# CSI 3540

## Structures, techniques et normes du Web

# Services Web

**Objectif:**

- Concepts de base liés aux **services Web**
- **Génération automatiquement** d'un service et d'un client

**Lectures:**

- Web Technologies (2007) § 9 Pages 486–502

# Plan

1. Motivation
2. Générer automatiquement des services et des clients
3. Présentation du laboratoire 10

# Motd

- Implémentation de **Quake II** reposant sur **HTML 5**
  - Canvas (WebGL)
  - Web Storage API
- code.google.com/p/quake2-gwt-port/

# Services Web

- **“The Internet is quickly replacing the desktop as the medium of choice for modern application development.”**

J.T. Howerton (2007) Service-Oriented Architecture and Web 2.0. IT Pro, mai/juin 2007

# Services Web

- On a beaucoup parlé d'**Applications Web**
- Qu'est-ce qu'un **Service Web** ?
- Au sense large : c'est un système informatique pour supporter des intactions **machine-machine** à travers un réseau

# Services Web

- À la base de **SOA** (Service-Oriented Architecture)
- “A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a **uniform means to offer, discover, interact with, and use capabilities to produce desired effects** (...)” OASIS

# SOA pour l'entreprise

- Les applications d'une entreprise doivent :
  - gérer une base d'usagers
  - offrir des services authentification
  - utiliser des algorithmes d'encryption
  - offrir des services de journalisation (logging)
  - etc.

# SOA pour l'entreprise

- Moins de programmation
- Moins de documentation
- Réduction des coûts
- SQA (Software Quality Assurance) accrue
- Standardisation accrue
- Sécurité accrue

# SOA pour l'entreprise

- La possibilité de créer un point de défaillance unique

# Discussion

- **Quels sont les besoins pour supporter des interactions machine-machine?**
  - Il faut décrire les opérations
  - Il faut des protocoles de communications
  - Il faut possiblement des registres

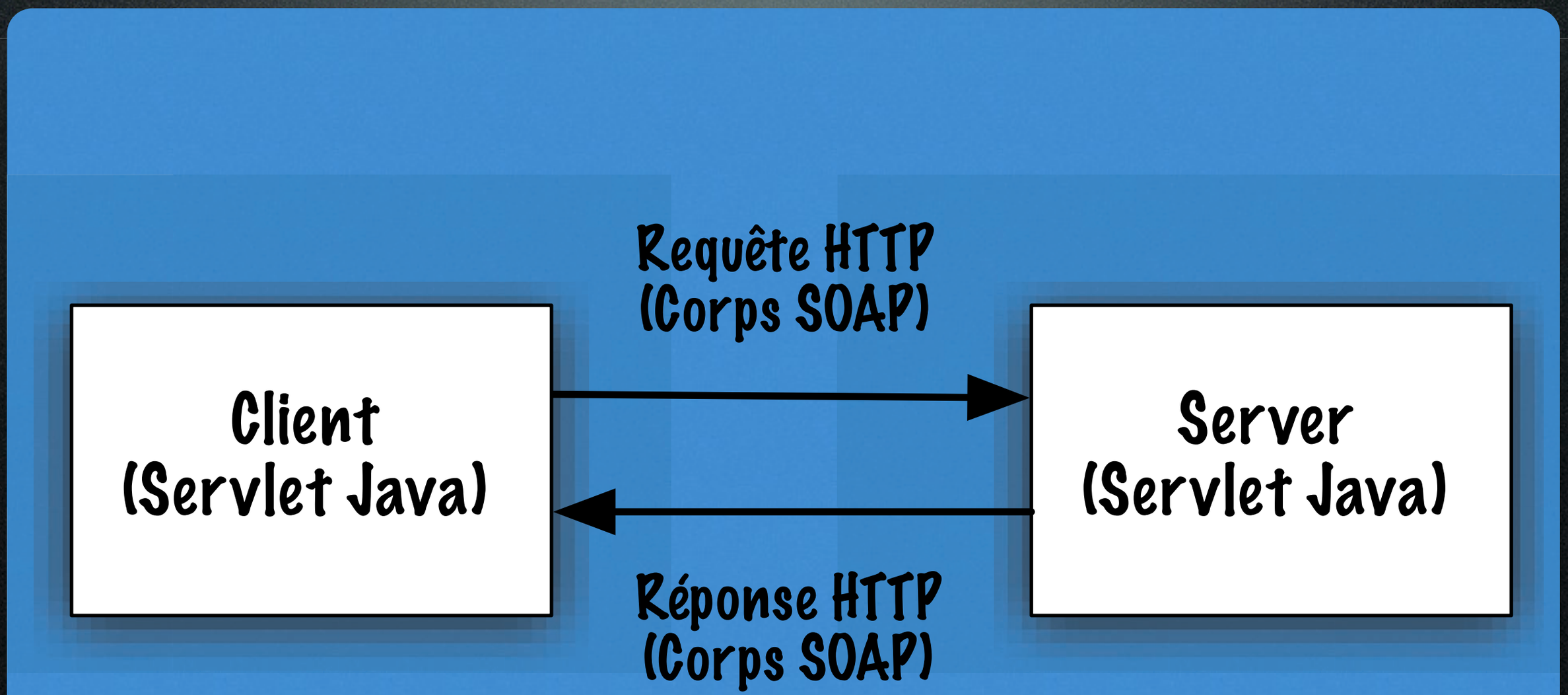
# Remarques

- **SOAP** (Simple Object Access Protocol) – Ce vocabulaire XML est généralement invisible, caché derrière « message exchange pattern »
- **REST** (REpresentational State Transfer) est une architecture logicielle
- Cette introduction se limite aux services Web qui reposent sur SOAP (JAX-WS)

# Service Web

- Spécifiquement : un **Service Web** supporte les échanges de **messages XML** (**SOAP**) entre **clients** et **serveurs** à l'aide de **HTTP**

# Service Web


Requête HTTP
(Corps SOAP)
Client
(Servlet Java)
Server
(Servlet Java)
Réponse HTTP
(Corps SOAP)

# Pile de protocoles de communication

| **Couche** | **Technologie** |
|---|---|
| Répertoire | UDDI |
| Description | **WSDL** |
| Message | **SOAP** |
| Transport | HTTP |

# Services Web

- Le vocabulaire XML **WSDL** (**Web Services Description Language**) décrit les opérations supportées par le serveur
- Les clients et les serveurs s'échangent des messages XML dont le vocabulaire est **SOAP** (**Simple Object Access Protocol**, **Service Oriented Architecture Protocol**)

WSDL et SOAP : deux technologies clés des services Web

- Ils reposent sur un ensemble de protocoles **XML**
- Grand **choix de langages de programmation** pour le développement des services **et** clients
- **Java** est un excellent choix pour plusieurs raisons : **multiplateformes**, plusieurs bibliothèques, **fortement typé**, etc.
- Grand **choix d'environnement** (OS, CPU) pour les clients et serveurs

# Remarques

- Puisque les **Services Web** reposent sur des technologies **XML**
- Ils sont **indépendants** de tout langage de programmation, environnement, SE, etc.
- Autant côté **client** que **serveur**

# Exemples de services

- **B2B** : Business-to-business
- **Agents** : EBay, Amazon, etc.
- **Services** : suivi de collis, météo, captcha, etc.
- **Grid** : Calculs distribués
- **Sciences** : banques de données, etc.

# Granularité

- Les services Web ne s'appliquent pas qu'aux grandes applications (telles que les accès aux inventaires)
- Une application de commerce électronique peut, par exemple, faire appel à un service Web pour le traitement des **transactions de cartes de crédit**

# Exemples spécifiques

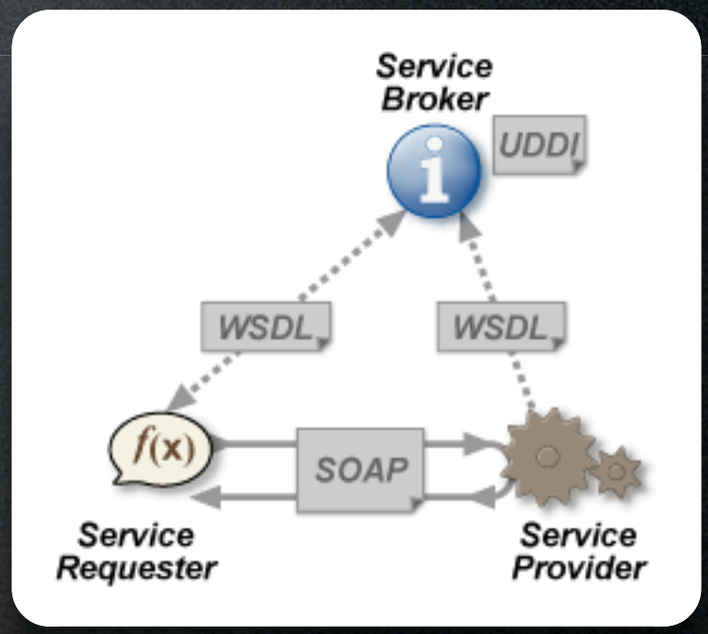
- **Amazon Web Services**
  [ aws.amazon.com ] :
    - Amazon E-Commerce Service (Amazon ECS 4.0)
    - Amazon Simple Storage Service (Amazon S3)
    - Amazon Flexible Payments Service (Amazon FPS)

# Exemples spécifiques

- **api.google.com/GoogleSearch.wsdl**
- **www.saq-b2b.com**

# Répertoires de services

- http://www.XMethods.com
- http://www.webserviceX.NET
- http://www.RemoteMethods.com
- http://www.WebServiceList.com
- http://www.StrikeIron.com
- ...
- http://www.google.com

