Servicios Web: SOAP y REST

Caso de estudio:
Proyecto Axis (SOAP)

Dr. Víctor J. Sosa Sosa
Laboratorio de Tecnologías de Información
Cinvestav-Tamaulipas
vjsosa@tamps.cinvestav.mx

axis

Objetivos y temas de la sesión

- Introducción a los Servicios Web con SOAP y REST
- Estándares de interoperabilidad para los Servicios Web: portabilidad usando los APIs de Java (recordar que Java ya es oficialmene Open Source bajo GPLv2).
- Conocer algunas opciones para producir y consumir servicios Web con software libre:

axis

  - Proyecto Axis (Java): Desglosamos ejemplo sencillo (construcción, despliegue e invocación de servicios).
    - http://ws.apache.org/axis/ o http://ws.apache.org/axis2/

mono

  - Proyecto Mono (C#): Descripción breve.
    - http://www.mono-project.com/

Tuesday, March 10, 2015

Servicios Web (Web Services)

Definiciones

- Componente de software reutilizable y distribuido que ofrece una funcionalidad concreta, independiente tanto del lenguaje de programación en que está implementado como de la plataforma de ejecución
- Aplicaciones auto-contenidas que pueden ser descritas, publicadas, localizadas e invocadas sobre la Internet (o cualquier otra red).
- Actividades relacionadas con los Servicios Web llevadas en el Web Consortium (W3C): http://www.w3.org/2002/ws/Activity.html

Servicios como abstracción

- La **computación Orientada a Servicio** se fundamenta en una comunicación que se abstrae del modelo de comunicación propio del lenguaje y de la plataforma de ejecución
  - No queremos "saber" si el servicio está programado en Java, Lisp, C, C++,Fortran, etc…
  - No quiero saber si tengo que invocar un procedimiento, método, función, …
  - No quiero saber nada de estructuras de datos en Java, Lisp, C, C++
  - No quiero saber nada de UNIX, Windows,…

Algunas Tecnologías para Servicios Web

Servidores de aplicaciones para servicios Web:

- Axis y el servidor Jakarta Tomcat (de Apache): ws.apache.org/axis/
- Proyecto Mono: http://www.mono-project.com/
- Java Web Services Development Pack (JWSDP) de Sun Microsystems (basado en Jakarta Tomcat): http://java.sun.com/webservices/
- IBM Lotus Domino a partir de la versión 7.0: www.lotus.com
- ColdFusion MX de Macromedia: http://www.adobe.com/products/coldfusion/
- JOnAS (parte de *ObjectWeb* una iniciativa de código abierto) : http://wiki.jonas.objectweb.org/
- Microsoft .NET: http://msdn.microsoft.com/netframework/
- Novell exteNd (basado en la plataforma J2EE): http://www.novell.com/documentation/extend5
- WebLogic: http://www.beasys.com
- WebSphere: http://www.ibm.com/websphere
- Zope es un servidor de aplicaciones Web orientado a objetos desarrollado en el lenguaje de programación Python: http://www.zope.org/
- VERASTREAM de Attachmate WRQ para modernizar o integrar aplicaciones host IBM y VT : http://www.attachmate.com/
- Proyecto NuSoap para PHP: http://dietrich.ganx4.com/nusoap/

Servicios de W3C: tecnologías asociadas

- **HTTP/HTTPS:** Protocolo ampliamente aceptado para transportar los datos
- **XML (Extensible Markup Language):** se usa para estructurar o darle formato a la información contenida en los servicios.
- **SOAP (Simple Object Oriented Protocol):** es usado para definir el protocolo de invocación/servicio.
- **WSDL (Web Services Description Language):** se usa para describir los servicios disponibles.
- **UDDI (Universal Description, Discovery, and Integration):** se utiliza para listar los servicios que están disponibles.
- WS-Security, XML-Signature, XML-Encryption, (esquemas para manejo de seguridad).

SOAP en breve…

- Protocolo basado en XML para intercambio de información
  - Reglas de codificación para instancias de tipos de datos
  - Convenciones para representar invocaciones RPC
- Diseñado para procesamiento entre sistemas ligeramente acoplados
  - Sin manejo de referencias remotas
- Usado con XML Schema
- Independiente del transporte
- *SOAP con Attachments* permite empacar datos de cualquier tipo.

