Architectures Web Services WS-*

Alexandre Denis – Alexandre.Denis@inria.fr

Inria Bordeaux – Sud-Ouest
France

ENSEIRB PG306

Services

- Motivations
  - Intégration d'applications à gros grain
  - Unité : le « service »
- Hétérogénéité
  - Applications conçues indépendamment
  - Pas de modèle commun
  - Intégration par les protocoles communs et la description

Service - SOA

- Définition selon l'*Open Group* :
  - *Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation.*
  - *Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services.*
  - *A service:*
    - *Is a logical representation of a **repeatable** business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)*
    - *Is **self-contained***
    - *May be **composed** of other services*
    - *Is a "**black box**" to consumers of the service*
- **~ composants ?**

Architecture générale : accès à un service

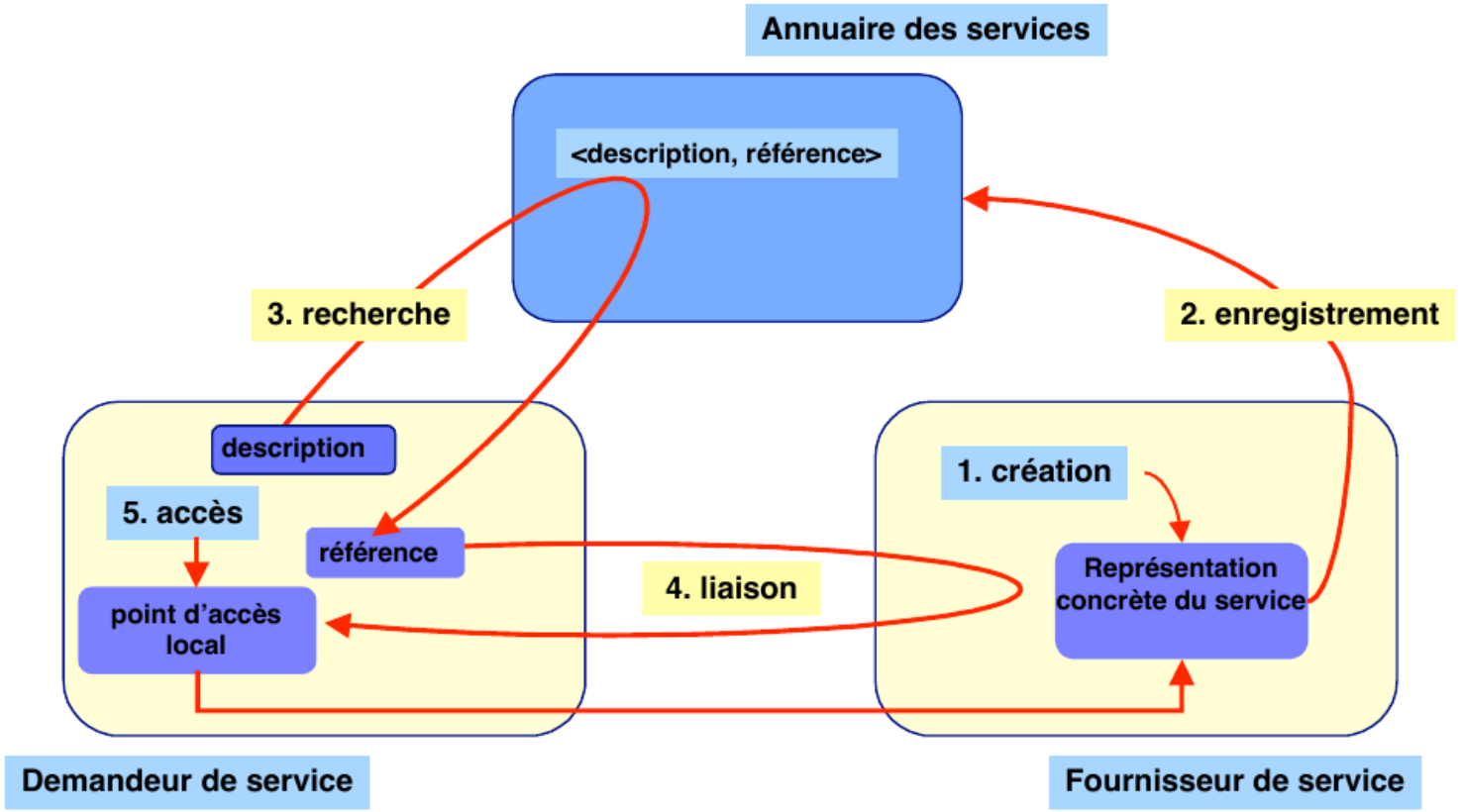


Figure : Krakowiak

Définition Web Services

- Une infrastructure independante des langages et des systèmes permettant des interactions faiblement couplées et interopérables entre des applications distribuées sur Internet.
  - séparation de la spécification et de l'implementation
  - faiblement couplé, car basé sur l'échange de messages
  - interopérable, car basé sur des standards

