# INF157 - Utilisation des Réseaux

## Licence 3 Informatique

Arnaud Pecher (repris par Damien Magoni)

Bureau 322, Bâtiment A30, LaBRI
Université de Bordeaux

Licence 3 Informatique - Bordeaux

*Bibliographie* :

- X. Blanc, "Web Services", Lip6
- J.-M. Chauvet, "Services Web avec SOAP, WSDL, UDDI, ebXML ...", Eyrolles
- A. Dulaunoy, "Le cadre des services webs"
- K. Topley, "Java Web Services in a nutshell", O'Reilly

*Hyperliens*

- `http ://services.xmethods.net :80/soap`
- `http ://www.zend.com/php5/articles/php5-SOAP.php`

# Historique

## Pourquoi les Services Web ?

### Web :

- Tim Berner Lee créa le premier client Web en 1990 ;
- Conçu pour une interface Homme-Machine ;
- Evolution vers machine-machine ;
- Analyse syntaxique automatique difficile ;
- Interface instable et propre à  chaque technologie/site.

# Les ancètres

Insatisfaction avec SUN RPC, DCOM, RMI, ... :

- Sortir des problèmes de compatibilités systèmes, langage et format ;
- Sortir du problème propriétaire et créer des standards ;
- Utiliser l'infrastructure Internet existante ;
- Simplifier et limiter le temps de développement ;

## SUN Remote Procedure Call

- Première approche pour standardiser les interactions entre un client et un serveur ;
- Le serveur offre des procédures (identifiées par un nom) et le client demande au serveur l'accès à une procédure donnée avec les valeurs (paramètres) ;
- XDR (eXternal Data Representation) solutionne le problème de la représentation binaire des données.

# Origines (2)

## Microsoft DCOM

Microsoft DCOM (Distributed Component Object Model) COM -> DCOM -> COM+)

- Pas uniquement des procédures (comme SUN RPC) mais aussi un interfaà§age avec des objets ou appels de méthodes ;
- DCOM se veut neutre par rapport à  la plateforme, au langage et même au transport ;
- DCOM est resté dans l'environnement Microsoft pour des questions de standardisation ;
- DCOM est propriétaire.

## CORBA (Common Object Request Broker Architecture)

- Fonctionne avec un ordonnanceur (ORB) pour les requêtes et les transferts vers un serveur donné ;
- Utilisé sur des très larges projets ;
- IDL (Interface Design Language) est un langage de description des services offerts par l'application ;
- La complexité de CORBA est souvent une source de problèmes...

## Services web

Ensemble de standards, permettant à des serveurs webs d'offrir des services.

*Standards* :

- XML : toutes les données à échanger sont formatées en XML. Ce codage peut être effectué par SOAP ou XML-RPC ;
- Protocoles communs : des données en XML peuvent être transportées entre les applications en utilisant des protocoles communs tels que HTTP, FTP, SMTP et XMPP ;
- WSDL : l'interface publique au service Web est décrite (en XML) par ce protocole ;
- UDDI : permet à des applications de rechercher le service web dont elles ont besoin.

## SOAP - Simple Object Access Protocol

Protocole permettant d'appeler les méthodes d'un objet distant, bati sur XML.

Le transfert se fait le plus souvent à l'aide du protocole HTTP.

Le protocole SOAP est composé de deux parties :

- une enveloppe, contenant des informations sur le message lui-même (pour acheminement et traitement),
- un modèle de données, définissant le format du message (informations à transmettre).

SOAP est une recommendation du W3C, initialement défini par Microsoft et IBM.

## WSDL - Web Services Description Language

Langage décrivant une interface publique d'accès à un Service Web.

C'est une description basée sur le XML qui indique comment communiquer pour utiliser le service, en précisant :

- le protcole de communication ;
- le format de message requis.

## UDDI

Technologie d'annuaire basée sur XML et plus particulièrement destinée aux services web. Un annuaire UDDI permet de localiser sur le réseau le service Web recherché. Il repose sur le protocole de transport SOAP.

L'annuaire UDDI est consultable de différentes manières :

- les pages blanches comprennent la liste des entreprises ainsi que des informations associées à ces dernières ;
- 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.

# Avantages des services Web

- les services Web fournissent l'interopérabilité entre divers logiciels fonctionnant sur diverses plateformes ;
- les services Web utilisent des standards et protocoles ouverts ;
- les protocoles et les formats de données sont au format texte dans la mesure du possible, facilitant ainsi la compréhension du fonctionnement global des échanges ;
- basés sur le protocole HTTP, les services Web peuvent fonctionner au travers de nombreux firewalls sans necessiter des changements sur les règles de filtrage.

# Inconvénients des services Web

- les normes de services Web dans les domaines de la sécurité et des transactions sont actuellement inexistantes ou très limitées comparées à des normes ouvertes plus mà»res de l'informatique répartie telles que CORBA ;
- les services Web souffrent de performances faibles comparée à d'autres approches de l'informatique répartie telles que le RMI, CORBA, ou DCOM ;
- par l'utilisation du protocole HTTP, les services Web peuvent contourner les mesures de sécurité mises en place au travers des firewalls ;
- rien ne permet pour l'instant d'assurer la qualité d'execution d'un Service Web. Il n'y a donc aucune qualité de service associée à ces derniers.

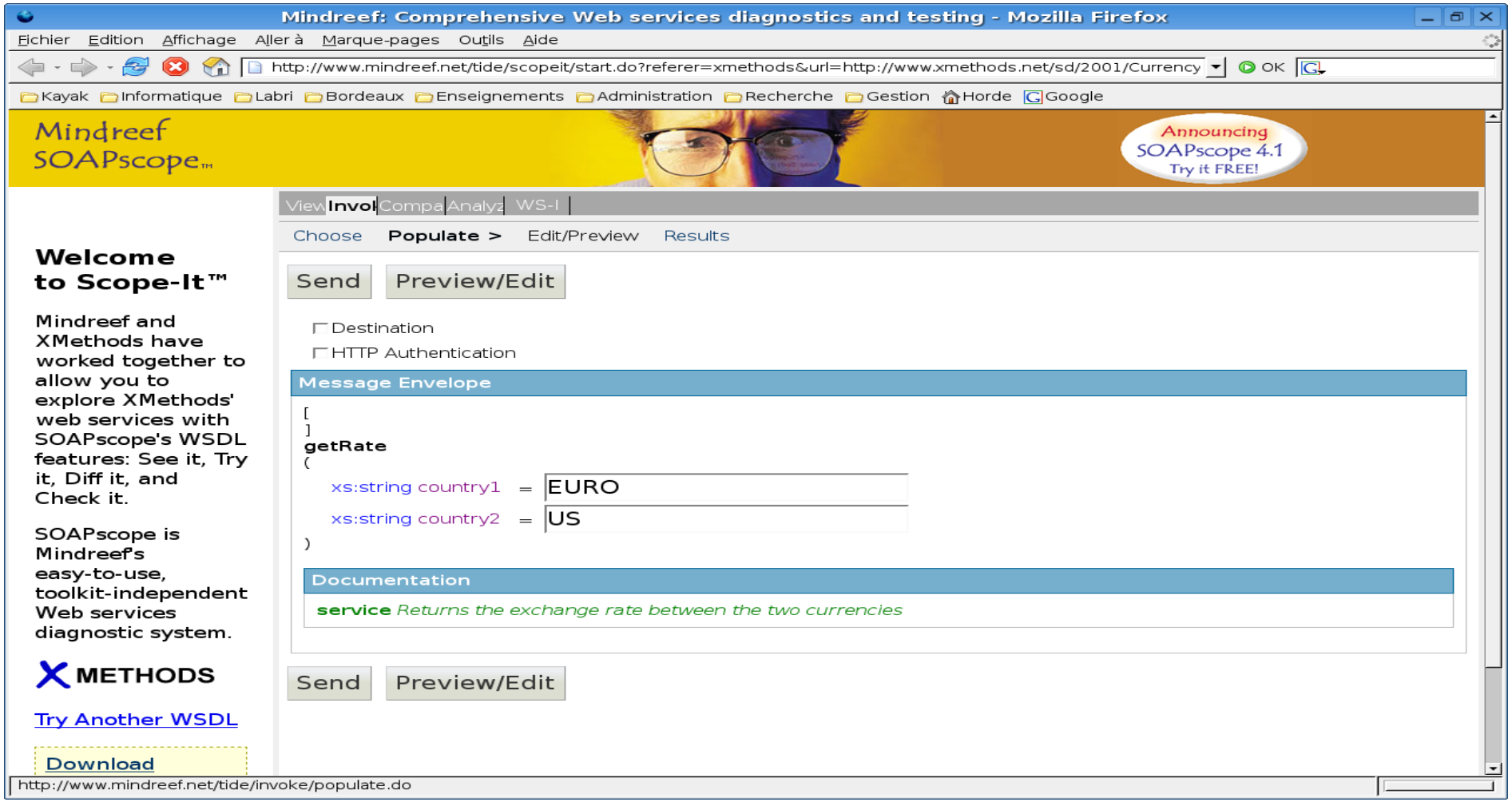
# Cours 5
## SOAP

# Plan

# Expérimentations : convertisseur Euro/Dollar

http ://www.xmethods.com/

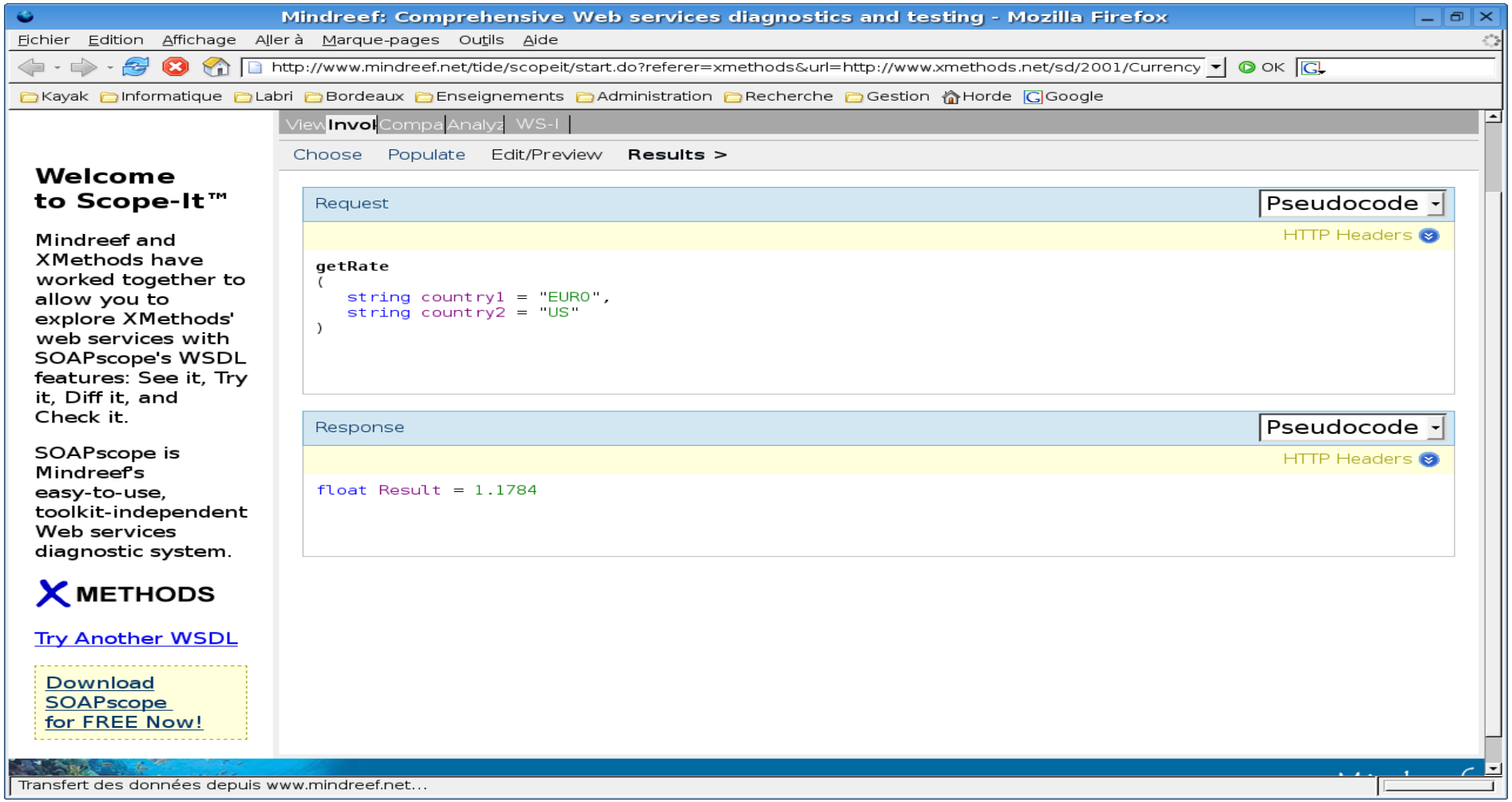
## Test de l'application en ligne : conversion de 1 euro en dollars



