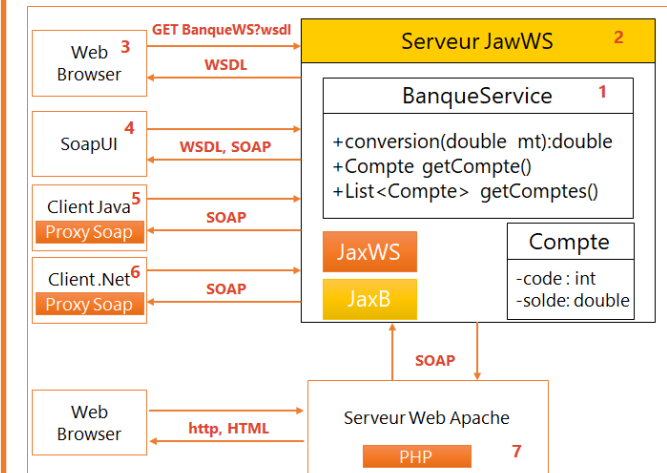
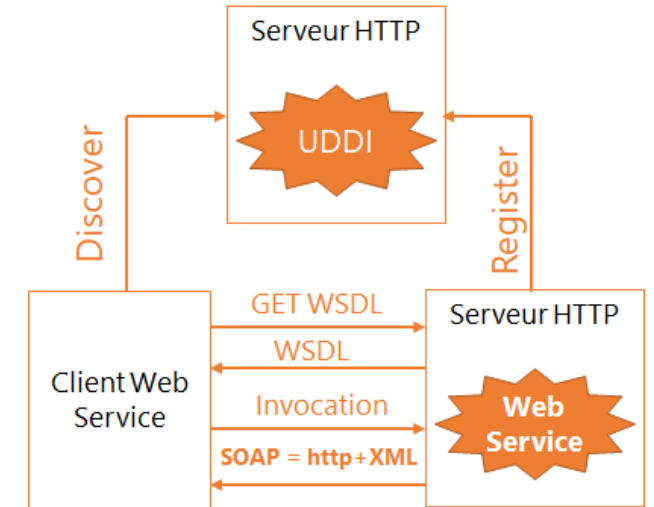
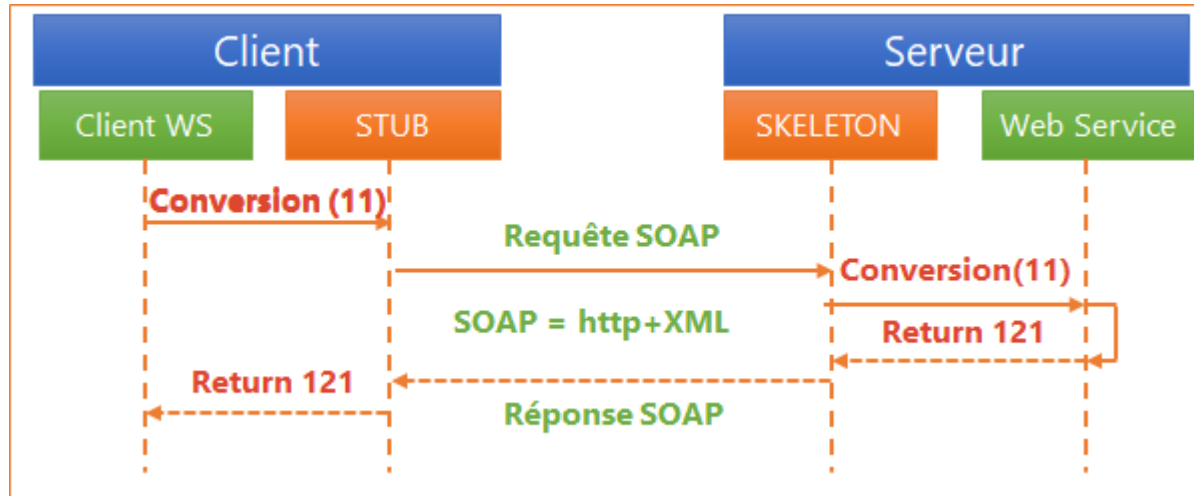


Systemes Distribués : Web Services : SOAP, WSDL, UDDI



ENSET



Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

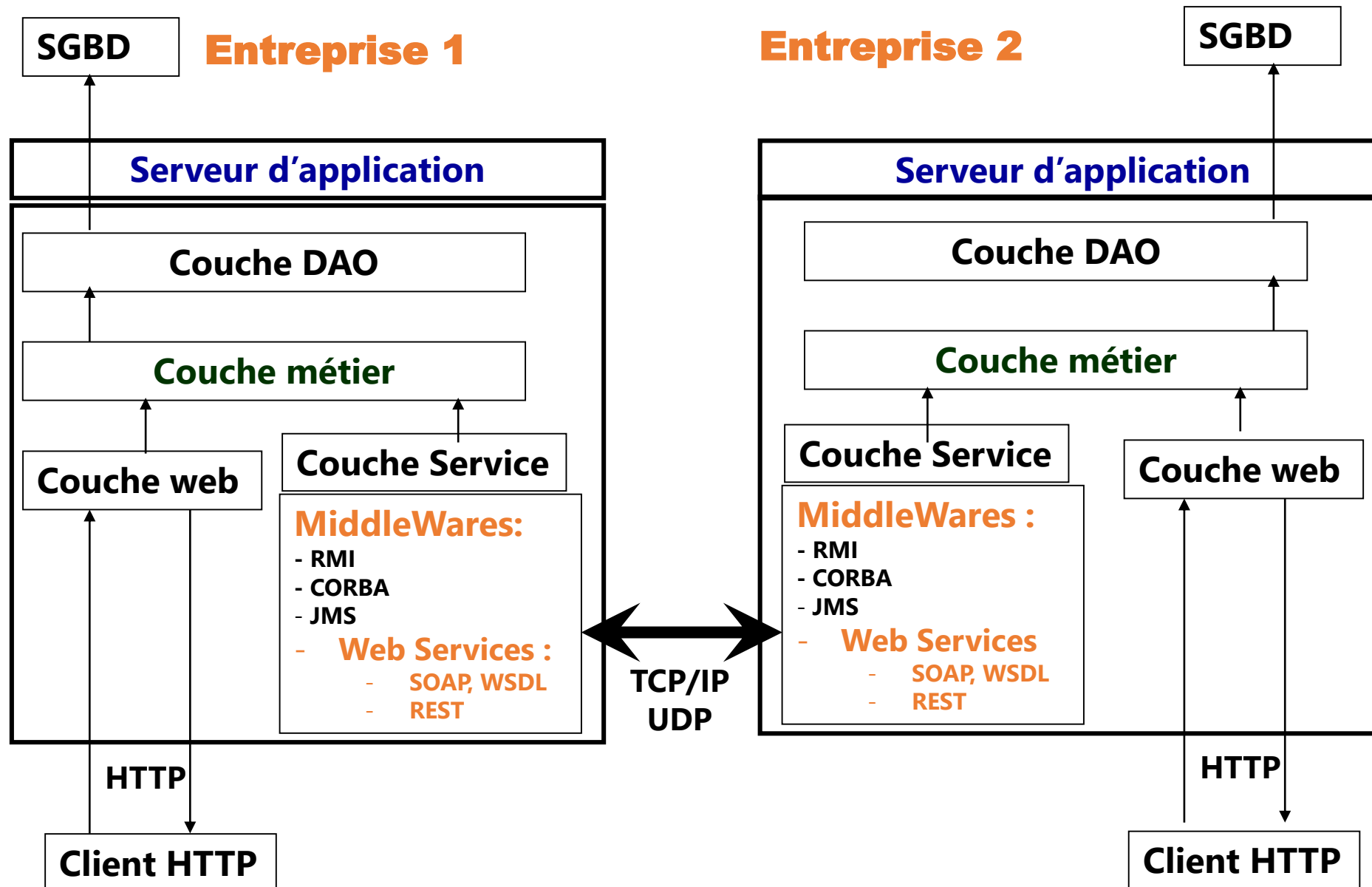
Email : med@youssfi.net

Supports de cours : <http://fr.slideshare.net/mohamedyoussfi9>

Chaîne vidéo : <http://youtube.com/mohamedYoussfi>

Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Architectures Distribuées

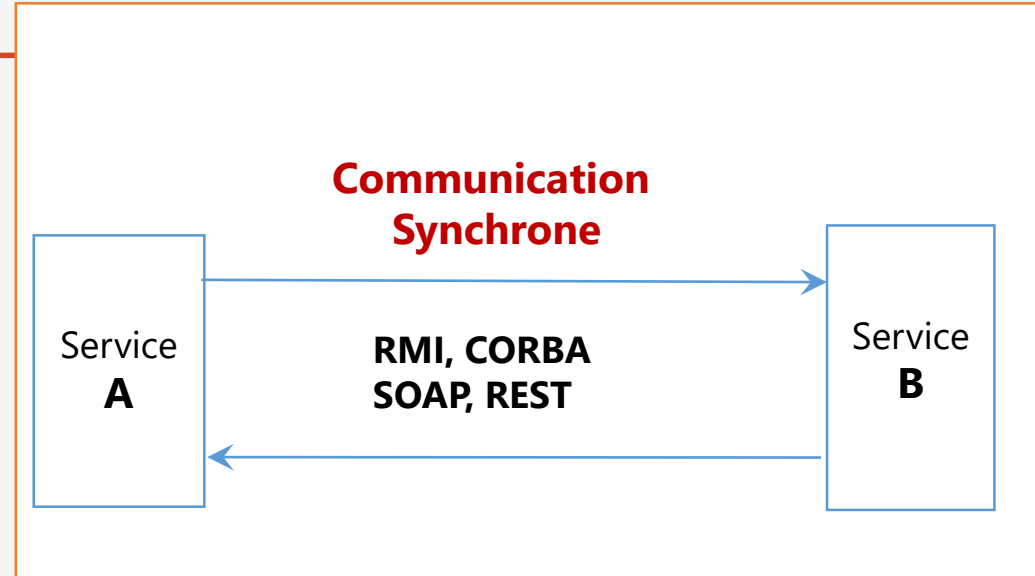


Middlewares Synchrones et Asynchrones

Communication Synchrone :

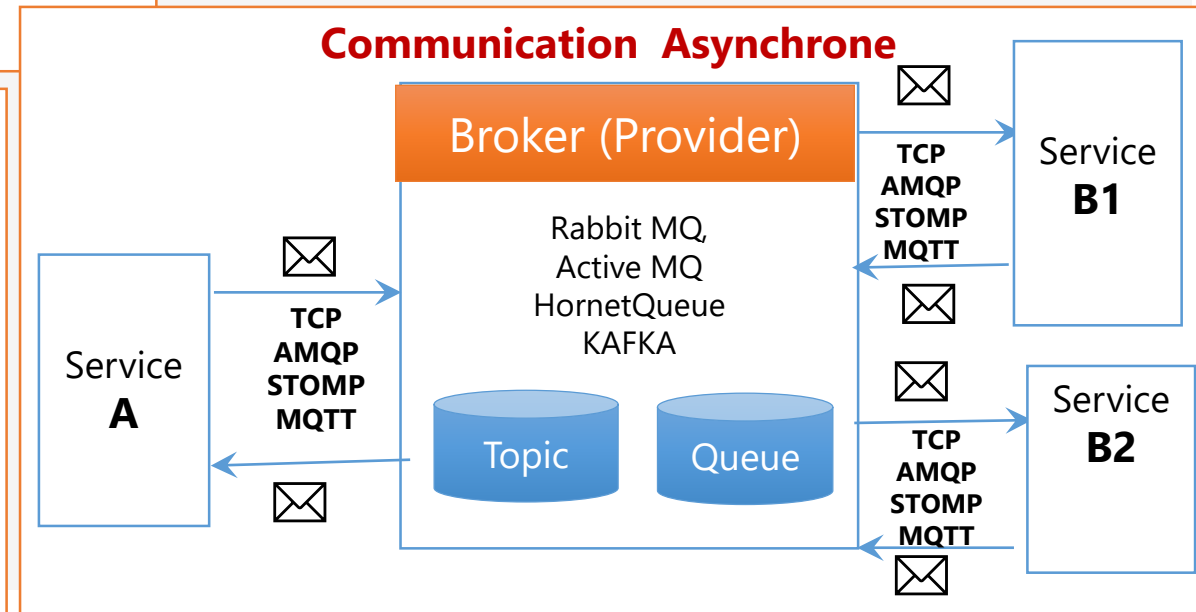
Dans une communication synchrone :

- Un service A dans une machine M1 établie une connexion avec un service B dans une autre machine M2.
- Si le service B n'est pas disponible, il n'y aura pas de communication
- Si oui, Le service A envoie ensuite une requête pour faire appel à une opération à distance de le service B.
- le service A est bloqué jusqu'à ce qu'il reçoit la réponse de le service B.
- Modèle de communication synchrone :
 - RMI : Remote Method Invocation
 - CORBA : Common Request Broker Architecture
 - HTTP : Hyper Text Transfer Protocol
 - Web services étendus : SOAP Over HTTP : Simple Object Access Protocol
 - Web Services RESTful) : REST Over HTTP : Representational Stat Transfer



Communication Asynchrone :

- Un service A établie une connexion vers un provider (Broker).
- Le service A envoie le message à une file d'attente du Broker pour le délivrer au service B
- Si le service B n'est pas disponible, cela n'empêche pas le service A d'envoyer le message. Dans ce cas là message est conservé dans la file d'attente
- Si B est connecté à la file d'attente du broker, ce dernier lui délivre le message.
 - Modèles de communications :
 - JMS, AMQP, STOMP, MQTT
 - RabbitMQ, ActiveMQ, KAFKA



Introduction aux web services

Les Web Services sont des composants web basés sur Internet (**HTTP**) qui exécutent des tâches précises et qui respectent un format spécifique (**XML**).

Ils permettent aux applications de faire appel à des fonctionnalités à distance en simplifiant ainsi l'échange de données.

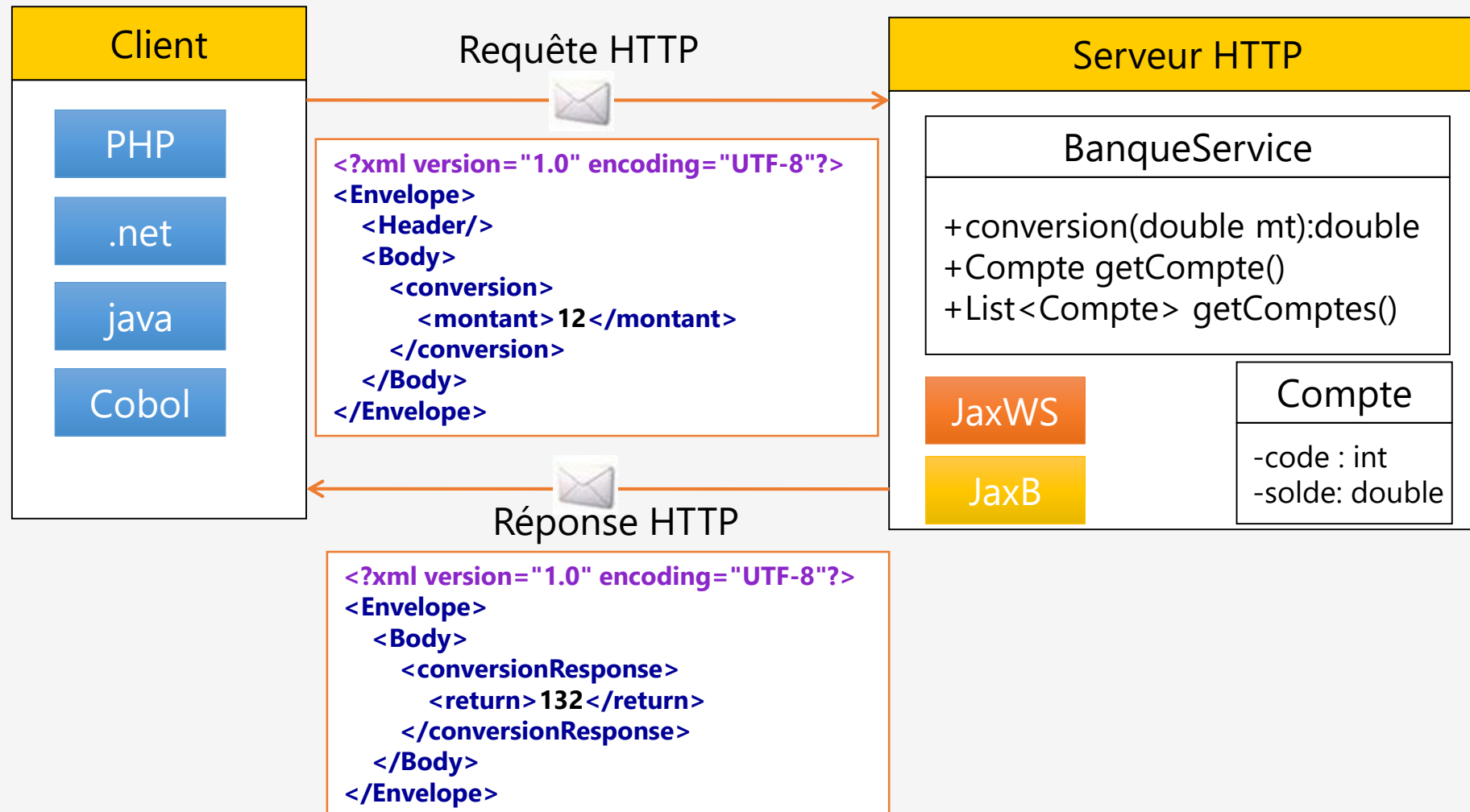
Les Web Services permettent aux applications de dialoguer à travers le réseau, indépendamment de

- leur plate-forme d'exécution
- et de leur langage d'implémentation.

