists. However, the goal of web services is to standardize the process, enabling "plug and play" functionality.

P2P Integration - Problems



Designing Web Services

■ Goals

- Enable universal interoperability.
 - Widespread adoption, ubiquity: fast!
 - Compare with the good but still limited adoption of the OMG's OMA.
 - Enable (Internet scale) dynamic binding.
 - Support a service oriented architecture (SOA).
 - Efficiently support both open (Web) and more constrained environments.
 - Diminish the pain of incompatibility (languages, operating systems, & network protocols)
- **Web services** is an effort to build a distributed computing platform for the Web

Designing Web Services

■ Requirements

- Based on standards. Pervasive support is critical.
- Minimal amount of required infrastructure is assumed.
 - Only a minimal set of standards must be implemented.
- Very low level of application integration is expected.
 - But may be increased in a flexible way.
- Focuses on messages and documents, not on APIs.

Web Services Model

Web service applications are encapsulated, loosely coupled Web “components” that can bind dynamically to each other

Why Web Services?

- Two key factors
 - ubiquity
 - ease of use
- Interoperable
 - OS and language neutral
 - Java & .NET integration: simple and cheap
- Everybody supports or will support web services
 - need to support web services to facilitate integration
- Non-invasive
 - Based on ubiquitous protocols: HTTP/SMTP
 - Complements existing technologies

Statistics (Cristal Ball) - USA

- Gartner estimates the web service software market to be \$1.7 billion by 2003
- Evans Data estimates that 63% of all developers will be working on web-service projects by the end of 2003
 - 37% are working on web-service projects already
- Butler Group says over 76% of CIOs believe web services to be highly significant for the way they deliver IT services
- Jupiter Media Matrix recently found that 60% of CEOs planned to use web services in 2002
- A recent InfoWorld survey showed that 28% of companies are now using web services
 - another 23% are planning to introduce them this year

Component Tecnology & WS

- CORBA, EJB, and COM solve many integration problems
- However...
 - Each technology erroneously assumes that it is ubiquitous
 - Industry support for each component standard is fragmented
 - Complex Development
 - Result: "Islands of Interoperability"

Integration Problem

- It's hard to Integrate
 - Object model differences
 - Communication protocol differences
 - Type system differences
 - Exception handling differences:
 - especially runtime vs. application exceptions
- Example: calling from CORBA to EJB
 - requires reverse-mapping EJB into CORBA IDL
 - resulting IDL is not clean

The mapping Problem

- Whenever a new technology (such as Web Services) comes along, some attempt to map it directly to their existing technologies
- This never really works
 - Partially integration:
 - COM-CORBA mapping and the reverse Java-to-IDL mapping
 - But impedance mismatches between the systems cause pain and do not allow for full interworking

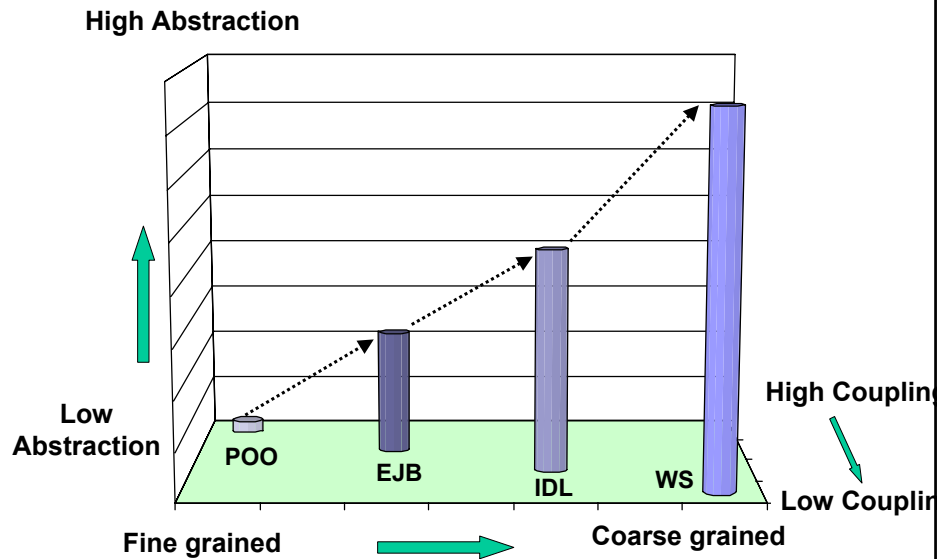
Abstractions & Integration Types

- **Method-oriented integration:**
 - Based on exposing through one technology the methods of a service implemented using a different technology
 - Component systems are largely method-oriented
 - Services exposed in this manner must be properly abstracted for the different technology:
 - hide implementation technology details
- **Document-oriented integration:**
 - Based on mapping the data understood by one system into data understood by another
 - Message-Oriented Middleware (MOM) and Enterprise Application Integration (EAI) systems are often document-oriented

Web Services Everywhere?

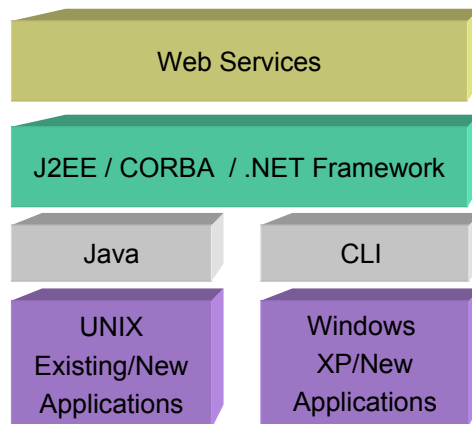
- Sometimes the hype gets a little too much...
- Web services are complementary to technologies such as J2EE, CORBA, etc.
 - they do not replace them!
- They give us a web-friendly “least common denominator” for systems integration
 - definitely the simplest way to integrate .NET, CORBA, J2EE, etc.

Granularity, Coupling & Abstraction

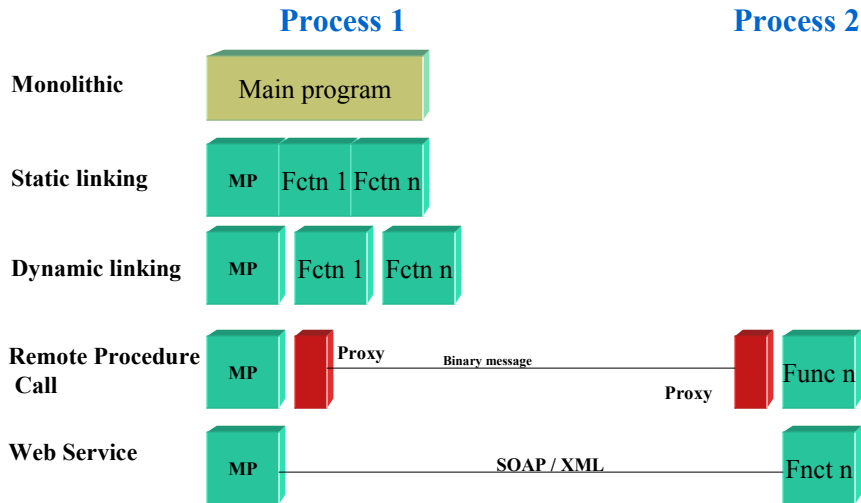


Web Services Offer

- A new (& needed) layer of abstraction

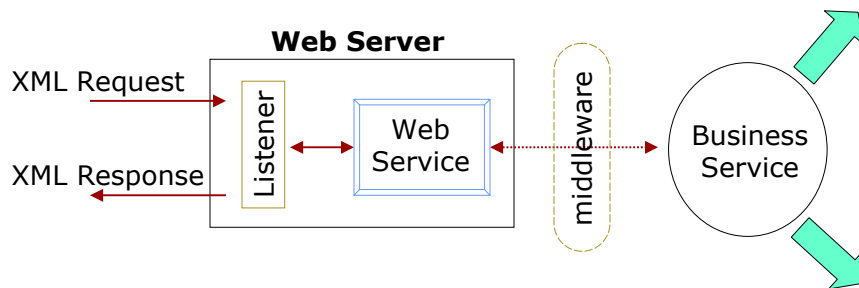


Web Services: Evolution of Technology



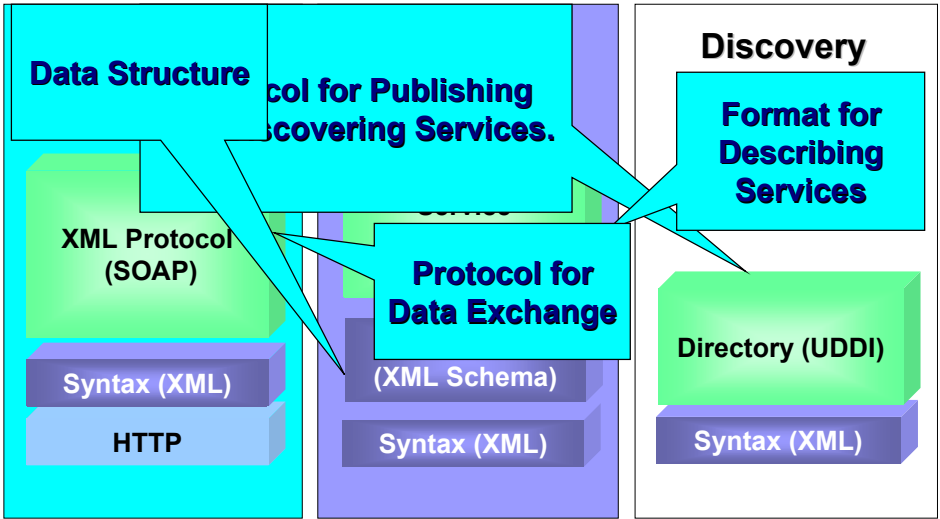
Architectural Perspective

- A web service is essentially a standards-based facade, or wrapper for accessing non-standardized middleware components

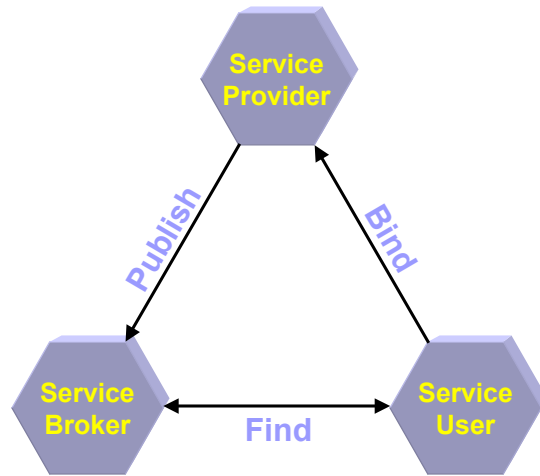


Web Services Framework

Web Services Stack

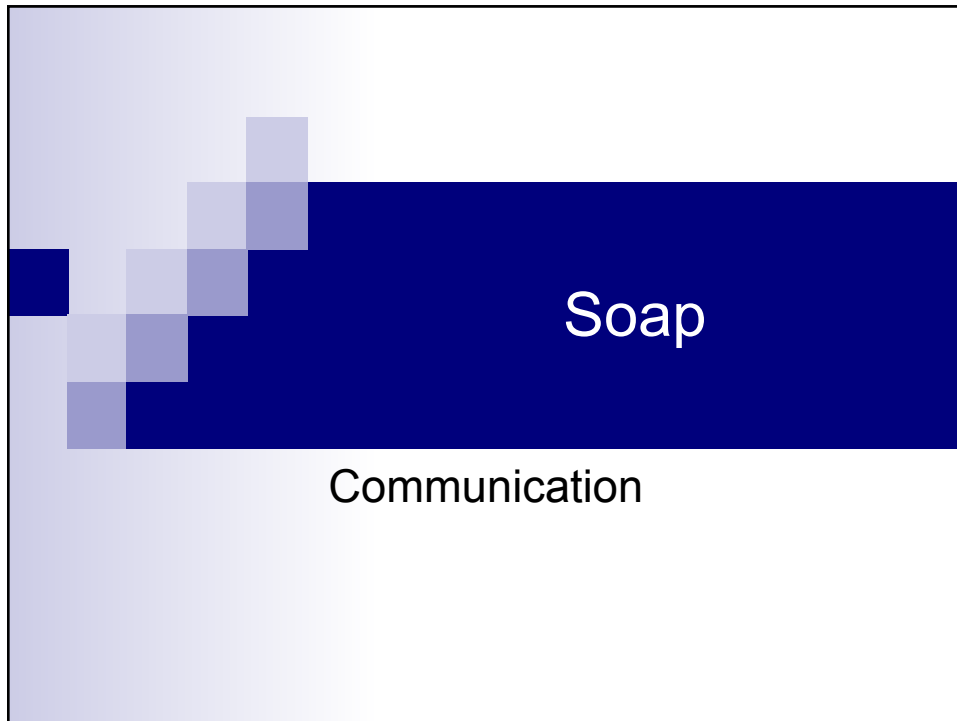


Service-Oriented Architecture



Web Services Protocols





Motivation

- Many Distributed applications communicate using remote procedure calls (RPC) between distributed objects like DCOM and CORBA.
- HTTP isn't designed for those objects, so RPC calls aren't easily adapted to the Internet.
- Security problems exist for those methods of RPC, so most firewalls and proxy servers are set to block this traffic.
- HTTP is supported by all Internet browsers and servers, so SOAP presents a nice protocol for doing RPC.