Estructura del mensaje

SOAP1.1

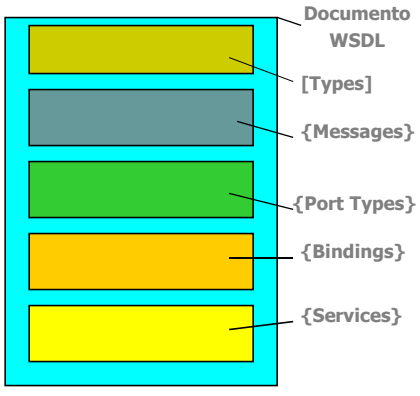
```
<soap:Envelope
      xmlns:soap="http://schemas.xmlsoap.org/soap/encoding/
          "
      soap:encodingStyle="http://schemas.xmlsoap.org/soap/e
          ncoding/"
      <soap:Header>
        ... ...
      </soap:Header>
      <soap:Body>
          <!--  Código del usuario aqui -->
       <soap:Fault>
         ... ...
       </soap:Fault>
      </soap:Body>
</soap:Envelope>
```


WSDL en breve…

- Un documento WSDL describe
  - *Qué* puede hacer el servicio
  - *Dónde* reside
  - *Cómo* invocarlo
- Los WSDLs son como IDLs pero mucho más flexible y extendible
- Define enlaces para SOAP1.1, HTTP GET/POST y MIME
- Las descripciones WSDL pueden ser hechas desde un registro UDDI

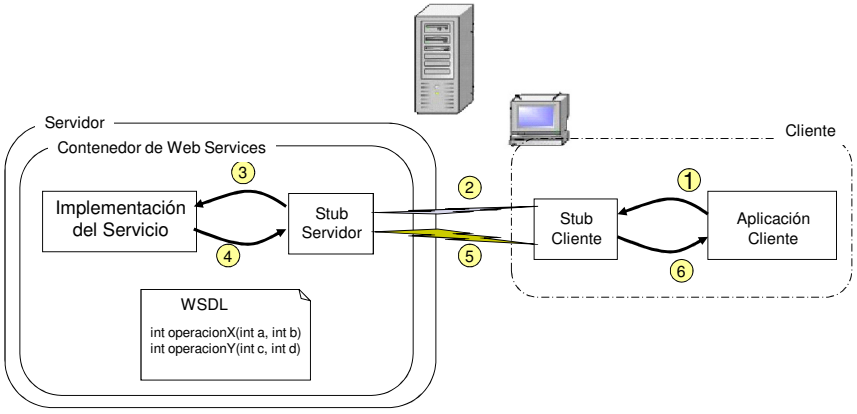
Estructura de documento

WSDL1.1



Web Services: Invocación básica (I)

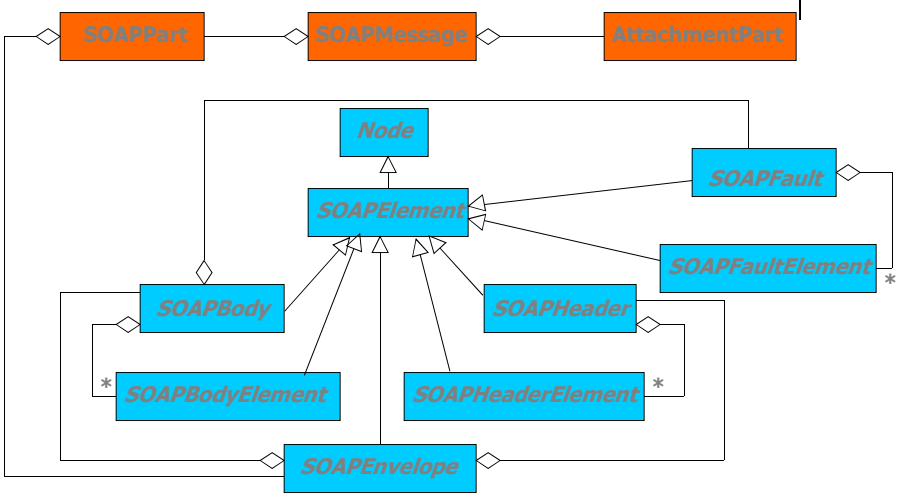
Web Services: Invocación básica (II)



APIs de Java para Servicios Web

- Mensajes SOAP manejados como objetos Java
  - SAAJ (SOAP with Attachments API for Java)
- Modelo de Programación (RPC)
  - JAX-RPC (JSR101), JSR109, EJB2.1
- Modelo de Programación (Message-oriented)
  - JAX-WS (evoultion of JAX-RPC)
- Acceso a descripciones WSDL
  - JWSDL (JSR110)
- Acceso a registros de Servicios Web
  - JAXR (Java API for XML Registries)
- Servicios Web con REST
  - JAX-RS (Java API for RESTful Web Services)

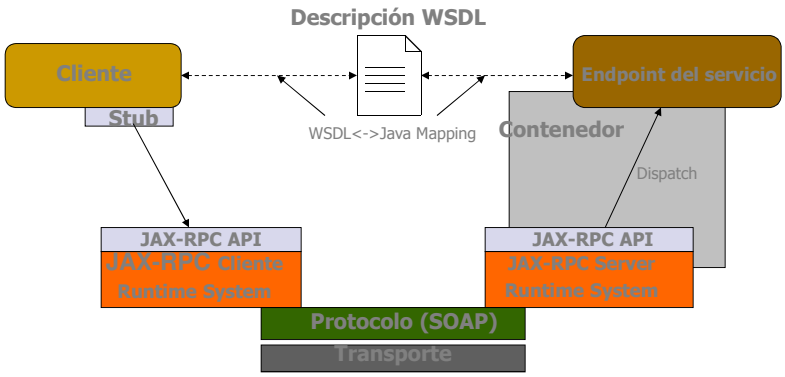
Modelo de Objetos de SAAJ



JAX-RPC

- Traslada de WSDL/XML a Java
- Traslada de Java a WSDL/XML
- Gestiona Mensajes SOAP con archivos adjuntos
- API para Cliente
  - Clases generadas desde WSDL
  - Proxy Dinámico
  - Interfaz de llamada dinámica DII
- Gestor de mensajes SOAP
- Asociación de tipos extensible