# Expérimentations : convertisseur (2)

http ://www.xmethods.com/

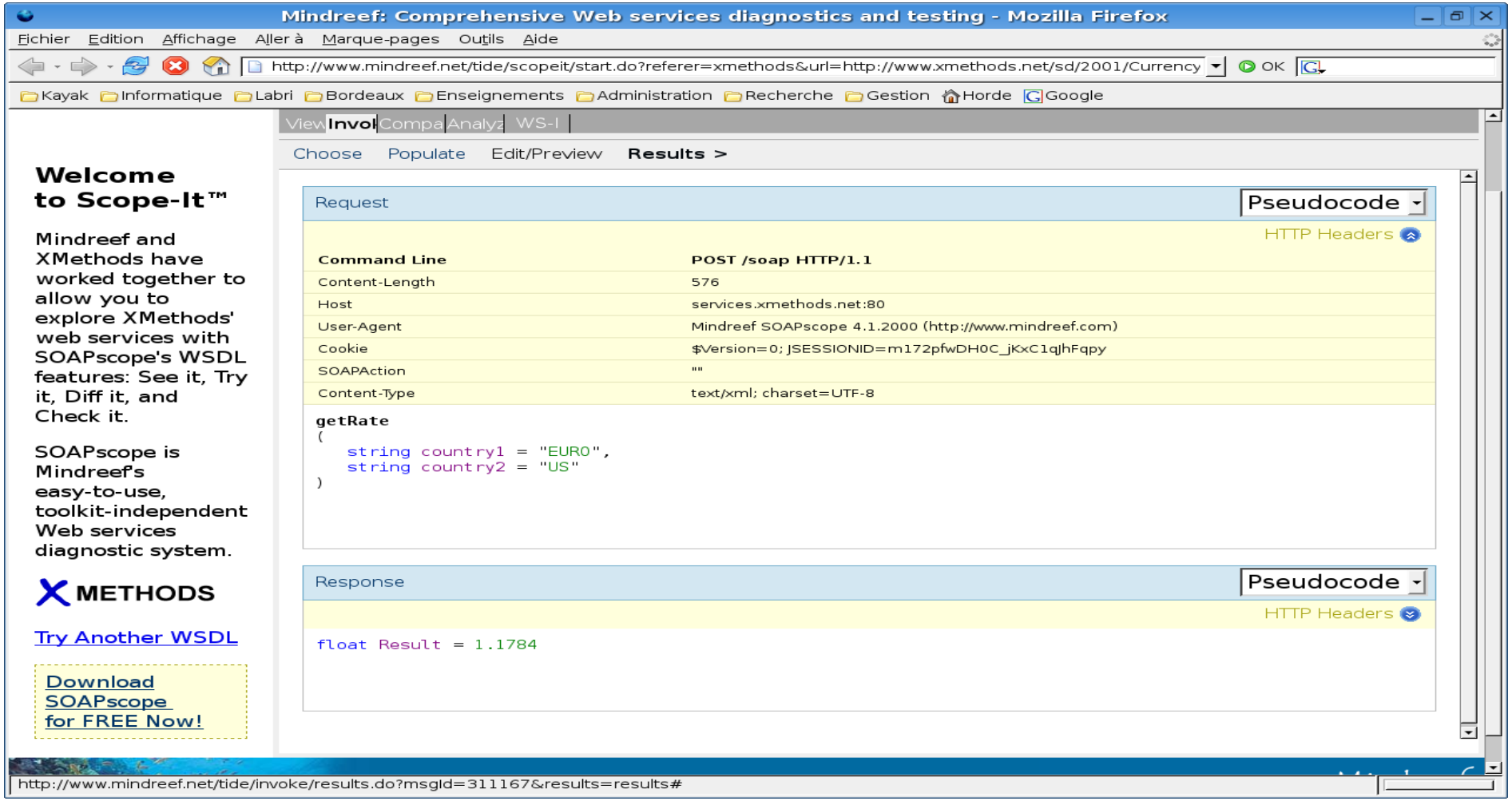
## Test de l'application en ligne : le résultat



# Expérimentations : convertisseur (3)

## Analyse de l'envoi

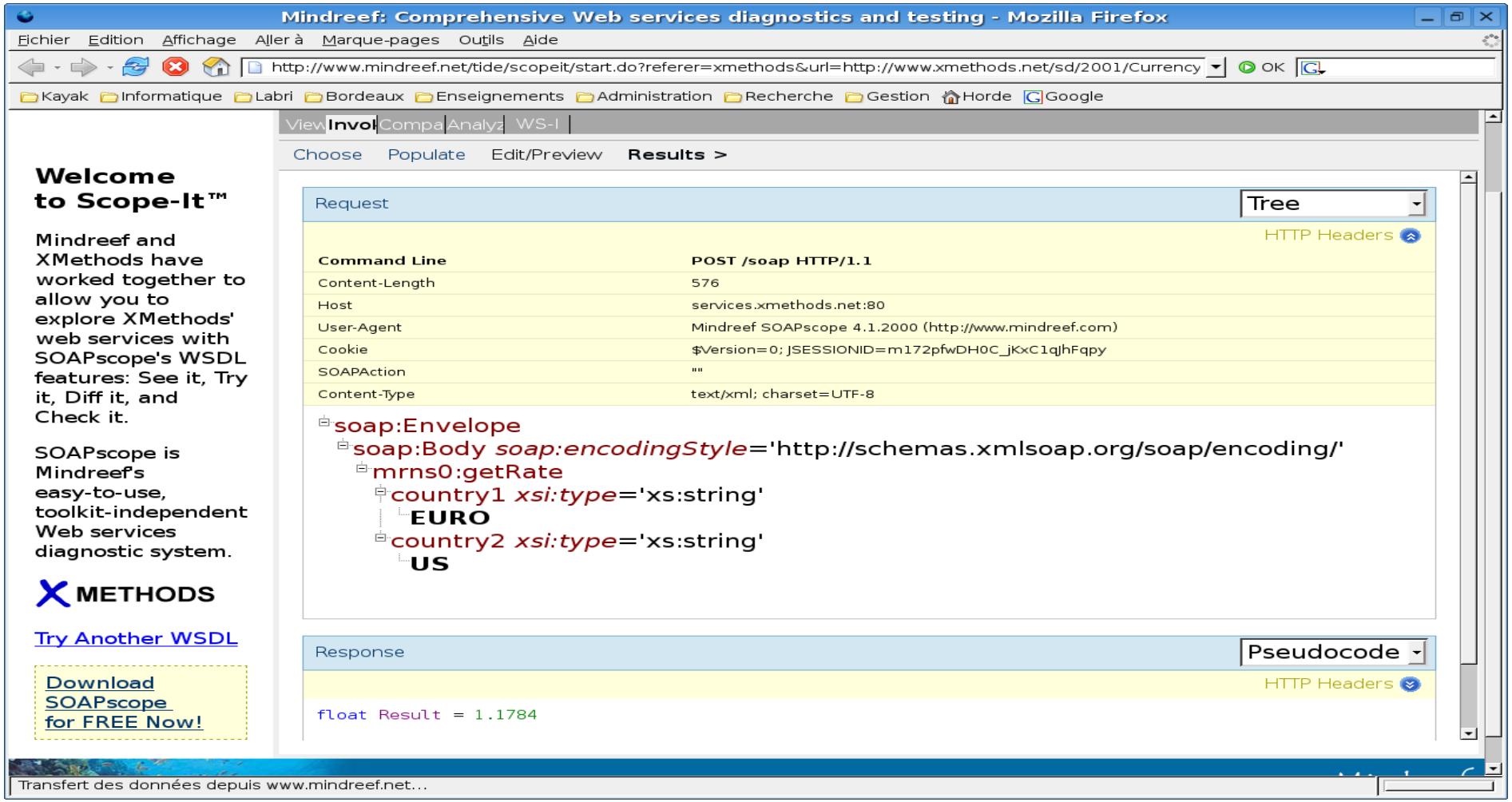
### Entêtes HTTP de l'envoi



# Expérimentations : convertisseur (4)

## Analyse de l'envoi

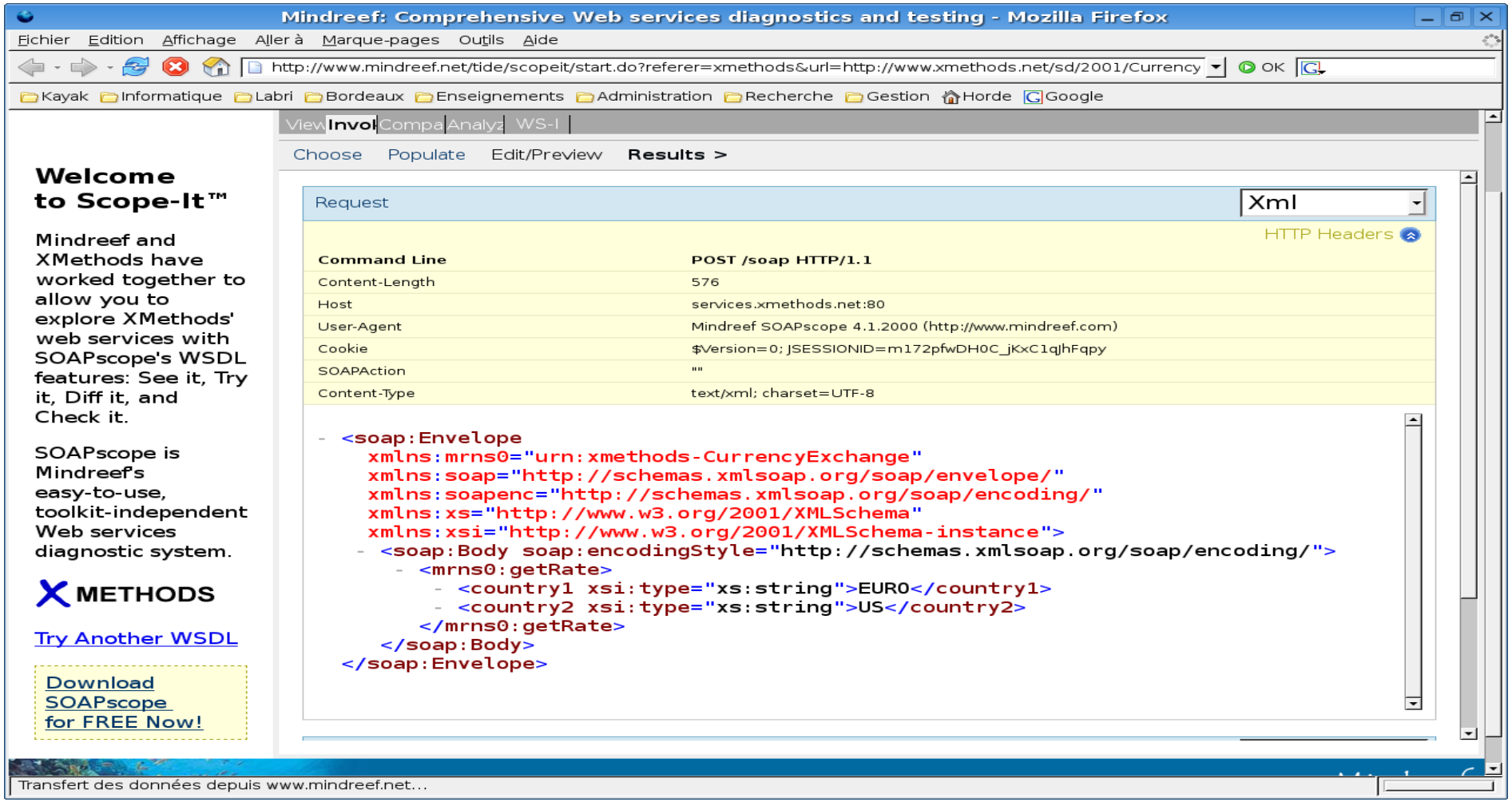
### Données XML envoyées par HTTP :