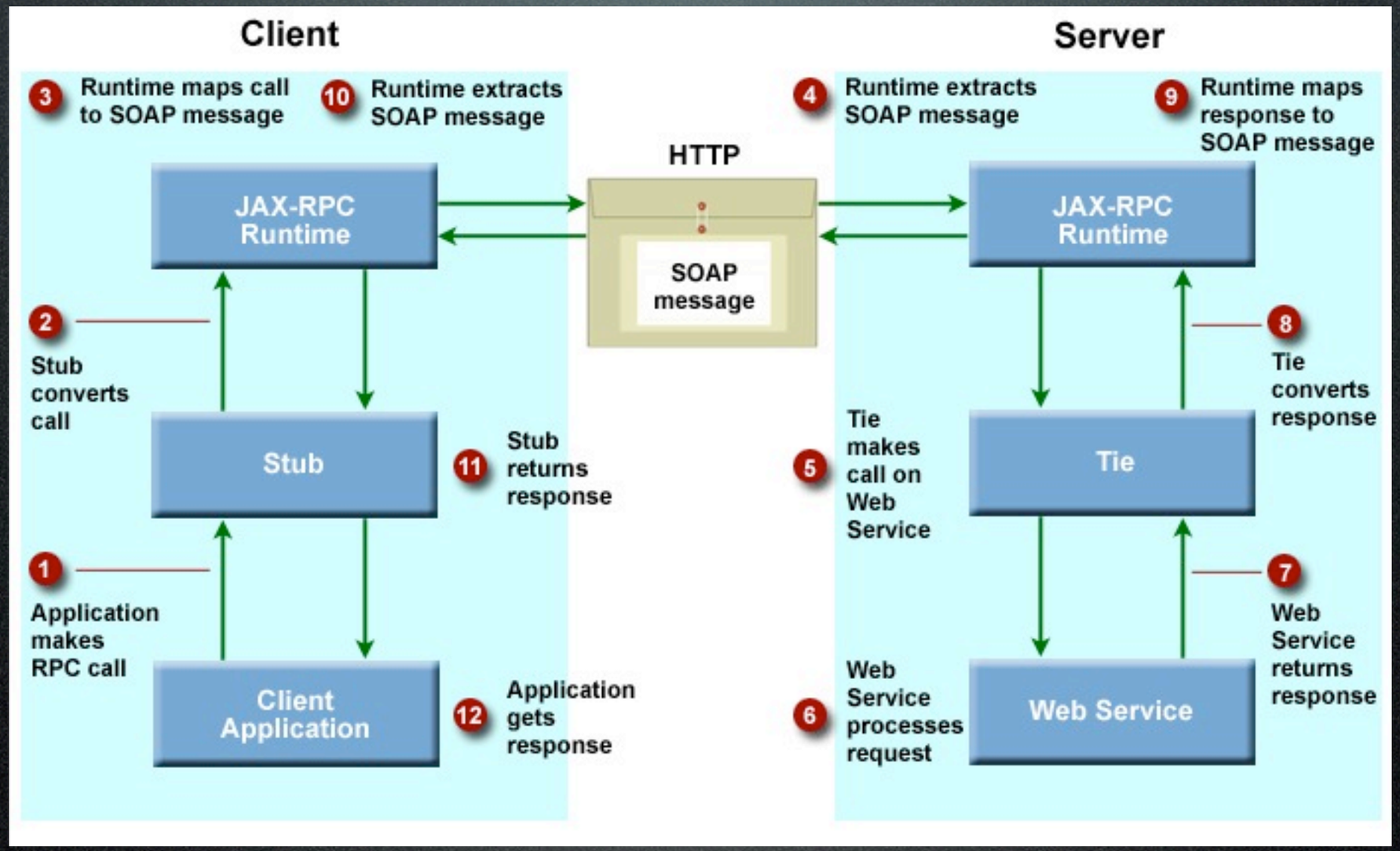


# Comment ?

- Les applications échangent, via **HTTP**, des documents **XML** dont le vocabulaire est **SOAP**
- **SOAP** sert à représenter des données structurées (objets et tableaux)
- Les opérations d'un service sont décrites à l'aide de **WDSL** (Web Services Definition Language)

# Comment ?

- Puisque la technologie est **mature**, **standardisée**, et **ordinolingue** (machine-readable)
- Il existe plusieurs outils pour la **génération automatique** de code pour les serveurs et les clients
- Peu de programmation**!**




Source : http://java.sun.com/developer/technicalArticles/WebServices/WSPack2/jaxrpc.html

# Message HTTP requête

```
POST /converter/currency HTTP/1.1
content-type: text/xml; charset="utf-8"
content-length: 513
soapaction: ""
host: localhost:8080

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
              xmlns:ns0="http://tempuri.org/wsdl"
              xmlns:ns1 ="http://tempuri.org/types"
              env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
   <ns0:fromDollars>
     <double_1 xsi:type="xsd:double">1.0</double_1>
   </ns0:fromDollars>
  </env:Body>
</env:Envelope>
```

Invoke la méthode fromDollars et la valeur du paramètre, de type double, est 1.0

# Message HTTP réponse

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.4
SOAPAction: ""
Content-Type: text/xml; charset="utf-8"
Transfer-Encoding: chunked
Date: Sat, 11 Dec 2004 17:50:31 GMT
Server: Sun-Java-System/JWSDP-1.3

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
              xmlns:ns0="http://tempuri.org/types"
              env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <ans1:fromDollarsResponse xmlns:ans1 ="http://tempuri.org/wsdl">
      <result href="#ID1"/>
    </ans1:fromDollarsResponse>
    <ns0:ExchangeValues id="ID1" xsi:type="ns0:ExchangeValues">
      <dollars xsi:type="xsd:double">1.0</dollars>
      <euros xsi:type="xsd:double">0.746826</euros>
      <yen xsi:type="xsd:double">102.56</yen>
    </ns0:ExchangeValues>
  </env:Body>
</env:Envelope>
```

La réponse (valeur de retour de l'appel de méthode) est de type ExchangeValues et comprend 3 valeurs, toutes de type double

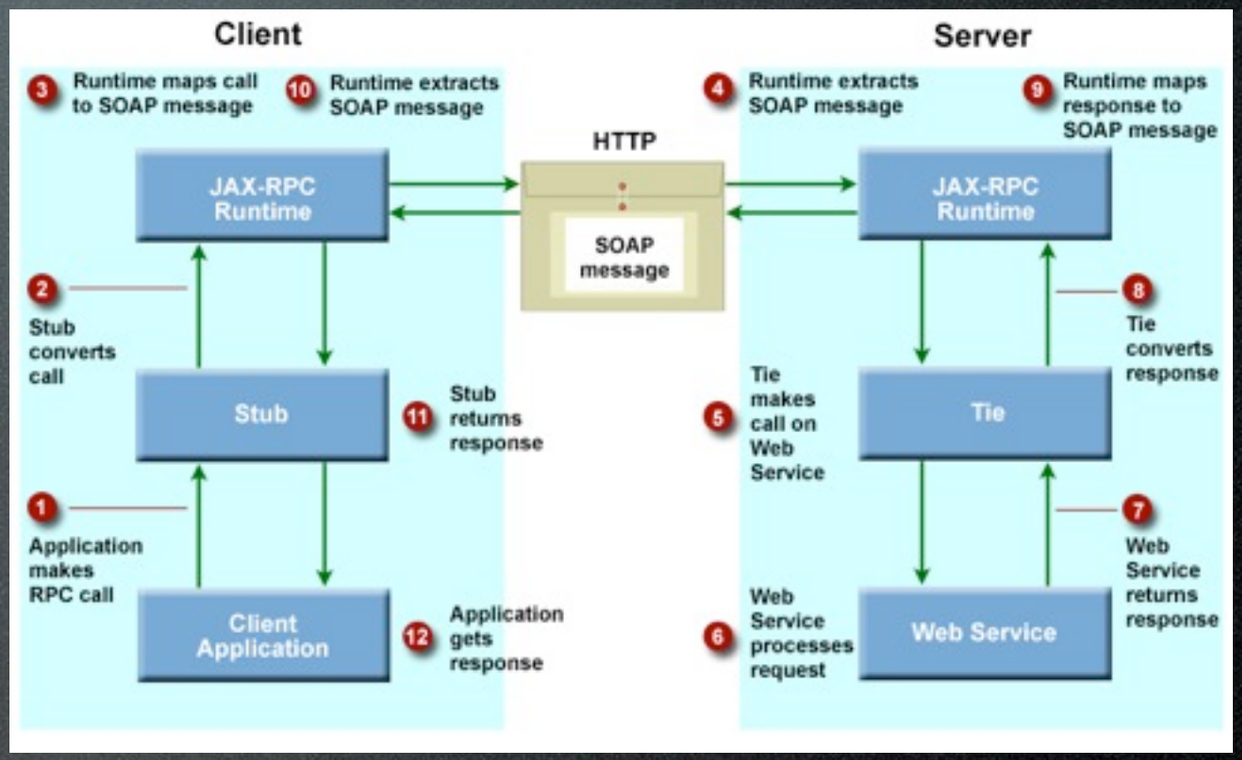
# Comment?

- Tant du côté serveur que client, les programmes développés interagissent avec un **objet mandataire** qui lui **interagit avec l'environnement d'exécution**
- Du côté client, l'objet mandataire est une souche (**stub**)
- Du côté serveur, l'objet mandataire est une attache (**tie**)

# Comment?

- Du côté client, l'objet mandataire est une souche (**stub**) reproduit les fonctionnalités du service localement

- Du côté serveur, l'objet mandataire est une attache (**tie**) reçoit les requêtes du client et fait appel aux méthodes correspondantes du service



- « A **stub** is a client-side proxy for a remote object which is responsible for communicating method invocations on remote objects to the server where the actual remote object implementation resides. A client's reference to a remote object, therefore, is actually a reference to a local stub. »

- « A **tie** for a remote object is a server-side entity similar to a skeleton, but which communicates with the client using the IIOP protocol. »

- « A **skeleton** for a remote object is a JRMP protocol server-side entity that has a method that dispatches calls to the actual remote object implementation. »

# Service Web : côté serveur

- Une **interface** Java spécifie les **opérations**
- Une ou plusieurs classes implémentent ces opérations
- **La génération du Servlet est automatisée**

# Service Web : côté client

- Créer un **objet mandataire** (proxy)
  - Un objet local qui représente le service
  - Les accès au service se font par le biais d'appels de méthodes de l'objet mandataire

# Messages SOAP

- Tant du côté serveur que du côté client, l'**environnement d'exécution** (runtime), Java-WS, se charge de la génération ainsi que de l'analyse syntaxique des **messages SOAP** représentant les appels de méthodes et les valeurs de retours

# Remarques

- **Java SE 6** supporte **JAX-WS 2.1** (Java API for XML-Web Services)
  - **apt** - annotation processing tool
  - **wsgen** - generates JAX-WS portable artifacts used in JAX-WS web services (côté serveur?)
  - **wsimport** - generates JAX-WS portable artifact (côté client?)

# Remarques

- **GlassFish 3** supporte **JAX-WS 2.2** (Java API for XML-Web Services)
  - Il installer, via « Update Center », **Metro 2.0**

# Premier exemple

## Motd et Java SE 6

# Motd

- Considérons d'abord un exemple complet, la théorie suivra
- L'exemple est simple, tâchez de déceler les similitudes avec une application plus complexe (service météo, inventaire, etc.)
- **Problématique** : concevoir un service Web qui génère une nouvelle citation (**mot du jour**) pour chaque appel

- **Applications :**
  - Un client de courriel pourrait faire appel au service afin d'insérer une citation dans la signature des messages (**GUI**)
  - Un système informatique pourrait faire appel au service afin d'afficher une citation lors de l'ouverture de session (**ligne de commande**)
  - Une application Web (JSPX, par exemple) pourrait faire appel au service afin d'ajouter un élément dynamique à la page (**application Web**)

# Motd : côté serveur

- Il faut **décrire** le service à l'aide d'une **interface** :