SOAP - Goal

- The goal behind the SOAP/ XMLP work is a protocol that is neutral on everything except the XML representation of data:
 - Transport
 - Programming language
 - Object model
 - Operating system
 - Etc.

SOAP Features

- Lightweight communication protocol
- Support different models:
 - one-way, request/response, multicast, etc..
- Designed to communicate via HTTP
 - Not restricted to
- Not tied to any component technology
- Not tied to any programming language
- Based on XML
- Simple and extensible

SOAP Message Exchange Model

- Fundamentally one-way
- Can be combined to achieve patterns like request/response
- Messages can be routed along a message path

SOAP Message - General Structure

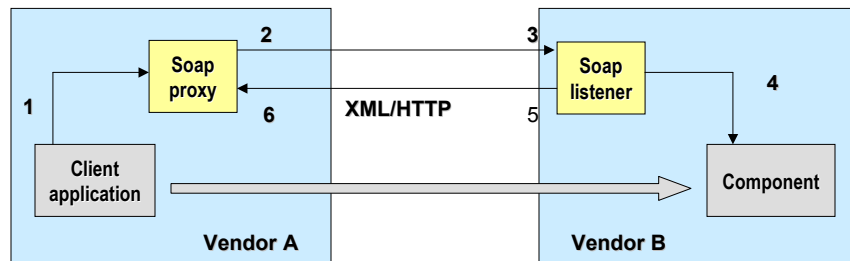
- Envelope:
 - Defines the content of the message
 - MUST be associated with SOAP envelope namespace:
<http://www.w3.org/2001/06/soap-envelope>
- Header (optional):
 - contains header information
 - application specific information about the SOAP message.
- Body:
 - contains call and response information

That's the basics...example

A simple SOAP XML document requesting the price of soap.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP



1. Client application makes a call
2. Soap proxy intercepts the call, constructs and transmits XML request message
3. Soap listener receives, parses and validates request
4. Listener calls component message
5. Listener takes the result of the call, constructs and transmits the XML response
6. Proxy receives and parses the response, returning the result to the client

This process is transparent for the client and the component

SOAP RPC Conventions

- Information needed for a method call:
 - The URI of the target object (namespace)

- Example:

```
<SOAP-ENV:Body>  
  <p:GetQuote xmlns:p="http://example.com/StockQuotes">  
    <Symbol>AMZN</Symbol>  
  </p:GetQuote>  
</SOAP-ENV:Body>
```

SOAP RPC Conventions

- Information needed for a method call:
 - The URI of the target object
 - A method name

- Example:

```
<SOAP-ENV:Body>  
  <p:GetQuote xmlns:p="http://example.com/StockQuotes">  
    <Symbol>AMZN</Symbol>  
  </p:GetQuote>  
</SOAP-ENV:Body>
```

SOAP RPC Conventions

- Information needed for a method call:
 - The URI of the target object
 - A method name
 - The parameters to the method

- Example:

```
<SOAP-ENV:Body>  
  <p:GetQuote xmlns:p="http://example.com/StockQuotes">  
    <Symbol>AMZN</Symbol>  
  </p:GetQuote>  
</SOAP-ENV:Body>
```

SOAP RPC Conventions

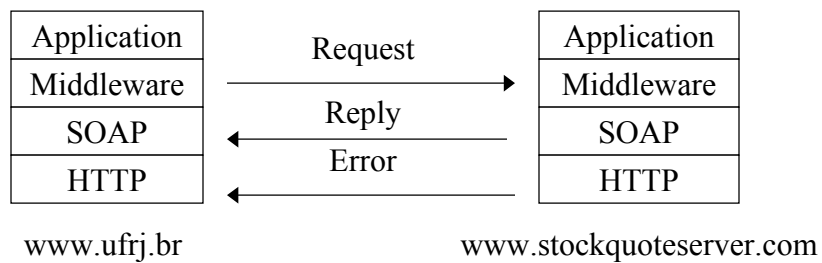
- Information needed for a method call:
 - The URI of the target object
 - A method name
 - The parameters to the method
 - Optional header data

- Example:

```
<SOAP-ENV:Body>  
  <p:GetQuote xmlns:p="http://example.com/StockQuotes">  
    <Symbol>AMZN</Symbol>  
  </p:GetQuote>  
</SOAP-ENV:Body>
```

An example

- `www.stockquoteserver.com`
- `float GetLastTradePrice(symbol)`
- The process:



SOAP HTTP Request Example

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```

↑ HTTP Header
↓ SOAP Extensions

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>AMZN</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

↑ XML payload

SOAP HTTP Response Example

HTTP/1.1 200 OK

Content-Type: text/xml; charset="utf-8"

Content-Length: nnnn

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >

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

</SOAP-ENV:Envelope>
```

SOAP + Attachments

- Mechanism for encoding SOAP msgs in a MIME multipart structure and associating it with any parts in that structure
- SOAP msg
 - Root of the structure
 - Refers to attachments using uri
 - [cid:prefix](#) (content ID)

SOAP + Attachments example

- Start with a SOAP message...

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Body>
    ..
    <Person>
    </Person>
    ..
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP + Attachments example

- Add a link to an external image...

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Body>
    ..
    <Person>
      <Picture href="http://example.com/myPict.jpg" />
    </Person>
    ..
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP + Attachments example

- Add MIME packaging...

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
  type=text/xml;
  start="<soapmsg.xml@example.com>"
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: soapmsg.xml@example.com

<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Body>
    ..
    <Person>
      <Picture href="http://example.com/myPict.jpg" />
    </Person>
    ..
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
--MIME_boundary--
```

SOAP + Attachments example

- Add MIME part and link to it...

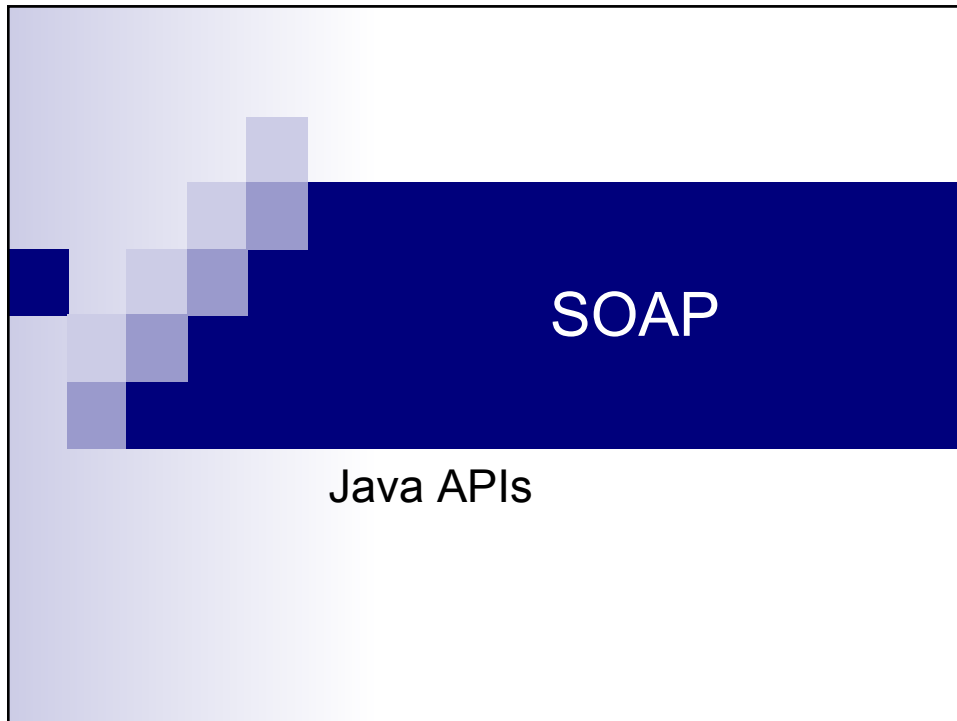
```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
  start="<soapmsg.xml@example.com>"

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: soapmsg.xml@example.com

<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Body>
    <Person>
      <Picture href="cid:myPict.jpg@example.com" />
    </Person>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/jpg
Content-Transfer-Encoding: binary
Content-ID: <myPict.jpg@example.com>

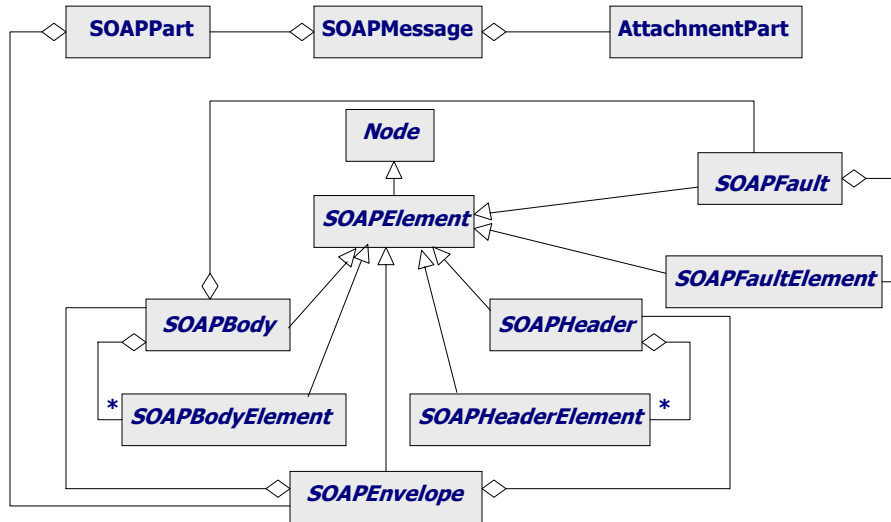
..binary image..
--MIME_boundary--
```



Java API for Web Services

- Based on remote procedure call (RPC) mechanism
 - client's remote method call is converted to a SOAP msg and sent to the service as a HTTP request
 - service receives the request, translates the SOAP msg into a method call and invokes it
 - and vice versa with the result of the method call
- Implementation details invisible to client and service
- SOAP messages as Java objects
 - SAAJ (SOAP with Attachments API for Java)
- Programming Model
 - JAX-RPC client can access a web service not running on Java platform
 - JAX-RPC service can be accessed by a non Java client