# Expérimentations : convertisseur (5)

## Analyse de l'envoi

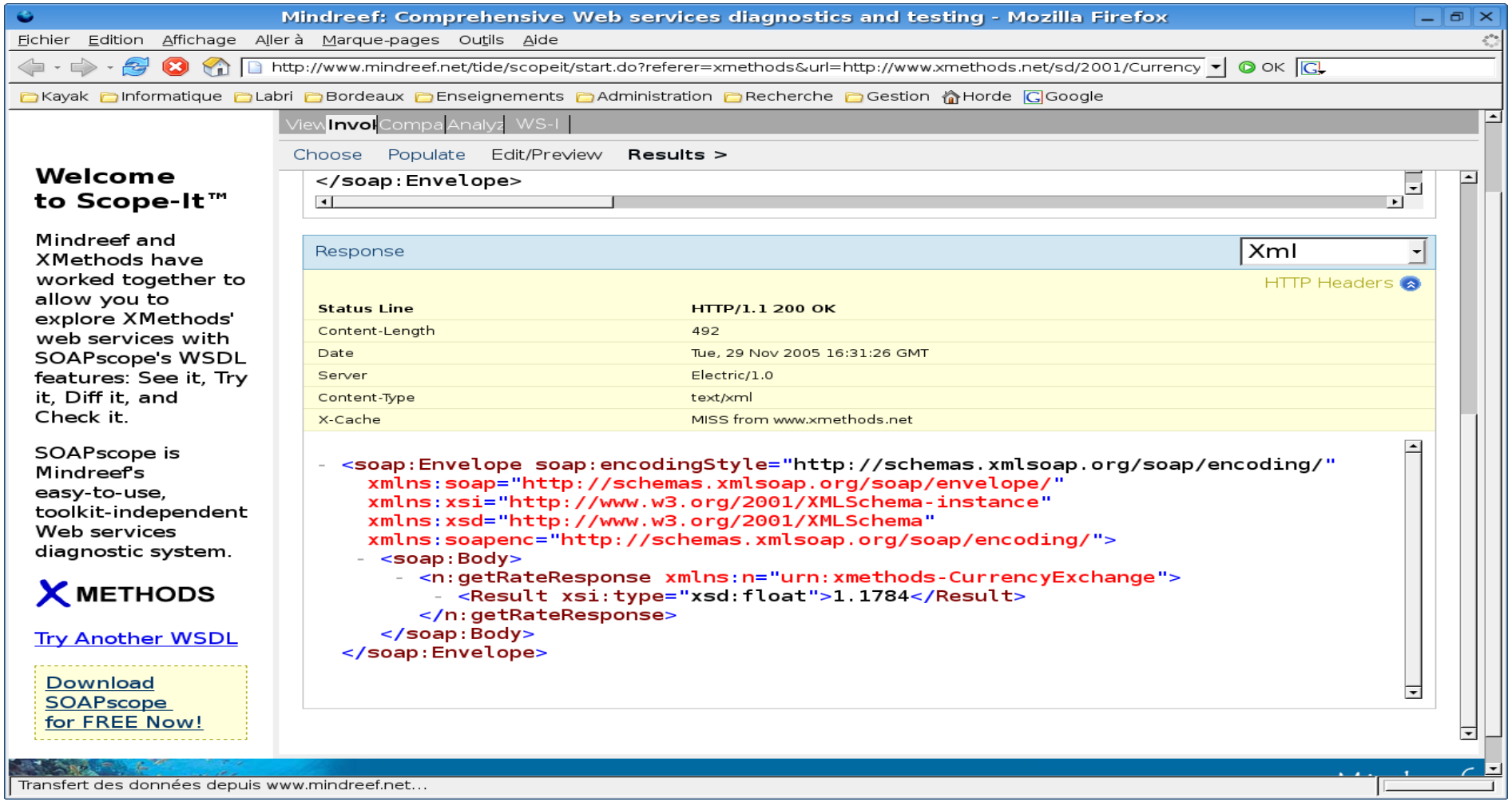
### Code des données XML envoyées par HTTP :



# Expérimentations : convertisseur (6)

## Analyse de la réponse

### Code des données XML retournées par HTTP :



# Site de test : http ://www.webservicex.net/

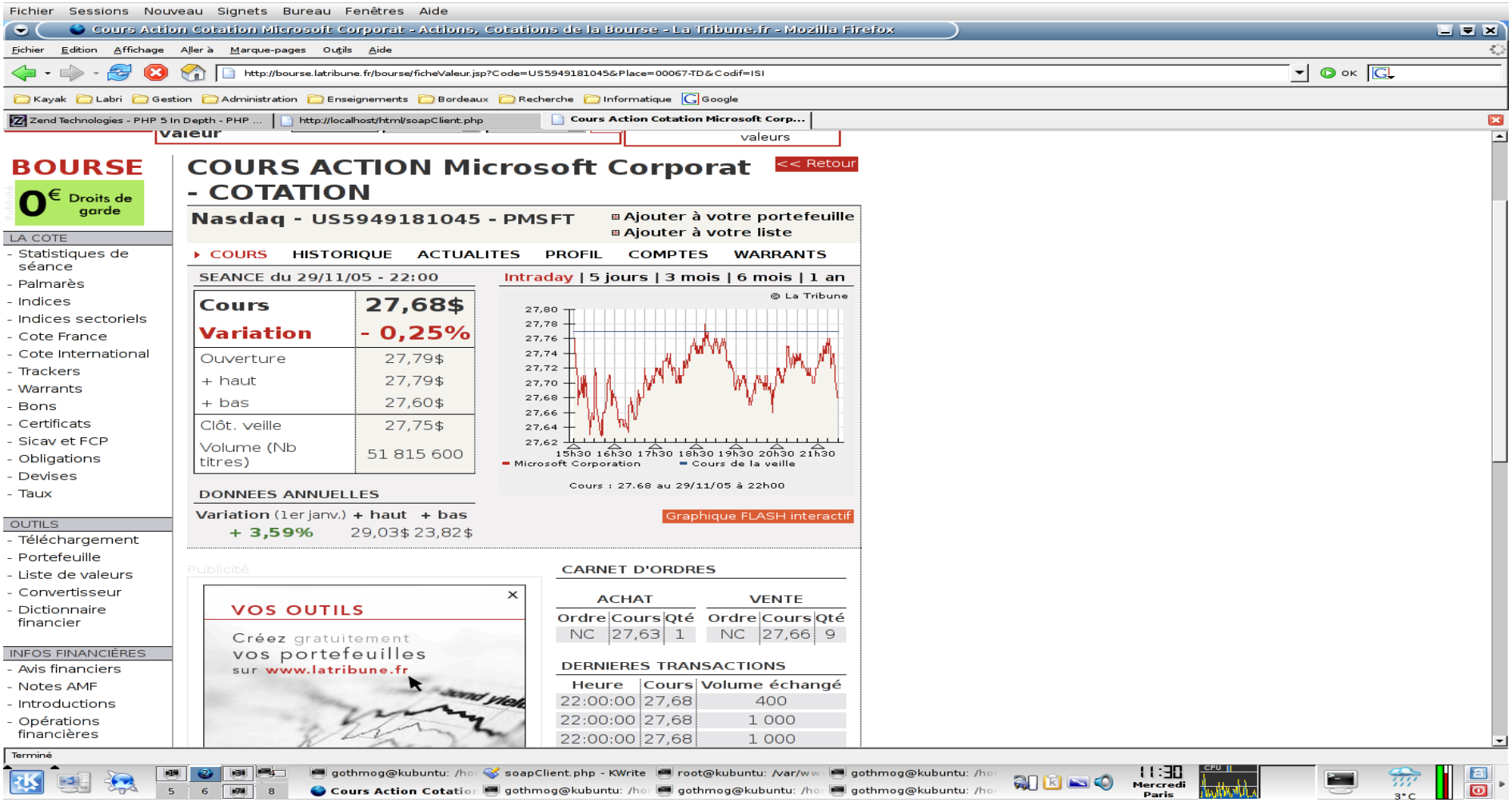
## Exercice

1. se connecter sur `www.webservicex.net`
2. tester quelques services :
   - Stock Quote ;
   - Currency Convertor ;
   - Global Weather ;
   - Translation Engine ; (Bonjour Monde !)
   - Country Details ;
   - SendSMSWorld ;
   - Validate Email Address ;
   - SendFax ;
   - GeoIPService