```
package util;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService
@SOAPBinding( style = Style.RPC )

public interface Motd {
    @WebMethod
    public String getMotd();
}
```

# Motd : implémentation

```
package util;
import javax.jws.WebService;
@WebService( endpointInterface = "util.Motd" )
public class MotdImpl implements Motd {
    private static String[] mots = {
        "L'homme sage apprend de ses erreurs, L'homme plus sage .... (Confusius)",
        "Le sage ne dit pas ce qu'il sait, le sot ne sait pas ce qu'il dit. (Sagesse turque)",
        "L'esprit est le contraire de l'argent, moins on en a, plus on est satisfait. (Voltaire)",
        "Cerveau. Appareil avec lequel nous pensons que nous pensons. (Ambrose Bierce)",
        "On mesure l'intelligence d'un individu ... (Emmanuel Kant)",
        "Tu ne peux pas tout enseigner à un homme ; .... (Galilée)"
    };
    public String getMotd() {
        return mots[ (int) ( Math.random() * mots.length ) ];
    }
}
```

**La programmation côté serveur est terminée!**

# Service Endpoint Interface (SEI)

- « A **service endpoint interface** or **service endpoint implementation** (SEI) is a Java **interface** or **class**, respectively, that declares the methods that a client can invoke on the service. An interface is not required when building a JAX-WS endpoint. The web service implementation class implicitly defines an SEI. »

# Package javax.jws

| Enum Summary | |
|---|---|
| WebParam.Mode | The direction in which the parameter flows |

| Annotation Types Summary | |
|---|---|
| HandlerChain | Associates the Web Service with an externally defined handler chain. |
| Oneway | Indicates that the given @WebMethod has only an input message and no output. |
| WebMethod | Customizes a method that is exposed as a Web Service operation. |
| WebParam | Customizes the mapping of an individual parameter to a Web Service message part and XML element. |
| WebResult | Customizes the mapping of the return value to a WSDL part and XML element. |
| WebService | Marks a Java class as implementing a Web Service, or a Java interface as defining a Web Service interface. |

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES All Classes

*Java™ Platform Standard Ed. 6*

# Programme Test (optionnel – ne fait partie de l'exemple)

```
import util.*;
class Main {
    public static void main( String[] args ) {
        Motdl m = new MotdImpl();
        System.out.println( m.getMotd() );
    }
}
```

```
> java Main
Le sage ne dit pas ce qu'il sait, le sot ne sait pas ce qu'il dit. (Sagesse turque)
```

# Faisons le point

- Nous avons conçu un composant logiciel pour afficher une nouvelle citation pour chaque appel de la méthode **getMotd()**
- Une **interface** décrit ce composant
- Nous avons créé et **validé** son implémentation
- Ce qui suit nous permettra d'ajouter les fonctionnalités nécessaires pour en faire un **Service Web**

# MotdPublisher (programme autonome)

```
package util;
import javax.xml.ws.Endpoint;;
public class MotdPublisher {
    public static void main( String[] args ) {
      Endpoint.publish( "http://localhost:9876/motd", new MotdImpl() );
    }
}
```

# Instructions

- **javac -d bin src/util/ * .java**
- **javac -cp bin -d bin src/Main.java**
- **java -cp bin Main**

L'esprit est le contraire ...(Voltaire)

- **java -cp bin util.MotdPublisher**
- **curl http://localhost:9876/motd?wsdl**

```
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://util/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             targetNamespace="http://util/"
             name="MotdImplService">
  <types></types>
  <message name="getMotd"></message>
  <message name="getMotdResponse">
    <part name="return" type="xsd:string"></part>
  </message>
  <portType name="Motd">
    <operation name="getMotd">
      <input message="tns:getMotd"></input>
      <output message="tns:getMotdResponse"></output>
    </operation>
  </portType>
  <binding name="MotdImplPortBinding" type="tns:Motd">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"></soap:binding>
    <operation name="getMotd">
      <soap:operation soapAction=""></soap:operation>
      <input>
        <soap:body use="literal" namespace="http://util/"></soap:body>
      </input>
      <output>
        <soap:body use="literal" namespace="http://util/"></soap:body>
      </output>
    </operation>
  </binding>
  <service name="MotdImplService">
    <port name="MotdImplPort" binding="tns:MotdImplPortBinding">
      <soap:address location="http://localhost:9876/motd"></soap:address>
    </port>
  </service>
</definitions>
```

# Client Ruby

```
#! /usr/bin/ruby

require 'soap/wsdlDriver'
wsdl_url = 'http://localhost:9876/motd?wsdl'
service = SOAP::WSDLDriverFactory.new( wsdl_url ).create_rpc_driver
result = service.getMotd
puts "Motd :: #{ result }"
```

```
> motdClient
Motd :: L'esprit est le contraire de l'argent... (Voltaire)
```

# Client Perl (nécessite SOAP::Lite)

```
#! /usr/bin/perl
use SOAP::Lite;
my $url = 'http://127.0.0.1:9876/motd?wsdl';
my $service = SOAP::Lite->service( $url );
print "Motd :: ", $service->getMotd();
```

```
> motdClient
Motd :: L'esprit est le contraire de l'argent... (Voltaire)
```

# Client Java (version 1)

```
package client;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.URL;

import util.*;

public class MotdClient  {

    public static void main( String[] args ) throws java.net.MalformedURLException {

        URL url = new URL( "http://localhost:9876/motd?wsdl" );
        QName qname = new QName( "http://util/", "MotdImplService" );
        Service service = Service.create( url, qname );
        Motd eif = service.getPort( Motd.class );
        System.out.println( "Motd :: " + eif.getMotd() );

    }

}
```

# Client Java (version 2)

- **wsimport -p client -keep http://localhost:9876/motd?wsdl**

  produit **Motd.java** et **ModImplService.java**

- **Client :**

```
package client;
public class MotdClient {
    public static void main( String[] args ) {
      MotdImplService service = new MotdImplService();
      Motd port = service.getMotdImplPort();
      System.out.println( port.getMotd() );
    }
}
```

# Remarques

- Tant côté serveur que client, le programmeur n'a pas à produire le document XML (**WSDL**) qui décrit le service, ni à manipuler les documents XML (**SOAP**) qui servent aux appels de méthodes et encodage des valeurs de retour

- www.site.uottawa.ca/~turcotte/teaching/csi-3540/lectures/lab-10/01_motd.jar

# Premier exemple

## Motd et Glassfish

# Glassfish

La description d'un service à l'aide d'une interface est optionnelle. Ici, il n'y a que l'implémentation.

```
package util;
import javax.jws.WebService;