Ils s'inscrivent dans la continuité d'initiatives telles que

- CORBA (*Common Object Request Broker Architecture*, de l'OMG) en apportant toutefois une réponse plus simple, s'appuyant sur des technologies et standards reconnus et maintenant acceptés de tous.

Architecture de base des Web Services



Requête SOAP avec POST

Entête de la requête

Post /WebServicePath HTTP/1.1

accept: application/xml

Content-Type : application/xml

***** saut de ligne *****

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">  
  <SOAP-ENV:Header/>  
  <SOAP-ENV:Body>  
    <conversion xmlns="http://bk/test">  
      <montant xmlns="">12</montant>  
    </conversion>  
  </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

corps de la requête HTTP

Réponse SOAP :

Entête de la réponse

HTTP/1.0 200 OK

Date : Wed, 05Feb02 15:02:01 GMT

Server : Apache/1.3.24

Mime-Version 1.0

Last-Modified : Wed 02Oct01 24:05:01GMT

Content-Type : application/xml

Content-length : 4205

***** saut de ligne *****

```
<?xml version="1.0" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns1="http://bk/test">
  <soapenv:Body>
    <ns1:conversionResponse>
      <return>132.0</return>
    </ns1:conversionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

corps de la réponse

Concepts fondamentaux des web services

Le concept des Web Services s'articule actuellement autour des trois concepts suivants :

- **SOAP (Simple Object Access Protocol)**

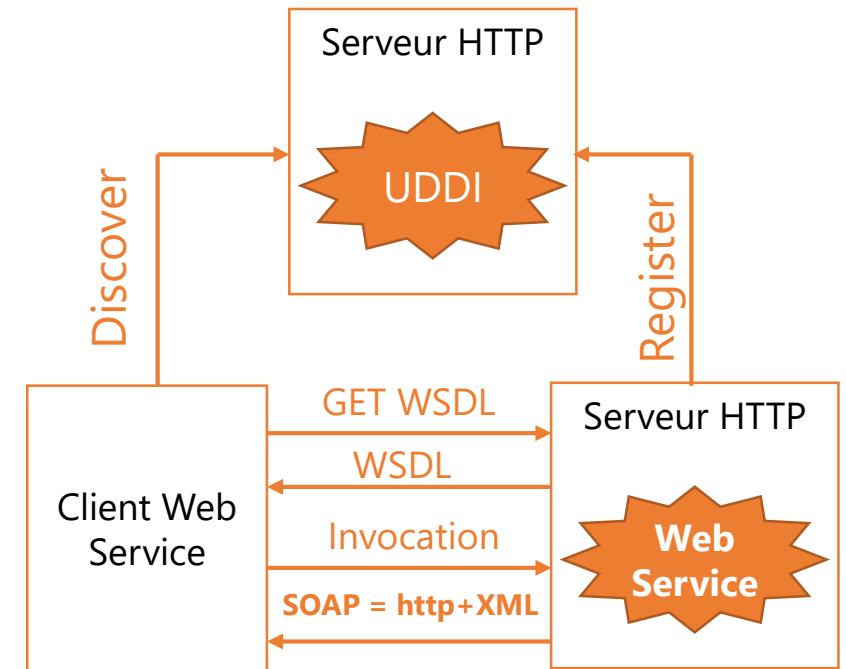
- est un protocole d'échange inter-applications indépendant de toute plate-forme, basé sur le langage XML.
- Un appel de service SOAP est un flux ASCII encadré dans des balises XML et transporté dans le protocole HTTP.

- **WSDL (Web Services Description Language)**

- donne la description au format XML des Web Services en précisant les méthodes pouvant être invoquées, leurs signatures et le point d'accès (URL, port, etc..).
- C'est, en quelque sorte, l'équivalent du langage IDL pour la programmation distribuée CORBA.

- **UDDI (Universal Description, Discovery and Integration)**

- normalise une solution d'annuaire distribué de Web Services, permettant à la fois la publication et l'exploration (recherche) de Web Services.
- UDDI se comporte lui-même comme un Web service dont les méthodes sont appelées via le protocole SOAP.

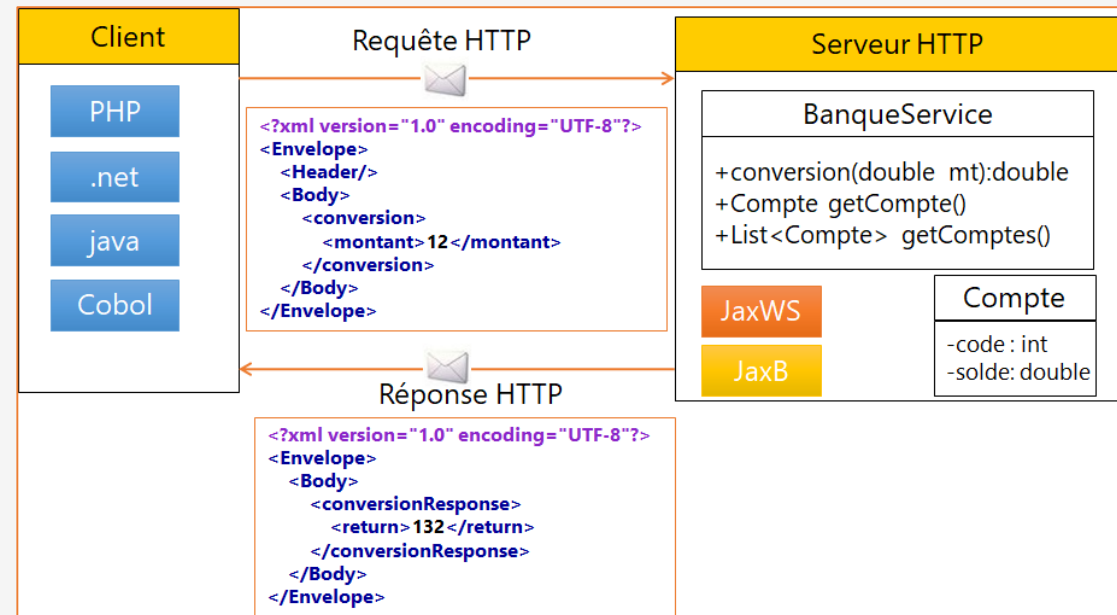


Mise en œuvre des web services avec JAX-WS

JAX-WS

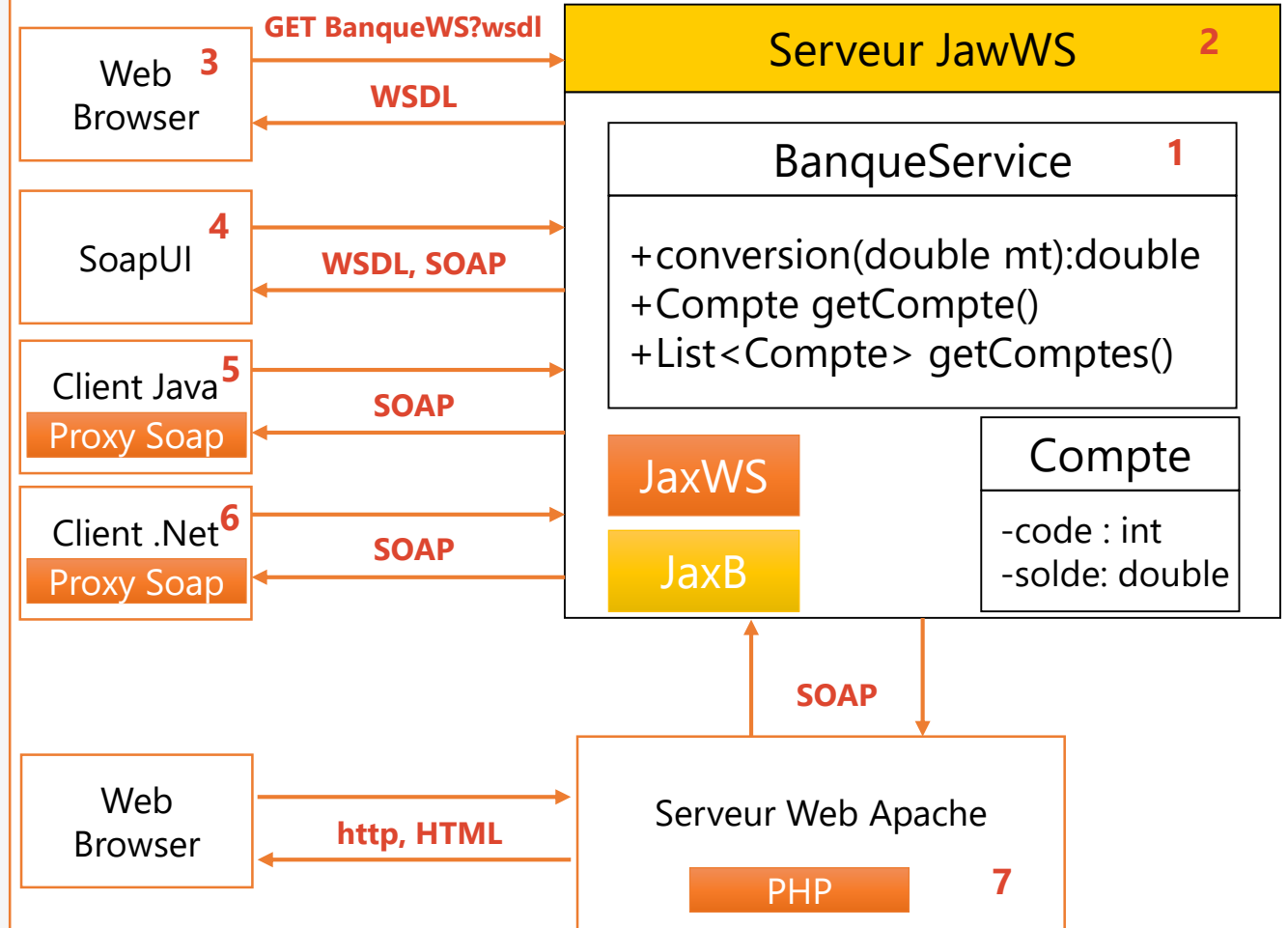
- JAX-WS est la nouvelle appellation de JAX-RPC (Java API for XML Based RPC) qui permet de développer très simplement des services web en Java.
- JAX-WS fournit un ensemble d'annotations pour mapper la correspondance Java-WSDL. Il suffit pour cela d'annoter directement les classes Java qui vont représenter le service web.
- Dans l'exemple ci-dessous, une classe Java utilise des annotations JAX-WS qui vont permettre par la suite de générer le document WSDL. Le document WSDL est auto-généré par le serveur d'application au moment du déploiement :

```
@WebService(serviceName="BanqueWS")
public class BanqueService {
    @WebMethod(operationName="ConversionEuroToDh")
    public double conversion(@WebParam(name="montant")double mt){
        return mt*11;
    }
    @WebMethod
    public String test(){ return "Test"; }
    @WebMethod
    public Compte getCompte(){ return new Compte (1,7000); }
    @WebMethod
    public List<Compte> getComptes(){
        List<Compte> cptes=new ArrayList<Compte>();
        cptes.add (new Compte (1,7000)); return cptes;
    }
}
```

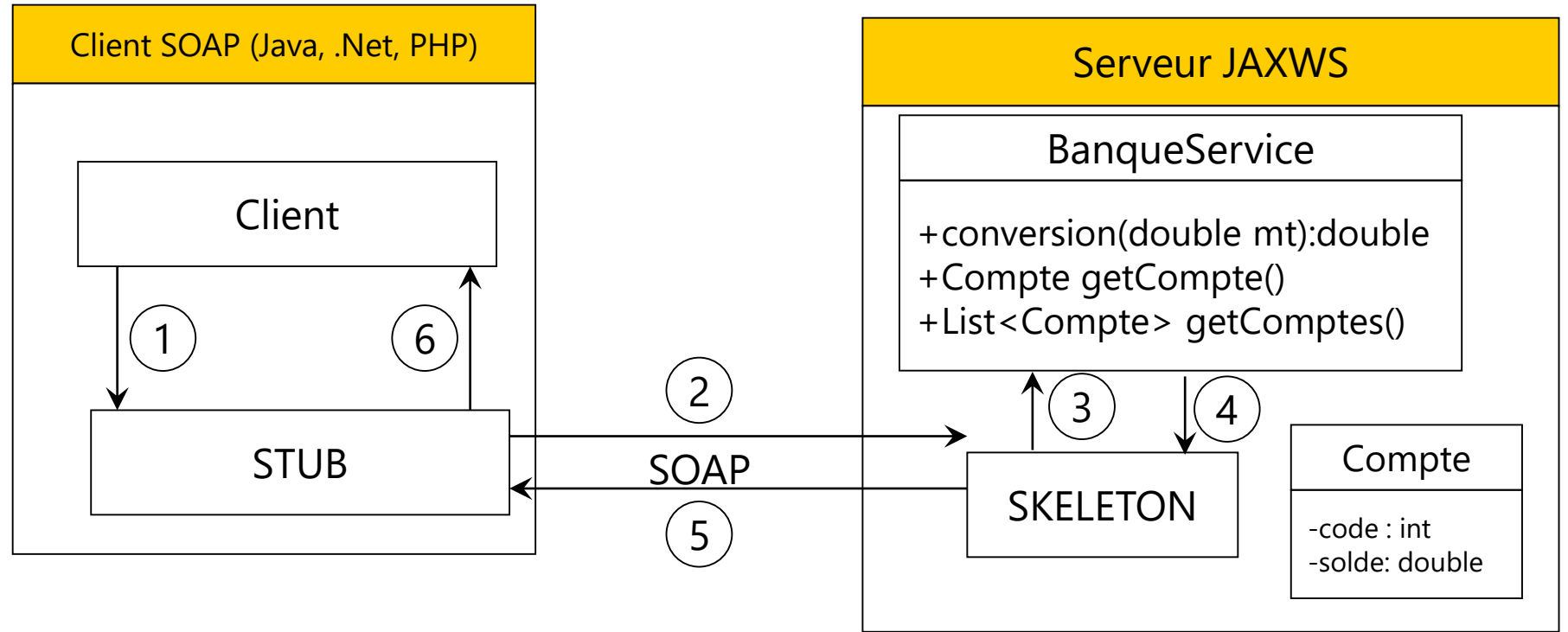


Application

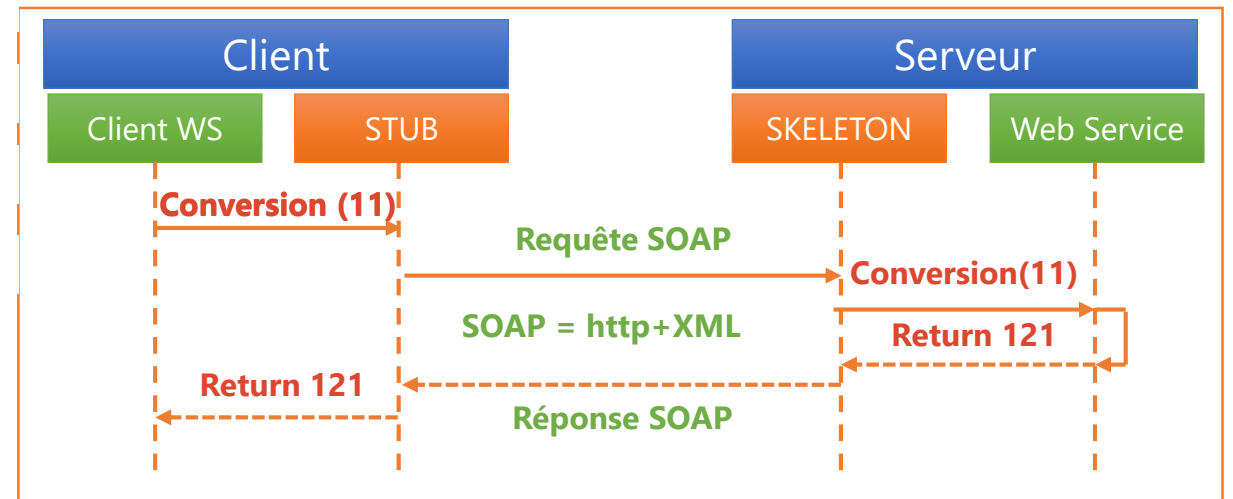
1. Créer un Web service qui permet de :
 - Convertir un montant de l'auro en DH
 - Consulter un Compte
 - Consulter une Liste de comptes
2. Déployer le Web service avec un simple Serveur JaxWS
3. Consulter et analyser le WSDL avec un Browser HTTP
4. Tester les opérations du web service avec un outil comme SoapUI ou Oxygen
5. Créer un Client SOAP Java
6. Créer un Client SOAP Dot Net
7. Créer un Client SOAP PHP
8. Déployer le Web Service dans un Projet Spring Boot



Architecture 1



- 1 Le client demande au stub de faire appel à la méthode conversion(12)
- 2 Le Stub se connecte au Skeleton et lui envoie une requête SOAP
- 3 Le Skeleton fait appel à la méthode du web service
- 4 Le web service retourne le résultat au Skeleton
- 5 Le Skeleton envoie le résultat dans une la réponse SOAP au Stub
- 6 Le Stub fournit le résultat au client



Implémentation du Web Service JaxWS

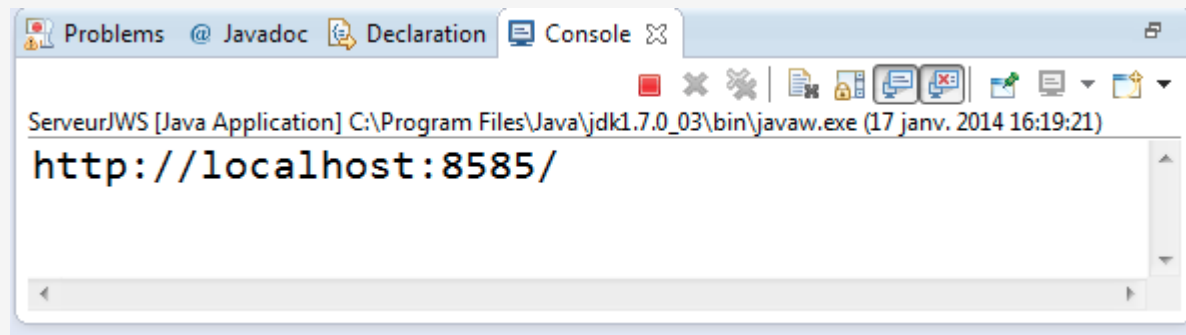
```
package ws;
import java.util.*;
import javax.jws.*;
import metier.Compte;
@WebService(serviceName="BanqueWS")
public class BanqueService {
    @WebMethod(operationName="ConversionEuroToDh")
    public double conversion(@WebParam(name="montant")double mt){
        return mt*11;
    }
    @WebMethod
    public Compte getCompte(@WebParam(name="code")Long code){
        return new Compte (code,7000,new Date());
    }
    @WebMethod
    public List<Compte> getComptes(){
        List<Compte> cptes=new ArrayList<Compte>();
        cptes.add (new Compte (1L,7000,new Date()));
        cptes.add (new Compte (2L,7000,new Date()));
        return cptes;
    }
}
```