- Version simple :
  - Web Service = architecture SOA construite avec technologies « web »

Positionnement des Web Services

- Fondamentalement :
  - Rien de nouveau...
  - RPC, composants, annuaire, concepts connus
- Schéma d'intégration à grande échelle
  - On encapsule une application complète plutôt qu'un objet
  - Grain plus gros
- Technologies « web » plus vendeuses que CORBA
  - On peut faire du SOA en CORBA
  - Certaines implémentations Web Service ont aussi une interface CORBA
    - ex.: Apache CXF

Web Services – perspective historique

- XML-RPC - 1998
  - RPC minimaliste sur HTTP + XML
- SOAP – 1998
  - Idem
- Standard W3C SOAP – 2003
  - WSDL, UDDI, etc.
- **WS-*** - 2003-aujourd'hui
  - **Florilège de normes** (80+) formant l'architecture Web Service
  - WS-Adressing, WS-Enumeration, WS-Policy, WS-Routing, WS-Transfer, WS-Security, WS-Eventing, WS-Transaction, etc.
- Architecture REST - ~2007
  - Autre approche, différente des WS-*

Deux écoles de Services Web

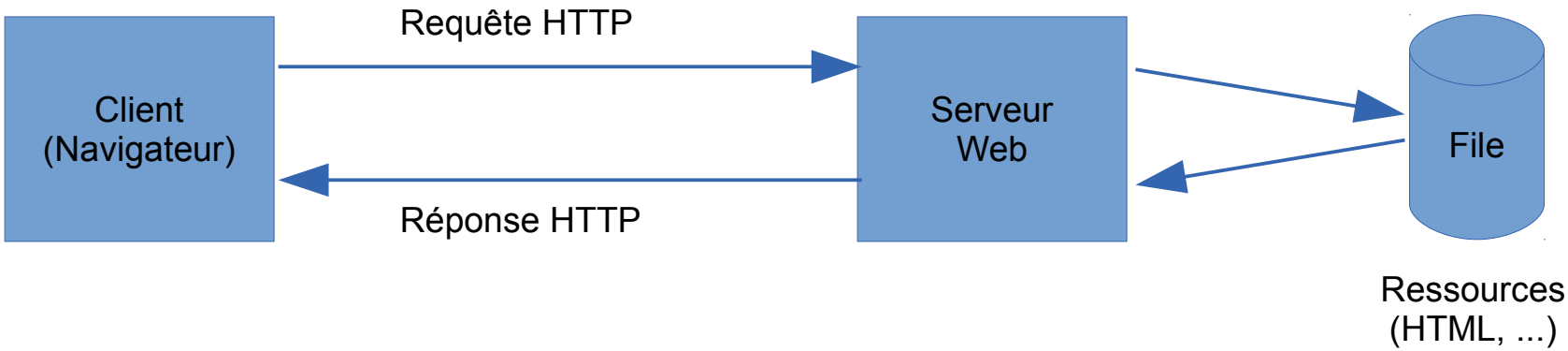
- **WS-***
  - Web Services suivant les normes W3C (et OASIS, OGF, WS-I, Microsoft)
  - Essentiellement SOAP, XML, WSDL, UDDI
  - Orienté « applications d'entreprises »
- **REST**
  - Un style d'architecture, pas une norme
  - L'interface est le protocole réseau
  - Orienté « Web 2.0 », applications légères
  - c.f. prochaine séance

Les protocoles Web

- Protocoles conçu initialement pour les sites web
- Protocole de transport : HTTP
  - Ou SMTP…
- Encodage : XML
- Normalisation par le W3C

Rappels HTTP

- HTTP – Hypertext Transfer Protocol
  - Protocole de transport sur port TCP 80
  - Protocole sans état
  - Adressage URI – Uniform Ressource Identifier