@WebService
public class Motd {
    private static String[] mots = {
        "L'homme sage apprend de ses erreurs, L'homme plus sage .... (Confusius)",
        "Le sage ne dit pas ce qu'il sait, le sot ne sait pas ce qu'il dit. (Sagesse turque)",
        "L'esprit est le contraire de l'argent, moins on en a, plus on est satisfait. (Voltaire)",
        "Cerveau. Appareil avec lequel nous pensons que nous pensons. (Ambrose Bierce)",
        "On mesure l'intelligence d'un individu ... (Emmanuel Kant)",
        "Tu ne peux pas tout enseigner à un homme ; .... (Galilée)"
    };
    @WebMethod
    public String getMotd() {
        return mots[ (int) ( Math.random() * mots.length ) ];
    }
}
```

**La programmation côté serveur est terminée!**

# Glassfish

1. Assurez-vous que **Metro** est installé!
2. Déployez le .war
3. Testez
4. Obtenez l'URL du WSDL et développé un client

Installed Components

http://localhost:4848/updateCenter/installed.jsf

Google

Home | Version | Logout | Help

**User:** admin | **Domain:** domain1 | **Server:** localhost

# GlassFish™ v3 Administration Console

There are 25 update(s) available.

**Tree**

- Common Tasks
  - Registration
  - GlassFish News
  - Enterprise Server
  - Applications
  - Lifecycle Modules
  - Resources
    - JDBC
    - Connectors
    - Resource Adapter Configs
  - Configuration
    - JVM Settings
    - Logger Settings
    - Web Container
    - EJB Container
    - Ruby Container
    - Security
    - Transaction Service
    - HTTP Service
    - Virtual Servers
    - Network Config
    - Thread Pools
    - Admin Service

| | | | | |
|---|---|---|---|---|
| glassfish-registration | Application Servers | 3.0-74.2 | 289KB | stable.glassfish.org |
| glassfish-web | Application Servers | 3.0-74.2 | 3MB | stable.glassfish.org |
| python2.4-minimal | System Tools | 2.4.5.0-38.2493 | 8MB | stable.glassfish.org |
| pkg | System Tools | 1.122.2-38.2493 | 6MB | stable.glassfish.org |
| glassfish-jcdi | Application Servers | 3.0-74.2 | 1MB | stable.glassfish.org |
| javadb-common | Databases and Tools | 10.5.3.0-1 | 66KB | stable.glassfish.org |
| wxpython2.8-minimal | System Tools | 2.8.10.1-38.2493 | 44MB | stable.glassfish.org |
| glassfish-web-profile | Application Servers | 3.0-74.2 | 35KB | stable.glassfish.org |
| glassfish-jca | Application Servers | 3.0-74.2 | 1MB | stable.glassfish.org |
| glassfish-hk2 | Application Servers | 3.0-74.2 | 643KB | stable.glassfish.org |
| glassfish-jts | Application Servers | 3.0-74.2 | 392KB | stable.glassfish.org |
| glassfish-management | Application Servers | 3.0-74.2 | 834KB | stable.glassfish.org |
| felix | OSGi Service Platform Release 4 | 2.0.2-0 | 726KB | stable.glassfish.org |
| glassfish-jdbc | Application Servers | 3.0-74.2 | 770KB | stable.glassfish.org |
| glassfish-jsf | Application Servers | 2.0.2-10 | 2MB | stable.glassfish.org |
| metro | Web Services | 2.0-29 | 10MB | dev.glassfish.org |
| glassfish-corba-base | Application Servers | 3.0.0-41 | 463KB | stable.glassfish.org |
| glassfish-web-incorporation | Application Servers | 3.0-74.2 | 35KB | stable.glassfish.org |
| glassfish-scripting | Application Servers | 3.0-74.2 | 250KB | stable.glassfish.org |
| glassfish-jta | Application Servers | 3.0-74.2 | 95KB | stable.glassfish.org |
| glassfish-upgrade | Application Servers | 3.0-74.2 | 146KB | stable.glassfish.org |

Edit Application

http://localhost:4848/updateCenter/installed.jsf

Google

Home | Version | Logout | Help

**User:** admin | **Domain:** domain1 | **Server:** localhost

GlassFish™ v3 Administration Console

There are 25 update(s) available.

**Tree**

- Common Tasks
  - Registration
  - GlassFish News
  - Enterprise Server
  - **Applications**
  - Lifecycle Modules
- Resources
  - JDBC
  - Connectors
  - Resource Adapter Configs
- Configuration
  - JVM Settings
  - Logger Settings
  - Web Container
  - EJB Container
  - Ruby Container
  - Security
  - Transaction Service
  - HTTP Service
  - Virtual Servers
  - Network Config
  - Thread Pools
  - Admin Service

# Edit Application

Save | Cancel

Modify an existing application or module.

**Name:** motd

**Status:** ☑ **Enabled**

**Virtual Servers:** server

Associates an Internet domain name with a physical server

**Context Root:** /motd

Path relative to server's base URL

**Description:**

**Location:** ${com.sun.aas.instanceRootURI}/applications/motd/

**Libraries:**

**Modules and Components (4)**

| Module Name | Engines | Component Name | Type | Action |
|---|---|---|---|---|
| motd | [web, webservices] | ---------- | ---------- | Launch |
| motd | | jsp | Servlet | |
| motd | | default | Servlet | |
| motd | | Motd | Servlet | View Endpoint |

Web Service Endpoint Information

http://localhost:4848/updateCenter/installed.jsf

Google

Home | Version | Logout | Help

**User:** admin | **Domain:** domain1 | **Server:** localhost

GlassFish™ v3 Administration Console

There are 25 update(s) available.

Tree

- Common Tasks
  - Registration
  - GlassFish News
  - Enterprise Server
  - **Applications**
  - Lifecycle Modules
  - Resources
    - JDBC
    - Connectors
    - Resource Adapter Configs
  - Configuration
    - JVM Settings
    - Logger Settings
    - Web Container
    - EJB Container
    - Ruby Container
    - Security
    - Transaction Service
    - HTTP Service
    - Virtual Servers
    - Network Config
    - Thread Pools
    - Admin Service

# Web Service Endpoint Information

Back

View details about a web service endpoint.

| | |
|---|---|
| **Application Name:** | motd |
| **Tester:** | /motd/MotdService?Tester |
| **WSDL:** | /motd/MotdService?wsdl |
| **Endpoint Name:** | Motd |
| **Service Name:** | http://util/ |
| **Port Name:** | MotdPort |
| **Deployment Type:** | 109 |
| **Implementation Type:** | SERVLET |
| **Implementation Class Name:** | util.Motd |
| **Endpoint Address URI:** | /motd/MotdService |
| **Namespace:** | util.Motd |
| **Description:** | |

MotdService Web Service Tester

http://localhost:8080/motd/MotdService?Tester

Google

# MotdService Web Service Tester

This form will allow you to test your web service implementation (WSDL File)

---

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

### Methods :

public abstract java.lang.String util.Motd.getMotd()

getMotd ()

---

Method invocation trace

http://localhost:8080/motd/MotdService?Tester

Google

# getMotd Method invocation

---

### Method parameter(s)

| Type | Value |
|---|---|

---

### Method returned

java.lang.String : "**Cerveau. Appareil avec lequel nous pensons que nous pensons. (Ambrose Bierce)**"

---

### SOAP Request

---

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:getMotd xmlns:ns2="http://util/"/>
    </S:Body>
</S:Envelope>
```

---

### SOAP Response

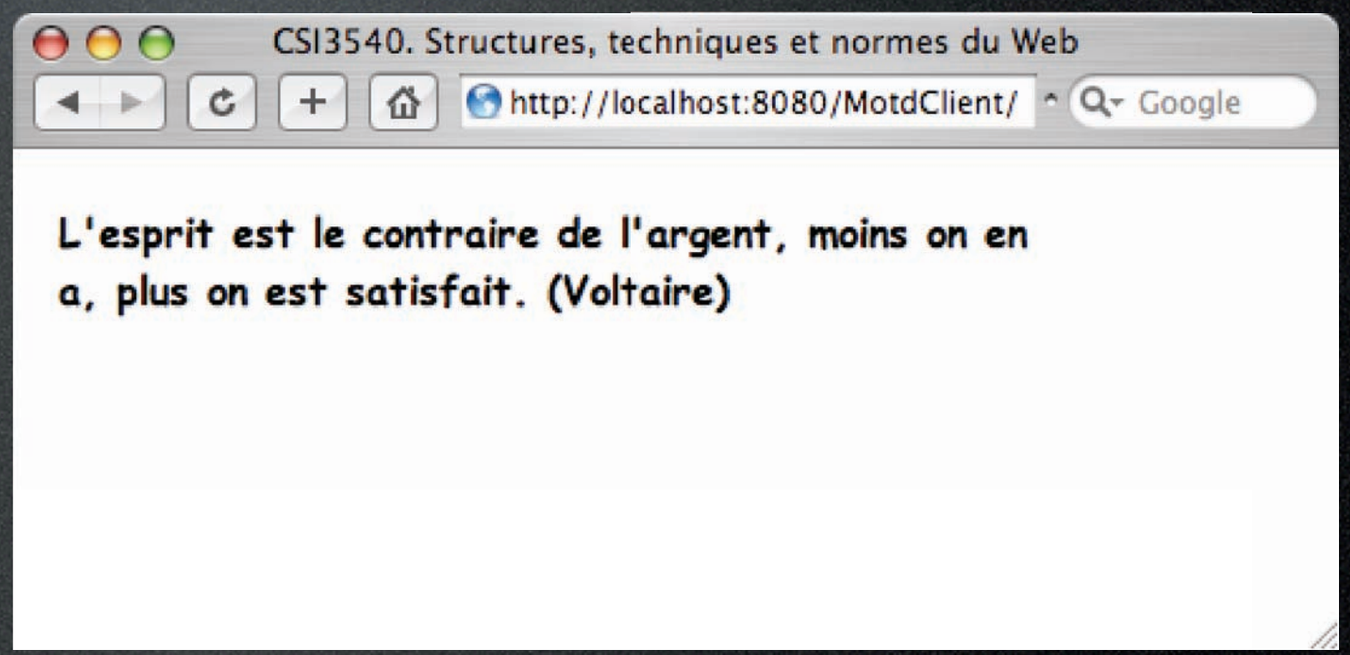
---

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:getMotdResponse xmlns:ns2="http://util/">
            <return>Cerveau. Appareil avec lequel nous pensons que nous pensons. (Ambrose Bierce)</return>
        </ns2:getMotdResponse>
    </S:Body>
</S:Envelope>
```

---

# Application (Web)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:core="http://java.sun.com/jsp/jstl/core"
      lang="fr-CA">

 <jsp:useBean scope="application" id="m" class="client.MotdBean"/>

 <head>
  <title>CSI3540. Structures, techniques et normes du Web</title>
 </head>

 <body>
  <p>
   <b>${ m.motd }</b>
  </p>
 </body>
</html>
```



# Application Web

1. **wsimport -p client -keep http://localhost:8080/motd/MotdService?wsdl**

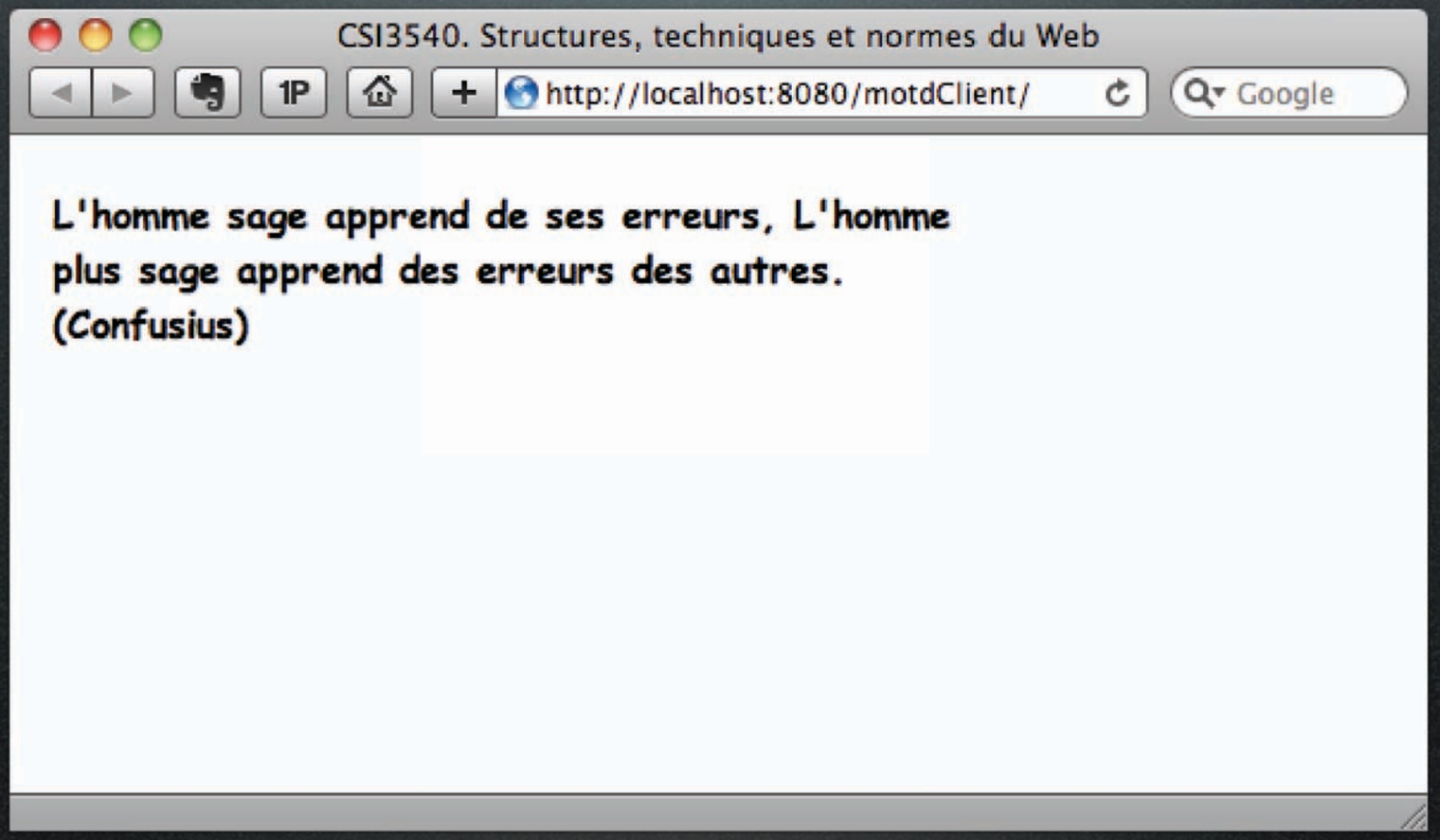
   Produit **MotdService.java** (et autres classes nécessaires)