JAX-RPC: Arquitectura Física



Apache axis

Axis2: http://axis.apache.org/

axis1: http://axis.apache.org/axis/

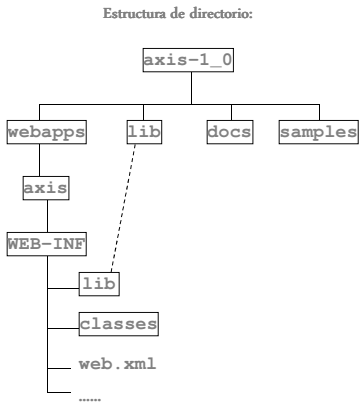
Apache Axis

- Máquina de procesamiento SOAP
  - Sistema cliente JAX-RPC
  - Sistema servidor JAX-RPC (basado en Servlet)
  - Implementación SAAJ
  - Arquitectura flexible y extensible
  - Herramientas, ejemplos, documentación, …
  - Un buen lugar donde aprender Servicios Web !!
- Open-source, auspiciado por Apache Software Foundation

axis

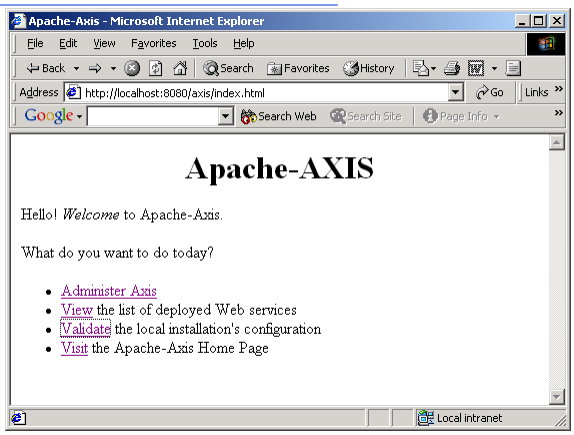
Instalación y Despliegue de Apache Axis

- Asegurarse de tener
  - J2SE SDK 1.4 o posterior
  - Un contenedor de Servlets (ej. Tomcat)
- descargar `xml-axis-rc1-bin.zip` de http://xml.apache.org/axis
- Descomprimirlo y revisar el árbol de directorios. Note que Axis corre como Servlet.
- Desplegar Axis.
  - Copiar el directorio `webapps\axis` al directorio `webapps` de Tomcat.
  - Alternativamente, modificar `server.xml` de Tomcat.
- Correr Tomcat: lanzar **`bin\startup`** del directorio raíz de Tomcat



Probar el despliegue..

- Direccionar el navegador a http://localhost:8080/axis



Un ejemplo sencillo…

•`AddFunction`: Una clase Java sencilla con un método que suma dos enteros. Note la extensión del archivo: `.jws` (refiere a Java Web Service).

•Desplegar. Sólo copiamos el archivo `AddFunction.jws` al directorio `webapps/axis`.

```
// File: AddFunction.jws
public class AddFunction {
  int addInt(int a, int b){
    return(a+b);
  }
}
```

• Examinamos su descripción: WSDL.

Dirigimos el navegador a:

http://localhost:8080/axis/AddFunction.jws?wsdl

Escribiendo un programa Cliente

- Existen muchas formas para escribir un programa Cliente:
  - Usando Interfaces de Invocación Dinámica (DII)
  - Usando la generación de los Stubs desde el archivo de descripción del servicio WSDL
  - Usando un proxy dinámico
- Analizaremos cada una de ellas.