```
package metier;
import java.util.Date;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Compte {
    private Long code;
    private double solde;
    @XmlTransient
    private Date dateCreation;
    // Constructeurs
    // Getters et setters
}
```

Simple Serveur JAX WS

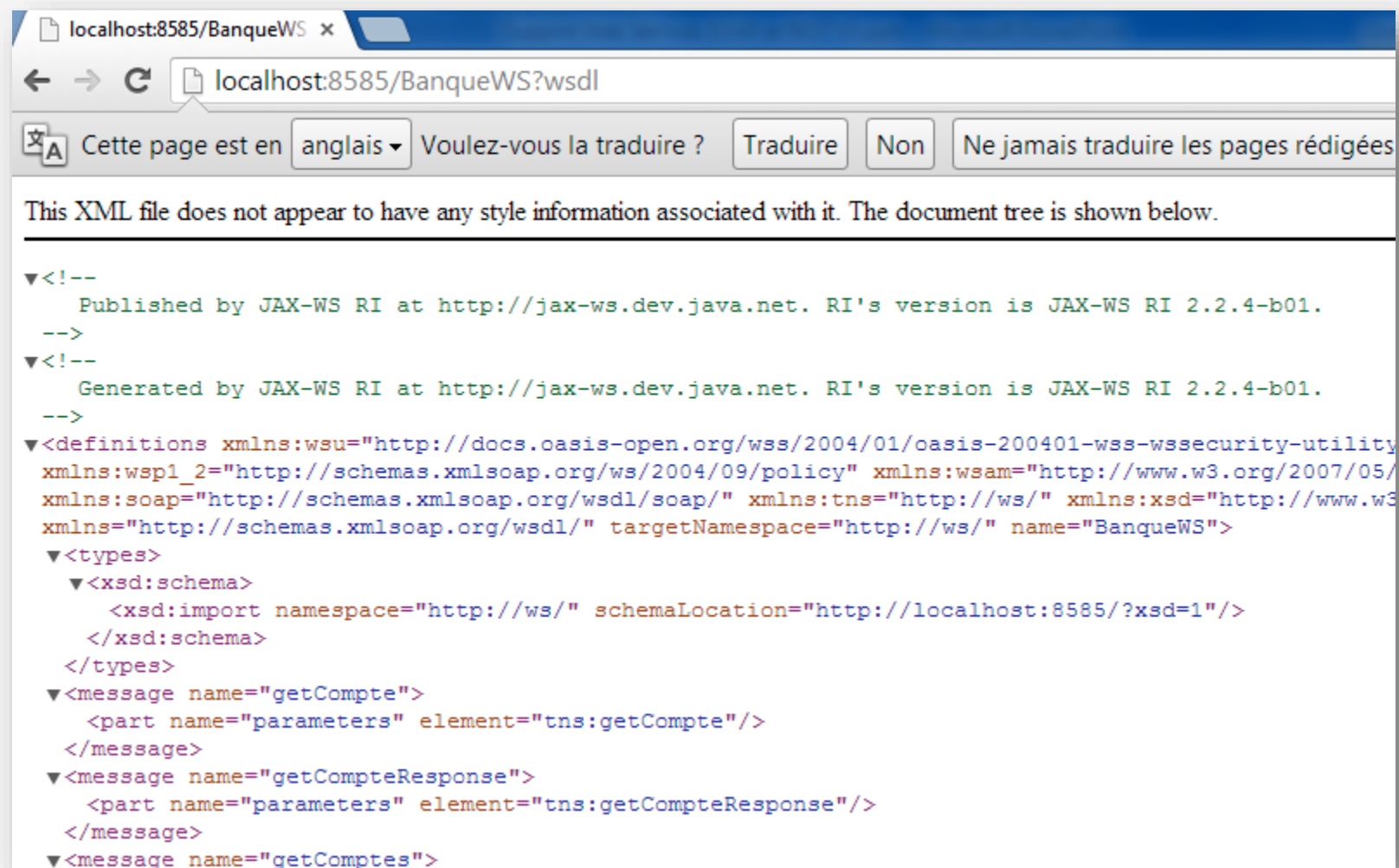
```
import javax.xml.ws.Endpoint;
import ws.BanqueService;
public class ServeurJWS {
    public static void main(String[] args) {
        String url="http://localhost:8585/";
        Endpoint.publish(url, new BanqueService());
        System.out.println(url);
    }
}
```

Exécution du serveur Java



Analyser le WSDL

Pour Visualiser le WSDL, vous pouvez utiliser un navigateur web



Déployer un Web service dans un projet Spring Boot

Web
Service

@Component

@WebService

```
public class BanqueService {  
    // Opérations du web service  
}
```

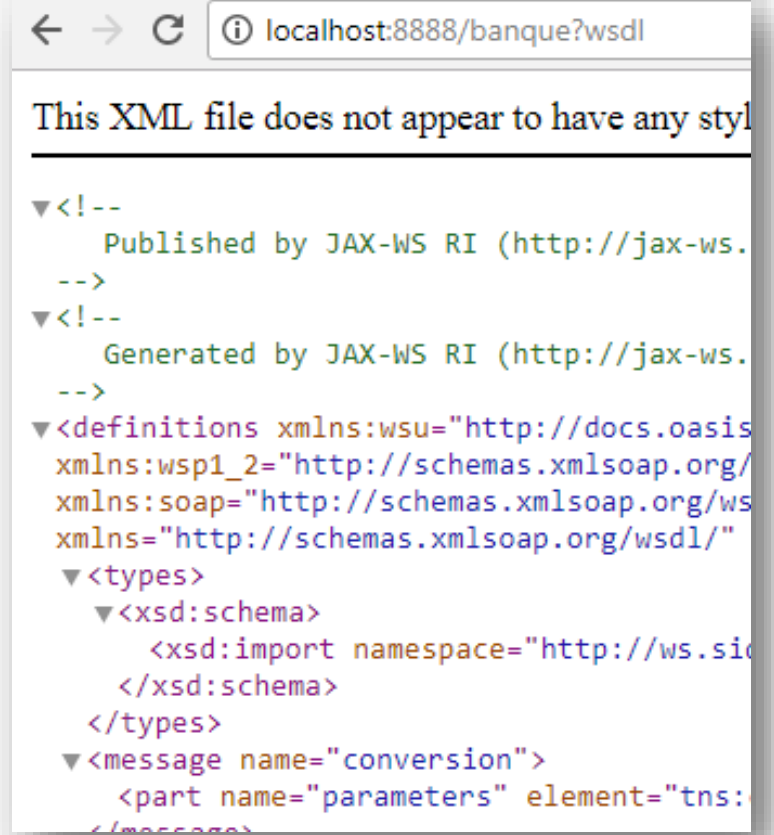
Classe de configuration

@Configuration

```
public class MyConfig {
```

@Bean

```
    public SimpleJaxWsServiceExporter getJWS() {  
        SimpleJaxWsServiceExporter exporter=new SimpleJaxWsServiceExporter();  
        exporter.setBaseAddress("http://0.0.0.0:8888/banque");  
        return exporter;  
    }  
}
```



← → ↻ ⓘ localhost:8888/banque?wsdl

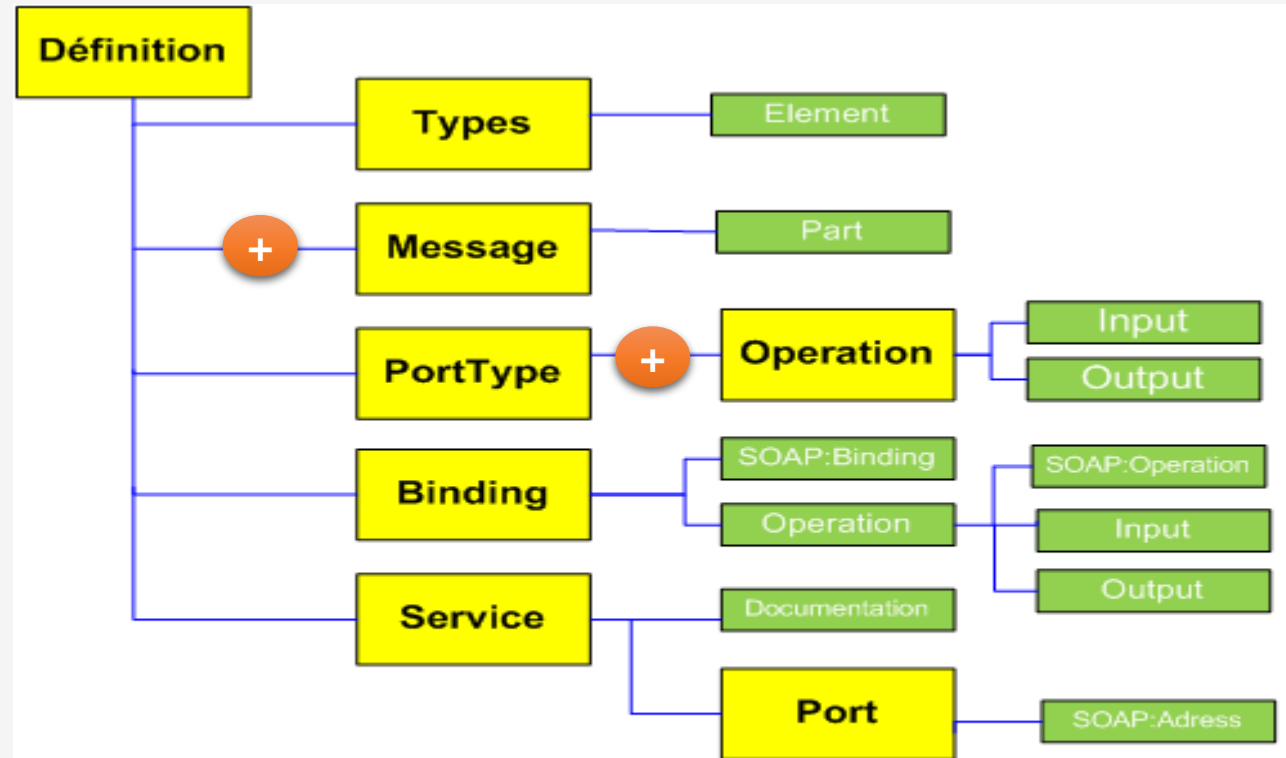
This XML file does not appear to have any style sheet associated with it.