2. Créer le JavaBean

3. Déployez l'application

# Application Web

```
package client;
public class MotdBean {

    private Motd port;

    public MotdBean() {
        MotdService service = new MotdService();
        port = service.getMotdPort();
    }

    public String getMotd() {
        return port.getMotd();
    }
}
```


CSI3540. Structures, techniques et normes du Web
http://localhost:8080/motdClient/
Google
L'homme sage apprend de ses erreurs, L'homme plus sage apprend des erreurs des autres.
(Confusius)

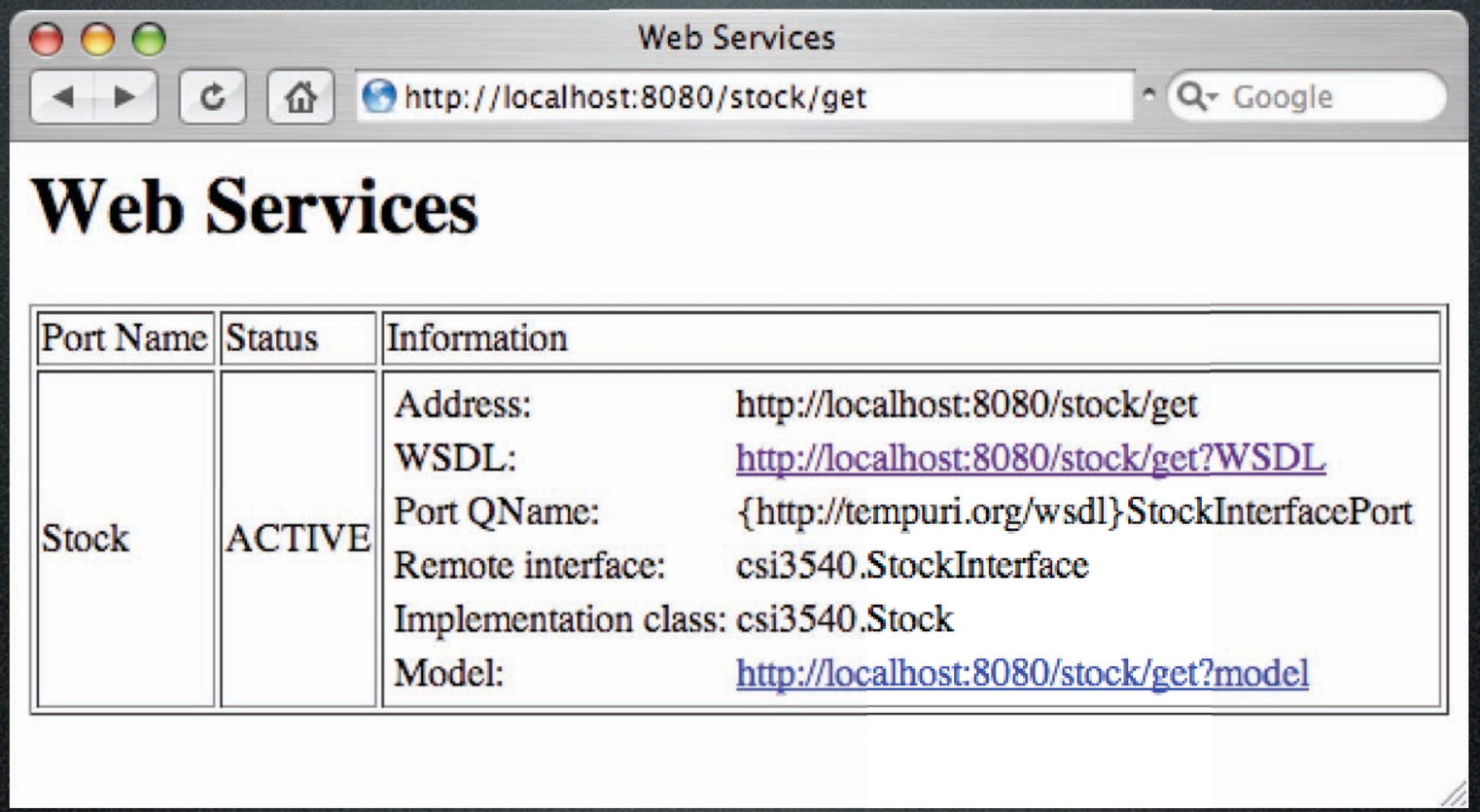
# Remarques

- Tant côté serveur que client, le programmeur n'a pas à produire le document XML (**WSDL**) qui décrit le service, ni à manipuler les documents XML (**SOAP**) qui servent aux appels de méthodes et encodage des valeurs de retour