Escribir el cliente requerirá más trabajo que escribir el servicio ;-)

AddFunctionClient – usando DII

```
// Archivo: lección1\client\dii\AddFunctionClient.java

import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.namespace.QName;

public class AddFunctionClient {
  public static void main(String [] args) {
    try {
      String endpoint = "http://localhost:8080/axis/AddFunction.jws";
      Service  service = new Service();
      Call   call = (Call) service.createCall();
      call.setOperationName(new QName(endpoint, "addInt"));
      call.setTargetEndpointAddress( new java.net.URL(endpoint) );
      Integer ret = (Integer)call.invoke(new Object[]{new Integer(5), new Integer(6)});
      System.out.println("addInt(5, 6) = " + ret);
    } catch (Exception e) {
          System.err.println("Execution failed. Exception: " + e);
    }
  }
}
```

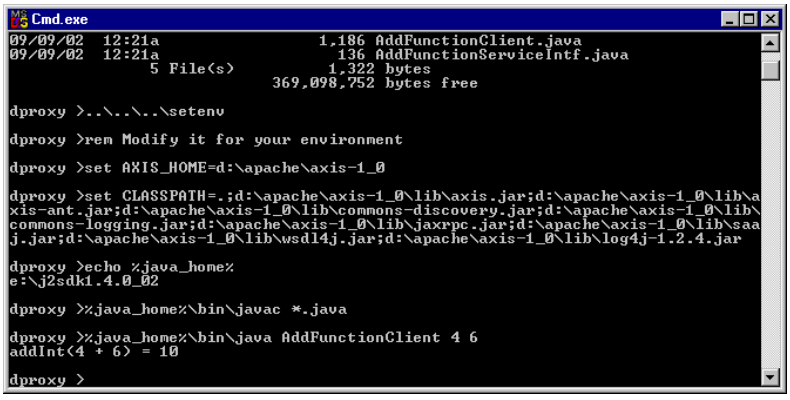

Compilando y ejecutando el Cliente usando DII

```
Cmd.exe
09/09/02  12:21a                 1,045 AddFunctionClient.java
09/11/02  12:32p                 1,619 AddFunctionClient.class
               5 File(s)          2,664 bytes
                            369,098,752 bytes free

dii >echo %java_home%
e:\j2sdk1.4.0_02

dii >..\..\..\setenv

dii >rem Modify it for your environment

dii >set AXIS_HOME=d:\apache\axis-1_0

dii >set CLASSPATH=.;d:\apache\axis-1_0\lib\axis.jar;d:\apache\axis-1_0\lib\axis
-ant.jar;d:\apache\axis-1_0\lib\commons-discovery.jar;d:\apache\axis-1_0\lib\com
mons-logging.jar;d:\apache\axis-1_0\lib\jaxrpc.jar;d:\apache\axis-1_0\lib\saaj.j
ar;d:\apache\axis-1_0\lib\wsdl4j.jar;d:\apache\axis-1_0\lib\log4j-1.2.4.jar

dii >%java_home%\bin\javac AddFunctionClient.java

dii >%java_home%\bin\java AddFunctionClient 4 6
addInt(4 + 6) = 10

dii >
```

AddFunctionClient – usando Proxy Dinámico

```
// File: lección1\client\dproxy\AddFunctionClient.java
import javax.xml.namespace.QName;
import javax.xml.rpc.*;
public class AddFunctionClient {
  public static void main(String [] args) {
    try {
      String wsdlUrl = "http://localhost:8080/axis/AddFunction.jws?wsdl";
      String nameSpaceUri = "http://localhost:8080/axis/AddFunction.jws";
      String serviceName = "AddFunctionService";
      String portName = "AddFunction";
      ServiceFactory serviceFactory = ServiceFactory.newInstance();
      Service afs = serviceFactory.createService(new java.net.URL(wsdlUrl),
                new QName(nameSpaceUri, serviceName));
      AddFunctionServiceIntf afsIntf = (AddFunctionServiceIntf)afs.getPort(
                new QName(nameSpaceUri, portName), AddFunctionServiceIntf.class);
      System.out.println("addInt(5, 3) = " + afsIntf.addInt(5, 3));
      } catch (Exception e) {
         System.err.println("Execution failed. Exception: " + e);
      }
   }
}
```


Compilando y ejecutando el Cliente usando Proxy Dinámico



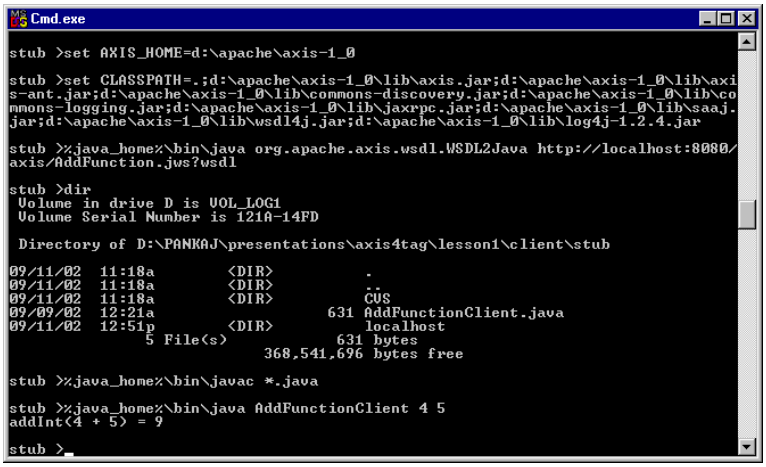
AddFunctionClient – usando generación de Stubs

Genera los stubs:

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

```
// File: lección1\client\stub\AddFunctionClient.java

Import localhost.*;
public class AddFunctionClient{
  public static void main(String [] args) {
    try {
      AddFunctionService afs = new AddFunctionServiceLocator();
      AddFunction af = afs.getAddFunction();
      System.out.println("addInt(5, 3) = " + af.addInt(5, 3));
    } catch (Exception e) {
      System.err.println("Execution failed. Exception: " + e);
    }
  }
}
```


Generando Stubs, Compilando y Ejecutando el Cliente



Descriptores de Despliegue

- El despliegue de JWS es simple, pero tiene limitaciones :
  - Debemos tener el código fuente
  - No se pueden especificar mapeo de tipos personalizados, handlers, etc.
- WSDD (Web Services Deployment Descriptors) permite despliegues más flexibles
  - Handlers en el path de solicitud o respuesta
  - Mapeo de tipos personalizados
  - Se pueden usar diferentes transportes – HTTP/S, TCP/IP, DIME (Microsoft),..
  - Diferentes despachadores – Java Class, EJB, Servlet
  - …

Añadiendo complejidad al ejemplo…

•`AddFunction1`: Clase en Java con un método que suma dos números complejos (`Complex`). El tipo `Complex` es una clase Java definida por el usuario.

•Desplegamos.

- •**Compilamos fuentes**
- •**Copiamos archivos .class**
- •**Escribimos el descriptor de despliegue**
- •**Corremos el `AdminClient`.**

•Examinamos su descripción WSDL. Dirigimos el navegador a:

```
// File: Complex.java
public class Complex {
  public Complex(){}
  public double getR(){ … }
  public void setR(double r){ … }
  …
  public Complex add(Complex c){ … }
```

```
// File: AddFunction1.java
public class AddFunction1 {
  public Complex addComplex (Complex a, Complex b)
  {
    return a.add(b);
  }
}
```

http://localhost:8080/axis/services/AddFunction1Service?wsdl

Descriptor de despliegue

```
// File: leccion\service\deploy.wsdd

<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
   <handler name="print" type="java:LogHandler"/>
   <service name="AddFunction1Service" provider="java:RPC">
   <requestFlow>
     <handler type="print"/>
   </requestFlow>
  <parameter name="className" value="AddFunction1"/>
  <parameter name="allowedMethods" value="*"/>
  <beanMapping qname="myNS:Complex" xmlns:myNS="urn:BeanService"
               languageSpecificType="java:Complex"/>
 </service>
</deployment>
```

Nótese:

(1) xmlns:java

(2) Un handler en el path de la petición

(3) Despacha a un proveedor RPC

(4) Mapeo del tipo Bean

Desplegando el servicio…

```
Cmd.exe
 Volume Serial Number is 121A-14FD

 Directory of D:\PANKAJ\presentations\axis4tag\lesson2\service

09/11/02  11:18a        <DIR>          .
09/11/02  11:18a        <DIR>          ..
09/11/02  11:18a        <DIR>          CVS
09/09/02  12:21a                   105 AddFunction1.java
09/09/02  12:21a                   472 Complex.java
09/09/02  12:21a                 1,112 LogHandler.java
09/09/02  12:21a                   559 deploy.wsdd
09/09/02  12:21a                 1,846 readme.html
09/09/02  12:21a                   111 undeploy.wsdd
               9 File(s)          4,205 bytes
                        368,345,088 bytes free

D:\PANKAJ\presentations\axis4tag\lesson2\service>%java_home%\bin\javac *.java

D:\PANKAJ\presentations\axis4tag\lesson2\service>copy *.class d:\jakarta-tomcat-
4.0.1\webapps\axis\Web-Inf\classes
AddFunction1.class
Complex.class
LogHandler.class
        3 file(s) copied.

D:\PANKAJ\presentations\axis4tag\lesson2\service>%java_home%\bin\java org.apache
.axis.client.AdminClient deploy.wsdd
- Processing file deploy.wsdd
- <Admin>Done processing</Admin>

D:\PANKAJ\presentations\axis4tag\lesson2\service>_
```