```
▼<!--  
    Published by JAX-WS RI (http://jax-ws.  
    -->  
▼<!--  
    Generated by JAX-WS RI (http://jax-ws.  
    -->  
▼<definitions xmlns:wsu="http://docs.oasis  
    xmlns:wsp1_2="http://schemas.xmlsoap.org/  
    xmlns:soap="http://schemas.xmlsoap.org/ws  
    xmlns="http://schemas.xmlsoap.org/wsdl/"  
    ▼<types>  
        ▼<xsd:schema>  
            <xsd:import namespace="http://ws.sic  
            </xsd:schema>  
        </types>  
    ▼<message name="conversion">  
        <part name="parameters" element="tns:  
    </message>
```


Structure du WSDL

Un document WSDL se compose d'un ensemble d'éléments décrivant les types de données utilisés par le service, les messages que le service peut recevoir, ainsi que les liaisons SOAP associées à chaque message.

Le schéma suivant illustre la structure du langage WSDL qui est un document XML, en décrivant les relations entre les sections constituant un document WSDL.



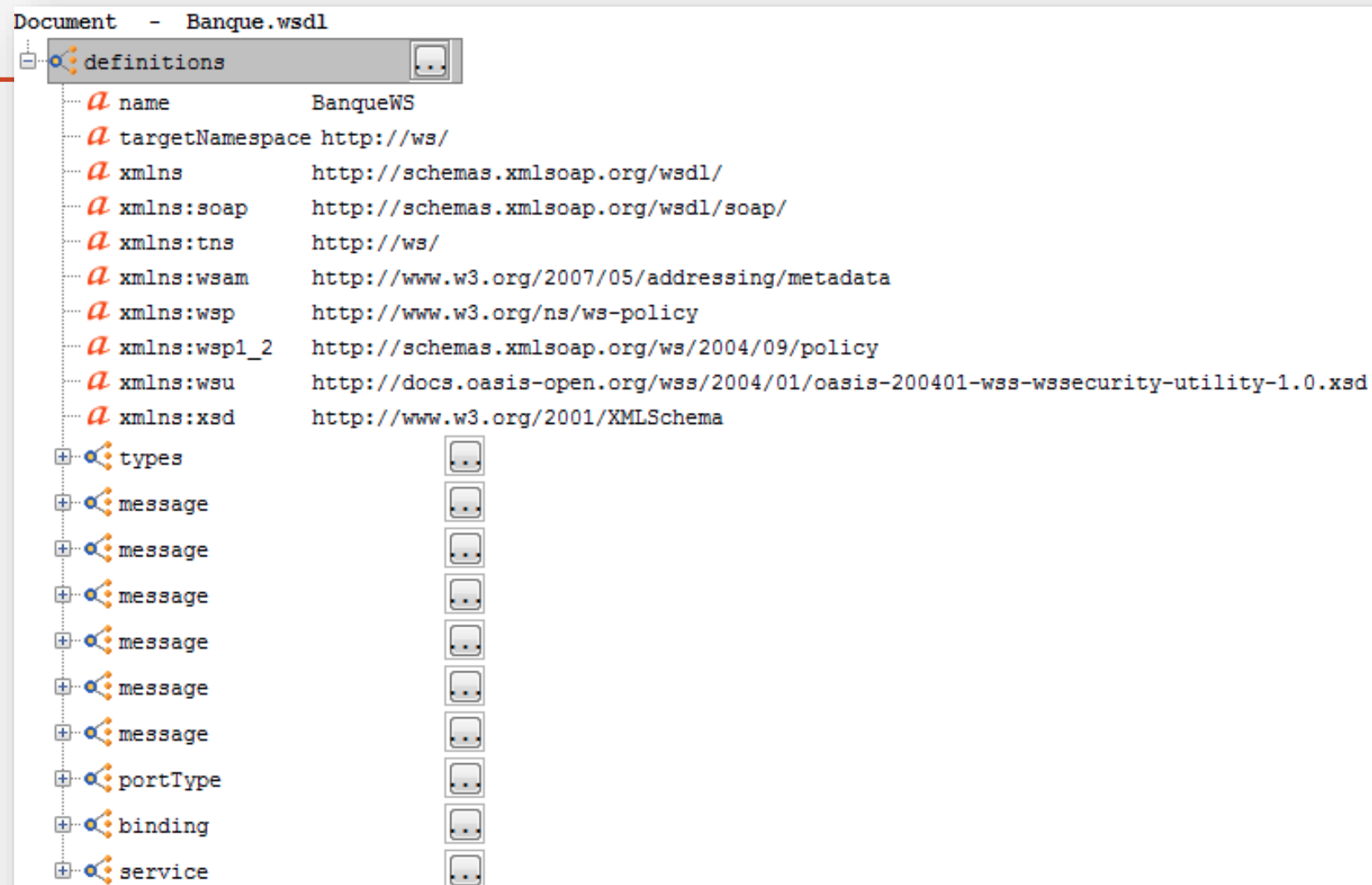
Structure d'un document WSDL

Structure du WSDL

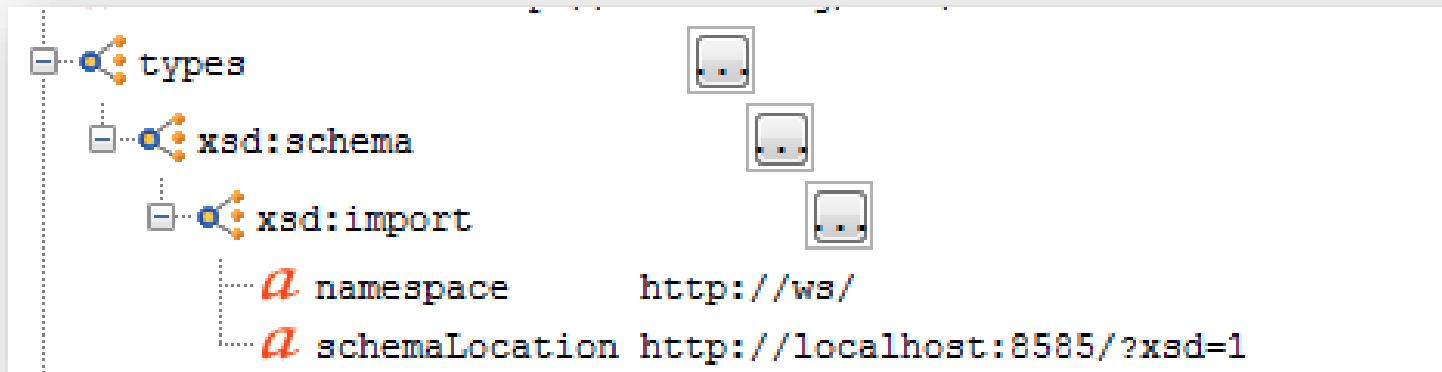
Un fichier WSDL contient donc sept éléments.

- **Types** : fournit la définition de types de données utilisés pour décrire les messages échangés.
- **Messages** : représente une définition abstraite (noms et types) des données en cours de transmission.
- **PortTypes** : décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou d'erreurs.
- **Binding** : spécifie une liaison entre un <portType> et un protocole concret (SOAP, HTTP...).
- **Service** : indique les adresses de port de chaque liaison.
- **Port** : représente un point d'accès de services défini par une adresse réseau et une liaison.
- **Opération** : c'est la description d'une action exposée dans le port.

Structure du WSDL



Elément Types



XML Schema

← → ↻ 📄 localhost:8585/?xsd=1

📄 Cette page est en anglais ▼ Voulez-vous la traduire ? Traduire Non Ne jamais traduire les pages rédigées en anglais

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<xs:schema xmlns:tns="http://ws/" xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0" targetNamespace="http://ws/">
  <xs:element name="ConversionEuroToDh" type="tns:ConversionEuroToDh"/>
  <xs:element name="ConversionEuroToDhResponse" type="tns:ConversionEuroToDhResponse"/>
  <xs:element name="compte" type="tns:compte"/>
  <xs:element name="getCompte" type="tns:getCompte"/>
  <xs:element name="getCompteResponse" type="tns:getCompteResponse"/>
  <xs:element name="getComptes" type="tns:getComptes"/>
  <xs:element name="getComptesResponse" type="tns:getComptesResponse"/>
  <xs:complexType name="ConversionEuroToDh">
    <xs:sequence>
      <xs:element name="montant" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ConversionEuroToDhResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="getCompte">
    <xs:sequence>
      <xs:element name="code" type="xs:long" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="getCompteResponse">
    <xs:sequence>
      <xs:element name="return" type="tns:compte" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="compte">
    <xs:sequence>
      <xs:element name="code" type="xs:long" minOccurs="0"/>
      <xs:element name="solde" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="getComptes">
    <xs:sequence/>
  </xs:complexType>
  <xs:complexType name="getComptesResponse">
    <xs:sequence>
      <xs:element name="return" type="tns:compte" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

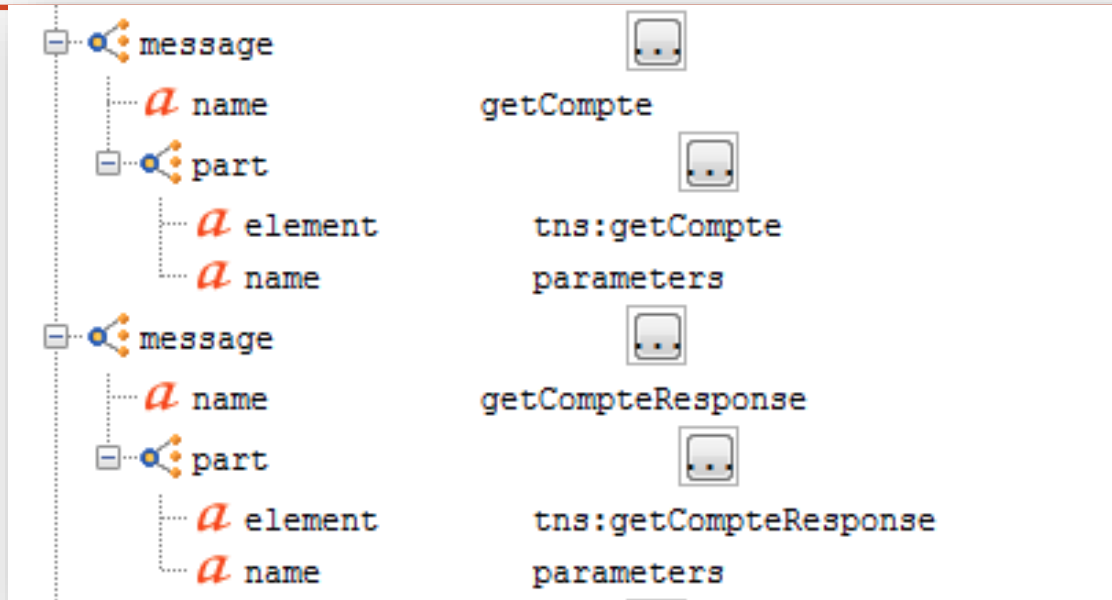
XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://ws/" xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://ws/">
  <xs:element name="ConversionEuroToDh" type="tns:ConversionEuroToDh"></xs:element>
  <xs:element name="ConversionEuroToDhResponse" type="tns:ConversionEuroToDhResponse"/>
  <xs:element name="compte" type="tns:compte"></xs:element>
  <xs:element name="getCompte" type="tns:getCompte"></xs:element>
  <xs:element name="getCompteResponse" type="tns:getCompteResponse"></xs:element>
  <xs:element name="getComptes" type="tns:getComptes"></xs:element>
  <xs:element name="getComptesResponse" type="tns:getComptesResponse"></xs:element>
  <xs:complexType name="ConversionEuroToDh">
    <xs:sequence>
      <xs:element name="montant" type="xs:double"></xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ConversionEuroToDhResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:double"></xs:element>
    </xs:sequence>
  </xs:complexType>
```

XML Schema

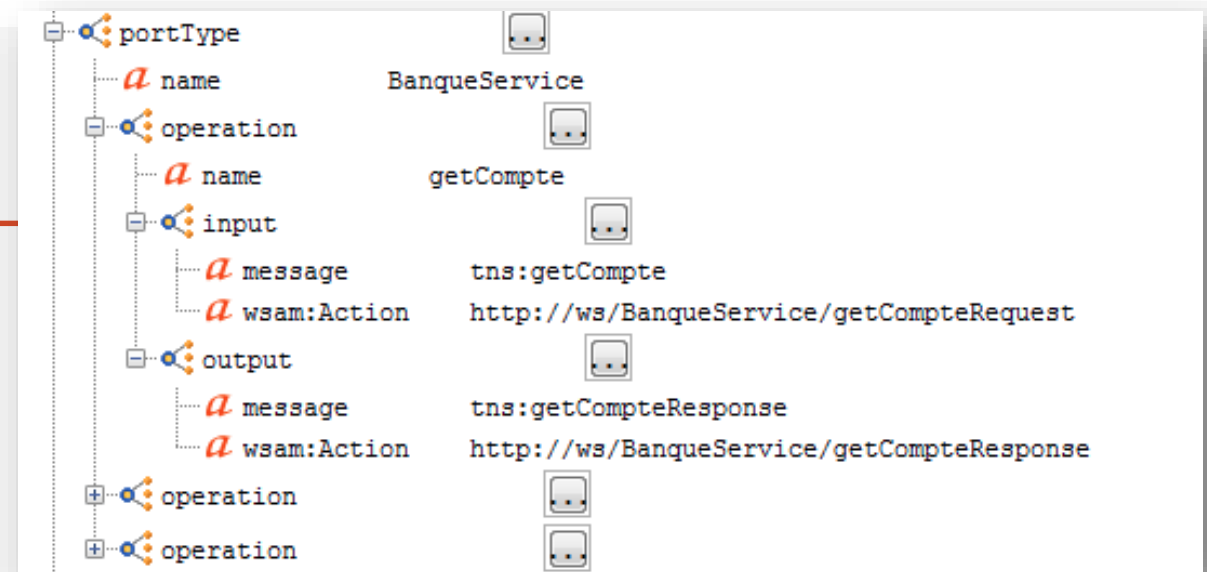
```
<xs:complexType name="getCompte">
  <xs:sequence>
    <xs:element name="code" type="xs:long" minOccurs="0"> </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="getCompteResponse">
  <xs:sequence>
    <xs:element name="return" type="tns:compte" minOccurs="0"> </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="compte">
  <xs:sequence>
    <xs:element name="code" type="xs:long" minOccurs="0"> </xs:element>
    <xs:element name="solde" type="xs:double"> </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="getComptes">
  <xs:sequence> </xs:sequence>
</xs:complexType>
<xs:complexType name="getComptesResponse">
  <xs:sequence>
    <xs:element name="return" type="tns:compte" minOccurs="0" maxOccurs="unbounded"> </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Élément message



```
<message name="getCompte">  
  <part name="parameters" element="tns:getCompte"> </part>  
</message>  
<message name="getCompteResponse">  
  <part name="parameters" element="tns:getCompteResponse"> </part>  
</message>
```

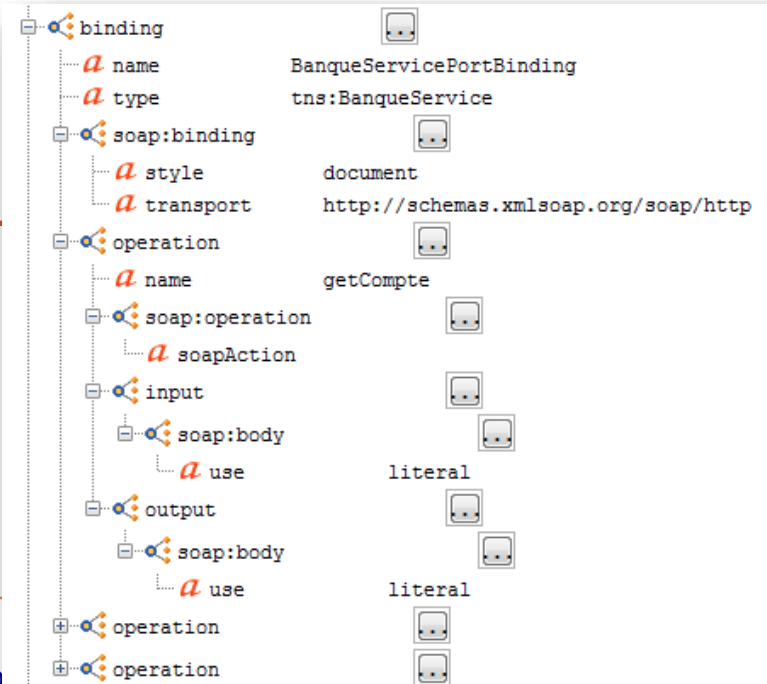

Elément portType



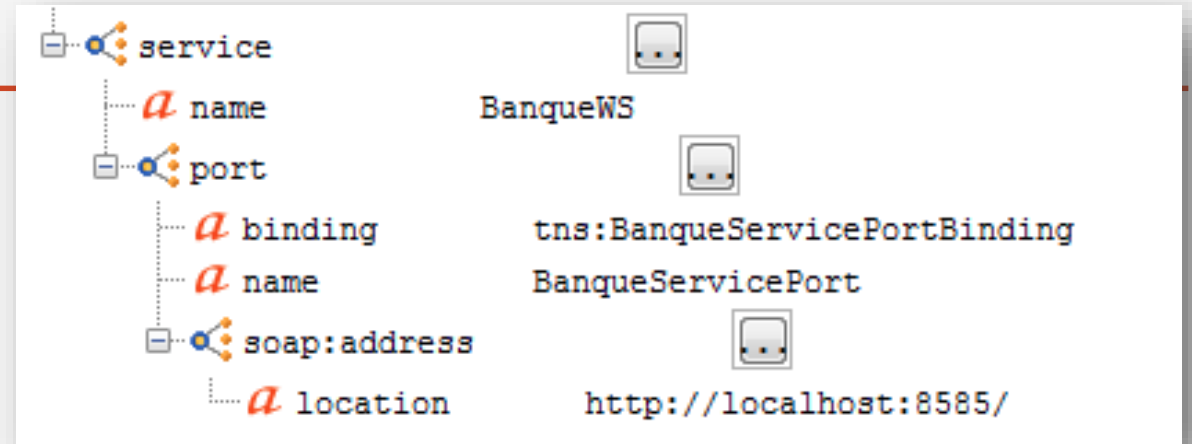
```
<portType name="BanqueService">
  <operation name="getCompte">
    <input wsam:Action="http://ws/BanqueService/getCompteRequest" message="tns:getCompte"> </input>
    <output wsam:Action="http://ws/BanqueService/getCompteResponse"
message="tns:getCompteResponse"> </output>
  </operation>
  <operation name="getComptes">
    <input wsam:Action="http://ws/BanqueService/getComptesRequest" message="tns:getComptes"> </input>
    <output wsam:Action="http://ws/BanqueService/getComptesResponse"
message="tns:getComptesResponse"> </output>
  </operation>
  <operation name="ConversionEuroToDh">
    ....
  </operation>
</portType>
```

Élément binding