- www.site.uottawa.ca/~turcotte/teaching/csi-3540/lectures/lab-10/01_motd.jar

# Laboratoire 10

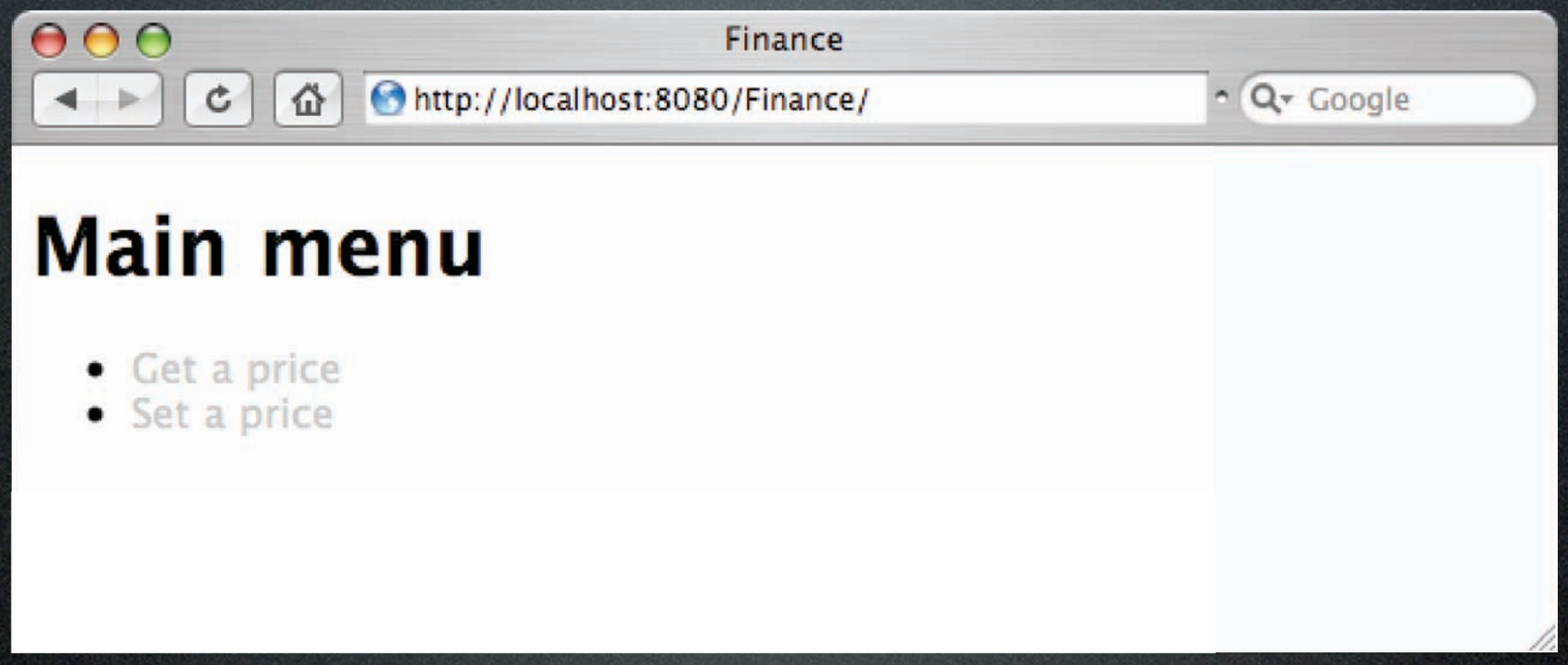
# Laboratoire : service



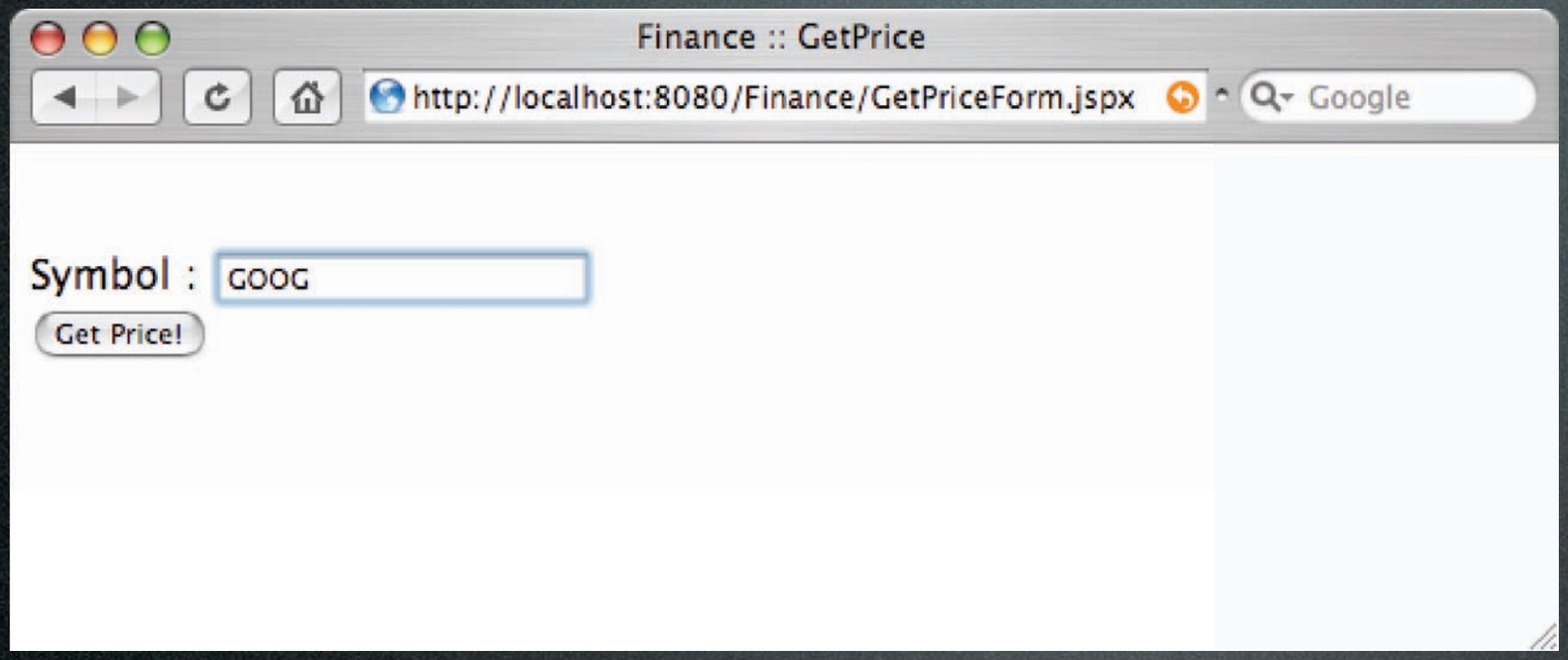
# Laboratoire : client

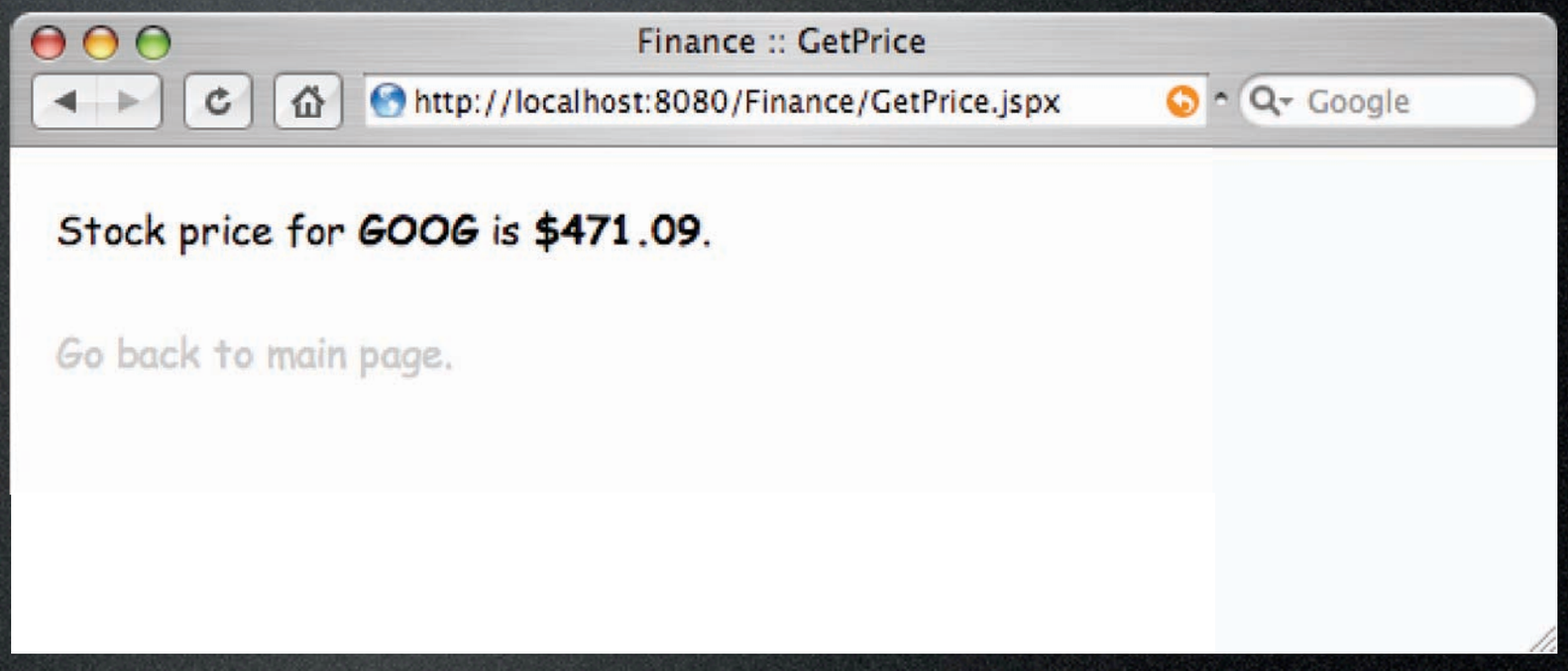
```
> java client/Main

Usage: java Main symbol
       java Main symbol price

> java client/Main AAPL 153.08

Stock price for AAPL set to 153.08

> java client/Main AAPL

Stock price for AAPL is 153.08

> java client/Main GOOG

Stock price for GOOG is -1.0

> java client/Main GOOG 471.09

Stock price for GOOG set to 471.09

> java client/Main GOOG

Stock price for GOOG is 471.09
```