AddFunction1Client – usando generación de Stubs

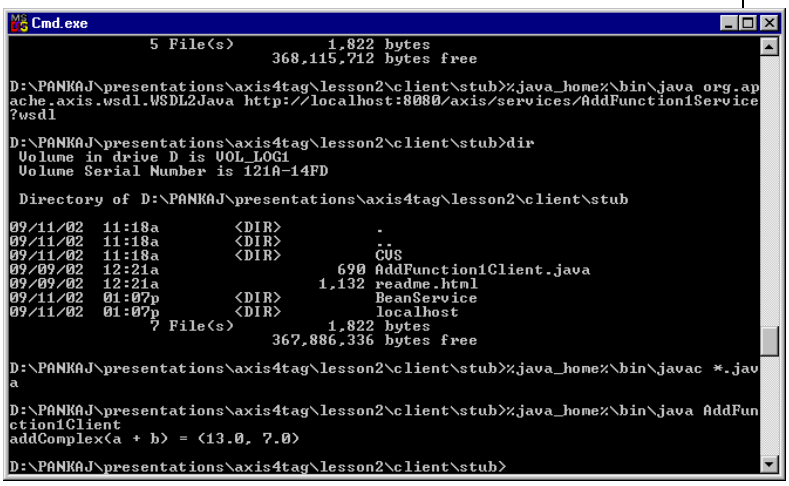
Generar los stubs:

```
java org.apache.axis.wsdl.WSDL2Java \
  http://localhost:8080/axis/services/AddFunction1Service?wsdl
```

```
// File: leccion\client\stub\AddFunction1Client.java,
import localhost.*;
import BeanService.*;
public class AddFunction1Client {
  public static void main(String [] args) throws Exception {
    Complex a = new Complex();
    Complex b = new Complex();
    a.setR(10.0); a.setI(5.0);
    b.setR(3.0); b.setI(2.0);
    AddFunction1Service afs = new AddFunction1ServiceLocator();
    AddFunction1 af = afs.getAddFunction1Service();
    Complex ret = af.addComplex(a, b);
    System.out.println("addComplex(a + b) = ("
             + ret.getR() + ", " + ret.getI() + ")");
  }
}
```

Clase Generada

Ejecutando el cliente…



Axis: Características interesantes

- SOAP con archivos adjuntos
- Mapeo entre tipos personalizados (serializadores en forma de plugings)
- Invocación de una vía (One-way)
- Intercambio de documento
- Despachador a EJBs
- Transporte HTTPS y autenticación mutua.
- Autenticación basada en login y password
- …

Web Services with REST

REST

- Roy Fielding and his doctoral thesis, “Architectural Styles and the Design of Network-based Software Architectures.”
- Why is the Web so prevalent and ubiquitous?
- What makes the Web scale?
- How can I apply the architecture of the Web to my own applications?

- The set of the architectural principles given by Roy Fielding to answer these questions - REpresentational State Transfer (REST)

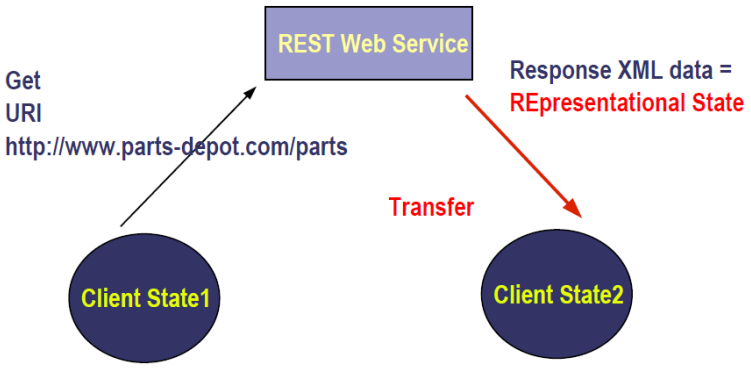
REST

REST stands for Representational State Transfer

- It is an architectural ***pattern*** for developing web services as opposed to a ***specification***.
- REST web services communicate over the HTTP specification, using HTTP vocabulary:
  - Methods (GET, POST, etc.)
  - HTTP URI syntax (paths, parameters, etc.)
  - Media types (xml, json, html, plain text, etc)
  - HTTP Response codes.

What is REST?

REpresentational **S**tate **T**ransfer



REST - set of principles

- *Addressable resources*

  -Resource oriented, and each resource must be addressable via a URI

  -The format of a URI is standardized as follows:

  scheme://host:port/path?queryString#fragment
- *A uniform, constrained interface*

  -Uses a small set of well-defined methods to manipulate your resources.

  -The idea behind it is that you stick to the finite set of operations of the application protocol you're distributing your services upon.

REST - set of principles

- *Representation-oriented*

  -Interaction with services using representations of that service.

  -Different platforms, different formats - browsers -> HTML, JavaScript -> JSON and a Java application -> XML?
- *Communicate statelessly*

  -Stateless applications are easier to scale.
- *Hypermedia As The Engine Of Application State (HATEOAS)*

  -Let your data formats drive state transitions in your applications.

Set of Constraints: Summary

- Client-Server: Separation of concerns
- Client-Stateless-Server: Visibility, Reliability, Scalability
- Caching : improves efficiency, scalability and user perceived performance, reduces average latency
- Uniform Interface: simplify overall system architecture and improved visibility of interactions
- Layered System: Simplifying components, Shared caching, Improved Scalability, Load balancing
- Code-On-Demand: Simplifies clients, Improves extensibility

HTTP and REST

A REST service framework provides a **controller** for routing HTTP requests to a request handler according to:

- The HTTP method used (e.g. GET, POST)
- Supplied path information (e.g /service/listItems)
- Query, form, and path parameters
- Headers, cookies, etc.