```
<binding name="BanqueServicePortBinding" type="tns:BanqueService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"> </soap:binding>
  <operation name="getCompte">
    <soap:operation soapAction=""> </soap:operation>
    <input>
      <soap:body use="literal"> </soap:body>
    </input>
    <output>
      <soap:body use="literal"> </soap:body>
    </output>
  </operation>
  <operation name="getComptes">
    ....
  </operation>
</binding>
```

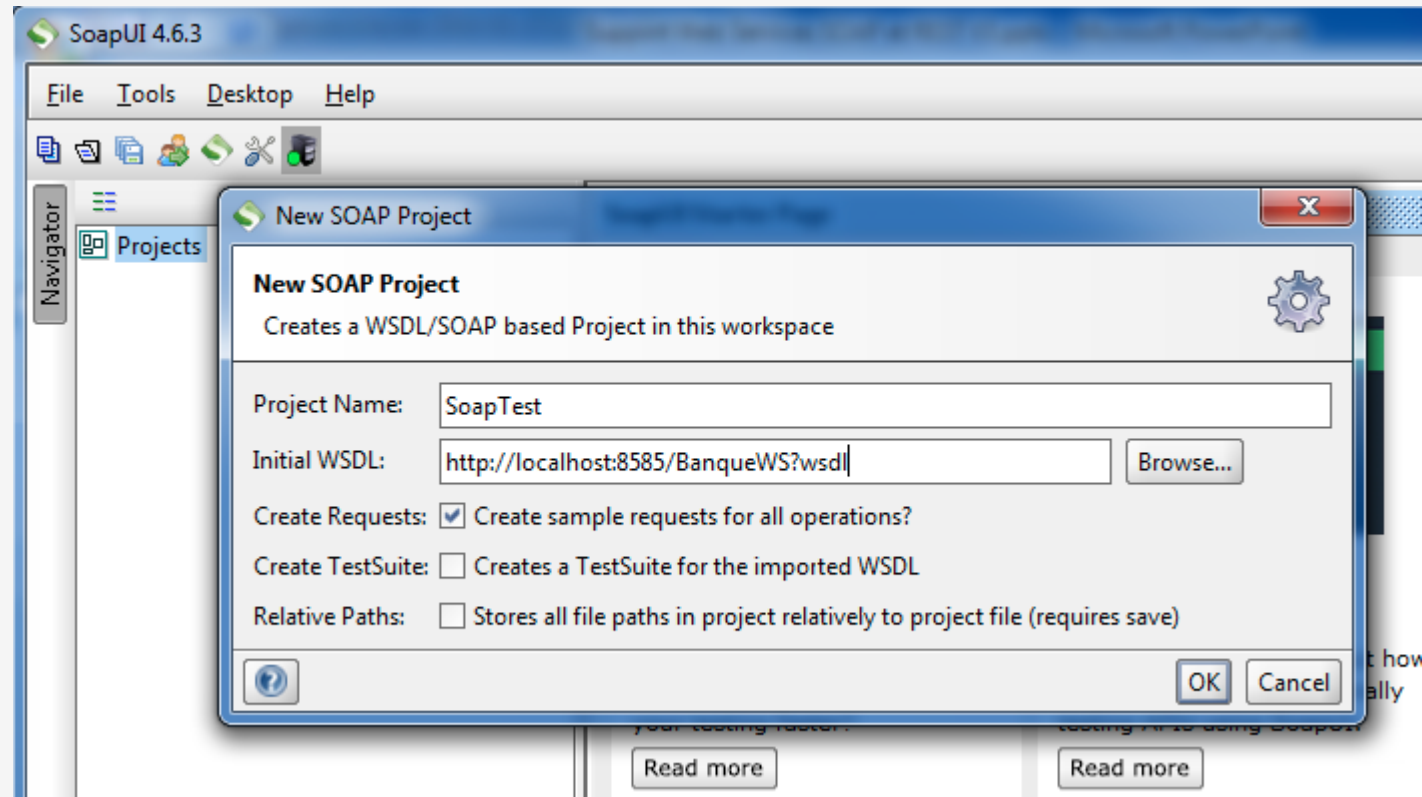


Élément service

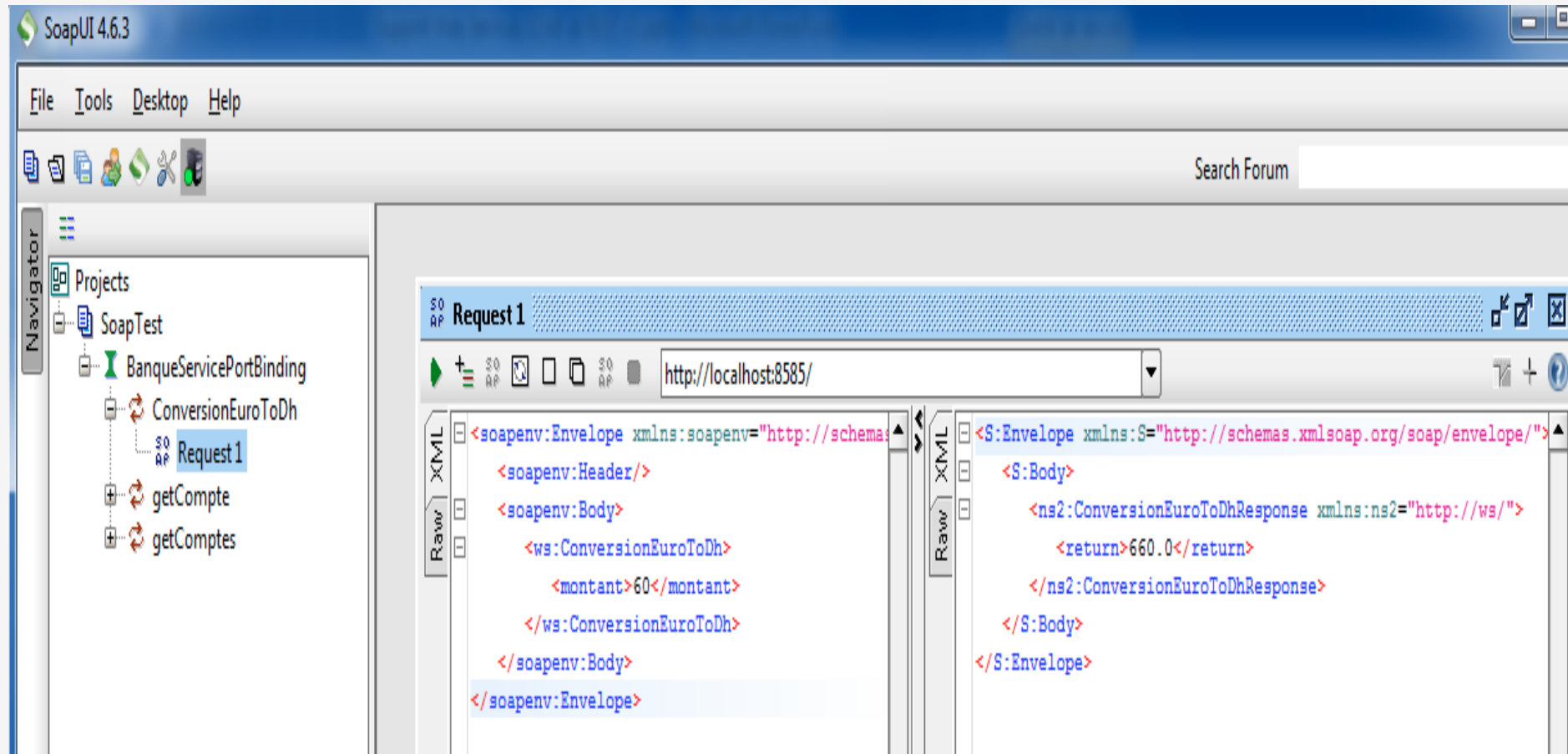


```
<service name="BanqueWS">  
  <port name="BanqueServicePort"  
binding="tns:BanqueServicePortBinding">  
    <soap:address location="http://localhost:8585/"> </soap:address>  
  </port>  
</service>
```

Tester les méthodes du web service avec un analyseur SOAP : SoapUI



Tester les méthodes du web service avec un analyseur SOAP : SoapUI



Tester la méthode conversion

Requête SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:ConversionEuroToDh>
      <montant>60</montant>
    </ws:ConversionEuroToDh>
  </soapenv:Body>
</soapenv:Envelope>
```

Réponse SOAP

- ```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:ConversionEuroToDhResponse xmlns:ns2="http://ws/">
 <return>660.0</return>
 </ns2:ConversionEuroToDhResponse>
 </S:Body>
</S:Envelope>
```

# Tester la méthode getCompte

## Requête SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
 <soapenv:Header/>
 <soapenv:Body>
 <ws:getCompte>
 <code>2</code>
 </ws:getCompte>
 </soapenv:Body>
</soapenv:Envelope>
```

## Réponse SOAP

- ```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getCompteResponse xmlns:ns2="http://ws/">
      <return>
        <code>2</code>
        <solde>7000.0</solde>
      </return>
    </ns2:getCompteResponse>
  </S:Body>
</S:Envelope>
```

Tester la méthode getComptes

Requête SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:getComptes/>
  </soapenv:Body>
</soapenv:Envelope>
```

Réponse SOAP

- ```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:getComptesResponse xmlns:ns2="http://ws/">
 <return>
 <code>1</code>
 <solde>7000.0</solde>
 </return>
 <return>
 <code>2</code>
 <solde>7000.0</solde>
 </return>
 </ns2:getComptesResponse>
 </S:Body>
</S:Envelope>
```



# Client Java

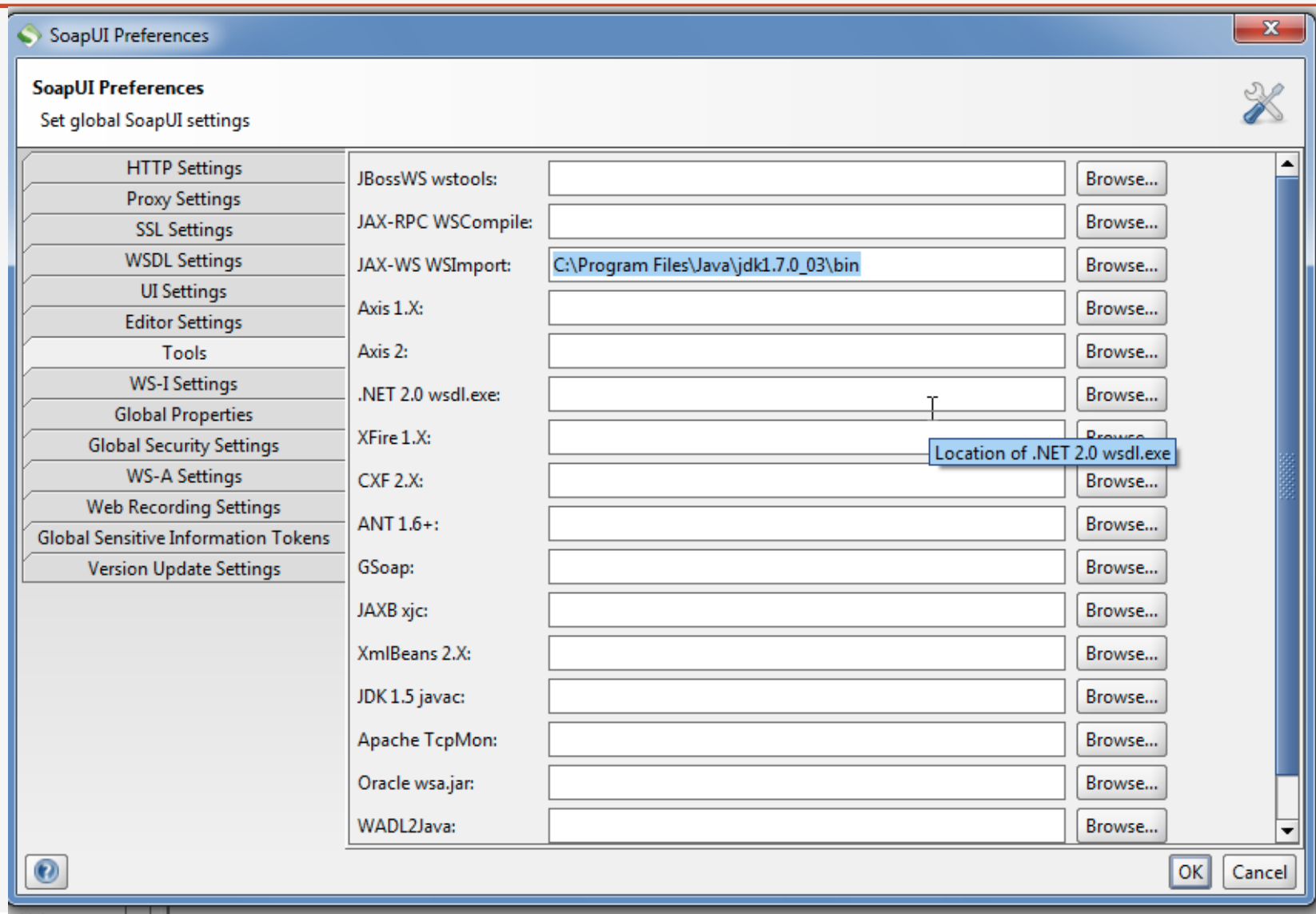
---

Créer un nouveau projet Java

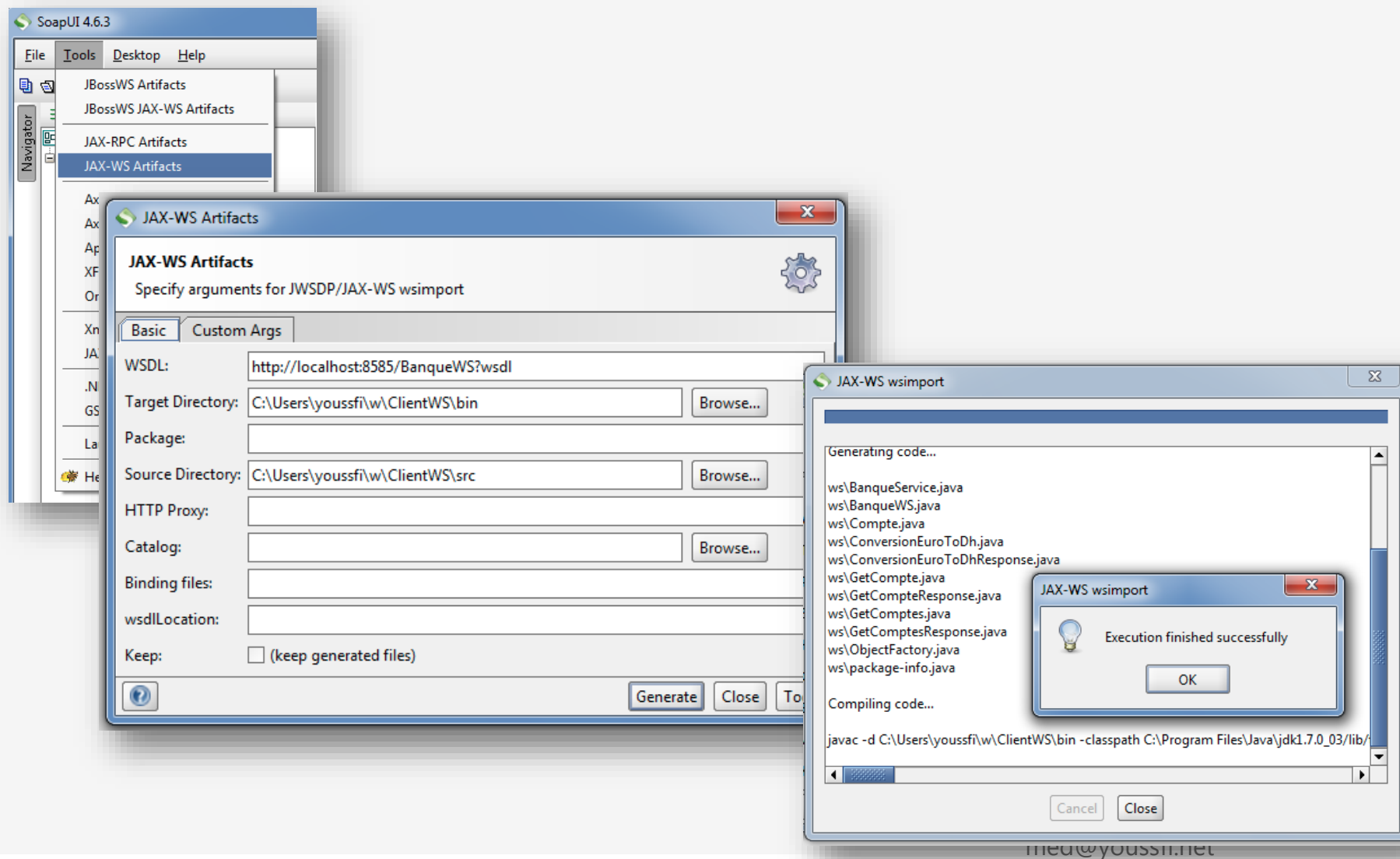
Générer un proxy

- SoapUI est l'un des outils qui peuvent être utilisés pour générer les artefacts client en utilisant différents Framework (Axis, CXF, JaxWS, etc...)
- Le JDK fournit une commande simple qui permet de générer un STUB JaxWS pour l'accès à un web service. Cette commande s'appelle **wsimport**.
- SoapUI a besoin de savoir le chemin de cette commande
- Avec la commande File > Preferences > Tools , vous pouvez configurer le ce chemin comme le montre la figure suivante :