# Laboratoire : client JSPX

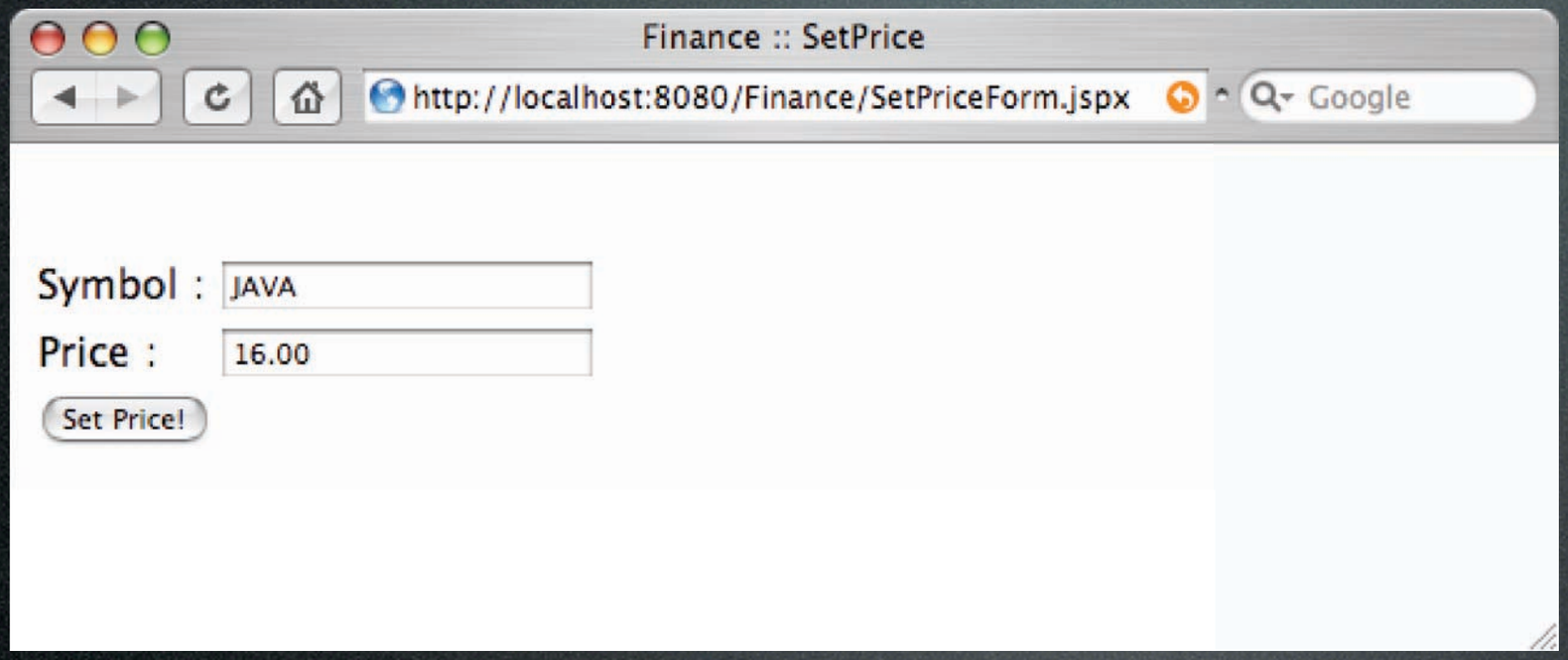


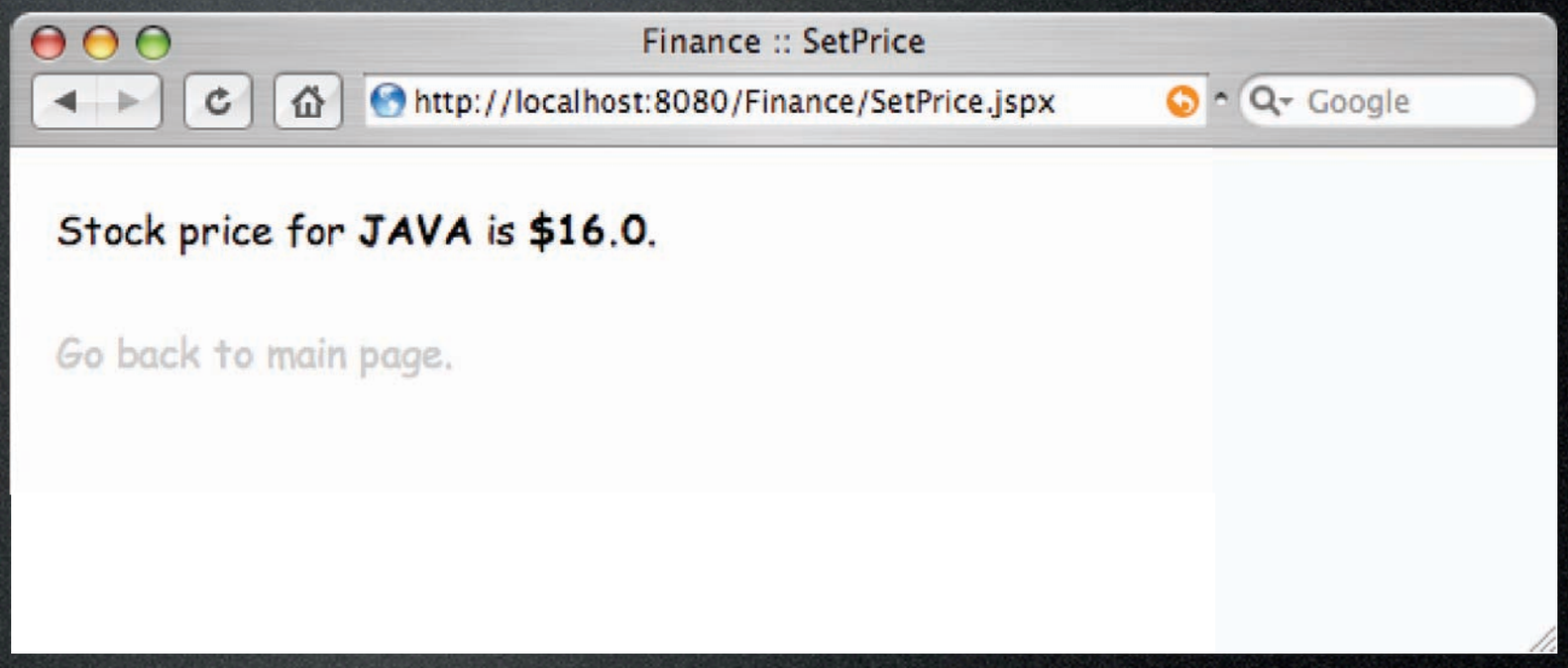
# Laboratoire : client JSPX



# Laboratoire : client JSPX



# Services Web

## Les détails

# Remarques

- Le **Service Web** peut être vu comme une abstraction dont le **contrat** est spécifié à l'aide du concept **d'interface** de Java
- Le dévelopement du service et des clients se fait sans avoir à implémenter les mécanismes pour la génération, le transport ou le décodage des messages (**XML**) **SOAP** liés aux appels de méthodes et valeurs de retour

# «Service Interface Definition»

- Côté serveur : l'interface sert à produire le document WSDL décrivant le service

```
package util;

public interface Motd {
    public String getMotd();
}
```

- Côté client : le document WSDL sert à (re-) produire l'interface décrivant le service

# «Service endpoint interface»

- Les types, des paramètres et valeurs de retours, ont certaines restrictions
- Aucunes constantes (static final)

# «Service Interface Definition»

- L'utilisation d'un document **WSDL**, afin de représenter le service, impose des contraintes sur les types des paramètres et valeurs de retour

# «Service endpoint interface»

- Les types, des paramètres et valeurs de retours, ont certaines restrictions
  - Types primitifs de Java (et wrappers)
  - java.lang.String, java.util.Date, java.util.Calendar, java.util.ArrayList, java.util.HashMap, entre autres
  - Classe ayant des attributs publiques de types primits ayant un constructeur sans paramètre

# «Service endpoint interface»

```
public class ExchangeValues {
    public double dollars;
    public double euros;
    public double yen;
}
```

# Protocoles

- Remote Procedure Call (JAX-**RPC**)
  - Protocole pour les **appels de méthodes** (procédures), généralement, à travers un réseau
- **SOAP** Version 1.2 (27 avril 2007)
  - Protocol XML pour l'**échange d'informations structurées** dans un environnement distribué

# Services Web

## Découverte de services

# Découverte de services

- «Most of the public **UDDI** registries have been **discontinued** in early 2006, however, due to the limited quality of the contained data, for searching public services they have never been a good source.»
- «Simple application of **IR** technologies and later the use of **ontologies** to describe standard vocabulary are the most promising approaches for the near future.»

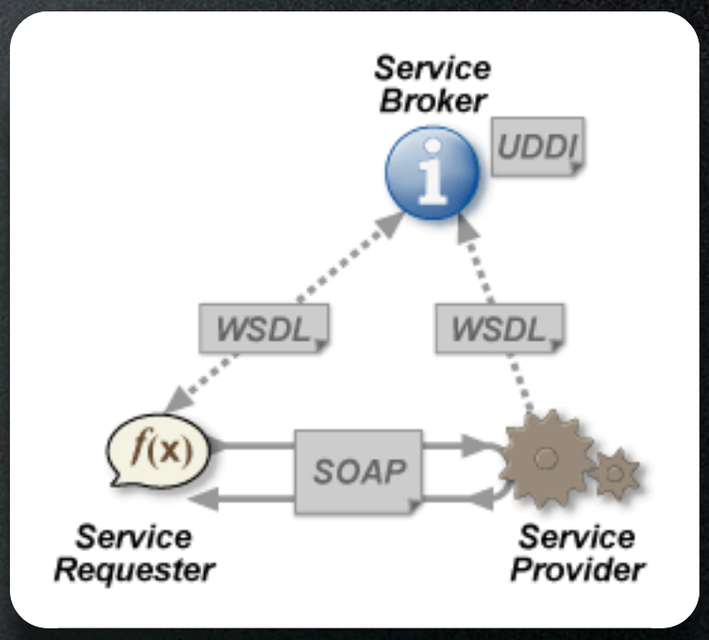
# Découverte de services

- [ http://www.deri.ie/fileadmin/documents/DERI-TR-2006-01-17.pdf ] 2008-04-06
- [ http://www.eswc2006.org/poster-papers/FP38-Bachlechner.pdf ] 2008-04-06

Toujours au stade de la recherche.

# Répertoires de services

- http://www.XMethods.com
- http://www.webserviceX.NET
- http://www.RemoteMethods.com
- http://www.WebServiceList.com
- http://www.StrikeIron.com
- ...
- http://www.google.com



# Remarques

# Alternatives au Java Web Services Developer Pack (Java WSDP)

- **Axis** [ http://ws.apache.org/axis ]
  - Axis 1.4 (22 avril 2006)
- **Metro** [ https://metro.dev.java.net ]
  - Metro 1.1 (19 décembre 2007)
  - Annotations : @WebService et @WebMethod
    - apt, wsimport, wsgen

# Résumé

- Peu de contraintes quand **1**) langage de programmation (SOAP), **2**) système d'exploitation, **3**) protocoles (http est largement supporté), **4**) ports (80 est ouvert par défaut pour bien des pare-feu)
- Enveloppe autour de systèmes existants
- Muli-langages de programmation
- Cryptage, authentification...

# Ressources

# Ressources

- Web Services Description Language (WSDL) Version 2.0 Part 0: Primer [ http://www.w3.org/TR/wsdl20-primer/ ] 2007
- Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language [ http://www.w3.org/TR/wsdl20/ ] 2007

# Ressources (suite)

- SOAP Version 1.2 : Primer [ http://www.w3.org/TR/soap12-part0 ] 2007
- SOAP Version 1.2 Part 1: Messaging Framework [ http://www.w3.org/TR/soap12-part1/ ] 2007

# Ressources (suite)

- Web Services @ W3C [ http://www.w3.org/2002/ws/ ] 2007
- Java Web Services At a Glance [ http://java.sun.com/webservices/ ] 2007
- Apache Web Services Project [ http://ws.apache.org/ ] 2007
- Web Services and Other Distributed Technologies [ http://msdn.microsoft.com/webservices ] 2007

# Ressources (suite)

- Java Web Services Developer Pack - Part 2: RPC Calls, Messaging, and the JAX-RPC and JAXM API [ http://java.sun.com/developer/technicalArticles/WebServices/WSPack2/ ] 2007
- The Java EE 5 Tutorial [ http://java.sun.com/javaee/5/docs/tutorial/doc/ ] 2007

# Ressources

- XML Schema Part 0: Primer Second Edition [ http://www.w3.org/TR/xmlschema-0 ] 2007
- XML: Looking at the Forest Instead of the Trees par Guy Lapalme [ http://www.iro.umontreal.ca/~lapalme/ForestInsteadOfTheTrees/ ] 2007

# Appendice : Motd

## Java Web Services Developer Pack v1.3 (Instructions Jackson 2007)

# Instructions désuètes

- Cette section présente les instructions pour compiler et exécuter les programmes liés à **Motd** à l'aide de **JWSDP 1.3** (Java Web Services Developer Pack v1.3)
- JAX-WS, maintenant intégré à **Java SE 6**, rend ces explications caduques

# Motd : côté serveur

- Il faut **décrire** le service à l'aide d'une **interface** :

```
package util;

public interface MotdInterface extends java.rmi.Remote {
    public String getMotd() throws java.rmi.RemoteException;
}
```

Définir un package;
extends java.rmi.Remote;
throws
java.rmi.RemoteException

# Motd : implémentation

```
package util;
public class Motd implements MotdInterface {
    private static String[] mots = {
        "L'homme sage apprend de ses erreurs, L'homme plus sage .... (Confusius)",
        "Le sage ne dit pas ce qu'il sait, le sot ne sait pas ce qu'il dit. (Sagesse turque)",
        "L'esprit est le contraire de l'argent, moins on en a, plus on est satisfait. (Voltaire)",
        "Cerveau. Appareil avec lequel nous pensons que nous pensons. (Ambrose Bierce)",
        "On mesure l'intelligence d'un individu ... (Emmanuel Kant)",
        "Tu ne peux pas tout enseigner à un homme ; .... (Galilée)"
    };
    public String getMotd() {
        return mots[ (int) ( Math.random() * mots.length ) ];
    }
}
```

**La programmation côté serveur est terminée!**

# Programme Test (optionnel – ne fait partie de l'exemple)

```
import util.*;
class Main {
    public static void main( String[] args ) {
        MotdInterface m = new Motd();
        System.out.println( m.getMotd() );
    }
}
```

```
> java Main
Le sage ne dit pas ce qu'il sait, le sot ne sait pas ce qu'il dit. (Sagesse turque)
```

# Faisons le point

- Nous avons conçu une application pour afficher une nouvelle citation pour chaque appel de la méthode **getMotd()**
- Une **interface** décrit cette application
- Nous avons créé et **validé** son implémentation
- Ce qui suit nous permettra d'ajouter les fonctionnalités nécessaires pour en faire un **Service Web**