# Plan

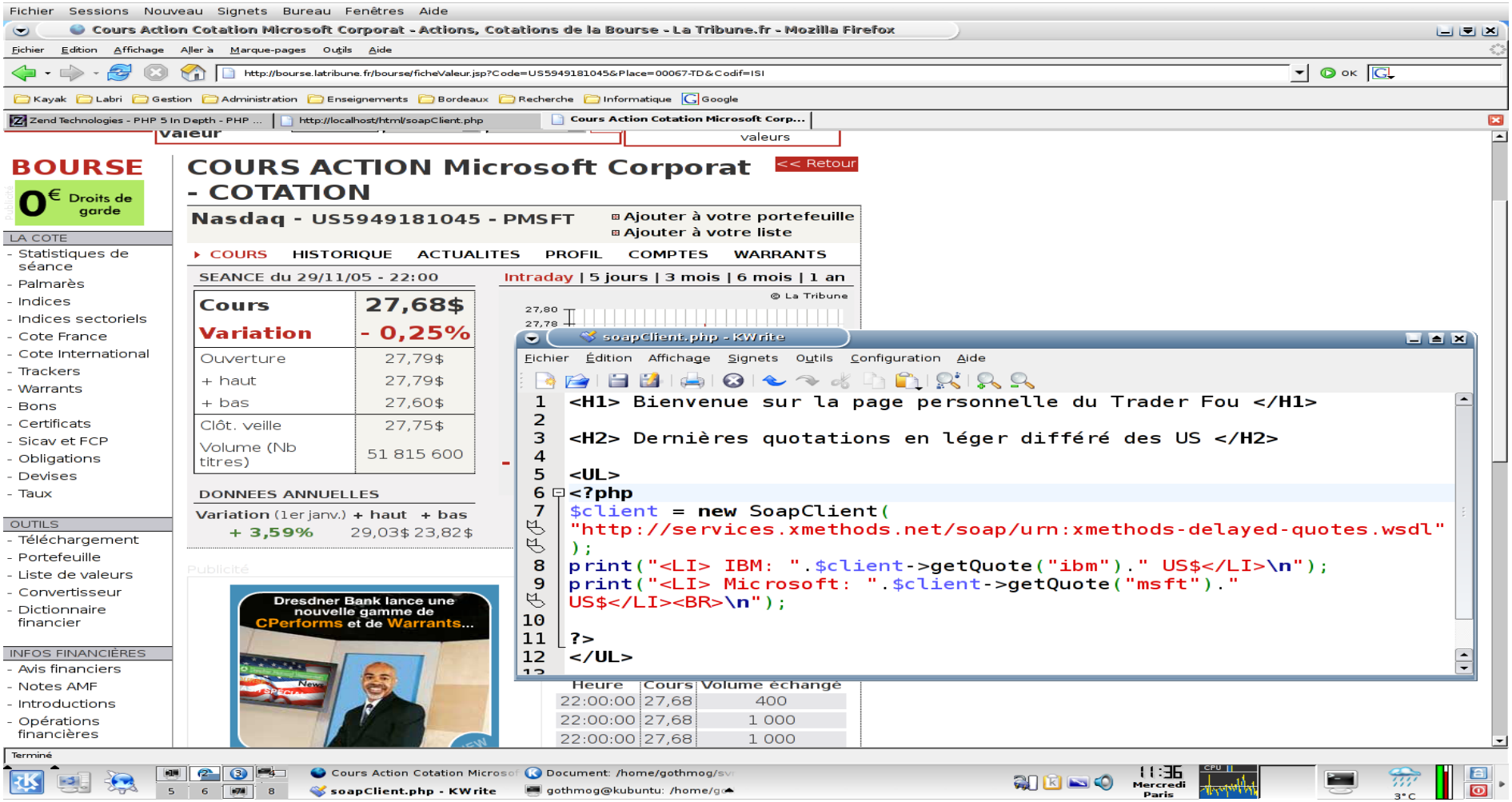
# Exemple : cours de Bourse US

## Cours de Microsoft



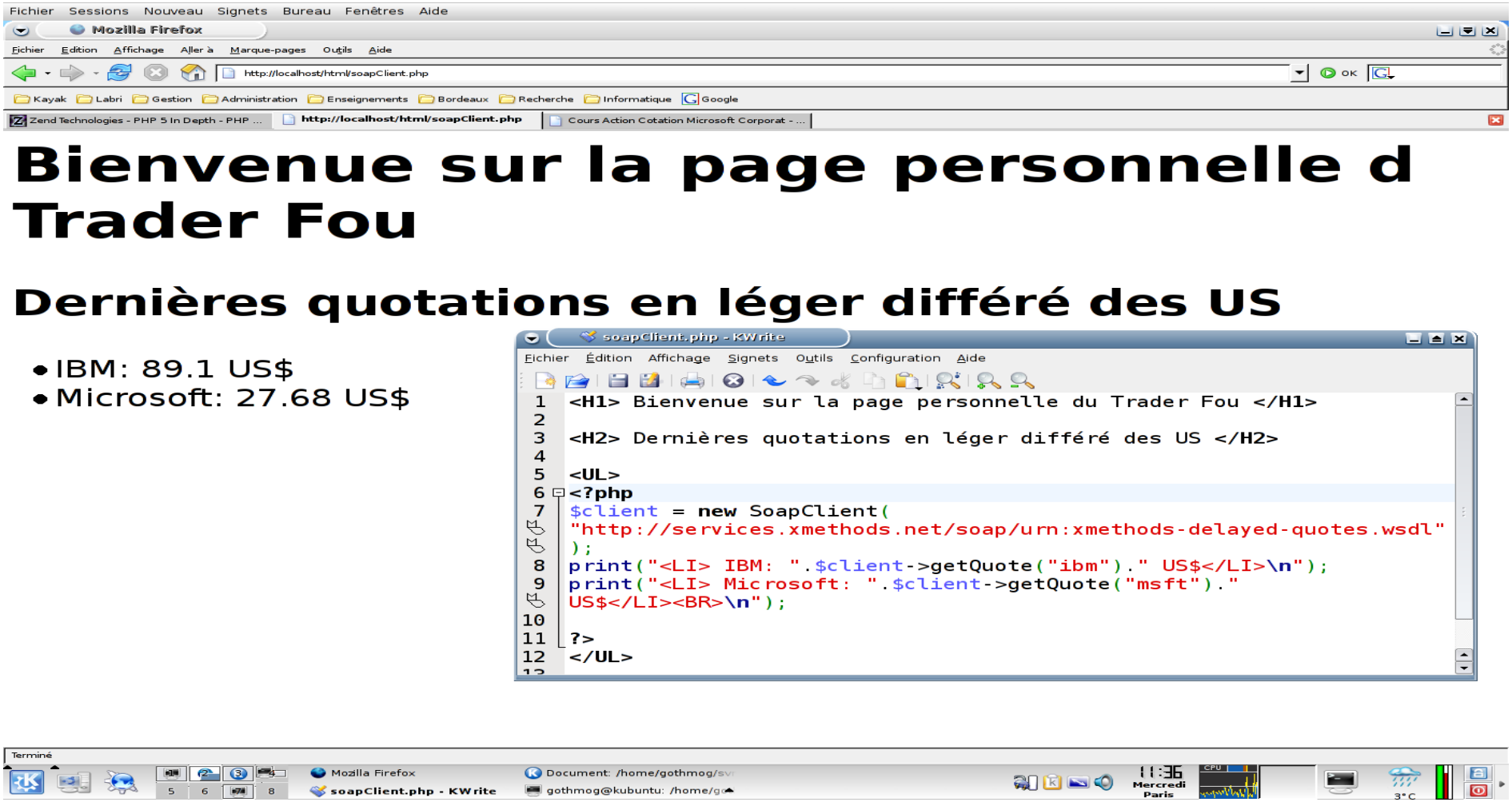
# Exemple : cours de Bourse US (2)

## Code client en PHP



# Exemple : cours de Bourse US (3)

## Notre page "trading" en PHP



# Client basique en PHP5

## Exemple de code pour récupérer un cours de bourse

```
<?
\$params['symbol']="ibm";
\$client =
new SoapClient("http://www.webservicex.net/stockquote.asmx?wsdl");
\$result = \$client->GetQuote(\$params);
\$quoteResult = \$result->GetQuoteResult; // resultat sous forme xml
\$xml = simplexml_load_string(\$quoteResult); // pour analyser le
    resultat
\$stockPrice = 0.0;
// <Stock>...<Last>56.34</Last>...</Stock>
if (property_exists(\$xml, "Stock") == true) { // balise Stock presente
        \$stockInfo = \$xml->Stock;
 if (property_exists(\$stockInfo, "Last")) \$stockPrice = \$stockInfo->
     Last;
}
echo "Resultat: ".\$stockPrice."<br>";
?>
```

## Exercice

A vous de jouer : modifier ce code pour ajouter la conversion en euros avec le cours exact

# Cours des actions en euros : traces

```
// Requete SOAP:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
    envelope/"
    xmlns:ns1="http://www.webserviceX.NET/">
<SOAP-ENV:Body>
<ns1:ConversionRate><ns1:FromCurrency>EUR</ns1:FromCurrency>
<ns1:ToCurrency>USD</ns1:ToCurrency></ns1:ConversionRate>
</SOAP-ENV:Body></SOAP-ENV:Envelope>


// Reponse SOAP:
<?xml version="1.0" encoding="utf-8"?><soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
        http://www.w3.org/2001/XMLSchema">
<soap:Body>
<ConversionRateResponse xmlns="http://www.webserviceX.NET/">
<ConversionRateResult>1.4182</ConversionRateResult>
</ConversionRateResponse></soap:Body></soap:Envelope>
```

# Cours des actions en euros : solution

```
<? echo "<h1> Ma page Trading </h1>\n";
\$params['symbol']="ibm";
\$client = new SoapClient("http://www.webservicex.net/stockquote.asmx?
    wsdl");
\$result = \$client->GetQuote(\$params);
\$quoteResult = \$result->GetQuoteResult;
// on a le resultat sous forme d'un document xml
\$xml = simplexml_load_string(\$quoteResult); // pour analyser le
    resultat
\$stockPrice = 0.0;
// <Stock>...<Last>56.34</Last>...</Stock>
if (property_exists(\$xml, "Stock") == true) { // balise Stock presente
 \$stockInfo = \$xml->Stock;
 if (property_exists(\$stockInfo, "Last")) \$stockPrice = \$stockInfo->
     Last;
}
\$params2['FromCurrency']="EUR"; \$params2['ToCurrency']="USD";
\$client = new SoapClient("http://www.webservicex.net/CurrencyConvertor
    .asmx?wsdl");
\$result = \$client->ConversionRate(\$params2);
\$convertResult = \$result->ConversionRateResult;
echo "Cours de l'action ".\$params['symbol']." : ".\$stockPrice." US\$<
    br>";
echo "Taux de conversion : 1 euro = ".\$convertResult." US\$<br>";
```

# Cours des actions en euros : ancienne sol.

```
<HR>
<H2> Dernières quotations en léger différé des US </H2>

<?php

function taux(){
$clientCurrencyExchange = new SoapClient(
"http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl" );
return ($clientCurrencyExchange->getRate("EURO","US"));
}

$clientDelayedQuote = new SoapClient(
"http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl");
$ibm = $clientDelayedQuote->getQuote("ibm");
$msft = $clientDelayedQuote->getQuote("msft");

$r = taux();

print("<UL><LI> IBM: $ibm $ = ".($ibm/$r)." euros</LI>\n");
print("<LI> Microsoft: $msft $ = ".($msft/$r)." euros </LI></UL>\n");

print("<CENTER><I>Taux de conversion actuel: 1 euro = $r $</I></CENTER>\n");
?>
```

---

## Bienvenue sur la page personnelle du Trader Fou

---

### Dernières quotations en léger différé des US

- IBM: 90.48 $ = 71.7355109807 euros
- Microsoft: 28.43 $ = 22.5402362642 euros

*Taux de conversion actuel: 1 euro = 1.2613 $*

# Les principales classes PHP5 de l'extension SOAP

- soapClient : permet de construire un client SOAP qui permettra d'interroger un service web ;
- soapFault : sert à  renvoyer une erreur SOAP , elle ne dispose pas de méthode hormis son constructeur ;
- soapHeader : permet d'envoyer ou de récupérer des en-têtes SOAP , selon le contexte où elle est appellée (client/serveur). Elle ne dispose pas de méthode hormis son constructeur ;
- soapParam : sert à  définir des paramètres en mode non WSDL. Elle ne dispose pas de méthode hormis son constructeur ;
- soapServer : permet la création de serveur SOAP en mode WSDL ou non ;
- soapVar : gère l'encodage des paramètres en mode non WSDL.

# soapClient

Cette classe permet de construire un client SOAP qui permettra d'interroger un service web.

Méthodes :

- __construct() : Constructeur de la classe ;
- __call() : Appelle une fonction SOAP ;
- __getFunctions() : Liste les fonctions SOAP disponibles en mode WSDL ;
- __getLastRequest() : Retourne la dernière requête SOAP ;
- __getLastResponse() : Retourne la dernière réponse SOAP ;
- __getTypes() : Retourne la liste des Types SOAP.