# Préférence générale de SoapUI

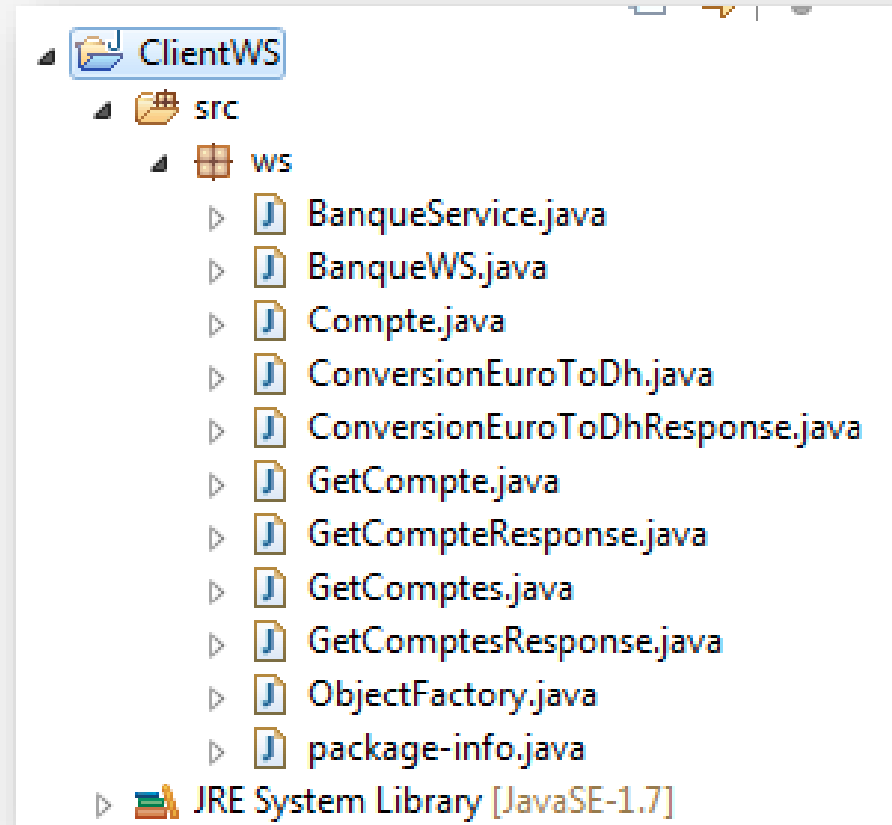


# Générer le STUB JaxWS



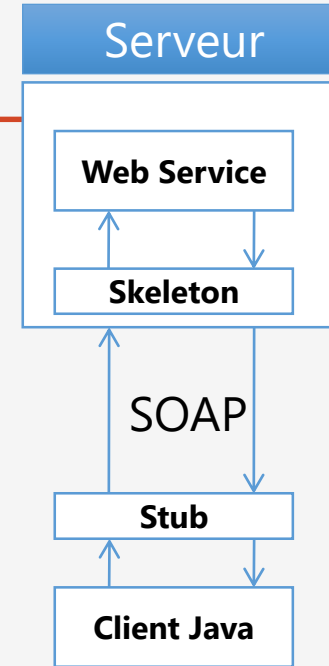
# Fichiers Générés

---



# Client Java

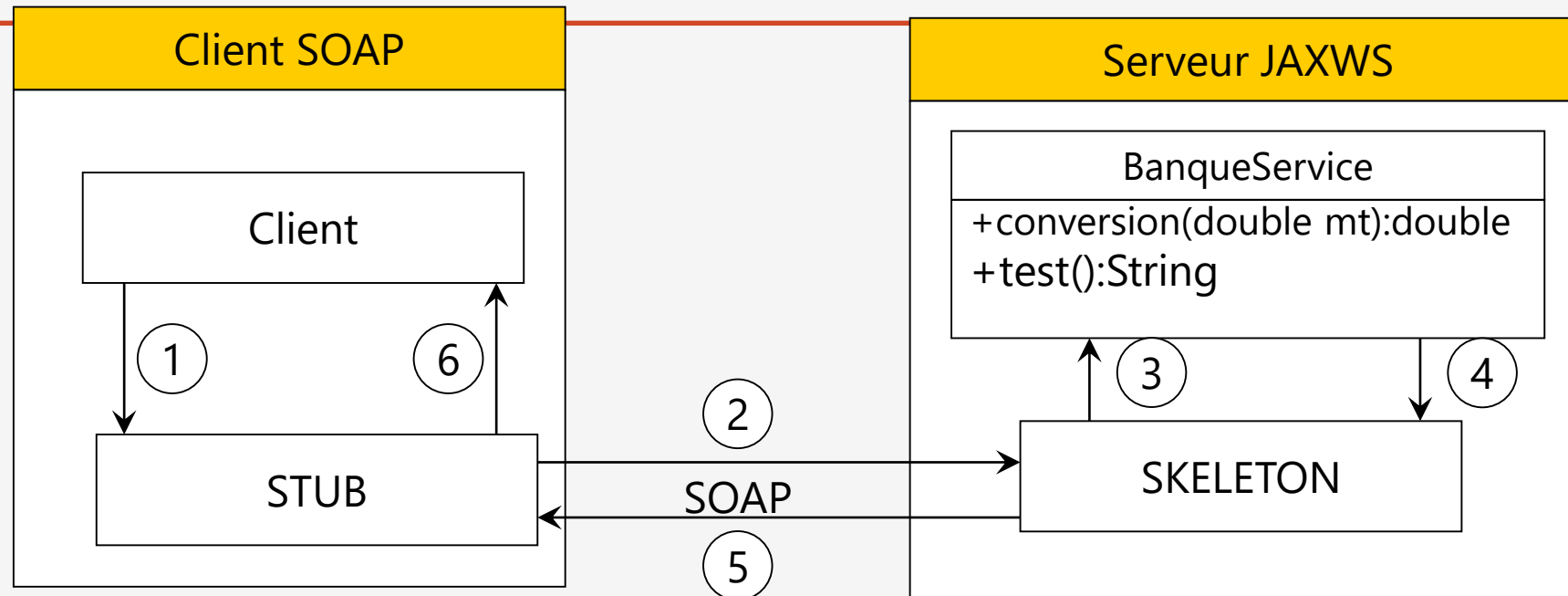
```
import java.util.List;
import ws.BanqueService;
import ws.BanqueWS;
import ws.Compte;
public class ClientWS {
 public static void main(String[] args) {
 BanqueService stub=new BanqueWS().getBanqueServicePort();
 System.out.println("Conversion");
 System.out.println(stub.conversionEuroToDh(9000));
 System.out.println("Consulter un compte");
 Compte cp=stub.getCompte(2L);
 System.out.println("Solde="+cp.getSolde());
 System.out.println("Liste des comptes");
 List<Compte> cptes=stub.getComptes();
 for(Compte c:cptes){
 System.out.println(c.getCode()+"----"+c.getSolde());
 }
 }
}
```



The screenshot shows a Java application window titled 'ClientWS (1) [Java Application] C:\Program Files\Java\'. The console output is as follows:

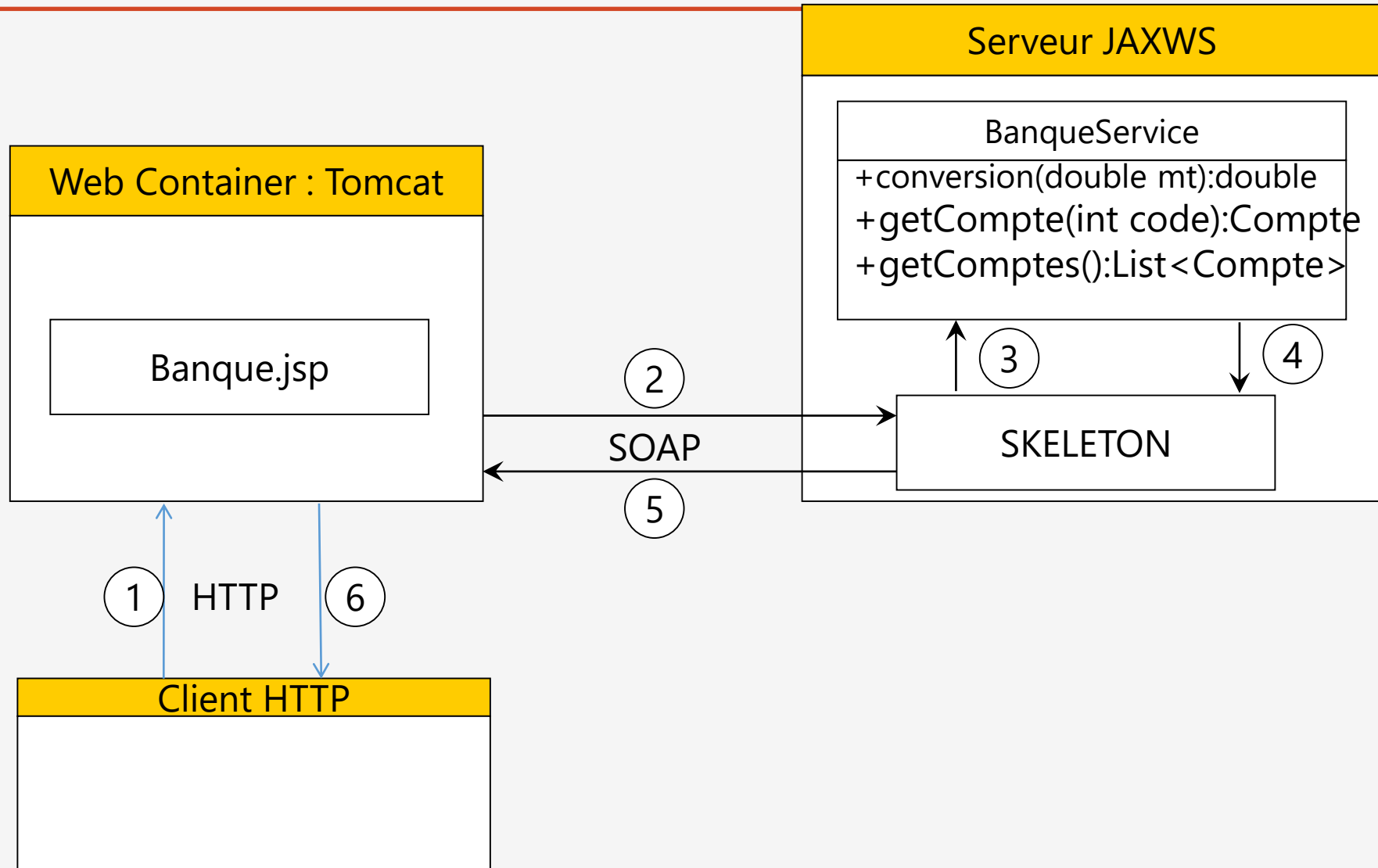
```
<terminated> ClientWS (1) [Java Application] C:\Program Files\Java\
Conversion
99000.0
Consulter un compte
Solde=7000.0
Liste des comptes
1----7000.0
2----7000.0
```

# Architecture



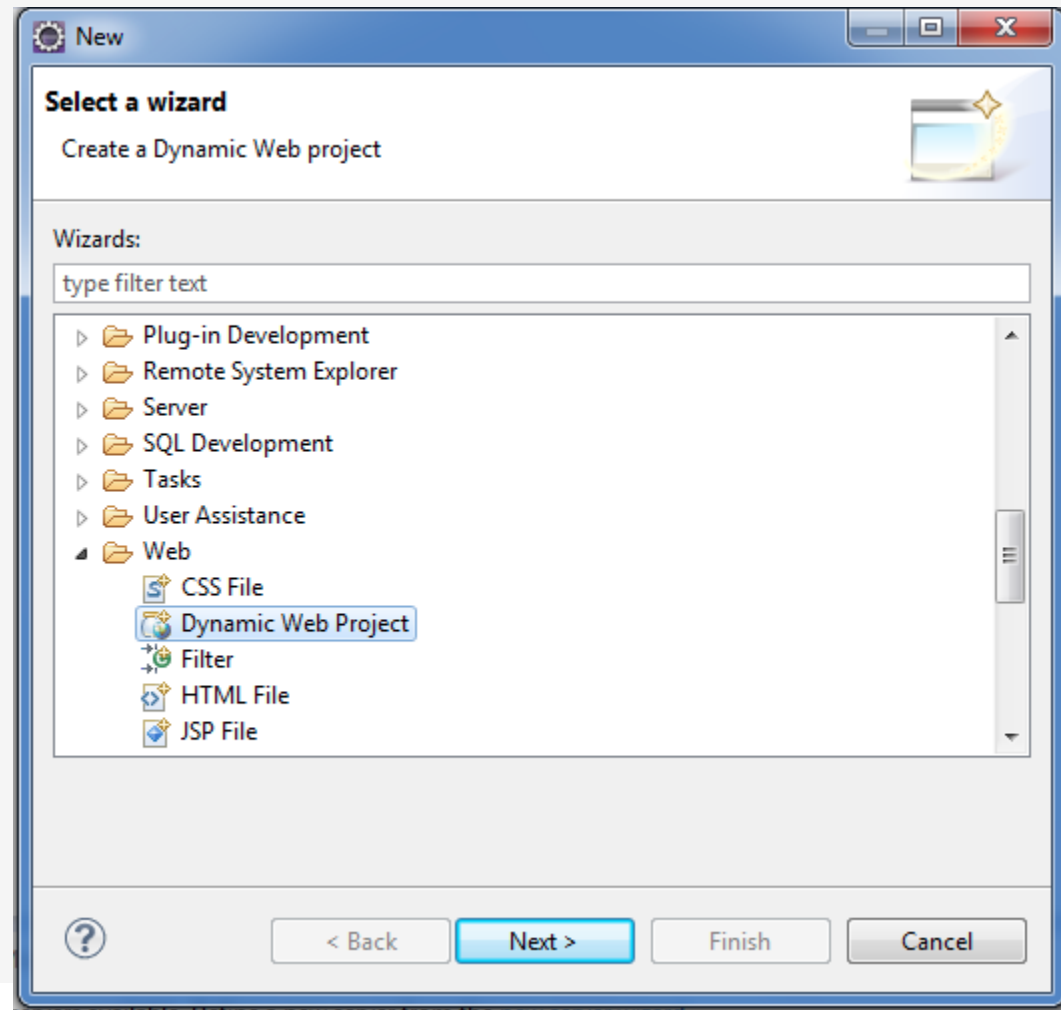
- 1 Le client demande au stub de faire appel à la méthode conversion(12)
- 2 Le Stub se connecte au Skeleton et lui envoie une requête SOAP
- 3 Le Skeleton fait appel à la méthode du web service
- 4 Le web service retourne le résultat au Skeleton
- 5 Le Skeleton envoie le résultat dans une la réponse SOAP au Stub
- 6 Le Stub fournie le résultat au client

# Client JSP



# Création d'un projet Web Dynamique basé sur Tomcat 7

Créer un projet Web Dynamique basé sur Tomcat 7





# Projet Web Dynamique

**New Dynamic Web Project**

**Dynamic Web Project**  
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

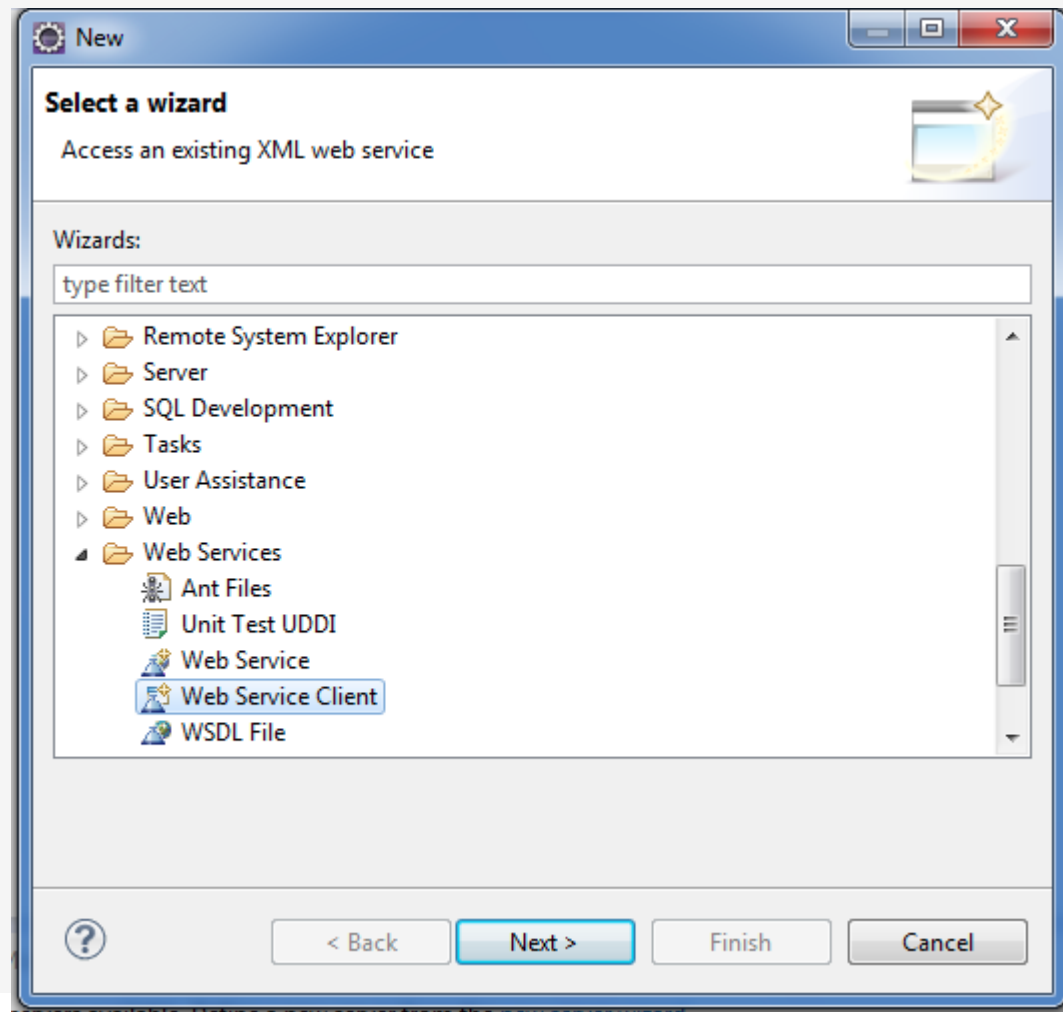
A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

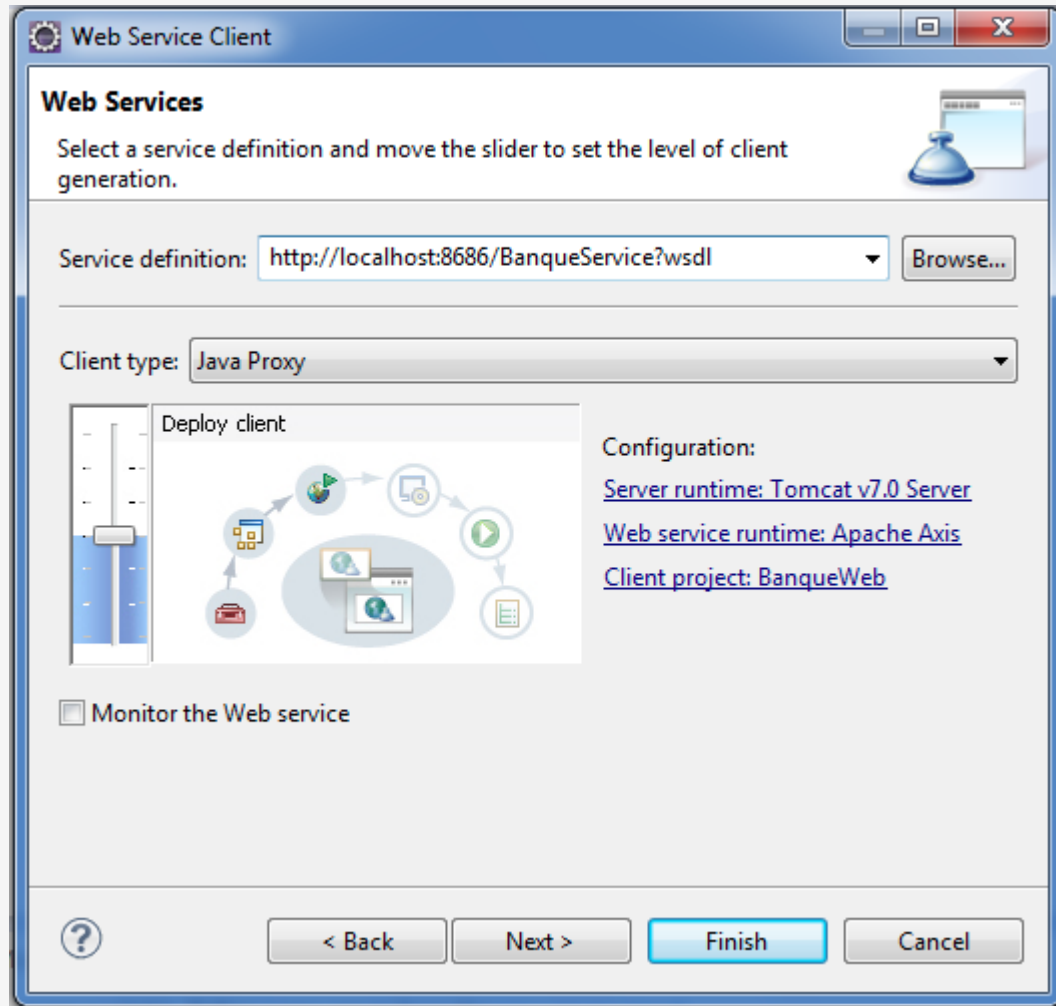
# Générer un proxy (Stub) pour le web Service