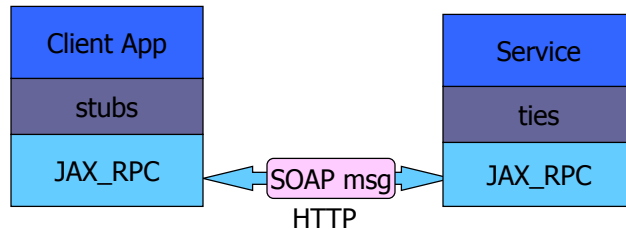
SAAJ Classes



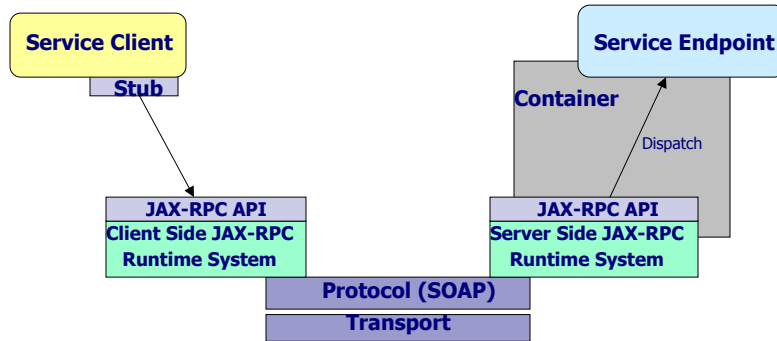
JAX-RPC

JAX-RPC client invocation models

- statically defined stub model
- dynamic proxy invocation model
- dynamic invocation interface (DII)



JAX-RPC Physical Architecture



A Simple Example

- Writing the Client Program
 - Calculator service
 - Using Dynamic Invocation Interface (DII)

Calculator Client – using JAX-RPC

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class CalcClient
{
    public static void main(String [] args) {
        try {
            String endpoint = "http://localhost:8080/axis/Calculator.jws";
            Service service = new Service();
            Call call = (Call) service.createCall();

            call.setTargetEndpointAddress( new java.net.URL(endpoint) );
            call.setOperationName(new QName("http://localhost:8080/axis/
            Calculator.jws/axis/Calculator.jws",
            "add" ) );

            Integer ret = (Integer) call.invoke( new Object[]
            { new Integer(5),
            new Integer(6) } );

            System.out.println("Result = " + ret);
        } catch (Exception e) {
            System.err.println("ERROR! : " + e.toString());
        }
    }
}
```

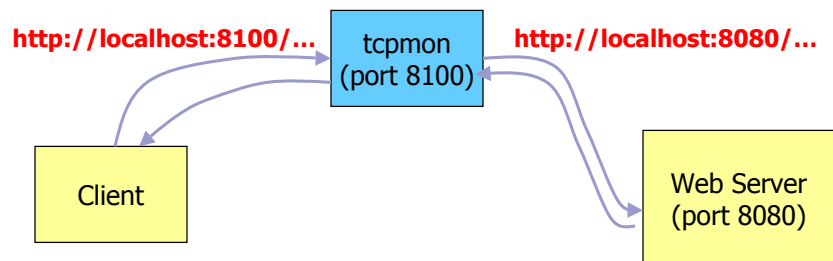
SOAP Demo

tcpmon

- org.apache.axis.utils package
 - `java org.apache.axis.utils.tcpmon`
 - watch classpath...
 - `listen port` is port that the client makes the request to (tcpmon port)
 - `target hostname` - host running web server
 - `target port` – port that web server running at

Viewing SOAP msgs

- `tcpmon` utility

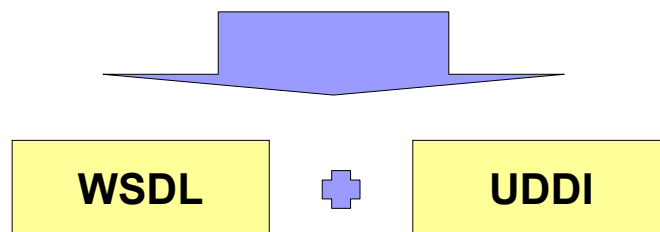


Analysing CalClient App

- The details of the actual SOAP protocol were handled by the **Call** and **Response** objects.
- The developer focused:
 - on finding the correct data to invoke the service,
 - Processing the response

Analysing CalClient App

- The SOAP client hard-coded:
 - URL, service ID, method name, parameters, etc.
- What if we could discover these things at runtime?



WSDL

Describing Web Services

What is WSDL?

- An XML format for describing web services

To the Consumer: How to use the service
(e.g., "To use my shopping cart, send these SOAP messages over HTTP to <http://myservice.net/cart>")

To the Server: How to configure the service
(e.g., "Make my shopping cart available using SOAP over HTTP at <http://myservice.net/cart>")

To the Registry: How to find the service
(e.g., "Find me all the shopping carts I can talk to.")

What's a WSDL File?

- A WSDL file is an XML document.
- The root is the **<definitions>** element:
 - defines the namespaces we'll use.

```
<definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/>
```

What's in a WSDL File?

- The **<definitions>** element has up to six different child elements:
 - **Types**
 - **Messages**
 - **Port types**
 - **Bindings**
 - **Ports**
 - **Services**

WSDL Elements by Example

WSDL Terms

Port Type

Operation



WSDL Elements by Example

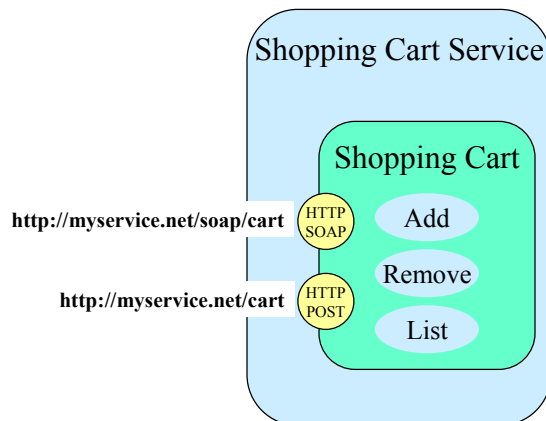
WSDL Terms

Port Type

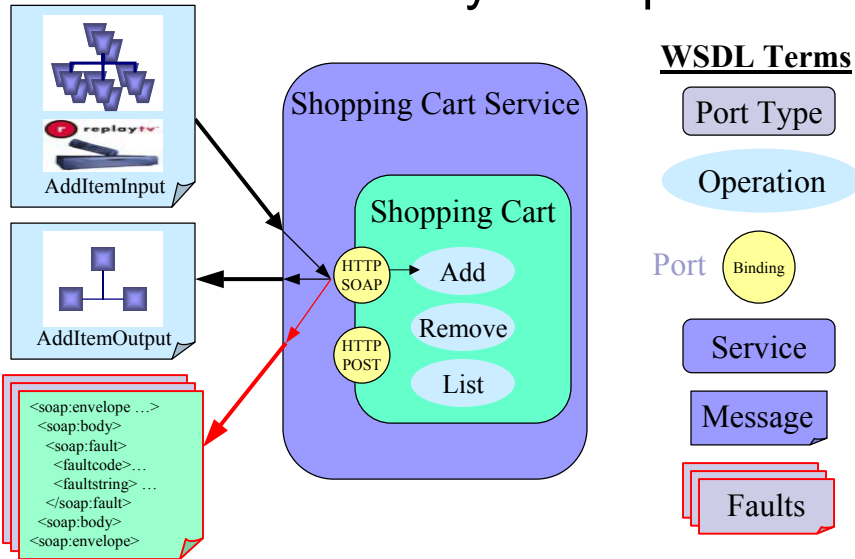
Operation

Port Binding

Service



WSDL Elements by Example



What does WSDL Describe?

- Abstract Concepts
 - Port Types (e.g., shopping Cart)
 - Operations (e.g., add, remove, list)
 - Messages (i.e., inputs, outputs and faults)
 - Data types (i.e., schemas)
- Concrete Bindings
 - Data encoding styles (e.g., SOAP encoding)
 - Transport protocols (e.g., HTTP, SMTP)
 - Network addresses (e.g., <http://myservice.net/cart>)

How does WSDL Describe Services?

```
<definitions name="ShoppingCartDefinitions"
  xmlns="http://schemas.xmlsoap.org/wsdl" targetNamespace= ... >
  <types> ... </types>
  <message name="AddItemInput"> ... </message>
  <message name="AddItemOutput"> ... </message>
  <portType name="ShoppingCart"> ... </portType>
  <binding name="CartHTTPXMLBinding" type="tns:ShoppingCart">...
  <binding name="CartSOAPBinding" type="tns:ShoppingCart">...
  <service name="ShoppingCartService">
    <port name="HTTPXMLCart" binding="tns:CartHTTPXMLBinding">
      ...
    <port name="SOAPCart" binding="tns:CartSOAPBinding"> ...
  </service>
  <import namespace="..." location="...">
</definitions>
```

Data Types

- You can define datatypes, which are typically based on XML Schema:
 - however, other type systems are permitted

```
<types>
  <schema targetNamespace="http://myservice.net/cart/types"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
    <complexType name="item"><all>
      <element name="description" type="xsd:string"/>
      <element name="quantity" type="xsd:integer"/>
      <element name="price" type="xsd:float"/>
    </all></complexType>
  </schema>
</types>
```

Messages

- Describes the flow of information
- A named collection of message parts
- Each part has a name and a type

```
<message name="AddItemInput">  
  <part name="cart-id" type="xsd:string"/>  
  <part name="item" type="carttypes:item"/>  
  <part name="image" type="xsd:base64Binary"/>  
</message>
```

Port Types

- A named collection of related operations
- Operations:
 - Have a signature (input, output and fault messages)
 - can be one-way, request-response, notifications or solicit-response

```
<portType name="ShoppingCart">  
  <operation name="AddItem">  
    <input message="tns:AddItemInput"/>  
    <output message="tns:ACK"/>  
    <fault name="BadCartID" message="tns:BadCartID"/>  
    <fault name="ServiceDown" message="tns:ServiceDown"/>  
  </operation>  
  <operation name="RemoveItem"> ... </operation>  
  <operation name="ListItems"> ... </operation>  
</portType>
```

Bindings

- Bindings describe the protocol used to access a service, as well as the data formats for the messages defined by a particular **portType**
- A binding is a named association of protocol details with a port type, its operations and its messages

Bindings

```
<binding name="CartHTTPPostBinding" type="tns:ShoppingCart">
  <http:binding verb="POST"/>
  <operation name="AddItem">
    <http:operation location="/AddItem"/>
    <input>
      <mime:content type="application/x-www-form-urlencoded"/>
    </input>
    <output>
      <mime:content type="application/x-www-form-urlencoded"/>
    </output>
    <fault name="BadCartID"> <mime:mimeXML/> </fault>
    <fault name="ServiceDown"> <mime ... /> </fault>
  </operation> ...
</binding>
```

How WSDL Describes: Bindings

```
<binding name="CartHTTPPostBinding"
  type="tns:ShoppingCart">
  <http:binding verb="POST"/>
  <operation name="AddItem">
    <http:operation location="/AddItem"/>
    <input> <mime:content ... /> </input>
    <output> <mime:content .... /> </output>
    <fault name="BadCartID"> <mime .../> </fault>
    <fault name="ServiceDown"> <mime ... /> </fault>
  </operation> ...
</binding>
```

Extensibility
Elements

Bindings: SOAP Binding