# 1. wscompile -define

- L'outil **wscompile** produit la description des méthodes et des types de données (**WSDL**) ainsi qu'un model

```
<?xml version="1.0" encoding="UTF-8"?>
 <configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service name="Motd"
           targetNamespace="http://tempuri.org/wsdl"
           typeNamespace="http://tempuri.org/types"
           packageName="util">
    <interface name="util.MotdInterface" />
  </service>
 </configuration>
```

# 1. wscompile -define

- wscompile -define
  -d web/WEB-INF
  -classpath web/WEB-INF/classes
  -model web/WEB-INF/model.xml.gz
  config.xml

Sur une même ligne

- produit :
  - web/WEB-INF/Motd.wsdl
  - web/WEB-INF/model.xml.gz

```
<definitions name="Motd"
             targetNamespace="http://tempuri.org/wsdl"
             xmlns:tns="http://tempuri.org/wsdl"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types/>
  <message name="MotdInterface_getMotd"/>
  <message name="MotdInterface_getMotdResponse">
    <part name="result" type="xsd:string"/></message>

  <portType name="MotdInterface">
    <operation name="getMotd">
      <input message="tns:MotdInterface_getMotd"/>
      <output message="tns:MotdInterface_getMotdResponse"/></operation></portType>

  <binding name="MotdInterfaceBinding" type="tns:MotdInterface">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="getMotd">
      <soap:operation soapAction=""/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   use="encoded" namespace="http://tempuri.org/wsdl"/>
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   use="encoded" namespace="http://tempuri.org/wsdl"/>
      </output>
    </operation>
  </binding>

  <service name="Motd">
    <port name="MotdInterfacePort" binding="tns:MotdInterfaceBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/></port></service>

</definitions>
```

Fichier WSDL produit
(se prononce wiz'-dul)

# 2. WEB-INF/jaxrpc-ri.xml

- Le fichier **jaxrpc-ri.xml** ressemble au descripteur de déploiement **web.xml**
- Le fichier **web.xml** sert à définir les servlets, et à leur associer une URL
- De même, le fichier **jaxrpc-ri.xml** sert à définir des points de service (endpoints), et à leur associer une URL

# 2. WEB-INF/jaxrpc-ri.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices  version="1.0"
              xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
              targetNamespaceBase="http://tempuri.org/wsdl"
              typeNamespaceBase="http://tempuri.org/types"
              urlPatternBase="/motd">

  <endpoint
    name="Motd"
    displayName="Serveur motd"
    description="Retourne le mot du jour."
    interface="util.MotdInterface"
    model="/WEB-INF/model.xml.gz"
    implementation="util.Motd"/>

  <endpointMapping
    endpointName="Motd"
    urlPattern="/get" />

</webServices>
```

# 3. motd-temp.war

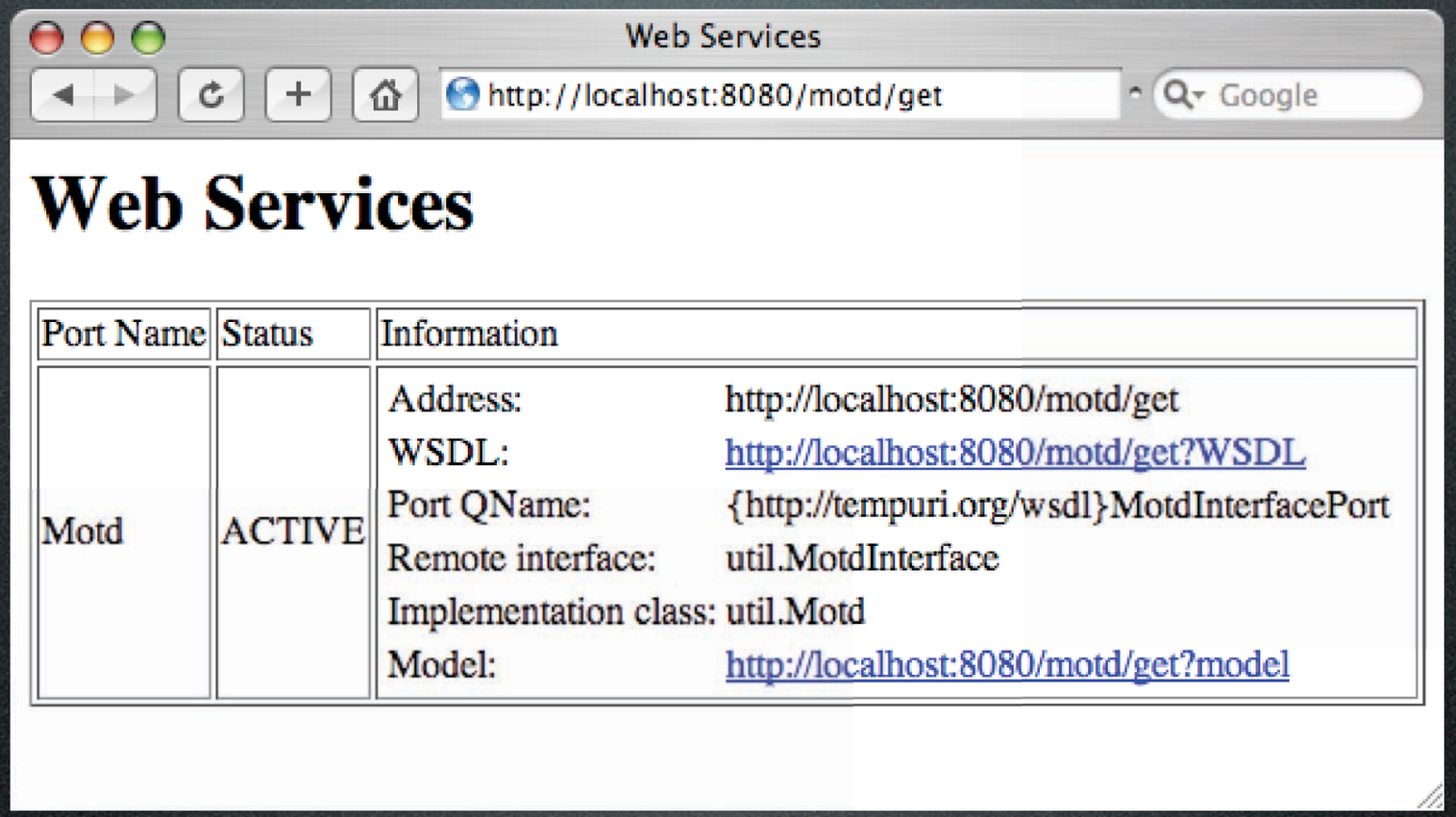
- jar cvf motd-temp.war WEB-INF
- Contenu de l'archive :
  - WEB-INF/classes/util/Motd.class
  - WEB-INF/classes/util/MotdInterface.class
  - WEB-INF/jaxrpc-ri.xml
  - WEB-INF/model.xml.gz
  - WEB-INF/Motd.wsdl
  - WEB-INF/web.xml

# 4. motd.war

- **wsdeploy -o motd.war motd-temp.war**
- contenu de l'archive :

```
WEB-INF/classes/util/Motd.class
WEB-INF/classes/util/Motd_SerializerRegistry.class
WEB-INF/classes/util/MotdInterface.class
WEB-INF/classes/util/MotdInterface_getMotd_RequestStruct.class
WEB-INF/classes/util/MotdInterface_getMotd_RequestStruct__Motd__SOAPSerializer.class
WEB-INF/classes/util/MotdInterface_getMotd_ResponseStruct.class
WEB-INF/classes/util/MotdInterface_getMotd_ResponseStruct__Motd__SOAPBuilder.class
WEB-INF/classes/util/MotdInterface_getMotd_ResponseStruct__Motd__SOAPSerializer.class
WEB-INF/classes/util/MotdInterface_Tie.class
WEB-INF/jaxrpc-ri-before.xml
WEB-INF/jaxrpc-ri-runtime.xml
WEB-INF/model.xml.gz
WEB-INF/Motd.wsdl
WEB-INF/web-before.xml
WEB-INF/web.xml
```

# 5. déployer le .war




Voilà! Le service est maintenant en fonction!

Appels par valeurs!

# Motd : 1. côté client

- Il faut créer un **fichier de configuration** :
  - indicant l'**URL** du **WSDL**
  - le nom du package

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="http://localhost:8080/motd/get?WSDL"
        packageName="client" />
</configuration>
```

# 2. wscompile

- mkdir classes src
- wscompile -gen -keep -d classes -s src config.xml
- Ce qui produit ceci (+classes/src/client/*.class) :

src/client/Motd.java
src/client/MotdInterface.java
src/client/MotdInterface_Stub.java
src/client/MotdInterface_getMotd_RequestStruct.java
src/client/MotdInterface_getMotd_RequestStruct_SOAPSerializer.java
src/client/MotdInterface_getMotd_ResponseStruct.java
src/client/MotdInterface_getMotd_ResponseStruct_SOAPBuilder.java
src/client/MotdInterface_getMotd_ResponseStruct_SOAPSerializer.java
src/client/Motd_Impl.java
src/client/Motd_SerializerRegistry.java

# 3. MotdBean

```
package client;

public class MotdBean {
    public String getMotd() {
      String result = null;
      MotdInterface motd = ( new Motd_Impl() ).getMotdInterfacePort();
      try {
        result = motd.getMotd();
      } catch ( java.rmi.RemoteException e ) {
        e.printStackTrace();
      }
      return result;
    }
}
```

«static stub», ci-haut, les objets mandataires sont créés lors du développement (wscompile);

La génération dynamique es aussi posssible.

# 4. Application (ligne de commande)

```
import client.MotdBean;

class Main {

    public static void main( String[] args ) {

        MotdBean m = new MotdBean();

        System.out.println( m.getMotd() );

    }
}
```

```
> java Main
Le sage ne dit pas ce qu'il sait, le sot ne sait pas ce qu'il dit. (Sagesse turque)
```

# 4. Application (ligne de commande)

```
> java -cp $GLASSFISH/lib/appserv-ws.jar:. Main
```

Le sage ne dit pas ce qu'il sait, le sot ne sait pas ce qu'il dit. (Sagesse turque)

# 4. Application (Web)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:core="http://java.sun.com/jsp/jstl/core"
      lang="fr-CA">

 <jsp:useBean scope="application" id="m" class="client.MotdBean"/>

 <head>
  <title>CSI3540. Structures, techniques et normes du Web</title>
 </head>

 <body>
  <p>
   <b>${ m.motd }</b>
  </p>
 </body>
</html>
```