HTTP-REST Vocabulary

A typical HTTP REST URL:

```
http://my.store.com/fruits/list?category=fruit&limit=20
```

protocol host name path to a resource query string

- The **protocol** identifies the transport scheme that will be used to process and respond to the request.
- The **host name** identifies the server address of the resource.
- The **path** and **query string** can be used to identify and customize the accessed resource.

HTTP-REST Vocabulary

HTTP Methods supported by REST:

- GET – Requests a resource at the request URL
  - Should not contain a request body, as it will be discarded.
  - May be cached locally or on the server.
  - May produce a resource, but should not modify on it.
- POST – Submits information to the service for processing
  - Should typically return the new or modified resource.
- PUT – Add a new resource at the request URL
- DELETE – Removes the resource at the request URL
- OPTIONS – Indicates which methods are supported
- HEAD – Returns meta information about the request URL

REST over HTTP – Uniform interface

- CRUD operations on resources
  - Create, Read, Update, Delete
- CRUD Operations — 4 main HTTP methods

	Verb	Noun
Create (Single)	POST	Collection URI
Read (Multiple)	GET	Collection URI
Read (Single)	GET	Entry URI
Update (Single)	PUT	Entry URI
Delete (Single)	DELETE	Entry URI

http://www.javapassion.com/webservices/RESTPrimer.pdf

HTTP-REST Request Basics

- The **HTTP request** is sent *from the client*.
  - Identifies the location of a **resource**.
  - Specifies the **verb**, or HTTP **method** to use when accessing the resource.
  - Supplies optional **request headers** (name-value pairs) that provide additional information the server may need when processing the request.
  - Supplies an optional **request body** that identifies additional data to be uploaded to the server (e.g. form parameters, attachments, etc.)

HTTP-REST Request Basics

Sample Client Requests:

- A typical client GET request:

```
GET /view?id=1 HTTP/1.1
User-Agent: Chrome
Accept: application/json
[CRLF]
```

Requested Resource (path and query string)
Request Headers
(no request body)

- A typical client POST request:

```
POST /save HTTP/1.1
User-Agent: IE
Content-Type: application/x-www-form-urlencoded
[CRLF]
name=x&id=2
```

Requested Resource (typically no query string)
Request Headers
Request Body (e.g. form parameters)

HTTP-REST Response Basics

- The **HTTP response** is sent *from the server*.
  - Gives the **status** of the processed request.
  - Supplies **response headers** (name-value pairs) that provide additional information about the response.
  - Supplies an optional **response body** that identifies additional data to be downloaded to the client (html, xml, binary data, etc.)

HTTP-REST Response Basics

Sample Server Responses:

```
HTTP/1.1 200 OK                          } Response Status
Content-Type: text/html                  } Response Headers
Content-Length: 1337
[CRLF]
<html>                                   } Response Body (content)
  <!-- Some HTML Content. -->
</html>
```

```
HTTP/1.1 500 Internal Server Error } Response Status
```

```
HTTP/1.1 201 Created                     } Response Status
Location: /view/7                        } Response Header
[CRLF]
Some message goes here.                  } Response Body
```

HTTP Request/Response As REST

15

http://www.javapassion.com/webservices/RESTPrimer.pdf

Producing REST Services

REST services in Java web applications can be implemented in several ways:

- As a plain Java Servlet
  - Adequate for very simple REST services.
  - Requires a lot of “boiler plate” code for complex services.
- Using a REST service framework.
  - Eliminates the need to write “boilerplate” code.
  - Typically integrates with other technologies, such as Spring.

Java provides the JAX-RS specification for use by providers of REST service frameworks.

Web Application Description Language (WADL)

- A machine-readable XML description of HTTP-based web applications (typically REST web services).
- Models the resources provided by a service and the relationships between them.
- Intended to simplify the reuse of web services that are based on the existing HTTP architecture of the Web.
- Platform and language independent and aims to promote reuse of applications beyond the basic use in a web browser.
- Was submitted to the World Wide Web Consortium by Sun Microsystems on 31 August 2009
- The REST equivalent of SOAP's Web Services Description Language (WSDL), which can also be used to describe REST web services.

WADL elements

- Application
- Grammar
- Include
- Resources
- Resource
- Resource Type
- Method
- Request
- Response
- Representation
- Param
- Link
- Doc

https://wadl.dev.java.net/wadl20090202.pdf

REST Vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user

- Using Web Services and SOAP, the request would look something like this:

```
<?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 pb="http://www.acme.com/phonebook">
                <pb:GetUserDetails>
                        <pb:UserID>12345</pb:UserID>
                </pb:GetUserDetails>
        </soap:Body>
</soap:Envelope>
```

REST Vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user

- And with REST? The query will probably look like this:
  http://www.acme.com/phonebook/UserDetails/12345

- GET /phonebook/UserDetails/12345 HTTP/1.1
  Host: www.acme.com
  Accept: application/xml

- Complex query:
  http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe

WS on the Java Stack

Although developers may implement REST web services however they choose, the Java Stack team is best equipped to support the following:

- Apache CXF
  - A JAX-RS web service framework
- Spring MVC
  - An MVC framework built upon the Spring Platform (does not implement the JAX-RS specification)

CXF Web Services Framework

Apache CXF is a robust framework designed specifically for producing and consuming web services:

- It is open-source and free to use.
- It supports several web service standards and JSR APIs.
- It provides tooling and configuration for JAX-WS and JAX-RS services.
- It provides integration with the Spring Application Framework, the core technology upon which most of the Java Stack is built.