Fichier > Nouveau > Web Service Client



# Génération du proxy

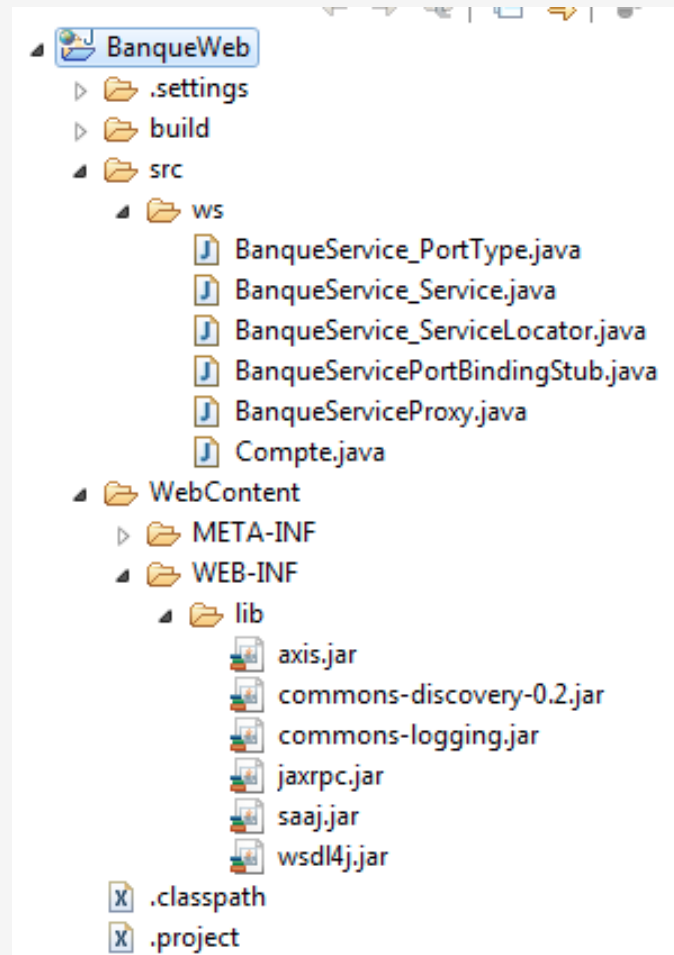
## Générer le proxy à partir du WSDL



# Fichier Générés

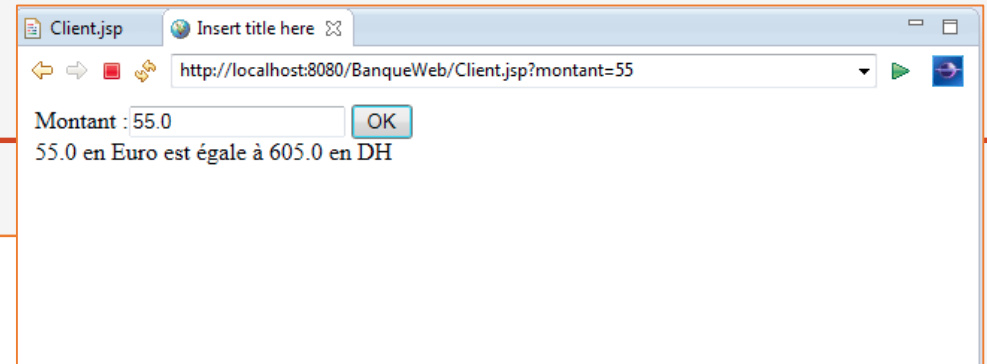
---

Le proxy généré est basé sur AXIS



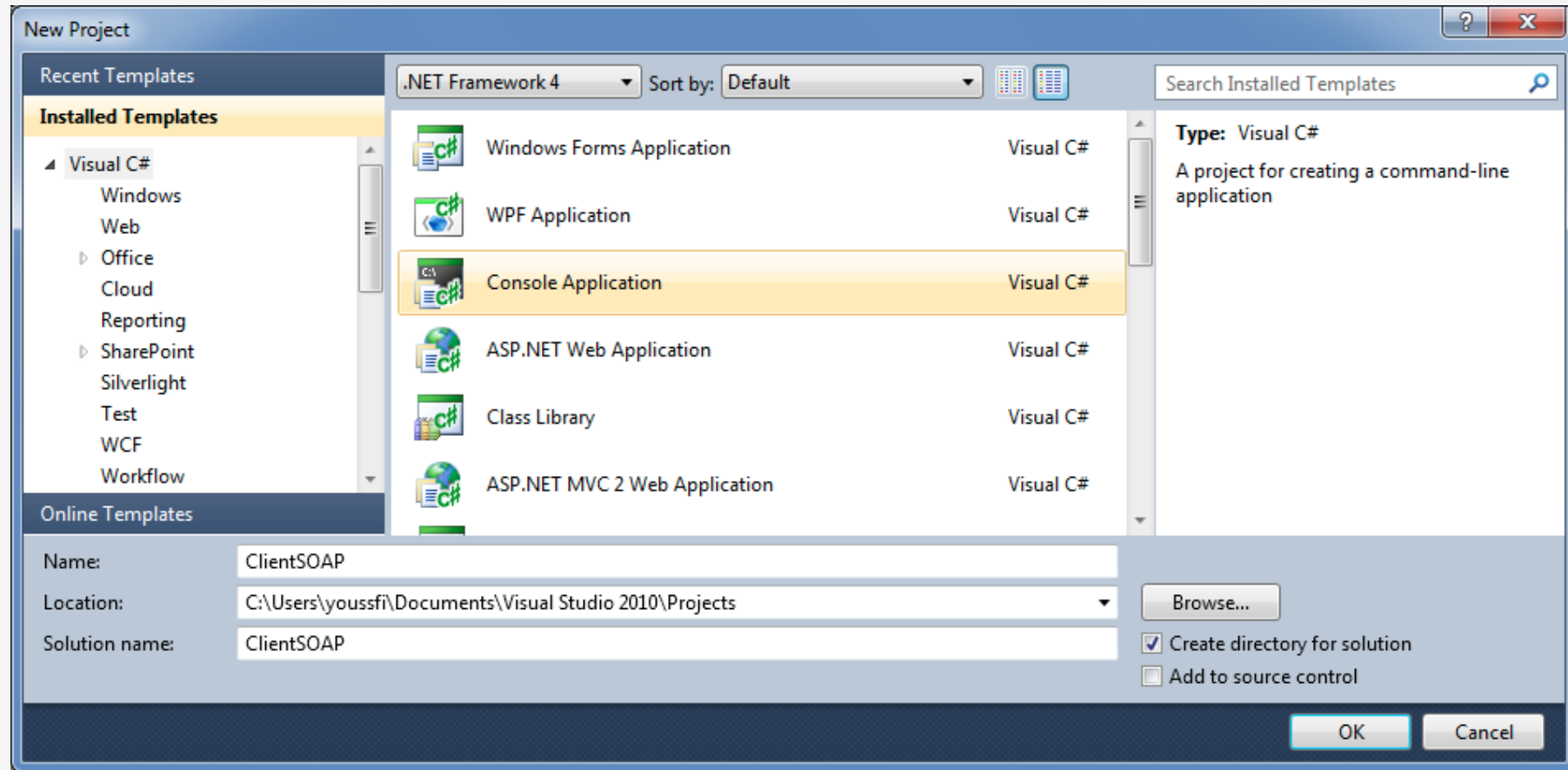
# Client JSP

```
<%@page import="ws.BanqueServiceProxy"%>
<%
 double montant=0; double resultat=0;
 if(request.getParameter("montant")!=null){
 montant=Double.parseDouble(request.getParameter("montant"));
 BanqueServiceProxy service=new BanqueServiceProxy();
 resultat=service.conversionEuroDH(montant);
 }
%>
<html><body>
 <form action="Client.jsp">
 Montant :<input type="text" name="montant" value="<%=montant%>">
 <input type="submit" value="OK">
 </form>
 <%=montant %> en Euro est égale à <%=resultat %> en DH
</body></html>
```

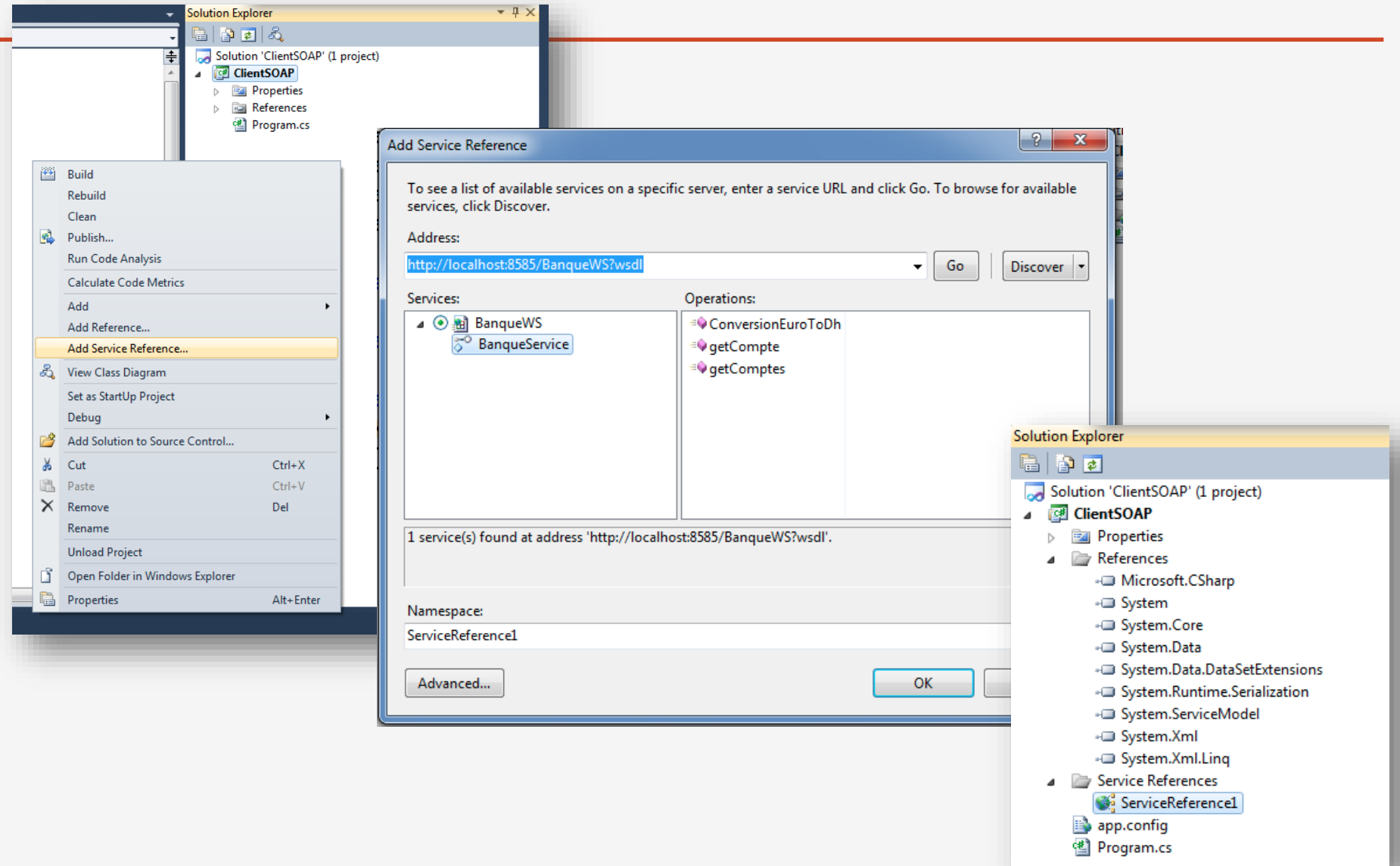


# Client SOAP avec .Net

En utilisant Visual Studio (2010), Créer un projet C# de type Console



# Générer Le Proxy client Soap

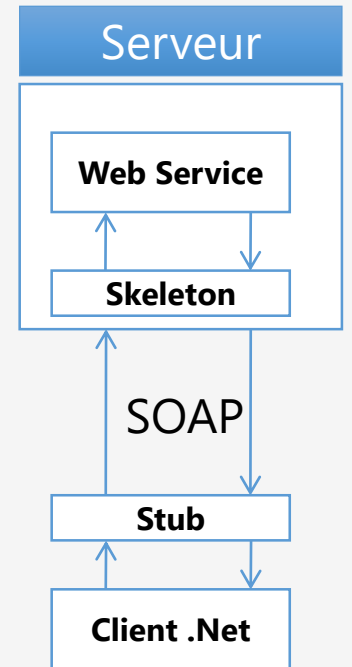


# Code C# du client

```
using System;
namespace ClientSOAP
{
 class Program
 {
 static void Main(string[] args)
 {
 ServiceReference1.BanqueServiceClient stub =
 new ServiceReference1.BanqueServiceClient();

 Console.WriteLine("----- Conversion -----");
 Console.WriteLine(stub.ConversionEuroToDh(34));
 Console.WriteLine("----- Consulter un Compte -----");
 ServiceReference1.compte cp = stub.getCompte(2L);
 Console.WriteLine("Solde=" + cp.solde);
 Console.WriteLine("----- Liste des comptes -----");
 ServiceReference1.compte[] cptes = stub.getComptes();
 for (int i = 0; i < cptes.Length; i++)
 {
 Console.WriteLine(cptes[i].code + "-----" + cptes[i].solde);
 }

 Console.ReadLine();
 }
 }
}
```



The screenshot shows the output of the C# client program running in a console window. The output matches the expected results from the code: a conversion of 34 Euros to 374 Dh, an account balance of 7000 for account 2, and a list of two accounts both with a balance of 7000.

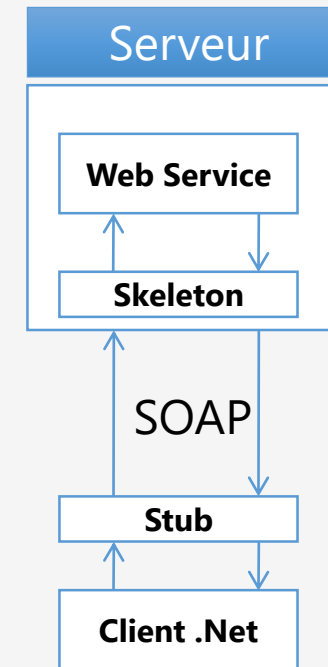
```
----- Conversion -----
374
----- Consulter un Compte -----
Solde=7000
----- Liste des comptes -----
1-----7000
2-----7000
-
```



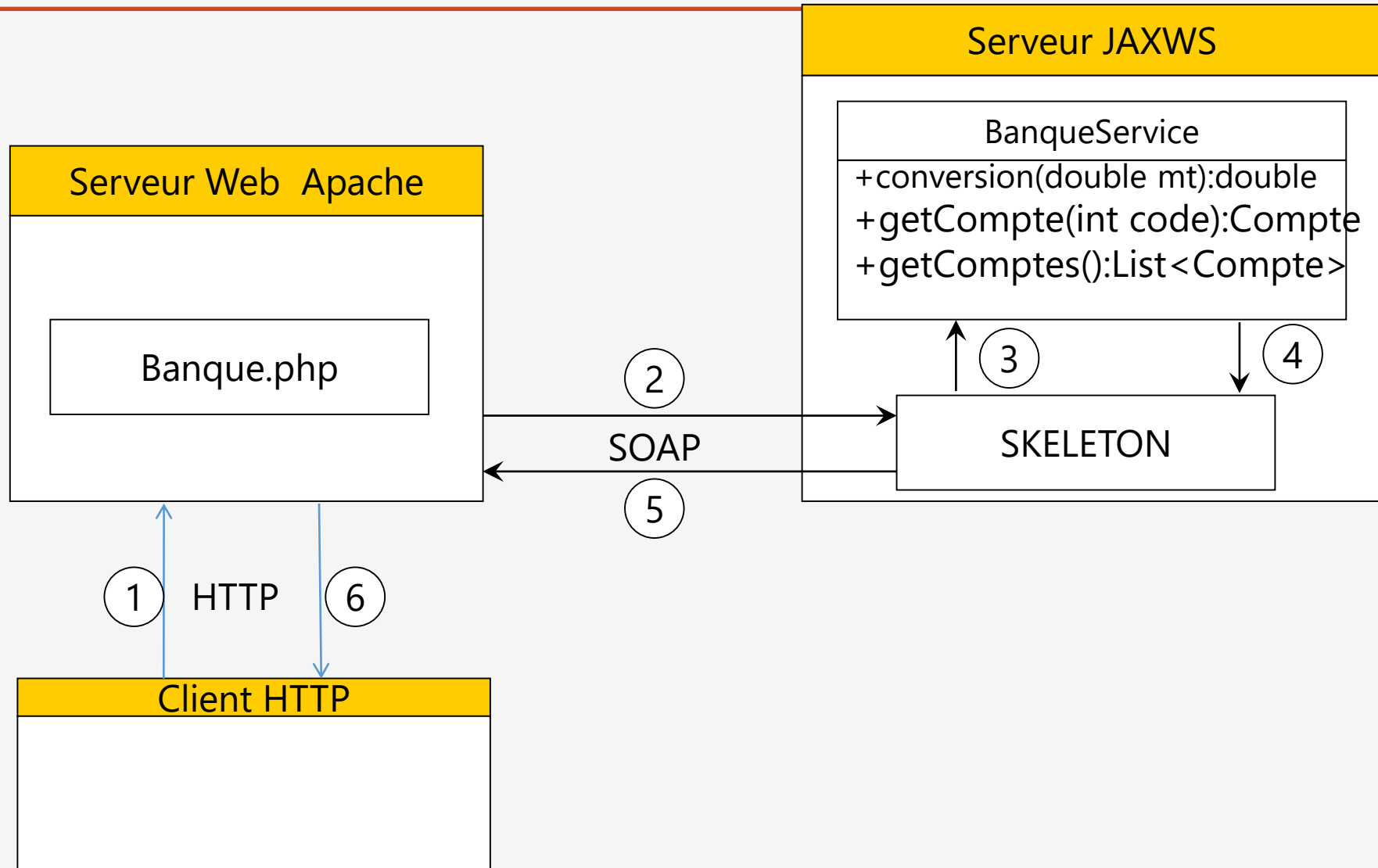
# Dessiner les composants graphique de l'interface

The screenshot shows a Windows application window titled "Form1". Inside the window, there is a user interface for a conversion tool. It features two text input fields: "Montant:" with the value "45" and "Résultat:" with the value "495". To the right of these fields are two buttons: "Conversion" (highlighted with a blue border) and "Comptes". Below the input fields is a table with three columns: an empty column, "CODE", and "SOLDE". The table contains three rows: the first row has a right-pointing triangle icon, the value "1", and "7000"; the second row has an empty cell, the value "2", and "7000"; the third row has an asterisk icon, an empty cell, and an empty cell. The table is set against a gray background.