```
<binding name="CartHTTPSOAPBinding" type="tns:ShoppingCart">
  <soap:binding style="RPC"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="AddItem">
    <soap:operation soapAction="http://myservice.net/cart/AddItem"/>
    <input>
      <soap:body use="encoded" namespace="http://myservice.net/cart"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <soap:body use="encoded" namespace="http://myservice.net/cart"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
    <fault name="BadCartID"> <soap:body use="encoded" namespace= ... />
    </fault>
    <fault name="ServiceDown"> <soap:body use= ... /> </fault>
  </operation> ...
</binding>
```

Ports

- A named association of a protocol binding with a **network address**

```
<port name="SOAPCart" binding="tns:SOAPCartBinding">  
  <soap:address location="http://myservice.net/soap/cart"/>  
</port>  
<port name="HTTPPostCart" binding="tns:HTTPPostCartBinding">  
  <http:address location="http://myservice.net/cart"/>  
</port>
```

<http://myservice.net/soap/cart> 

<http://myservice.net/cart> 

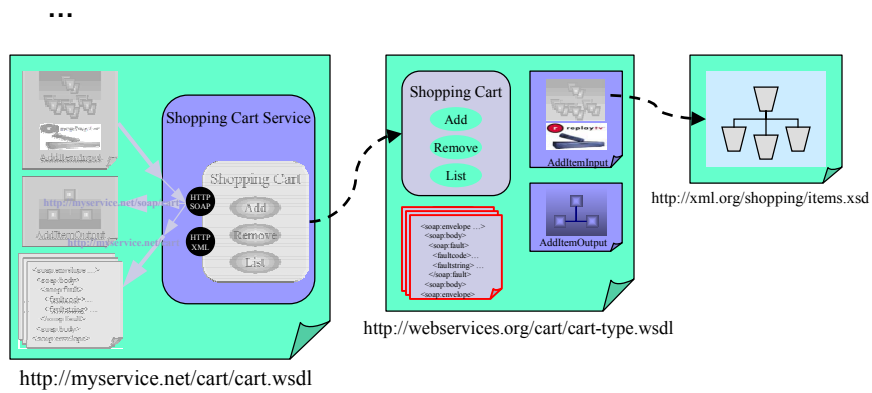
Services

- A related collection of ports

```
<service name="ShoppingCartService">  
  <documentation>A Shopping Cart for the Web</documentation>  
  <port name="HTTPPostCart" binding="tns:HTTPPostCartBinding">  
    <http:address location="http://myservice.net/cart"/>  
  </port>  
  <port name="SOAPCart" binding="tns:SOAPCartBinding">  
    <soap:address location="http://myservice.net/soap/cart"/>  
  </port>  
</service>
```

Reusing WSDL Elements

```
<import namespace="http://webservices.org/cart/cart-type"  
  location="http://services.org/cart/cart-type.wsdl"/>  
<binding xmlns:ws="http://webservices.org/cart/cart-type"  
  name="CartHTTPXMLBinding" type="ws:ShoppingCart">
```



SOAP & WSDL

Analysing CalcClient Code

SOAP and WSDL

- The SOAP code needs the URL of the SOAP router:
 - **String endpt=**
`call.setTargetEndpointAddress (endpt);`
- WSDL:
 - **<service ...**
 - **<port ...**
 - **<soap: address location= />**
 - **“http://localhost:8080/axis/Calculator.jws”**

SOAP and WSDL

- The SOAP code needs the (namespace) of the service:
 - `call.setOperationName(new QName(http://localhost:8080/axis..., args[1]))`
- WSDL:
 - **<binding ...**
 - **<soap: binding style=“ rpc” ...**
 - **<operation name= “add”>**
 - **<input> ...**
 - **<soap: body namespace= “http://localhost:8080/...”**

SOAP and WSDL

- The SOAP code needs the name of the method:
 - `call.setOperationName(new QName(http://localhost:8080/axis..., args[1]))`
- WSDL:
 - **<binding ...**
 - **<soap: binding style=" rpc" ...**
 - `<operation name= "add"`

SOAP and WSDL

- The SOAP code needs to fill in the arguments.
- WSDL
 - **<portType name=" Calculator">**
 - **<operation name="add">**
 - `<input message= "intf:addRequest " />`
- This tells us we need to use the **addRequest** message as the input to the method.

SOAP and WSDL

- Now we can look at the **addRequest** message:
- **<message name=" addRequest ">**
 - **<part name="val1" type="xsd:int"/>**
 - **<part name="val2" type="xsd:int"/>**
- **</ message>**
- **Therefore, we need two arguments (val1, val2) of type Integer to call the service**

SOAP and WSDL

- In practice, the SOAP client code is written using a well- known programming interface.
 - the name of the method and the arguments to it will be known beforehand.
- The URL of the SOAP router and the ID of the service can change,
 - The method name and parameters probably won't.
- The WSDL file provides all the information needed to invoke a Web service
 - Automate the task of invoking Web Services
 - Tools to hide the details of WebServices Technology



Web Services Platforms



What is a Web Services Platform?

- Marshals Data Types
 - Serializes native language objects into XML
 - De-serializes XML into native language objects
- Publishes a Service Endpoint
- Invokes a Service Handler or Chain of Handlers
- Generates WSDL Descriptions
- Generates client and server stubs from WSDL

Implementing WS

Java – Apache Axis

Apache Axis

■ A SOAP Processing Engine

- JAX-RPC Client System
- JAX-RPC Server System (Servlet based)
- SAAJ implementation
- Flexible and extensible architecture
- Tools, Examples, Documentation, ...
- A great tool to learn (and produce) about Web Services !!

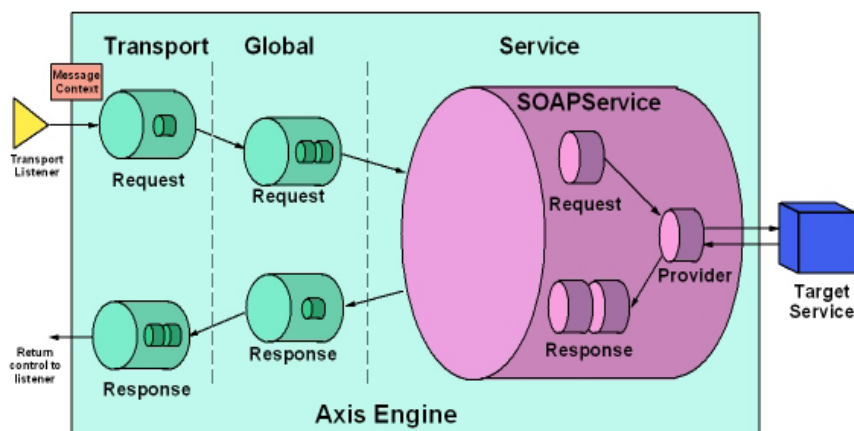
■ **Open-source**

- hosted by Apache Software Foundation

Axis Feature Set

- “Drop-in” deployment of SOAP services
- Support for all basic data types, and a type mapping system for defining serializers/deserializers
- Providers for RPC and message based SOAP services
- Automatic WSDL generation from deployed services
- WSDL2Java tool for building Java proxies and skeletons from WSDL documents
- Java2WSDL tool for building WSDL from Java classes.
- HTTP servlet-based transport

Axis Architecture



Install & Deploy Apache Axis

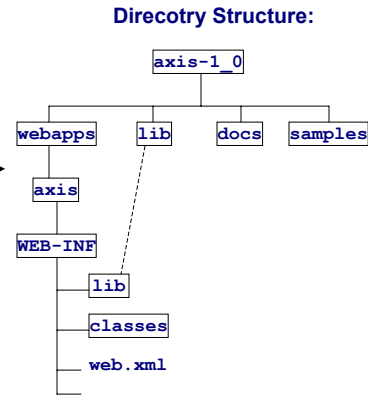
- Pre-requisites
 - J2SE SDK 1.3 or 1.4
 - A Servlet Container: i.e. Tomcat4.0.1

- Download:
 - `xml-axis-rcl-bin.zip`
 - <http://xml.apache.org/axis>

- Unzip it →
 - Axis runs as a Servlet.

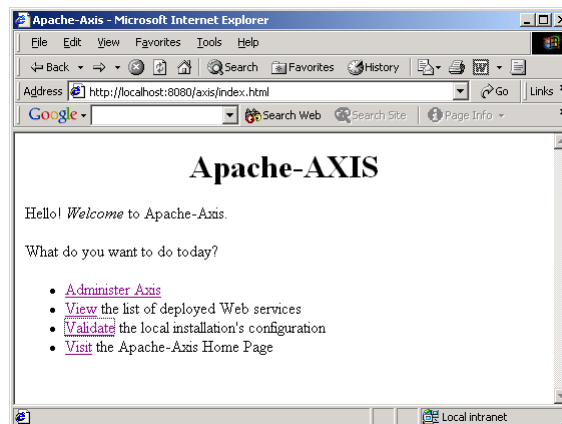
- Deploy Axis.
 - Copy `webapps\axis` tree to `webapps` directory of Tomcat.
 - Or modify `server.xml` of Tomcat.

- Run Tomcat:
 - `bin\startup` from Tomcat home.



Test the Deployment

- <http://localhost:8080/axis>
- Select **Validade** ..



Calculator Example

- **Calculator:**
 - A simple Java class with two methods to add and subtract two integers.
- The filename extension must be “.jws” (for Java Web Service).
- **Deployment:**
 - Copy the Calculator.jws file to webapps/axis directory.
- **Examine its WSDL description.**
 - <http://localhost:8080/axis/Calculator.jws?wsdl>

```
public class Calculator {  
    public int add(int val1, int val2){  
        return val1 + val2; }  
    public int subtract(int val1, int val2) {  
        return val1 - val2; }  
}
```

Writing the Client Program

- **Ways to write a Client program**
 - Using Dynamic Invocation Interface (DII)
 - Using generated Stubs from Service WSDL description
 - Using Dynamic Proxy
- **CLASSPATH Setup:**
 - xml-axis-beta2/lib/axis.jar
 - xml-axis-beta2/lib/jaxrpc.jar
 - xml-axis-beta2/lib/saaj.jar
 - xml-axis-beta2/lib/commons-logging.jar
 - xml-axis-beta2/lib/tt-bytecode.jar
 - xml-axis-beta2/lib/wsdl4j.jar
 - A JAXP-1.1 compliant XML parser such as xerces
 - + app resources

CalcClient – Generated Stubs

- Generate the stubs:

- `java org.apache.axis.wSDL.WSDL2Java \`
`http://localhost:8080/axis/Calculator.jws?wsdl`

WSDL clause	Java class(es) generated	Name
For each entry in the type section	A java class	
	A holder if this type is used as an inout/out parameter	
For each portType	A java interface	Calculator.java
For each binding	A stub class	CalculatorSOAPBindingStub.java
For each service	A service interface	CalculatorService.java
	A service implementation (the locator)	CalculatorServiceLocator.java

CalcClient Program – Generated Stubs

```
import localhost.*;

public class CalcClient{
    public static void main(String [] args) {
        try {
            CalculatorService srv = new CalculatorServiceLocator();
            Calculator calc = srv.getCalculator();
            System.out.println("addInt(5, 3) = " + calc.addInt(5, 3));
        }
        catch (Exception e) {
            System.err.println("Execution failed. Exception: " + e);
        }
    }
}
```


Custom Deployment

- Instant deployment (.jws) is simple, but has limitations:
 - You must have the source code
 - Can't specify custom type mappings, handlers etc.
- Java2WSDL: Building WSDL from Java
 - Step 1: Provide a Java interface or class
 - Step 2: Create WSDL using Java2WSDL
 - Step 3: Create Client/Server Bindings using WSDL2Java
 - Step 4: Use adminClient & Deployment Descriptor to deploy the service

Step 1: Provide a Java interface or class