CXF Web Services Framework

Apache CXF provides robust support for several web service patterns and specifications:

- JSR APIs: JAX-WS, JAX-RS, JSR-181 annotations, SAAJ
- WS-* specifications for web service interoperability.
- Rich support support for message transports, protocol bindings, content negotiation, data bindings, and so forth.
- Flexible, lightweight deployment in a variety of web application containers or stand-alone.
- Tooling for code generation
- Tools for WSDL and WADL publishing.

REST Services with JAX-RS

JAX-RS is a Java standard API for REST services:

- Services are annotation driven
- Provides support for data binding.
- Provides advanced APIs for content negotiation.

CXF provides an implementation of JAX-RS:

- Supports CXF filters, interceptors, and invokers to customize and extend the service.
- Configurable through Spring.
- Integrates with security providers.

REST Services with Spring MVC

Spring MVC is a model-view-controller framework built upon the Spring Application Framework.

- Annotation driven
- Supports a RESTful pattern of routing requests to web resources using HTTP vocabulary.
- Not an implementation of the JAX-RS specification.

JAX-RS or Spring MVC?

Some guidelines for choosing your solution:

- Both JAX-RS and Spring MVC can produce REST services.
- Spring MVC is a web application framework that can be used as service framework.
  - Provides better validation
  - Supports internationalization
- JAX-RS is a primarily a services framework.
  - Provides support for WADL generation
  - Can use CXF interceptors, filters, etc.
- Match the framework to the needs and purpose of the project.
- Don't mix both in same web application unless you need unique features from each.
  - If your project needs both, consider separate web applications.

JAX-RS Basics

JAX-RS applications consist of a hierarchy of resources:

- Resources are served up by a CXF controller servlet.
- Each REST resource is mapped to a request URI that is relative to the CXF controller servlet path.
- The relative path of each resource is mapped to a method on a JAX-RS annotated service bean that returns the resource.
- Service bean methods that return a resource must be annotated with a *single* JAX-RS HTTP method annotation (e.g @GET)
- Additional, optional annotations may be applied to the class, class fields, and method parameters to customize the service API.
- JAX-RS service beans form the “view” or public interface of your REST web service application.

JAX-RS Basics

An example REST service class:

```
package org.lds.tech.training.lab.ws;

import javax.ws.rs.GET;
import org.springframework.stereotype.Controller;

@Controller
public class HelloWebServiceRest {

    @GET
    public String sayHello() {
        return "Hello, World!";
    }
}
```

- At least one method must be annotated with an HTTP verb (e.g. @GET)
- The @Controller annotation makes the class discoverable by Spring

JAX-RS Basics

Example Spring configuration:

- Example location: WEB-INF/example-servlet.xml

```
<stack-rs:produce>
    <stack-rs:interfaces>
        <ref bean="helloWebServiceRest"/>
    </stack-rs:interfaces>
</stack-rs:produce>
```

- A reference to the JAX-RS annotated service bean is passed to the Stack RS “produce” namespace handler.
- Multiple service beans may be supplied under the Stack RS “interfaces” element.
- Each bean will be scanned by CXF for annotated resources that can be served up RESTfully.

JAX-RS Basics

Stack RS Namespace Usage:

- Element name: <stack-rs:produce/>
- Optional Attributes:
  - **secured** – whether to secure the endpoint
  - **extensions** – whether to support the use of .xml and .json extensions
  - **address** – the relative address of the REST service
  - **authentication-manager-ref**
- Child elements:
  - **interfaces** – JAX-RS annotated service beans
  - **providers** – provider beans for content negotiation
  - **in-interceptors**
  - **out-interceptors**

JAX-RS Basics

The JAX-RS resource hierarchy is described using "Web Application Descriptor Language", or WADL. Apache CXF generates a WADL descriptor to expose the following information about your service:

- All the resources available through REST calls.
- The relative path to each resource
- The HTTP method required to access each resource.
- How the HTTP response will represent, or format, each resource.

JAX-RS Basics

An example WADL descriptor:

```
<application xmlns="http://wadl.dev.java.net/2009/02"
             xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <resources
      base="http://localhost:8080/example/Services/rest">
    <resource path="/">
      <method name="GET">
        <response>
          <representation mediaType="application/octet-stream">
            <param name="result" style="plain" type="xs:string"/>
          </representation>
        </response>
      </method>
    </resource>
  </resources>
</application>
```