# Analyse des échanges SOAP

## Récupération des traces SOAP

```
<?php
      \$client = new SoapClient("stockquote.wsdl",array(
      "trace"       => 1,
"exceptions" => 0));
\$client->getQuote("ibm");
print "<pre>\n";
print "Request :\n".htmlspecialchars(\$client->__getLastRequest()) ."\n
    ";
print "Response:\n".htmlspecialchars(\$client->__getLastResponse()).."\n
    ";
print "</pre>";
?>
```

# Trace de la requête

## Requête SOAP

```
Request :
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns1="urn:xmethods-delayed-quotes"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns1:getQuote>
<symbol xsi:type="xsd:string">ibm</symbol>
</ns1:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# Trace de la réponse

## Réponse SOAP

```
Response:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns1="urn:xmethods-delayed-quotes"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns1:getQuoteResponse>
<Result xsi:type="xsd:float">98.42</Result>
</ns1:getQuoteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# Plan

# Cours des actions en euros

## Trame http de la requête

```
▷ Frame 43 (755 bytes on wire, 755 bytes captured)
▷ Ethernet II, Src: Intel_41:26:9b (00:04:23:41:26:9b), Dst: Netgear_9a:c4:94 (00:09:5b:9a:c4:94)
▷ Internet Protocol, Src: 192.168.0.2 (192.168.0.2), Dst: 64.124.140.30 (64.124.140.30)
▷ Transmission Control Protocol, Src Port: 2393 (2393), Dst Port: www (80), Seq: 1, Ack: 1, Len: 689
▽ Hypertext Transfer Protocol
  ▷ POST /soap HTTP/1.1\r\n
    Host: services.xmethods.net\r\n
    Connection: Keep-Alive\r\n
    User-Agent: PHP SOAP 0.1\r\n
    Content-Type: text/xml; charset=utf-8\r\n
    SOAPAction: ""\r\n
    Content-Length: 511\r\n
    \r\n
▽ eXtensible Markup Language
  ▽ <?xml
      version="1.0"
      encoding="UTF-8"
      ?>
  ▽ <SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ns1="urn:xmethods-CurrencyExchange"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    ▽ <SOAP-ENV:Body>
      ▽ <ns1:getRate>
        ▽ <country1>
            EURO
            </country1>
        ▽ <country2>
            US
            </country2>
          </ns1:getRate>
        </SOAP-ENV:Body>
      </SOAP-ENV:Envelope>
```

# Cours des actions en euros

## Trame http de la réponse

```
▷ Frame 47 (775 bytes on wire, 775 bytes captured)
▷ Ethernet II, Src: Netgear_9a:c4:94 (00:09:5b:9a:c4:94), Dst: Intel_41:26:9b (00:04:23:41:26:9b)
▷ Internet Protocol, Src: 64.124.140.30 (64.124.140.30), Dst: 192.168.0.2 (192.168.0.2)
▷ Transmission Control Protocol, Src Port: www (80), Dst Port: 2393 (2393), Seq: 1, Ack: 690, Len: 709
▽ Hypertext Transfer Protocol
  ▷ HTTP/1.1 200 OK\r\n
    Date: Sat, 21 Oct 2006 13:20:03 GMT\r\n
    Server: Electric/1.0\r\n
    Content-Type: text/xml\r\n
    Content-Length: 492\r\n
    X-Cache: MISS from www.xmethods.net\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    \r\n
▽ eXtensible Markup Language
  ▽ <?xml
      version='1.0'
      encoding='UTF-8'
      ?>
  ▽ <soap:Envelope
      xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
      xmlns:xsd='http://www.w3.org/2001/XMLSchema'
      xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
      soap:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
    ▽ <soap:Body>
      ▽ <n:getRateResponse
          xmlns:n='urn:xmethods-CurrencyExchange'>
        ▽ <Result
            xsi:type='xsd:float'>
            1.2613
            </Result>
          </n:getRateResponse>
        </soap:Body>
      </soap:Envelope>
```

# Client Java direct

## Service web taux de change

```
import java.net.*; import java.io.*; import java.util.*;

class SoapRawClient{

	public static void main( String[] args ) throws Exception{
		Socket s = new Socket("64.124.140.30",80);

		String requeteSoap = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\r\n<SOAP-ENV:Envelope xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:ns1=\"urn:xmethods-CurrencyExchange\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:SOAP-ENC=\"http://schemas.xmlsoap.org/soap/encoding/\"
SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\"><SOAP-ENV:Body><ns1:getRate><country1>EURO</country1><country2>US</country2></ns1
:getRate></SOAP-ENV:Body></SOAP-ENV:Envelope>";
		String enteteHTTP = "POST /soap HTTP/1.1\r\nHost: services.xmethod.net\r\nConnection: Keep-Alive\r\nUser-Agent: java raw\r\nContent-Type: text/xml;
charset utf-8\r\nSOAPAction: \"\"\r\nContent-Length:"+requeteSoap.length()+"\r\n";

//		System.out.println("Envoi de la requete:");
		PrintWriter pw = new PrintWriter( s.getOutputStream() );
		pw.println(enteteHTTP+"\r\n"+requeteSoap);
		pw.flush();

		LineNumberReader lnr = new LineNumberReader( new InputStreamReader(s.getInputStream()) );
		String ligne="";
		double val = 0;
		while ((ligne=lnr.readLine())!=null){
//			System.out.println(ligne);
			StringTokenizer st = new StringTokenizer(ligne,"<>");
			while (st.hasMoreTokens()){
				String mot = st.nextToken();
				if (mot.startsWith("1."))
					val = new Double(mot).doubleValue();
			}
		}
		System.out.println("1 euro vaut "+val+" dollars us");

	}

}
```

# Axis

## Axis

Implémentation de SOAP par la fondation Apache :
http ://ws.apache.org/axis/

Librairies : `axis.jar`, `axis-ant.jar`, `commons-discovery-0.2.jar`, `commons-logging-1.0.4.jar`, `jaxrpc.jar`, `log4j-1.2.8.jar`, `saaj.jar`, `wsdl4j-1.5.1.jar`

Etapes pour créer un client d'un service web :

1. Génération du code de base des classes : `java -cp * :. org.apache.axis.wsdl.WSDL2Java urlWSDL`
2. Ecrire une classe pour consommer le service - cf exemple ci-dessous

```
package NET.webserviceX.www;
public class StockQuoteClient{
public static void main( String[] args ) throws Exception{
StockQuoteLocator sql = new StockQuoteLocator();
StockQuoteSoap sqs = sql.getStockQuoteSoap();
String cours = sqs.getQuote("IBM");
System.out.println("Cours de l'action IBM: "+cours);
}}
```

# Utilitaire wsimport

Génération du code de base du client : wsimport urlWSDL

Exemple de classe pour consommer le service :

```
package net.webservicex;
public class StockQuoteClient{
public static void main( String[] args ){
StockQuote sq = new StockQuote();
StockQuoteSoap sqs = sq.getStockQuoteSoap();
String cours = sqs.getQuote("IBM");
System.out.println("Cours de l'action IBM:"+cours);
}}
```

# Plan

# SOAP : entête

## SOAP Header : mécanisme d'extension du protocole SOAP

- la balise Header est optionnelle ;
- si la balise Header est présente, elle doit être le premier fils de la balise Envelope ;
- la balise Header contient des entrées ;
- une entrée est n'importe quelle balise incluse dans un namespace

# SOAP : corps

## SOAP Body : le corps contient le message à échanger

- la balise Body est obligatoire ;
- la balise Body doit être le premier fils de la balise Envelope (ou le deuxième si il existe une balise Header) ;
- la balise Body contient des entrées ;
- une entrée est n'importe quelle balise incluse optionnellement dans un namespace
- une entrée peut être une Fault.

# SOAP : erreur

SOAP Fault : balise permettant de signaler des cas d'erreur.

- La balise Fault contient les balises suivantes :
  - faultcode : un code permettant d'identifier le type d'erreur (Client, Server, VersionMismatch, MustUnderstand) ;
  - faultstring : une explication en langage naturel ;
  - faultactor : une information identifier l'initiateur de l'erreur ;
  - detail : définition précise de l'erreur.

# Encodage

Un message SOAP contient des données typées. Il faut donc définir un moyen d'encoder ces données.

Vocabulaire SOAP :

- Value (valeur d'une donnée)
  - Simple value (string, integers,etc) ;
  - Compound value (array, struct, ...) ;
- Type (d'une value) ;
  - Simple Type ;
  - Compound Type.

*Bibliographie* :

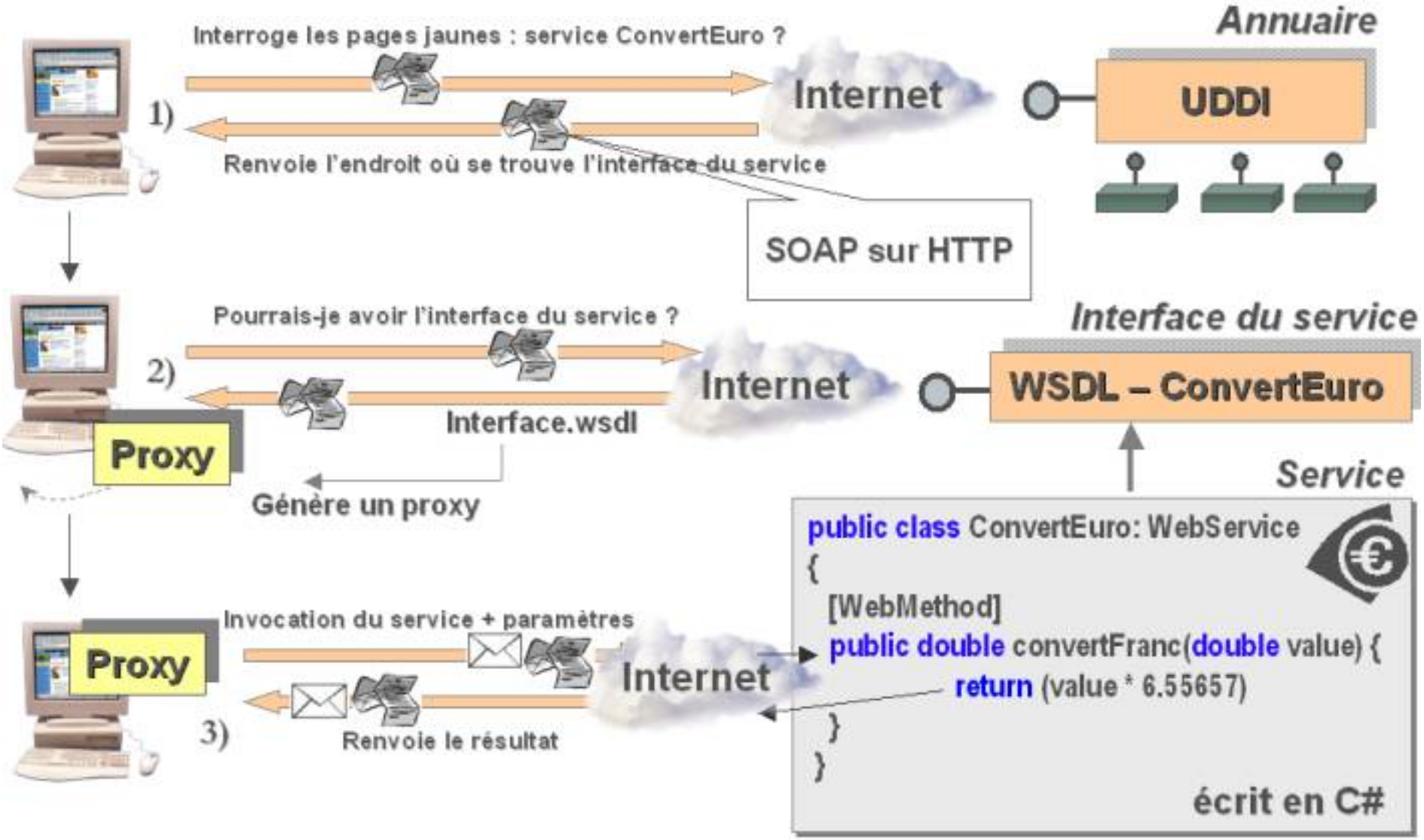
- J.-M. Chauvet, "Services Web avec SOAP, WSDL, UDDI, ebXML ...", Eyrolles
- K. Topley, "Java Web Services in a nutshell", O'Reilly

*Hyperliens*

- S. Jaber, `http ://www.dotnetguru.org/articles/webservices/WebServices.htm`
- `http ://www.zend.com/php5/articles/php5-SOAP.php`
- F. Rossi, `http ://apiacoa.org/teaching/webservices/`
- XML Schema : `http ://xmlfr.org/w3c/TR/xmlschema-0/`