Rappels HTTP

- Requêtes
  - `GET` - demander une ressource
  - `POST` - ajouter une nouvelle ressource (ex. : formulaire)
  - `HEAD` - demander uniquement l'en-tête HTTP
  - `TRACE` - echo de la requête
  - `CONNECT, PUT, DELETE`, ...
- Historique
  - Version 0.9 : requête GET, réponse HTML
  - Version 1.0 : gestion de cache, description du type MIME des ressources (content-type), ...
  - Version 1.1 : connexion persistante (keep-alive), négociation de contenu (accept-*), …
  - Version 2.0 : HTTP/2, RFC 7540, mai 2015 ; chiffrement+compression par défaut, connexions parallèles, mode push

Rappels HTTP - exemple

- Requête

Commande HTTP : GET

```
GET /HelloWorld.html HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.15)
Gecko/2009102815 Ubuntu/9.04 (jaunty) Firefox/3.0.15
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Thu, 19 Nov 2009 14:06:01 GMT
If-None-Match: W/"153-1258639561000"
Cache-Control: max-age=0
```

Header HTTP

Rappels HTTP - exemple

- Réponse

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"153-1258639561000"
Last-Modified: Thu, 19 Nov 2009 14:06:01 GMT
Content-Type: text/html
Content-Length: 153
Date: Tue, 24 Nov 2009 15:48:32 GMT
Connection: close

<html>
   <head>
      <title>Hello World!</title>
   </head>
   <body>
      <center>
         <h1>Hello World!</h1>
      </center>
   </body>
</html>
```

*type **MIME** de la ressource*

***corps** de la réponse*

Rappels XML

- XML – eXtensible Markup Language
  - Standard W3C depuis 1998
- Formalisme pour la description de données structurées
- Constructions de langages spécialisés
  - Utilisation de DTD ou Schema
- Dérivé du SGML – Standard Generalized Markup Language

Rappels XML - documents

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adresse>
  <lieu>
    <rue> avenue de l'Europe</rue>
    <numero>655</numero>
  </lieu>
  <ville>Montbonnot></ville>
  <code>38330</code>
</adresse>
```

- Document XML
  - Format texte
  - Structure : balises
- Deux niveau de correction
  - Document **bien formé**
    - Conforme à la **syntaxe** XML (caractères autorisés, balises bien imbriquées, etc.)
  - Document **valide**
    - Conforme au **modèle** spécifié par le DTD ou le Schema

Source: Krakowiak

Rappels XML - validation

- Deux modèles de spécification de structure
  - DTD – Document Type Definition
    - Formalisme spécifique
    - Simple, peu puissant, tombe en désuétude
  - XML Schema (XSD – XML Schema Definition)
    - Exprimé en XML
    - Standard actuel
- **Validation**
  - Vérification qu'un document XML est conforme au Schema

Rappels XML - Schema

Schema

```
<element name="rue" type="string"/>
<element name="numero" type="string"/>
<element name="ville" type="string"/>
<element name="code" type="string"/>
<element name="lieu">
  <complexType>
    <sequence>
      <element ref="rue"/>
      <element ref="numero"/>
    </sequence>
  </complexType>
</element>
<element name="adresse">
  <complexType>
    <sequence>
      <element ref="lieu"/>
      <element ref="ville"/>
      <element ref="code"/>
    </sequence>
  </complexType>
</element>
```

DTD

```
<!ELEMENT rue (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT lieu(rue, numero)>
<!ELEMENT adresse(lieu, ville, code)>
```