	CODE	SOLDE
▶	1	7000
	2	7000
*		



# Web Service Java et Client PHP

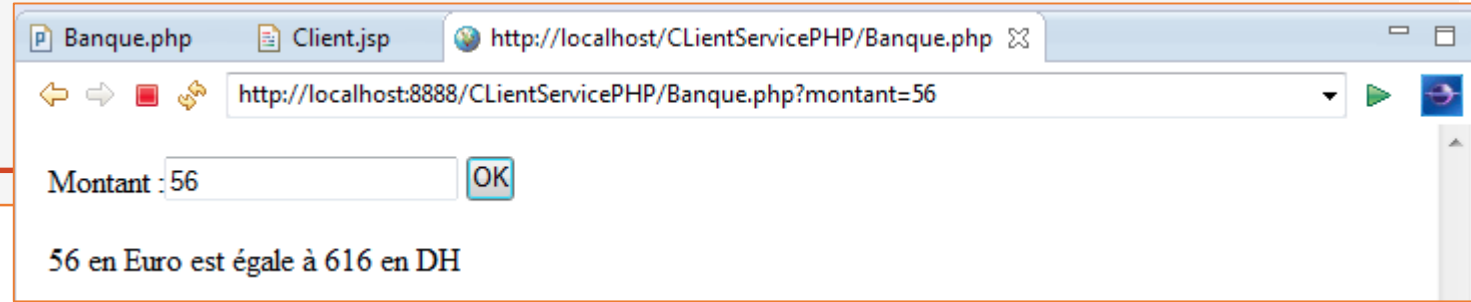


# Exemple de Client SOAP PHP

```
<?php
$client = new SoapClient('http://localhost:8585/BanqueWS?wsdl');
$params=new stdClass();
$params->montant=23;
$res=$client->__soapCall("conversionEuroDH",array($params));
//var_dump($res);
echo($res->return);
$params2=new stdClass();
$params2->arg0=2;
$res2=$client->__soapCall("getCompte",array($params2));
//var_dump($res2);
echo("Code=".$res2->return->code);
echo("
Solde=".$res2->return->solde);
$res3=$client->__soapCall("getComptes",array());
//var_dump($res3);
echo ("<hr/>");
foreach($res3->return as $cpte){
 echo("Code=".$cpte->code);
 echo("
Solde=".$cpte->solde);
 echo("
");
}
?>
```

# Code PHP

```
<?php
$montant=0;$resultat=0;
if (isset($_GET['montant'])){
$montant=$_GET['montant'];
$client = new SoapClient('http://localhost:8686/BanqueService?wsdl');
$params=new stdClass();
$params->montant=$montant;
$rep=$client->__soapCall("conversionEuroDH",array($params));
$resultat=$rep->return;
}
?>
<html>
<body>
<form action="Banque.php">
Montant :<input type="text" name="montant" value="<?php echo($montant)?>">
<input type="submit" value="OK">
</form>
<?php echo($montant)?> en Euro est égale à <?php echo($resultat)?> en DH
</body>
</html>
```



# Un autre exemple Client SOAP PHP

```
<?php
$mt=0;
if(isset($_POST['action'])){
 $action=$_POST['action'];
 if($action=="OK"){
 $mt=$_POST['montant'];
 $client=new SoapClient("http://localhost:8585/BanqueWS?wsdl");
 $param=new stdClass();
 $param->montant=$mt;
 $rep=$client->__soapCall("conversionEuroToDh",array($param));
 $res=$rep->return;
 }
 elseif($action=="listComptes"){
 $client=new SoapClient("http://localhost:8585/BanqueWS?wsdl");
 $res2=$client->__soapCall("getComptes",array());
 }
}

?>
```

← → ↺ localhost/ClientPHP/clientSOAP.php

Montant:

Résultat:

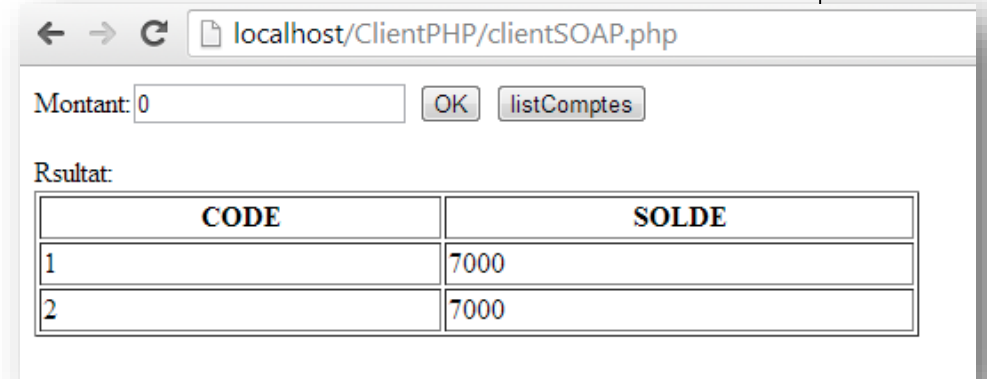
CODE	SOLDE
1	7000
2	7000

# Suite de l'exemple du Client SOAP PHP

```
<html>
<body>
 <form method="post" action="clientSOAP.php">
 Montant:<input type="text" name="montant" value="<?php echo($mt)?>">
 <input type="submit" value="OK" name="action">
 <input type="submit" value="listComptes" name="action">
 </form>
```

Rsltat:

```
<?php if (isset($res)){
 echo($res);
}
?>
<?php if(isset($res2)){?>
 <table border="1" width="80%">
 <tr>
 <th>CODE</th><th>SOLDE</th>
 </tr>
 <?php foreach($res2->return as $cp) {?>
 <tr>
 <td><?php echo($cp->code)?></td>
 <td><?php echo($cp->solde)?></td>
 </tr>
 <?php }?>
 </table>
<?php }?>
</body>
</html>
```



localhost/ClientPHP/clientSOAP.php

Montant: 0

Rsltat:

CODE	SOLDE
1	7000
2	7000

# UDDI

---

- L'annuaire des services UDDI est un standard pour la publication et la découverte des informations sur les services Web.
- La spécification UDDI est une initiative lancée par *ARIBA*, *Microsoft* et *IBM*.
- Cette spécification n'est pas gérée par le W3C mais par le groupe OASIS.
- La spécification UDDI vise à créer une plate-forme indépendante, un espace de travail (framework) ouvert pour la description, la découverte et l'intégration des services des entreprises.

# Consultation de l'annuaire

---

L'annuaire UDDI se concentre sur le processus de découverte de l'architecture orientée services (SOA), et utilise des technologies standards telles que XML, SOAP et WSDL qui permettent de simplifier la collaboration entre partenaires dans le cadre des échanges commerciaux.

L'accès au référentiel s'effectue de différentes manières.

- **Les pages blanches** : comprennent la liste des entreprises ainsi que des informations associées à ces dernières (coordonnées, description de l'entreprise, identifiants...).
- **Les pages jaunes** : recensent les services Web de chacune des entreprises sous le standard WSDL.
- **Les pages vertes** : fournissent des informations techniques précises sur les services fournis.



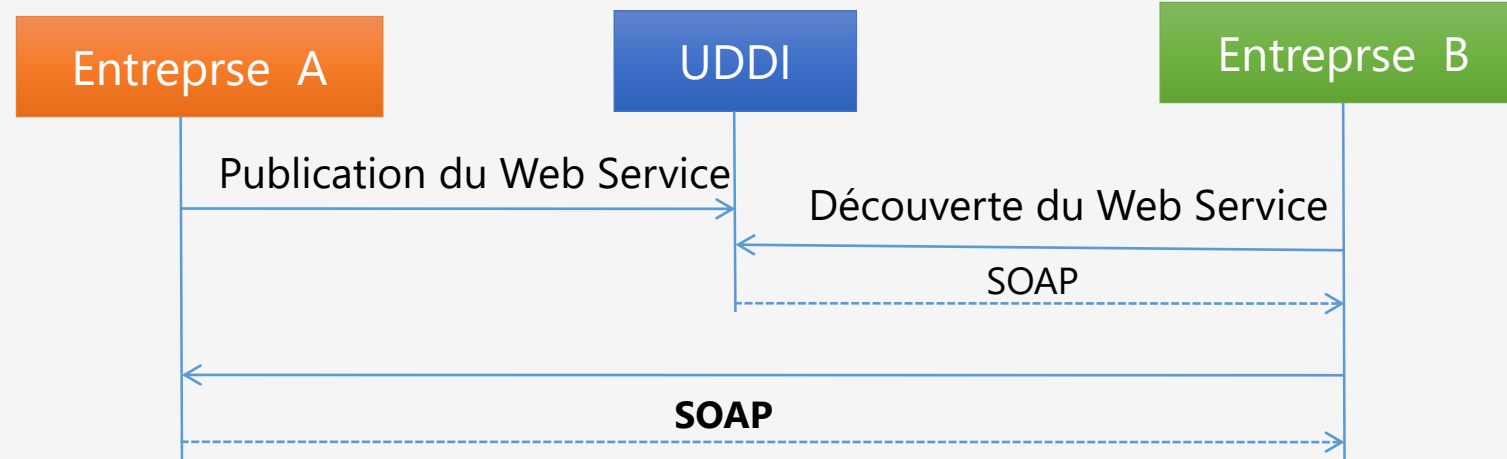
# Architecture

Les entreprises publient les descriptions de leurs services Web en UDDI, sous la forme de fichiers WSDL.

Ainsi, les clients peuvent plus facilement rechercher les services Web dont ils ont besoin en interrogeant le registre UDDI.

Lorsqu'un client trouve une description de service Web qui lui convient, il télécharge son fichier WSDL depuis le registre UDDI. Ensuite, à partir des informations inscrites dans le fichier WSDL, notamment la référence vers le service Web, le client peut invoquer le service Web et lui demande d'exécuter certaines de ses fonctionnalités.

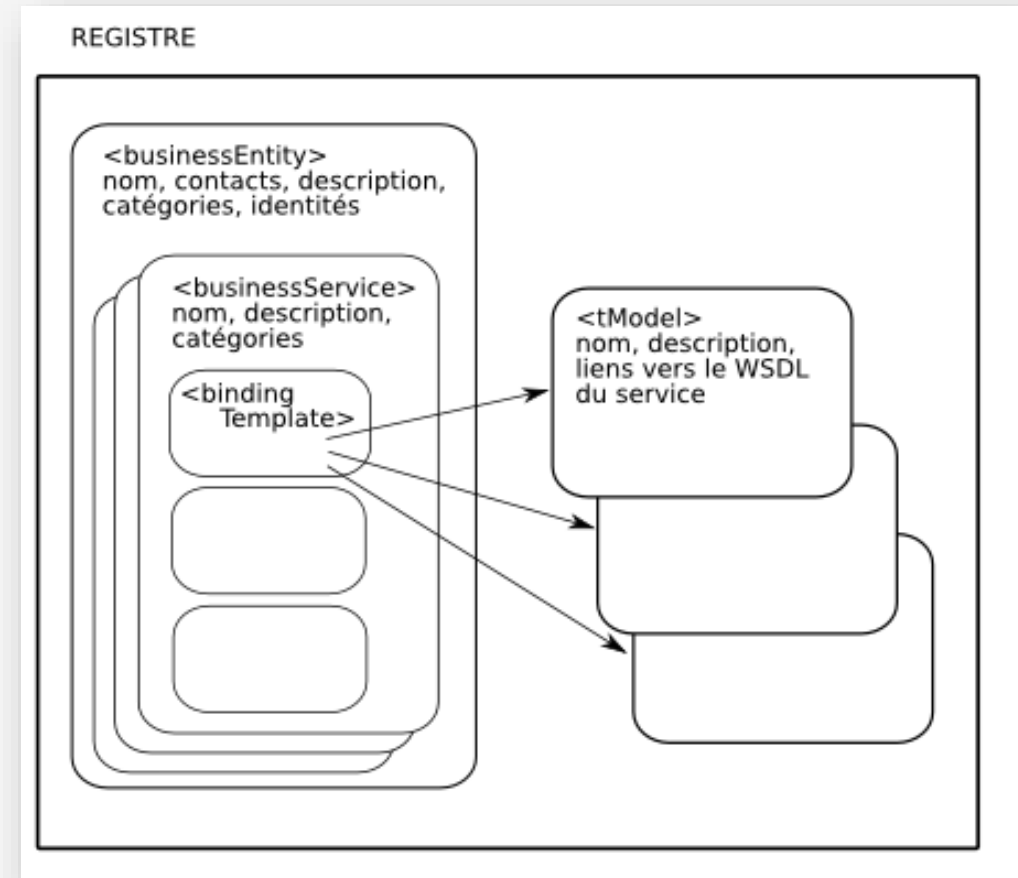
Le scénario classique d'utilisation de UDDI est illustré ci-dessous. L'entreprise B a publié le service Web S, et l'entreprise A est client de ce service :



# Structures de données UDDI

Un registre UDDI se compose de quatre types de structures de données,

- le **businessEntity**,
- Le **businessService**,
- le **bindingTemplate**
- et la **tModel**.



# BusinessEntity (entité d'affaires)

---

Les « businessEntities » sont en quelque sorte les pages blanches d'un annuaire UDDI. Elles décrivent les organisations ayant publié des services dans le répertoire.

On y trouve notamment

- le nom de l'organisation,
- ses adresses (physiques et Web),
- des éléments de classification,
- une liste de contacts
- ainsi que d'autres informations.

# BusinessService (service d'affaires)

---

Les « businessServices » sont en quelque sorte les pages jaunes d'un annuaire UDDI.

Elles décrivent de manière non technique les services proposés par les différentes organisations.

On y trouve essentiellement

- le nom et la description textuelle des services
- ainsi qu'une référence à l'organisation proposant le service
- et un ou plusieurs « bindingTemplate ».

# BindingTemplate (modèle de rattachement)

---

- UDDI permet de décrire des services Web utilisant HTTP, mais également des services invoqués par d'autres moyens (SMTP, FTP...).
- Les « bindingTemplates » donnent les coordonnées des services.
- Ce sont les pages vertes de l'annuaire UDDI.
- Ils contiennent notamment une description, la définition du **point d'accès** (une URL) et les éventuels « tModels » associés.

## tModel (index)

---

- Les « tModels » sont les descriptions techniques des services.
- UDDI n'impose aucun format pour ces descriptions qui peuvent être publiées sous n'importe quelle forme et notamment sous forme de documents textuels (XHTML, par exemple).
- C'est à ce niveau que WSDL intervient comme le vocabulaire de choix pour publier des descriptions techniques de services.

# L'interface UDDI

---

L'interface UDDI est définie sous forme de documents UDDI et implémentée sous forme de service Web SOAP.

Elle est composée des modules suivants :

- **Interrogation inquiry** : Cette interface permet de rechercher des informations dans un répertoire UDDI.
- **Publication** : Cette interface permet de publier des informations dans un répertoire UDDI.
- **Sécurité** : cette interface est utilisée pour obtenir et révoquer les jetons d'authentification nécessaires pour accéder aux enregistrements protégés dans un annuaire UDDI.
- **Contrôle d'accès et propriété custody and ownership transfer**: Cette interface permet de transférer la propriété d'informations (qui est à l'origine attribuée à l'utilisateur ayant publié ces informations) et de gérer les droits d'accès associés.
- **Abonnement Subscription** : Cette interface permet à un client de s'abonner à un ensemble d'informations et d'être averti lors des modifications de ces informations.