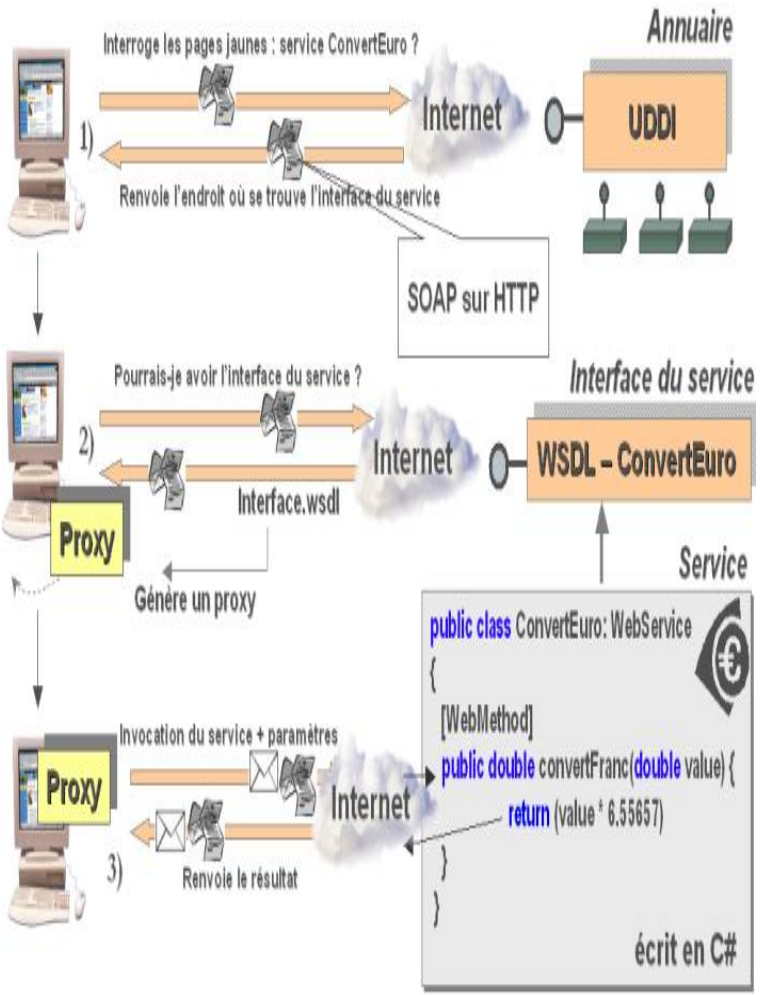
# Plan

# Scénario typique



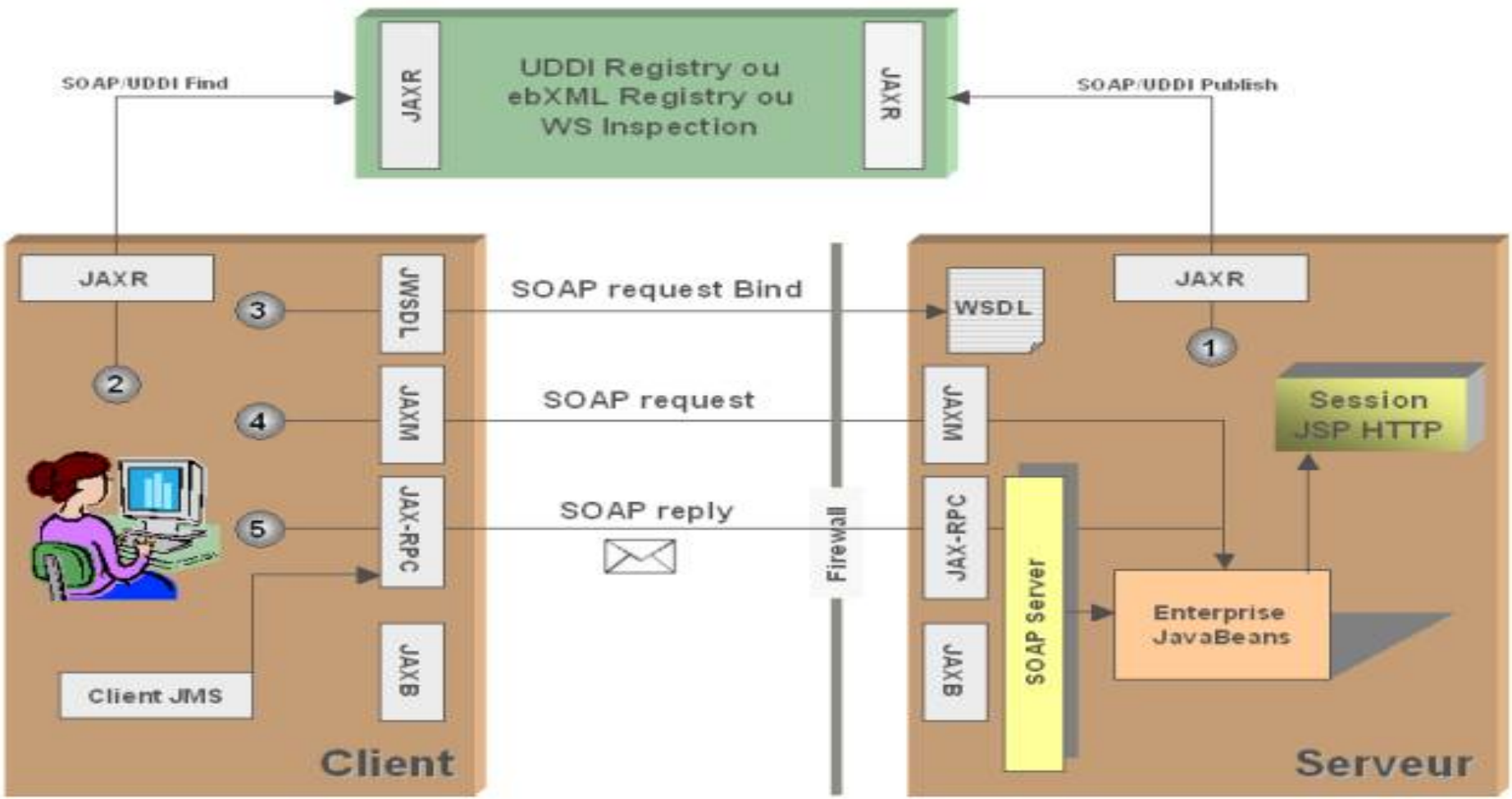

Source : `http ://www.dotnetguru.org/`

# Scénario



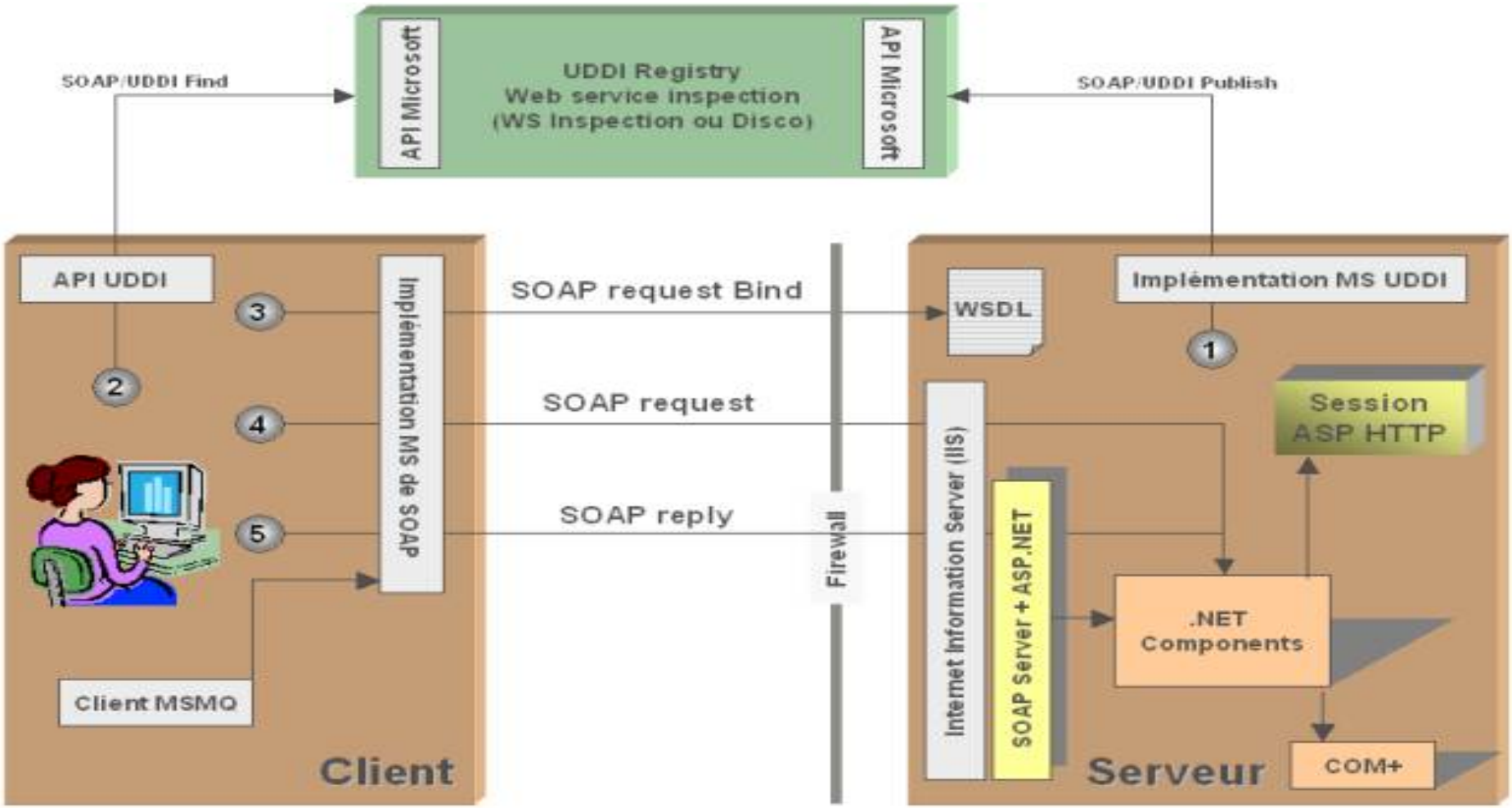

1. Le client demande un service : une recherche sémantique dans un annuaire UDDI donne la liste des prestataires potentiels ;
2. Une fois la réponse reçue (en XML), le client recherche l'interface du composant référencé dans l'annuaire. Cette interface est spécifiée en WSDL et décrit l'ensemble des services implémentés par l'objet distant ;
3. Il ne reste plus qu'à effectuer l'invocation du service. Pour ce faire, il faut fournir les paramètres attendus par l'interface et assurer la communication avec le serveur.

# L'architecture WebServices J2EE




Source : `http ://www.dotnetguru.org/`

# L'architecture WebServices .NET




Source : `http ://www.dotnetguru.org/`

# Plan

# Document WSDL

## Definitions

### Message

...

### Binding

<Operation> <input> ... </input> ...
</Operation>
<Operation> ... </Operation>
...

### PortType

<Operation> <input message> <output message> </Operation>
<Operation> ... </Operation>
...

### Service

<Port binding="..."> ... </Port>
<Port binding="..."> ... </Port>
...