```
package calculator;  
  
public interface Calculator extends java.rmi.Remote {  
    public int add(int in0, int in1)  
        throws java.rmi.RemoteException;  
    public int subtract(int in0, int in1)  
        throws java.rmi.RemoteException;  
}
```

Step 2 - Java2WSDL – Generating WSDL file

- `java org.apache.axis.wsdl.Java2WSDL \`
 - o calculator.wsdl \
 - l "http://localhost:8080/axis/services/Calculator" \
 - n "urn:Calculator"
 - p "custom.deploy=urn:Calculator" \
 - custom.deploy.Calculator



WSDL file "calculator.wsdl"

Step 3: Create Client/Server Bindings using WSDL2Java

- **Generate the classes:**
 - `java org.apache.axis.wsdl.WSDL2Java \`
 - `--server-side --skeletonDeploy true`
 - `calculator.wsdl`
 - **All classes generated without option "--server-side --skeletonDeploy true" plus:**

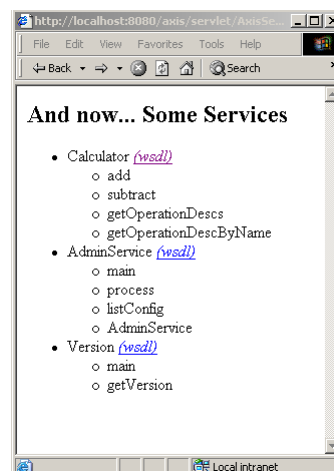
WSDL clause	Java class(es) generated	Name
For each binding	A skeleton class	CalculatorSoapBindingSkeleton.java
	An implementation template class	CalculatorSoapBindingImpl.java
For all services	One deployment descriptor	deploy.wsdd
	One Undeployment descriptor	Undeploy.wsdd

Step 4: Provide the service implementation

- Insert the service code in the Implementation template Class
- Compile the source code
- The service is ready for deployment

Step 5: Use adminClient & Deployment Descriptor to deploy the service

- `java org.apache.axis.client.AdminClient deploy.wsdd`
- Don't forget to start the Web server !
- Copy the server class files to
`TOMCAT_HOME\webapps\axis\WEB-INF\classes`



List of deployed services: `java org.apache.axis.client.AdminClient list`

Deployment Descriptors

■ WSDD (Web Services Deployment Descriptors)

- allow more flexible deployments
- Handlers in request or response path
- Custom type mappings
- Different transports – HTTP/S, TCP/IP, DIME
- Different Dispatchers – Java Class, EJB, Servlet

The Deployment Descriptor

```
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <!-- Services from CalculatorService WSDL service -->

  <service name="Calculator" provider="java:RPC">
    <parameter name="wsdlTargetNamespace" value="urn:custom.deploy"/>
    <parameter name="wsdlServiceElement" value="CalculatorService"/>
    <parameter name="wsdlServicePort" value="Calculator"/>
    <parameter name="className"
      value="deploy.custom.CalculatorSoapBindingSkeleton"/>
    <parameter name="wsdlPortType" value="Calculator"/>
    <parameter name="allowedMethods" value="**"/>
  </service>
</deployment>
```

Additional Features

- SOAP with Attachments
- Custom type mappings (Pluggable Serializers)
- One-way invocations
- Document exchange
- Dispatch to EJBs
- HTTPS transport and mutual authentication
- Username and password based authentication

Demo

Calculator



Demo

Consuming a Web Service
Using Delphi



WSDL Bindings

HTTP Post & Get

HTTP GET & POST Binding

- WSDL includes a binding for HTTP 1.1's GET and POST verbs:
 - This allows applications other than Web Browsers to interact with the site.
- The following protocol specific information may be specified:
 - An indication that a binding uses HTTP GET or POST
 - An address for the port
 - A relative address for each operation (relative to the base address defined by the port)

HTTP GET/POST Examples

- Consider 2 ports that are bound differently for a given port type.
- If the values being passed are part1=1, part2=2, part3=3, the request format would be as follows for each port:
 - port1:
 - GET, URL="http://example.com/o1?p1=1&p2=2&p3=3"
 - port2:
 - POST,
 - URL="http://example.com/o1",
 - PAYLOAD="p1=1&p2=2&p3=3"
- For each port:
 - the response is either a GIF or a JPEG image.

```

<definitions .... >
  <message name="m1">
    <part name="part1" type="xsd:string"/>
    <part name="part2" type="xsd:int"/>
    <part name="part3" type="xsd:string"/>
  </message>
  <message name="m2">
    <part name="image" type="xsd:binary"/>
  </message>
  <portType name="pt1">
    <operation name="o1">
      <input message="tns:m1"/>
      <output message="tns:m2"/>
    </operation>
  </portType>
  <service name="service1">
    <port name="port1" binding="tns:b1">
      <http:address location="http://example.com"/>
    </port>
    <port name="port2" binding="tns:b2">
      <http:address location="http://example.com"/>
    </port>
  </service>
  ...

```