# Signification des principales balises

Un document WSDL est organisé de la faà§on suivante :

- l'espace de noms principal est http ://schemas.xmlsoap.org/wsdl/ ;
- la racine du document est definitions ;
- les fils directs de la racine sont (dans l'ordre) :
  - types (facultatif) : système de types applicables aux données ;
  - message (nombre quelconque) : abstraction décrivant les données échangées ;
  - portType (nombre quelconque) : ensemble d'opérations implémenté par une terminaison ;
  - binding (nombre quelconque) : protocole concret d'accès à un port (terminaison identifiée de manière unique par une adresse Internet et une liaison), et format de spécification des messages et données pour ce port ;
  - service (nombre quelconque) : collection de ports ;

# Structure générale

- partie abstraite :
  - les types ;
  - les messages ;
  - les types de port.
- partie concrète :
  - les bindings ;
  - les services ;

La partie concrète propose une ou plusieurs réalisations de la partie abstraite (par exemple, un type de port peut être réalisé par SOAP+HTTP et/ou par SOAP+SMTP)

*Source : F. Rossi*

# Constructions générales

- les 'objets' définis en WSDL peuvent être placés dans un espace de noms, grâce à l'attribut `targetNamespace` de `wsdl :definitions` ;
- les références internes sont gérées grâce à l'introduction d'un préfixe pour cet espace de noms (en général tns) ;
- la construction `wsdl :import` permet de découper le document en plusieurs parties :
  - namespace : espace de noms dans lequel sont placés les 'objets' définis par le fichier inclus ;
  - location : emplacement du fichier inclus (URI) ;

*Source : F. Rossi*

# Balise Definitions

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="urn:webservice"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
name="WebServices" targetNamespace="urn:webservice">
```

# Définir les types

Les types WSDL sont décrits, via XML Schema, le modèle de métadonnées recommandé par W3C. La définition de types est inutile si on se contente des types de base.

Les types ainsi définis sont utilisés pour qualifier les arguments et les résultats des opérations des services webs.

Exemple

```
<types>
   <schema targetNamespace="http://example.com/titres.xsd"
           xmlns="http://www.w3.org/1999/XMLSchema">
   <element name="RequeteValeurCours">
           <complexType>
                <all><element name="Symbole" type="String"></all>
           </complexType>
   </element>
```

Le type "RequeteValeurCours" est une liste de chaînes de caractères.

# Définir les messages

- les messages sont spécifiés sans référence au protocole de transport ;

---

```
<message name="getNomRequest"><part name="id" type="xsd:Integer"/></message>
<message name="getNomResponse"><part name="return" type="xsd:String"/></message>
<message name="getPrenomRequest"><part name="id" type="xsd:Integer"/></message>
<message name="getPrenomResponse"><part name="return" type="xsd:String"/></message>
```

- 4 message sont définis :
  - getNomRequest de type Integer ;
  - getPrenomRequest de type Integer ;
  - getNomResponse de type String ;
  - getPrenomResponse de type String ;
- les messages sont de type élémentaire (i.e. ne sont pas d'un type défini dans le document WSDL)

# Définir les messages

- les messages sont spécifiés sans référence au protocole de transport ;

---

```
<message name="getNomRequest"><part name="id" type="xsd:Integer"/></message>
<message name="getNomResponse"><part name="return" type="xsd:String"/></message>
<message name="getPrenomRequest"><part name="id" type="xsd:Integer"/></message>
<message name="getPrenomResponse"><part name="return" type="xsd:String"/></message>
```

- les messages échangés par les services sont décrits par des éléments wsdl :message ;
- chaque message possède un nom (attribut name) et est constitué de parties ;
- chaque partie est décrite par un fils wsdl :part précisé par :
  - son nom (attribut name) ;
  - soit son type (attribut type) ;
  - soit directement le nom de l'élément qui la constitue (attribut element).

# Balises PortType/Operation

```
<portType name="EtudiantPortType">
<operation name="getNom"><input message="tns:getNomRequest"/><output message="tns:getNomResponse"/></operation>
<operation name="getPrenom"><input message="tns:getPrenomRequest"/><output message="tns:getPrenomResponse"/></operation>
</portType>
```

- un type de port ressemble à une interface Java ;
- les types de port sont décrits par des éléments wsdl :portType ;
- chaque type de port est identifié par un nom (attribut name) ;
- un type de port décrit un ensemble d'opérations, chacune précisée par un élément wsdl :operation, identifiée par un nom (attribut name) et contenant une spécification des messages échangés pour réaliser l'opération ;
- quatre modèles pour les opérations (basés sur le contenu de l'élément wsdl :operation) : envoi de message (One way), question - réponse (Request-response), sollicitation - réponse (Solicit-response), alerte (Notification)

*Source : F. Rossi*

# Opération

```
<portType name="EtudiantPortType">
<operation name="getNom"><input message="tns:getNomRequest"/><output message="tns:getNomResponse"/></operation>
<operation name="getPrenom"><input message="tns:getPrenomRequest"/><output message="tns:getPrenomResponse"/></operation>
</portType>
```

Trois sous-éléments possibles pour une opération :

- wsdl :input : message reà§u par le service ;
- wsdl :output : message produit par le service ;
- wsdl :fault : message d'erreur (produit par le service) ;

Chaque sous-élément est précisé par des attributs :

- name : donne un nom (au niveau de l'opération) au message (facultatif) ;
- message : type du message, référence à un message défini par un élément wsdl :message.

*Source : F. Rossi*

# Modèles d'opération

```
<portType name="EtudiantPortType">
<operation name="getNom"><input message="tns:getNomRequest"/><output message="tns:getNomResponse"/></operation>
<operation name="getPrenom"><input message="tns:getPrenomRequest"/><output message="tns:getPrenomResponse"/></operation>
</portType>
```

Les deux modèles les plus utilisés :

- envoi de message (approche message) : le client envoie un message et n'attend pas de réponse du service - un seul message wsdl :input ;
- question - réponse (approche RPC ou échange de documents) : le client envoie un message auquel le service répond - un wsdl :input, suivi d'un wsdl :output et d'éventuels wsdl :fault.

*Source : F. Rossi*

# Modèles avec dialogues inversés

```
<portType name="EtudiantPortType">
<operation name="getNom"><input message="tns:getNomRequest"/><output message="tns:getNomResponse"/></operation>
<operation name="getPrenom"><input message="tns:getPrenomRequest"/><output message="tns:getPrenomResponse"/></operation>
</portType>
```

Modèles moins utilisés :

- sollicitation - réponse (approche RPC inversée) : le client reà§oit un message du service et répond au service un wsdl :output, suivi d'un wsdl :input et d'éventuels wsdl :fault ;
- alerte : le client reà§oit un message du service mais ne doit pas répondre.

*Source : F. Rossi*

# Balise Binding (liaison)

```
<binding name="EtudiantBinding" type="tns:EtudiantPortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getNom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
<operation name="getPrenom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
</binding>
```

- une liaison propose une réalisation concrète d'un type de port ;
- élément racine wsdl :binding ;
- précisé par un attribut name (nom du binding) et par un attribut type (type de port concerné par le binding) ;
- contient un élément wsdl :operation pour chaque opération du type de port (attribut name pour indiquer l'opération concernée) ;
- chaque élément operation contient des éléments définissant la liaison des messages associés : wsdl :input, wsdl :output, wsdl :fault.

*Source : F. Rossi*

# Liaison SOAP (1)

```
<binding name="EtudiantBinding" type="tns:EtudiantPortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getNom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
<operation name="getPrenom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
</binding>
```

- WSDL définit d'abord une coquille vide pour le binding ;
- un binding se fait grâce à  des éléments additionnels ;
- la norme spécifie les éléments d'un binding vers SOAP :
  - espace de noms : http ://schemas.xmlsoap.org/wsdl/soap/
  - définit quelques éléments et attributs : soap :binding (fils direct de wsdl :binding), soap :operation (fils direct de wsdl :operation), soap :body, soap :header et soap :headerfault.

*Source : F. Rossi*

# Liaison SOAP (2)

```
<binding name="EtudiantBinding" type="tns:EtudiantPortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getNom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
<operation name="getPrenom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
</binding>
```

- soap :binding :
  - précise qu'on utilise SOAP ;
  - l'attribut transport indique le protocole utilisé pour l'échange des messages SOAP, (en général HTTP précisé par l'URI http ://schemas.xmlsoap.org/soap/http) ;
  - l'attribut style précise si SOAP doit fonctionner en mode rpc ou document (cf la description de soap :body).
- soap :operation : l'attribut soapAction donne la valeur du header HTTP correspondant.

*Source : F. Rossi*

# Corps SOAP

```
<binding name="EtudiantBinding" type="tns:EtudiantPortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getNom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
<operation name="getPrenom">
<soap:operation soapAction="urn:EtudiantAction"/>
<input><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="urn:xmethods" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/></output>
</operation>
</binding>
```

- soap :body :
  - précise le format des messages échangés par une opération ;
  - l'attribut parts permet de préciser les parties concernées ;
  - l'attribut use précise l'interprétation des messages.
- si style englobant document, le Body contient directement les messages ;
- si use encoded (classique) : l'attribut encodingStyle précise la représentation XML ;
- si use literal (style document) : parties de messages brutes.

*Source : F. Rossi*

# Balises Service/Port

```
<service name="Webservice">
<port name="EtudiantPort" binding="tns:EtudiantBinding">
<soap:address location="http://URL/soap_server.php?action=Etudiant"/>
</port>
</service>
```

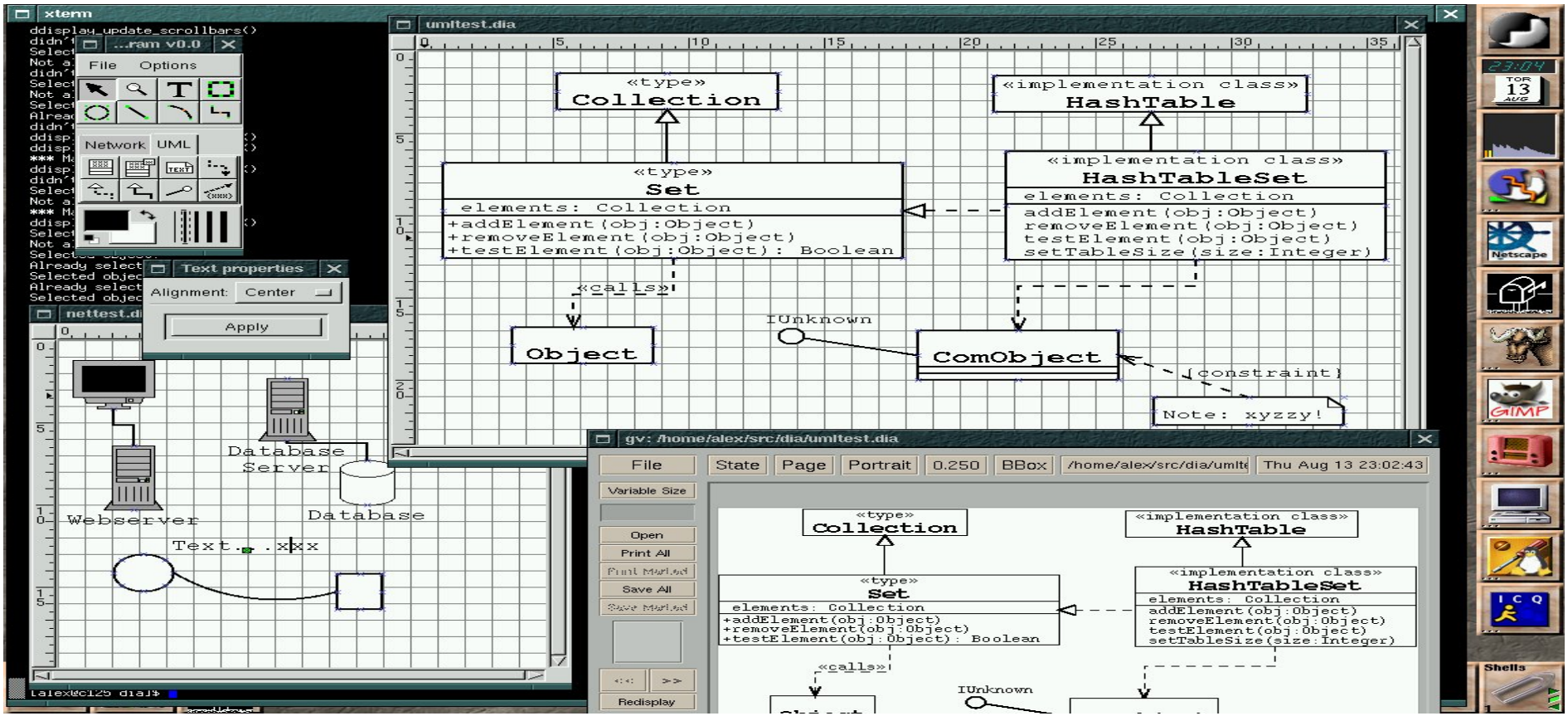
- un service est une collection de ports ;
- un élément wsdl :service portant un nom (attribut name) ;
- contenant un élément wsdl :port par port, précisé par un attribut name donnant le nom du port et un attribut binding donnant le nom du binding associé ;
- des éléments d'extension (sous-éléments de wsdl :port) précisent la définition ;
- pour SOAP, un élément soap :address précise l'URI du port grâce à son attribut location.

*Source : F. Rossi*

# Plan

# Obtenir Dia

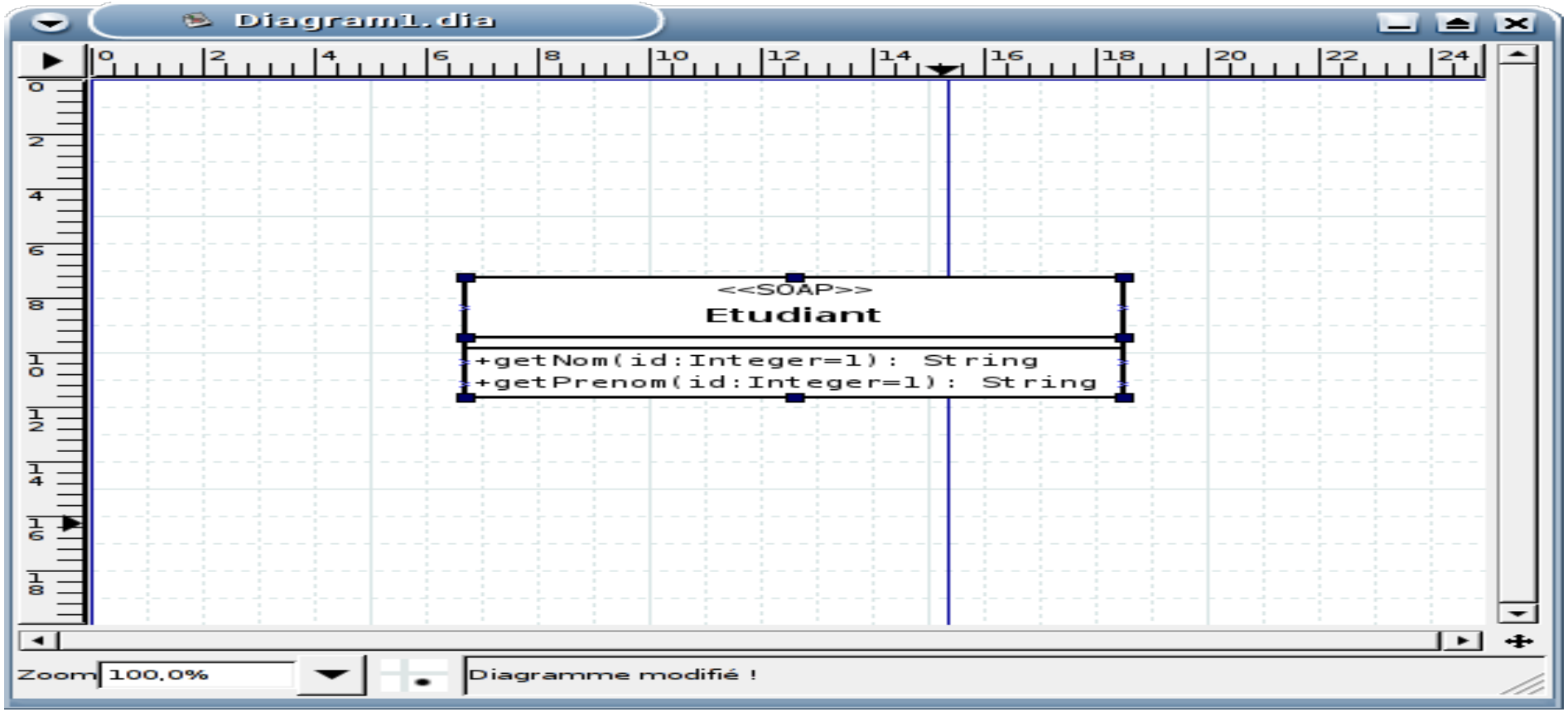
- page du projet : `http ://www.gnome.org/projects/dia/` ;
- plugin uml2php5 : `http ://uml2php5.zpmag.com/`



# Créer une classe Etudiant

Avec deux méthodes :

- String getNom( int id ) ;
- String getPrenom( int id ) ;



# Export en PHP5

## Classe : Etudiant