```

<binding name="b1" type="pt1">
  <http:binding verb="GET"/>
  <operation name="o1">
    <http:operation location="o1"/>
    <input> <http:urlEncoded/> </input>
    <output>
      <mime:content type="image/gif"/>
      <mime:content type="image/jpeg"/>
    </output>
  </operation>
</binding>
<binding name="b2" type="pt1">
  <http:binding verb="POST"/>
  <operation name="o1">
    <http:operation location="o1"/>
    <input>
      <mime:content type="application/x-www-form-urlencoded"/>
    </input>
    <output>
      <mime:content type="image/gif"/>
      <mime:content type="image/jpeg"/>
    </output>
  </operation>
</binding>
</definitions>

```


UDDI

Web Services Discovery

UDDI

- UDDI = Universal Description, Discovery and Integration of Web Services
- UDDI project (www.uddi.org) is an industry effort to define a searchable registry of services
 - started by IBM, MS and Ariba in Sept 2000
 - UDDI members > 200
 - currently at UDDI V3.0
 - plan to turn specification over to W3C at V3.0
- UDDI not restricted to SOAP services
 - can be used for simple web page, email address, up to full distributed apps

UDDI

Data within UDDI categorised into 3 types

**White
Pages**

- white pages
 - basic contact info about companies
 - allows businesses to include unique ids

**Yellow
Pages**

- yellow pages
 - general classification info about the company or service it offers (e.g. NAICS: industry codes, UNSPC: product & service clsf, ISO: geog)

**Green
Pages**

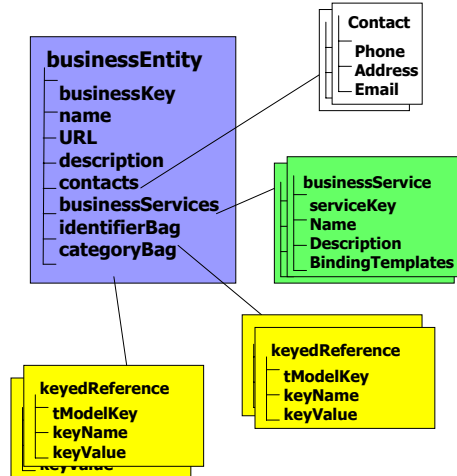
- green pages
 - technical info about a web service
 - reference to an external specification (e.g. WSDL)
 - address for invoking the service

Service Descriptions

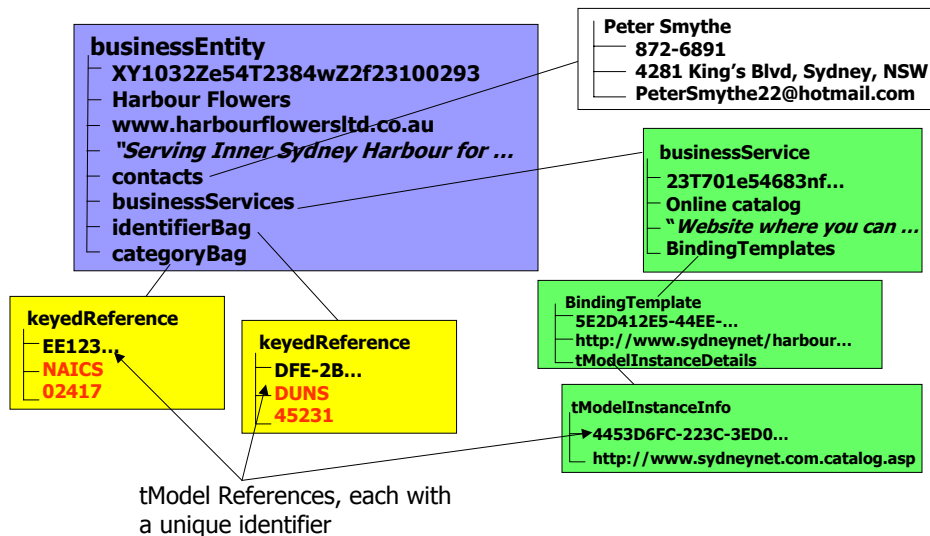
- Programmatically unique identifier for any given type of web service
- Used by standards bodies, ISVs, and developers to “publish” how their service works
- Used as a signature by web sites that implement those interfaces
- Stored in UDDI as “*tModels*”

UDDI Registration

- XML document
- Created by end-user company (or on their behalf)
- Can have multiple service listings
- Can have multiple taxonomy listings



Example of a Registration



How is UDDI used?

- Business analysts:

- to search for services
 - similar to search engine
 - portals needed

- Developers:

- to publish services
- to write software that use the discovered services

- incorporated into toolkits:

- to automate publishing of services

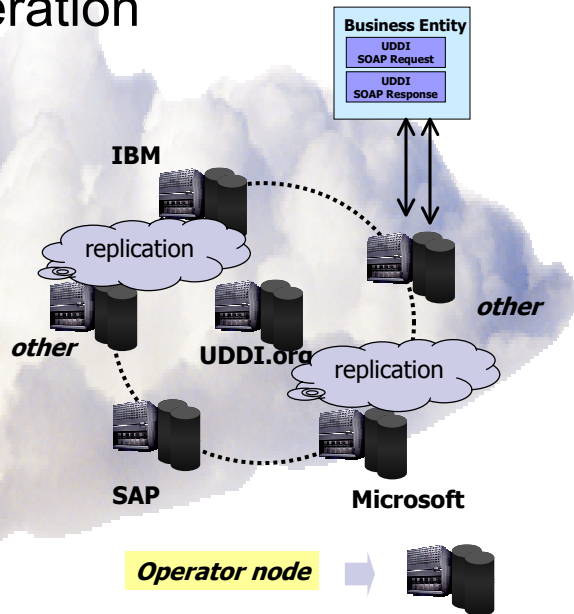
UDDI

Two parts

- technical specifications for building a distributed directory of business & web services, including
 - XML schema for describing the data structures used
 - API details for searching (*find*) & publishing (*publish*)
- UDDI Business Registry (“cloud services”)
 - fully operational implementation
 - launched in May 2001 by MS & IBM

Registry Operation

- Peer nodes (websites)
- Companies register with any node
- Registrations replicated on a daily basis
- Complete set of "registered" records available at all nodes
- Common set of SOAP APIs supported by all nodes



UBR

- content inserted into the UBR is done at a single node which becomes master owner of content
- content can be accessed from any node
- any business can set up an operator node
- companies can set up:
 - private nodes
 - private clouds

Private Registries

- Direct discovery
- Include additional security features
- Services only accessible within org or group of trusted parties
- Often charge is involved
- Can be used for internal or B2B operations
- Allows companies to provide personalised services for clients
- Companies are adopting private registries faster than public

Types of Private Registries

- e-marketplace UDDI
 - hosted by an industry, lists consortium member businesses and services
- portal UDDI
 - publishes a companies web services
 - *find* only available externally
- partner catalog UDDI
 - resides behind firewall
 - available to member companies only
 - *publish* and *find* restricted to authorised users
- internal UDDI
 - resides behind firewall
 - available to single organisation only

Why use UDDI?

Example: semi-conductor industry

- >400 companies members of RosettaNet consortium who created Partner Interface Processes (PIPs)
- PIPs = XML-based interfaces to allow exchange of data
- e.g. PIPs
 - PIP2A2 – query for product info
 - PIP3A2 – query price and availability of specific products
 - PIP3A3 – submit e-purchase order
 - PIP3B4 – query shipment status
- >80 individual PIPs registered within UBR

Advantages of UBR

- service provider
 - effective method of advertising
 - global visibility → market expansion
- service consumer
 - simplifies using web services by making tech specifications available
- general
 - don't have to pay to advertise or to discover

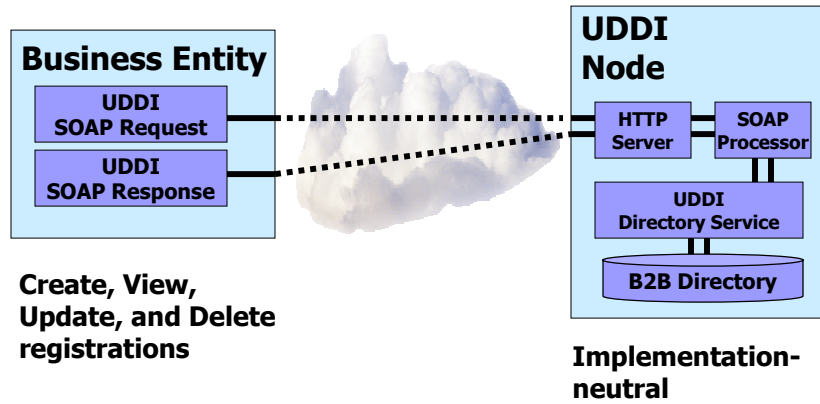
UDDI Software

- Applications servers incorporating UDDI server
 - BEA WebLogic, IBM WebSphere
- Open source UDDI project
 - jUDDI (www.juddi.org)
- J2EE compliant UDDI implementations:
 - Systinet, Cape Clear, HP, Oracle, SAP...

UDDI Limitations

- Still evolving
- Public Registries - data reliability?
 - no "last-updated" date
 - no checks for accuracy
- Quality-of-service of a web service?
 - how often can it be accessed?
 - scalability?
 - tech support?
 - Etc.

UDDI and SOAP



Registry WSI (SOAP Messages)

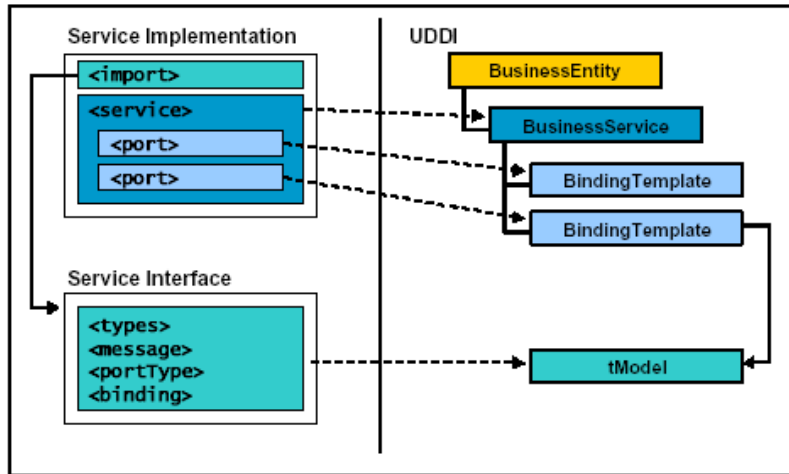
■ Inquiry API

- Find things
 - find_business
 - find_service
 - find_binding
 - find_tModel
- Get Details about things
 - get_businessDetail
 - get_serviceDetail
 - get_bindingDetail
 - get_tModelDetail

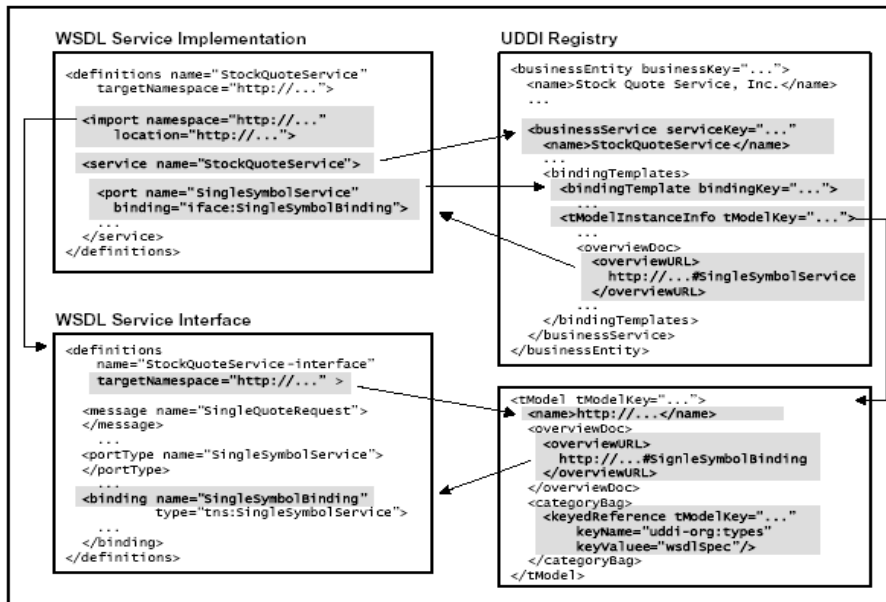
■ Publishers API

- Save things
 - save_business
 - save_service
 - save_binding
 - save_tModel
- Delete things
 - delete_business
 - delete_service
 - delete_binding
 - delete_tModel
- security...
 - get_authToken
 - discard_authToken

Mapping From WSDL to UDDI



Mapping From WSDL to UDDI (Example)



Working with UDDI

- Finding Services in UDDI
 - Services are found in UDDI using UDDI API
 - Services are found by name or by description
- Binding
 - Once the service requestor has located the service, it invokes the service at that location

Registries

- IBM's registry:
 - www.ibm.com/services/uddi/
- Microsoft's registry:
 - uddi.microsoft.com/
- SAP registry:
 - <http://uddi.sap.com/>
- NTT Communications:
 - <http://www.ntt.com/uddi>

UDDI4J

- IBM has released UDDI4J:
 - an **open source** implementation of the client side of UDDI.
 - AXIS, HP, WebSphere