```php
<?php
/**
 *
 * Code skeleton generated by dia-uml2php5 plugin
 * written by KDO kdo@zpmag.com
 */
class Etudiant {
        /**
         * @access public
         * @param Integer $id
         * @return String
         */
        public final  function getNom($id = 1) {
        }

        /**
         * @access public
         * @param Integer $id
         * @return String
         */
        public final  function getPrenom($id = 1) {
        }
}
?>
```

# Description du service

## Webservices.wsdl

```
<operation name="get">
<soap:operation soapAction="urn:AccesMysqlAction"/>
<input>
<soap:body use="encoded" namespace="urn:xmethods"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded" namespace="urn:xmethods"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>
<service name="Webservice">
<port name="AccesMysqlPort" binding="tns:AccesMysqlBinding">
<soap:address
location="https://miage.emi.u-bordeaux1.fr/webdav/test/soap_server.php
?action=AccesMysql"/>
</port>
</service>
```

Description du service, ici :

- un objet `AccesMysql` avec une méthode `get` ;
- implémentation sur le serveur `miage.emi.u-bordeaux.1.fr` (par https) ;
- via le script `webdav/test/soap_server.php` ayant pour paramètre `action=AccesMysql`.

# soapServer

Cette classe permet la création de serveur SOAP.

Méthodes :

- __construct() : : Constructeur de la classe ;
- addFunction() : : Exporte la, ou les fonctions passées en argument vers le client ;
- getFunctions() : : Liste les fonctions disponibles sur le serveur ;
- handle() : : Traite la requête SOAP envoyée par le client et retourne le résultat ;
- setClass() : : Définit la classe qui va traiter les requêtes SOAP envoyées au serveur ;
- setPersistence() : : Sauvegarde les données dans une session entre deux requêtes.

# chargement dynamique du service

## soap_server.php

```php
<?php
/**
 * Code generated by dia-uml2php5 plugin
 * written by KDO kdo@zpmag.com
 */

if (empty($_GET['action'])) {
        die('SOAP server error');
}

// disabling WSDL cache
ini_set('soap.wsdl_cache_enabled', FALSE);
$server = new SoapServer('Webservices.wsdl');

// Selecting SOAP action
switch ($_GET['action']) {
        case 'AccesMysql':
                require_once($_GET['action'].'.class.php');
                $server->setClass($_GET['action']);
                break;
        default :
                die('SOAP server error');
}
$server->setPersistence(SOAP_PERSISTENCE_SESSION);
$server->handle();
?>
```

Mécanisme générique, utilisable pour des services distincts ...

# Implémentation PHP du service

## AccesMysql.class.php

```php
<?php

class AccesMysql {
        /**
         * @access public
         * @param String $requete
         * @return Array
         */

public final function get($requete) {
$myhote="miage.emi.u-bordeaux1.fr:/tmp/mysql.sock";
$myuser="Consultant";
$mypass="Consultant";
$mybase="ProjetM1Stages";
$base_id = mysql_connect($myhote, $myuser,$mypass);
if (!$base_id)
{
echo "<center> Can't connect to database !</center>";
}
if (! mysql_select_db($mybase,$base_id))
{
echo "<center> Database unavailable !</center>";
}

$res = mysql_query("$requete") or die(mysql_error());
$i = 0;
while ($ligne = mysql_fetch_row($res)){
        $resultat[$i]=utf8_encode($ligne[0]);
                $i++;
                }
return $resultat;
}
```

# Déploiement

- classe accessible via un service web : @WebService

```
import javax.jws.WebService;
@WebService
public class MonService{
/* votre code */
```

- génération des classes du service : wsgen

```
wsgen -cp . MonService
```

- déploiement du service : Endpoint.publish

```
import javax.xml.ws.Endpoint;
[...]
Endpoint.publish("http://localhost:8080/MonService",
new MonService());
System.out.println("Service pret");
while(true);
```

- génération du fichier WSDL correspondant :
  URL http ://localhost :8080/MonService ?WSDL

# Les services webs des Dragons Maudits

## Gardien

- `String ouvrirCoffre( String login, String password )` : si `login=="Tetouillou"` et `password=="empereur"` retourner "le trésor des Dragons Maudits" ; retourner "une flêche empoisonnée" sinon.

## Instructions

1. implémenter en Java le service web Gardien et créer un client pour le consommer ;
2. idem en PHP.

**Remarque** : Java6 ne supporte pas le mode RPC (obsolète) mais seulement le mode Document tandis que PHP5 a des difficultés avec ce nouveau mode...

# Mise en place pas à pas en Java

## Etapes

1. implémenter le service Gardien.java et générer les classes support avec `wsgen -cp . webservice.Gardien`, et écrire une méthode main pour le déployer ;
2. vérifier son bon fonctionnement, en ouvrant le fichier wsdl dans un navigateur : ex. d'URL `http://localhost:8080/Gardien?wsdl` ;
3. générer les classes support du client avec `wsimport -keep urlWSDL` et donner une implémentation du client Client.java.

# Le gardien en Java

```
package webservice;

import javax.jws.WebService;
import javax.xml.ws.Endpoint;

@WebService
public class Gardien{

        private boolean autorisation( String login, String password){
                System.out.println("serveur du service web Gardien: demande autorisation");
                return (login.equals("Tetouillou")&&password.equals("empereur"));
        }

        public String ouvrirCoffre( String login, String password){
                if (autorisation(login,password)) return "le trésor des Dragons Maudits";
                return "une flêche empoisonnée";
        }

public static void main(String[] args) {
                        Endpoint.publish("http://localhost:8080/Gardien",new Gardien());
                        System.out.println("Service Gardien pret");
                        while(true);
}
}
```

# Le client en Java

```
package webservice;

public class Client{

        public static void main( String[] args ){
                Gardien g = (new GardienService()).getGardienPort();
                System.out.println("Bienvenue dans la salle du trésor des Dragons
Maudits");
                String login="Tetouillou"; String password="empereur";
                System.out.println("Identification soumise : login = "+login+" -
password = "+password);
                System.out.println("Resultat: "+g.ouvrirCoffre
("Tetouillou","empereur"));
        }

}
```

# Mise en place pas à pas en PHP

## Etapes

1. avec Dia et son plugin uml2PHP5, créer le code de base du service web Gardien en PHP5 ;
2. ajuster l'url du service dans le fichier Webservices.wsdl ;
3. compléter le code de Gardien.class.php pour implémenter le service ;
4. créer un client PHP pour l'invoquer.

# Le gardien et le client en PHP

```php
<?php
/**
 *
 * Code skeleton generated by dia-uml2php5 plugin
 * written by KDO kdo@zpmag.com
 */

class Gardien {


        /**
         * @access public
         * @param String $login
         * @param String $password
         * @return String
         */

        private function autorisation($login, $password){
                if (($login=="Tetouillou") && ($password=="empereur")) return true;
                return false;
        }

        public final  function ouvrirCoffre($login, $password) {
                if ($this->autorisation($login,$password)) return "le trésor des
Dragons Maudits";
                return "une flêche empoisonnée";
        }
}
?>
```

```php
<?php

        $client = new SoapClient("http://localhost/cours13/php/Webservices.wsdl");
        $login="Tetouillou"; $password="pschitt";
        echo "identification soumise: login = $login - password = $password <br />";
        echo "resultat: ".$client->ouvrirCoffre( $login, $password );
?>
```

# Client Java pour un serveur PHP (avec Axis)

1. génération d'un fichier WSDL adapté à l'utilitaire WSDL2Java
   1. création d'un fichier Java décrivant l'interface du service :

```
public class Etudiant{ public String getNom( int id ){return "
    bob";}
public String getPrenom( int id){return "greg";}}
```

   2. génération du fichier WSDL adapté

```
java -cp *:.
org.apache.axis.wsdl.Java2WSDL -o test.wsdl
-l"http://localhost/soap_server.php?action=Etudiant"  Etudiant
```

2. génération du code du client Java

```
java -cp *:.
org.apache.axis.wsdl.WSDL2Java test.wsdl
```

3. code pour invoquer le service

```
package DefaultNamespace;import java.rmi.*;
public class EtudiantTester {
public static void main(String [] args) throws Exception {
EtudiantService service = new EtudiantServiceLocator();
Etudiant port = service.getSoap_serverPhpActionEtudiant();
String prenom = port.getPrenom(2);
String nom = port.getNom(2);
System.out.println("Je suis "+prenom+" "+nom);}}
```