- Simplifies the task of interacting with a UDDI registry.
- UDDI4J web page:
 - **www-124.ibm.com/developerworks/projects/uddi4j**.

Architecting WS

Today Usage of Web Services

- Approach:
 - Direct exposure of components as Web Services
 - Is it Correct?

Granularity, coupling, amount of implementation details, and levels of abstraction at the component level are inappropriate for Web Services

Direct Export - Problems

- Exporting assumes that the exported component can stand alone
 - Only works for simple models
- What happens if operations return sequence of references, or complex data structures with embedded references?
Who does the mapping?
 - Direct one-to-one exporting of components to Web Services is too cumbersome, and it works only for simple demos

Web Services Examples

- Some web services are already accessible on the Internet
 - e.g., at <http://www.xmethods.net/>
 - Trivial RPC-oriented services, for example:
 - exchange rate calculators, stock quoters
- Too low-level
 - e.g., stock quoter requires polling:
 - no callbacks provided
 - Fundamental issues ignored:
 - e.g latency

Application Choreography

- Defines the “conversations” between cooperating applications that allow them to work together correctly
- Web Services choreographies must take business processes into account
 - trivial web services solve only trivial things
 - non-trivial web services must play a part in business processes
 - business-to-business integration (B2Bi) requires standardized choreographies

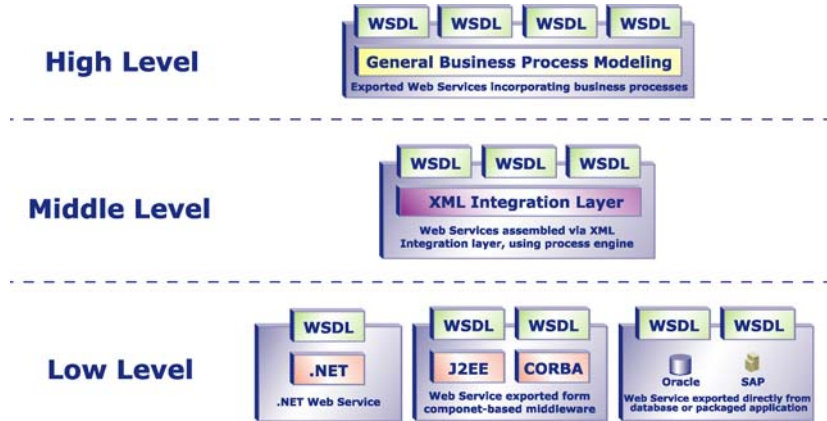
Web Service Choreographies

- CORBA, EJB, etc. choreographies not directly suitable for Web Services
 - too many implementation details leak through, e.g. exceptions, component interdependencies
- Instead, Web Services can integrate multiple back-end component technologies using process engines to avoid the need for more hand-coding
- Choreographies must completely hide component-level implementation details

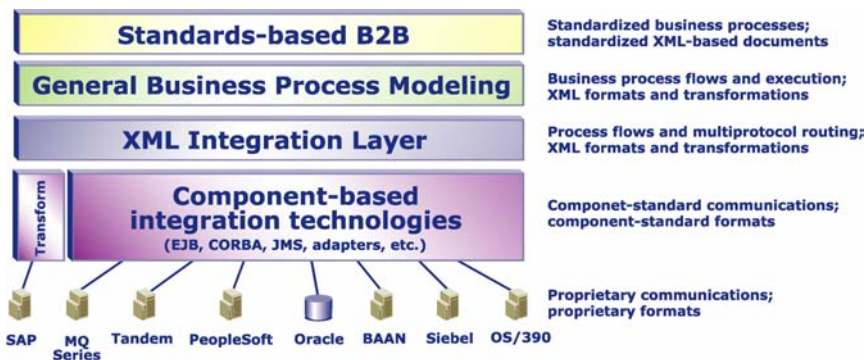
Web Service Choreographies(2)

- Web Services have no object model
 - CORBA, EJB, etc. choreographies can't be directly exposed to Web Services clients
 - Web Services must wrap and abstract existing component-based business logic
- Externally-exported Web Services choreographies require B2B standards basis
 - needed to ensure correct and legally-agreed interactions

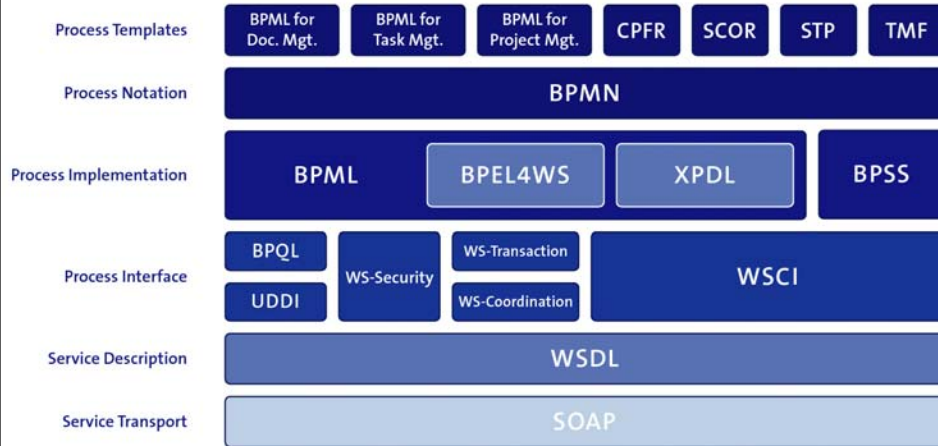
Web Services at Multiple Levels



Web Services System Architecture



Web Services Evolution - BPM



Applying Web Services Technology

Bioinformatics Application Domain

DEXA 2003

Structural Genomic Workflows Supported by Web Services

Presented by Fernanda Baião

Maria Cláudia Cavalcanti, Fernanda Baião,
Shaila C. Rössle, Paulo M. Bisch, Rafael Targino,
Paulo F. Pires, Maria Luiza Campos, Marta Mattoso



Federal University of Rio de Janeiro - Brazil

Presentation Outline

- Motivation
- Web services technology
- A bioinformatics workflow: The MHoLLine
- MHoLLine supported by Web Services
- Conclusion

Motivation

- in silico scientific experiments
 - programs and data resources
 - Multiple combinations – scientific workflows
 - Complex to manage
 - Typical approach
 - Script languages (e.g.: Perl)



Script languages

- Easy way to automate program calls
- Work, but...
 - Typically are specific and hard to re-use
 - Hard-coded parameter values and program calls
 - Difficult to cope with changes

Script languages

- Require
 - local infrastructure
 - Pre-installation of programs within local network
 - Data availability
 - Remote programs and data
 - Requires web interfaces availability
 - Screen scrapping
 - Brittle and unreliable work
 - What if there are no web interfaces available?

What do bioinformatics projects need?

 **Web services!**

- Provides flexibility and interoperability facilities in program and data access

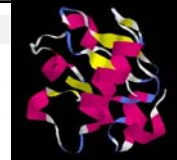
Web services have been pointed out in the bioinformatics area as a potential technology to allow biological data to be fully exploited
[L. Stein, Nature 417, 2002]



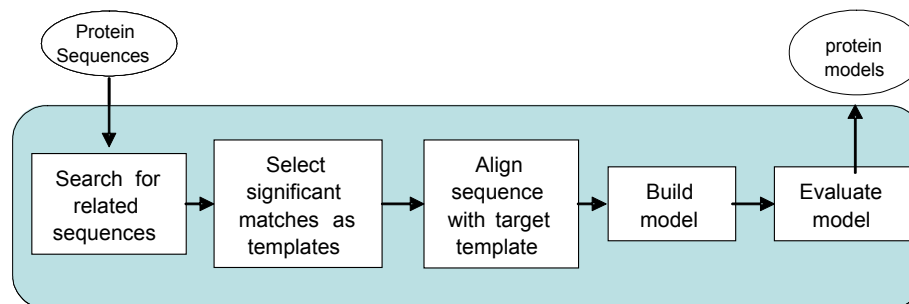
Why web services?

- Interoperability among programs from different platforms
 - SOAP protocol
- Service interface description (data and programs)
 - WSDL
 - Enables any program to understand how to interact
- Service publication
 - UDDI
- Service composition specification
 - BPEL4WS
- Extensibility

The MHoLLine Workflow



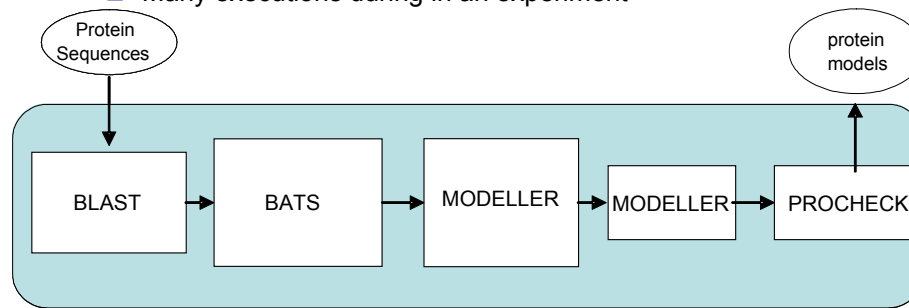
- Bioinformatics workflow for the MHoLLine structural genomic project, under development at IBCCF/UFRJ
- Sophisticated combination of programs that predict the most reliable structure for a sequence using related protein structures as templates



The MHoLine Workflow

Script language approach

- Local programs and data
- Specific set of programs and parameter values
- Partial re-executions are needed, but not allowed
 - High-cost full executions
 - Many executions during in an experiment



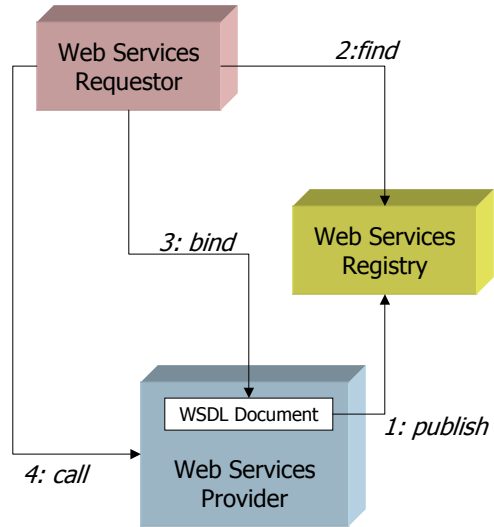
The MHoLine Workflow

Web Services approach

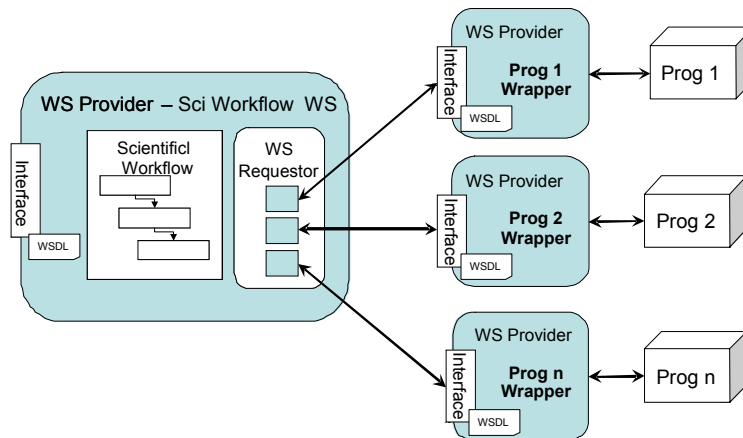
- Programs and data can be published and further browsed to build a composition of programs
 - Semantic information is available
- Programs involved in the MHoLine workflow were encapsulated into web services
 - Interface changes are handled by the web service wrapper

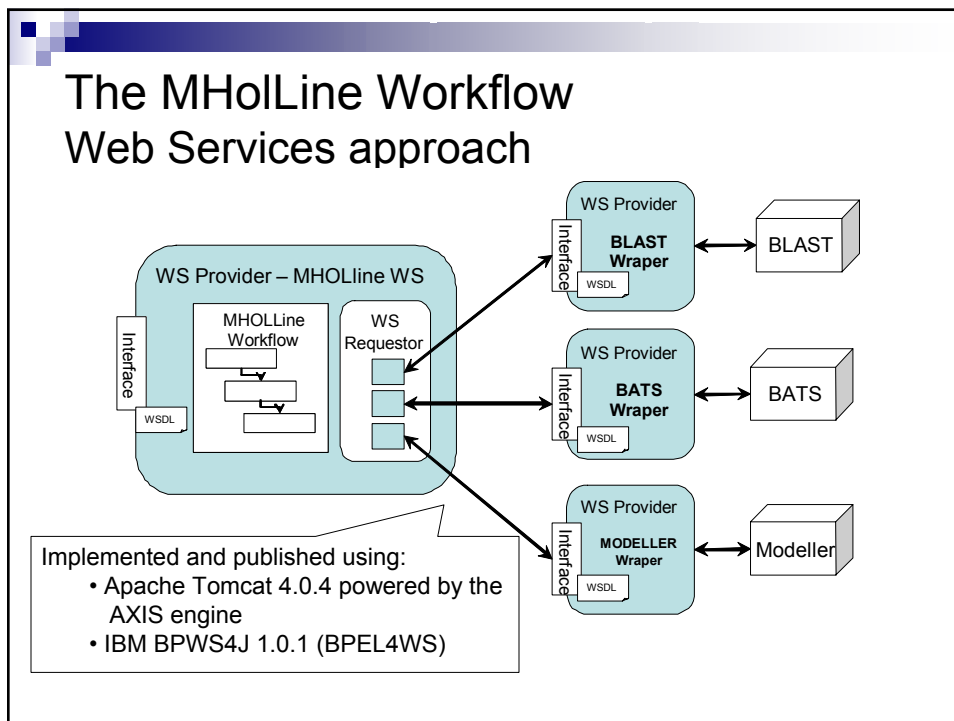
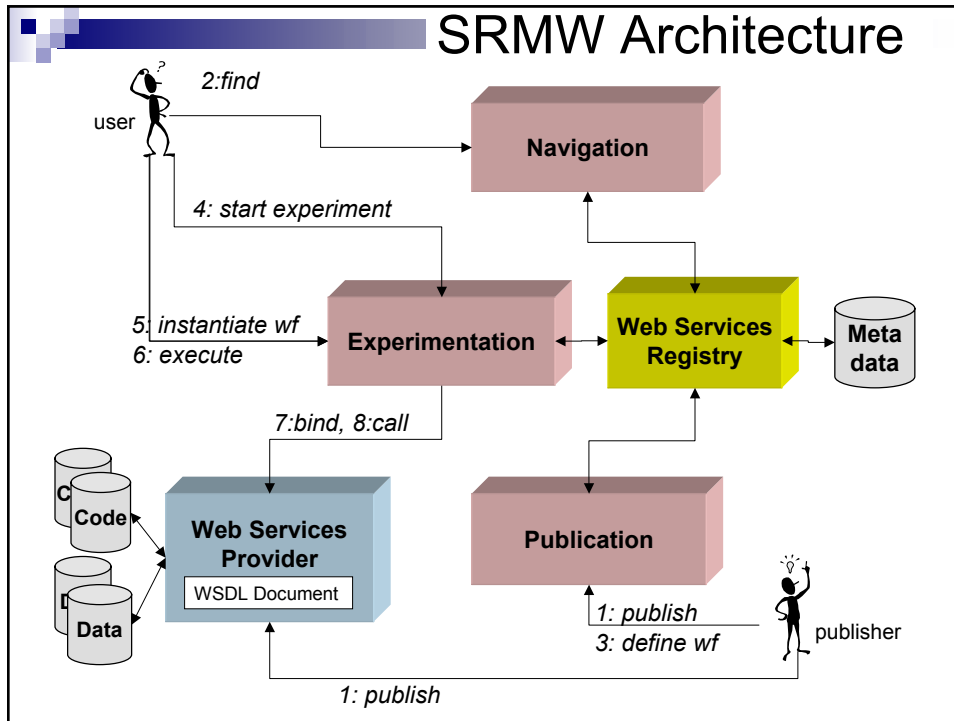
Web Services Architecture

- Standard based, open
- Service Provider
 - Access to remote codes installed in different platforms
 - Legacy codes and data sources can be available as Web services
 - WSDL describes programs and their binding information
- Service Registry
 - UDDI: services classification
 - WSDL is extensible
- Service Request
 - find: identify services
 - bind: gets WSDL
 - call: uses SOAP protocol



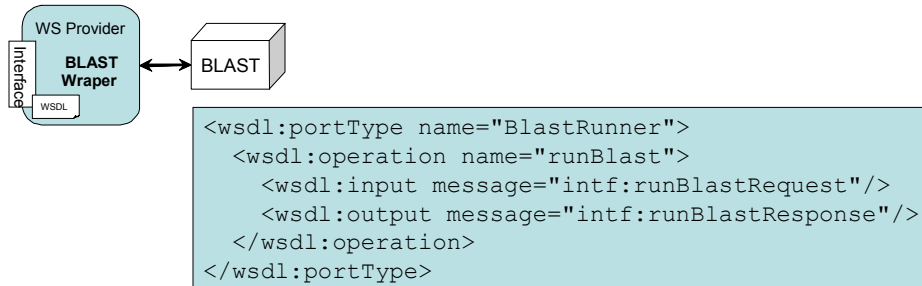
Scientific Workflows Web Services approach





The MHOline Workflow Web Services approach

- Step 1: Encapsulate MHOline programs as web services
 - WSDL for each program
 - Code for mapping input and output data between Web services and legacy programs
 - Web service is invoked through WSDL operations which includes input and output messages

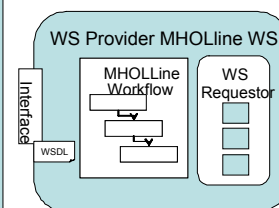


The MHOline Workflow Web Services approach

- Step 2: Define the MHOline Workflow
 - The BPEL4WS language was used to formally specify the composition of the programs
 - Each program is defined as a <partner>

```

<partners>
  <partner
    name="providerblast"
    serviceLinkType="blastns:blastSLT"/>
  ...
  <partner
    name="providermodeller"
    serviceLinkType="modns:modellerSLT"/>
</partners>
  
```



- Step 3: MHOline is published as a web service

The MHOline Workflow Web Services approach

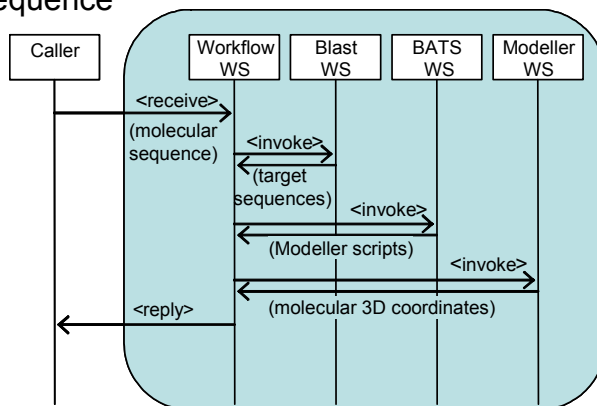
- Step 4: Program composition is defined
 - MHOline partners should be sequentially executed (<sequence>)
 - Input message (<receive>)
 - Service invocation (<invoke>)
 - Output message (<reply>)

```
<sequence name="WorkflowSequence">
  <receive name="receive" partner="caller" portType="tns:runnerPT"
    operation="runnerOP" container="request" createInstance="yes"/>
  ...
  <invoke name="invoke" partner="providerblast"
    portType="blastns:BlastRunner" operation="runBlast"
    inputContainer="blastrequest" outputContainer="blastresponse"/>
  ...
  <reply name="reply" partner="caller"
    portType="tns:runnerPT" operation="runnerOP"
    container="response"/>
</sequence>
```

165

The MHOline Workflow Web Services approach

- Step 5: Execute the MHOline workflow for a molecular sequence



The MHOline Workflow Web Services approach

- Discussion
 - Every step of the MHOLine workflow was automatically executed, with no human interaction
 - BPEL4WS is an expressive language
 - verify data input values
 - start parallel executions
 - allow workflow shortcuts
 - Different configurations of MHOLine workflow can be easily accommodated with the web services approach
 - different parameter values
 - alternative programs
 - Flexible for building other scientific workflows definitions

SRMW Architecture

- Web services alone are not enough
 - Handles interoperability among programs from different platforms
 - Encapsulates program and data resources
- **SRMW architecture**
 - Guides the use of the web service technology
 - Metamodel
 - Organizes semantic issues
 - Represents an application domain
 - Defines model and program categories, and data categories
 - e.g., BLAST x BLAST-P
 - Generic mechanisms for data and program representation
 - Provides semantic interoperability
 - Service publication
 - Enables navigation and interactive service composition
 - workflow definitions

Conclusion

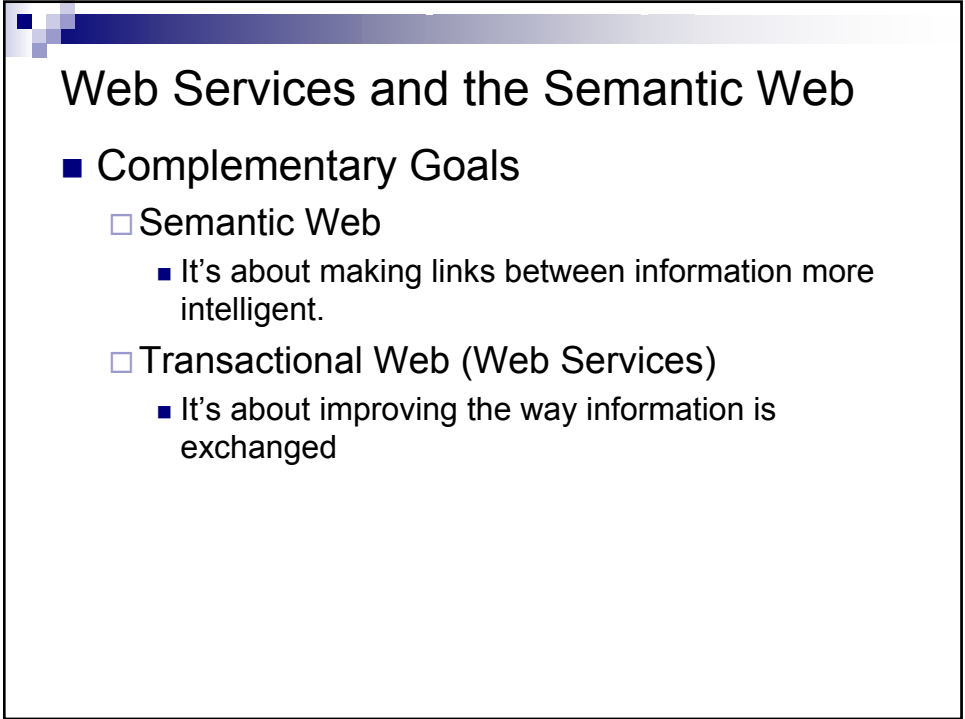
- Rapid growth of the bioinformatics area
 - Raises a lot of management issues to e-scientists
 - Vast amount of programs and data resources
- Web services is on the right track to a full-featured e-scientist lab
 - Need for interoperable, flexible and scalable environments
 - Superior when compared to the script languages approach
 - Overcomes platform incompatibilities
 - Allows ad-hoc services compositions, allows workflows as part of new service compositions

Conclusion

- Show how web services technologies can be used
 - bioinformatics workflow (“MHoLine”)
- Web services alone are not enough
 - lack application-related semantic descriptors
- Metadata support + web services within a framework that supports scientific workflows
 - SRMW architecture
- On the way to data provenance and derivation
 - User access is controlled
 - Web services messages could be easily interpreted to provide mechanisms for capturing and organizing user experiments



Web Services and the Semantic Web



Web Services and the Semantic Web


■ Complementary Goals

□ Semantic Web

- It's about making links between information more intelligent.

□ Transactional Web (Web Services)

- It's about improving the way information is exchanged



Web Services and the Semantic Web

- Different Points of View

- Semantic Web

- Derives the puzzle-pieces from the big picture

- Transactional Web

- Derives the big picture from the puzzle-pieces



Web Services and the Semantic Web

- Meeting in the middle

- Semantic Web

- Provides a formal data-model for Web Services

- Transactional Web

- Provides a technology foundation for the Semantic Web

Concluding Remarks

What business value do
Web Services provide?

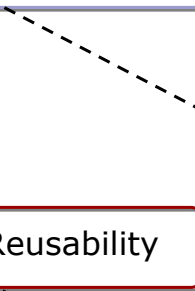
Business Value of Web Services

Fewer Development Risks

New revenue streams

System Reusability

Reduced time-to-market



Fewer Development Risks

- Web Services represent industry standards:
 - standardized protocols (HTTP and SMTP)
 - standardized message formats (SOAP and XML-RPC)
 - standardized service description (WSDL)
 - standardized registry service (UDDI)
- The use of common standards in development reduces the costs and risks involved because there are fewer variable factors

New Revenue Streams

- **Componentized E-Services**
 - Web Services represent the next step in the evolution of the Application Service Provider (ASP) business model
 - Stock quotation engines, sports scores, currency rates, loan amounts/rates, and traditional office applications will all be offered on a fee-per use or subscription basis
- **Dynamic Smart Services**
 - Real-time virtual auctions for everything from home loans, to used college textbooks, to gasoline
 - Ability to search for the best bargain in real-time

New Revenue Streams

■ Mobile Device Utilization

- Web Services open the door to the wireless world by relying upon standard industry formats and protocols
- Due to the small memory footprint, the utilization of standard formats and standard protocols is crucial
- Examples:
 - Purchase a coke with your cell phone
 - Allow doctors to perform a real-time query comparing a patient's symptoms with known diagnoses
 - Access your home's computer system, your bank account, and your travel arrangements via a handheld device

System Reusability

■ Increased Efficiency

- Maintain a library of interchangeable, extensible components that communicate via standardized formats over standardized protocols

■ "Develop Once, Sell Everywhere"

- Develop an application or suite of applications and have them expose their interfaces as Web Services, then resell this package to multiple clients
 - Create an application that processes insurance claims for a hospital. Then turn around and resell this solution to 10, 20, or 100 other hospitals. Since the system is accessed via standardized interfaces, integration is quick and easy

Reduced Time-to-Market

- If you are developing components and systems with:
 - fewer development risks
 - reusable code libraries
 - reusable systems that can be easily configured and accessed by other systems
- Then the result will be a **faster development cycle** and a **quicker deployment schedule**

If clients are charged exactly the same, but you are able to fulfill the needs of 10 times more clients than before...\$\$\$

Business Value of Web Services

Fewer Development Risks

New revenue streams

System Reusability

Reduced time-to-market

= \$\$\$

Web Services Technologies



Phases of Web Services Utilization

- Internal projects:
 - integration among divisions, etc.
- External reach:
 - select partners
- External market:
 - consumption on demand

Summary

- The Web services framework is being defined, standardized and supported by the industry at a record pace.
- Broad industry acceptance and standard compliance will make it ubiquitous.
- Will bring an unprecedented level of interoperability to Web applications.
- The benefits of Web services, however, are not limited to the Web!
 - The Web Services technology aggregates real Business Value
- The Web services Stack is already growing ...

Resources

- Web Services Activity
 - <http://www.w3.org/2002/ws/>
- SOAP W3C Working Drafts - 17 December 2001
 - <http://www.w3.org/2002/ws/#drafts>
 - SOAP 1.2
 - Primer
 - Messaging Framework
 - Adjuncts
 - XML Protocol Drafts
- WSDL W3C Note Released March 2001
 - <http://www.w3.org/TR/wsd/>
- WSDL Schema Specifications
 - Base Definition: <http://schemas.xmlsoap.org/wsd/>
 - SOAP Extension: <http://schemas.xmlsoap.org/wsd/soap/>



Resources

- UDDI

- <http://www.uddi.org>

- Web Services Directory

- <http://www.xmethods.net>

- <http://www.salcentral.com>

- Online Web Services Tools

- <http://www.soapclient.com>