Document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adresse>
  <lieu>
    <rue> avenue de l'Europe</rue>
    <numero>655</numero>
  </lieu>
  <ville>Montbonnot></ville>
  <code>38330</code>
</adresse>
```

Source: Krakowiak

Rappels XML - parsing

- Parsing XML : transformer un document XML en données exploitables
- Deux familles de parseurs génériques
  - DOM – Document Object Model
    - Document transformé en arboressence d'objets, puis navigation dans la structure en mémoire
  - SAX – Simple API for XML
    - Interface évènementielle, fonction utilisateur appelée sur chaque balise
- Alternative : parseur compilé
  - Un code de parseur spécifique est généré à partir du Schema et sort directement les données correctement typées

Rappels XML - namespace

- Espace de nom (*namespace*) : domaines séparés de définition des balises
- Définition d'un namespace
  - Dans les attributs de la balise XML englobante : `xmlns:toto=http://toto.net/todo.xsd`
  - URI indentifie le namespace
  - Ne coresspondant pas nécessairement à une URI accessible sur le web
    - Habituellement : URL du DTD/Schema, ou de la norme
  - Namespace par défaut : `xmlns=http://toto.net/`
- Utilisation d'un namespace
  - Préfixe de balise : `<toto:balise> ... </toto:balise>`
  - Sans préfixe : dans le namespace par défaut

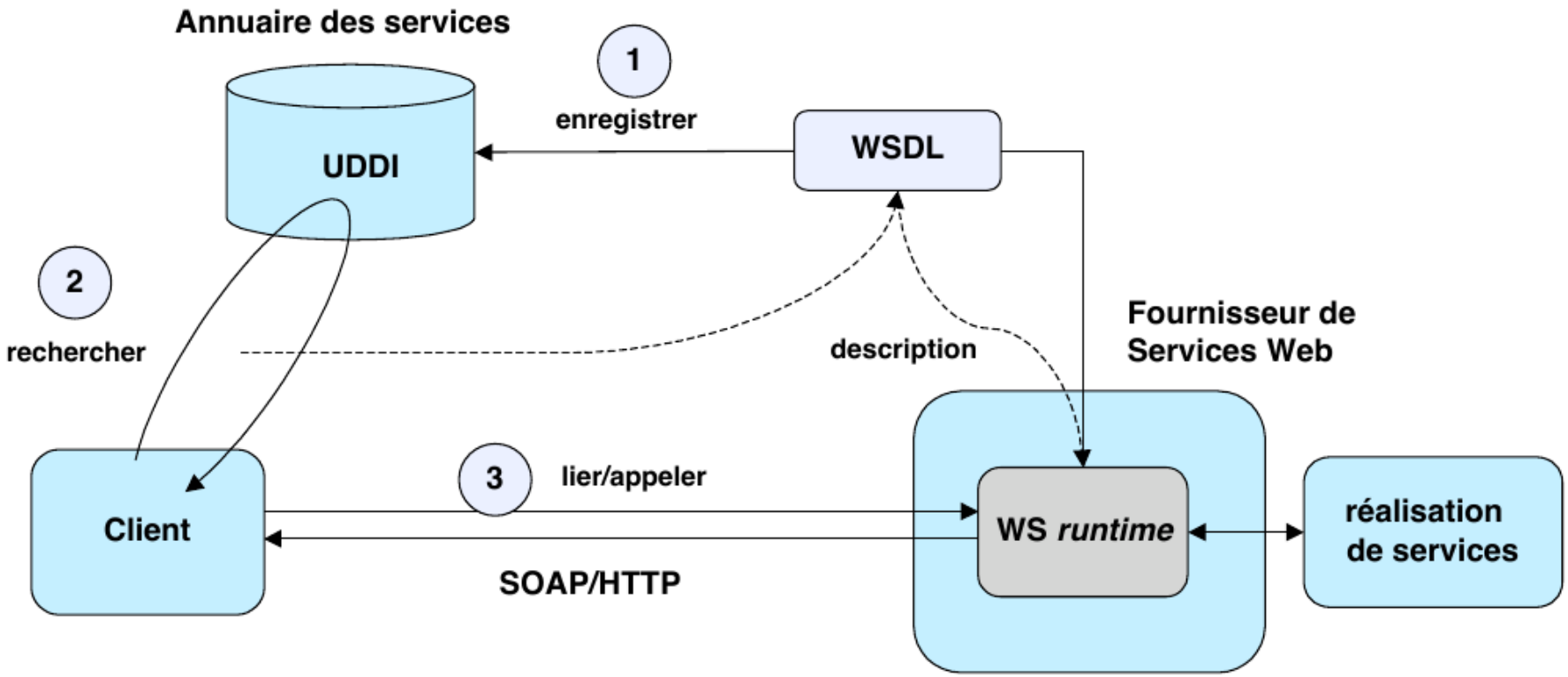
Service web minimal : RPC-XML

- Des RPC avec un encodage XML
  - Le typage est géré par le Schema XML

```
<methodCall>
  <methodName>meteo.temperature</methodName>
  <params>
    <param>
      <value>38330</value>
    </param>
  </params>
</methodCall>
```

```
<methodResponse>
  <params>
    <param>
      <value>12</value>
    </param>
  </params>
</methodResponse>
```

Big Picture des Web Services



Briques de base : WSDL, SOAP, UDDI

- **WSDL** – Web Service Description Language
  - Description de l'interface d'un service
- **SOAP** – Simple Object Access Protocol
  - Protocole de communication pour accéder aux services web
- **UDDI** – Universal Description, Discovery and Integration
  - Annuaire de services

Le protocole SOAP

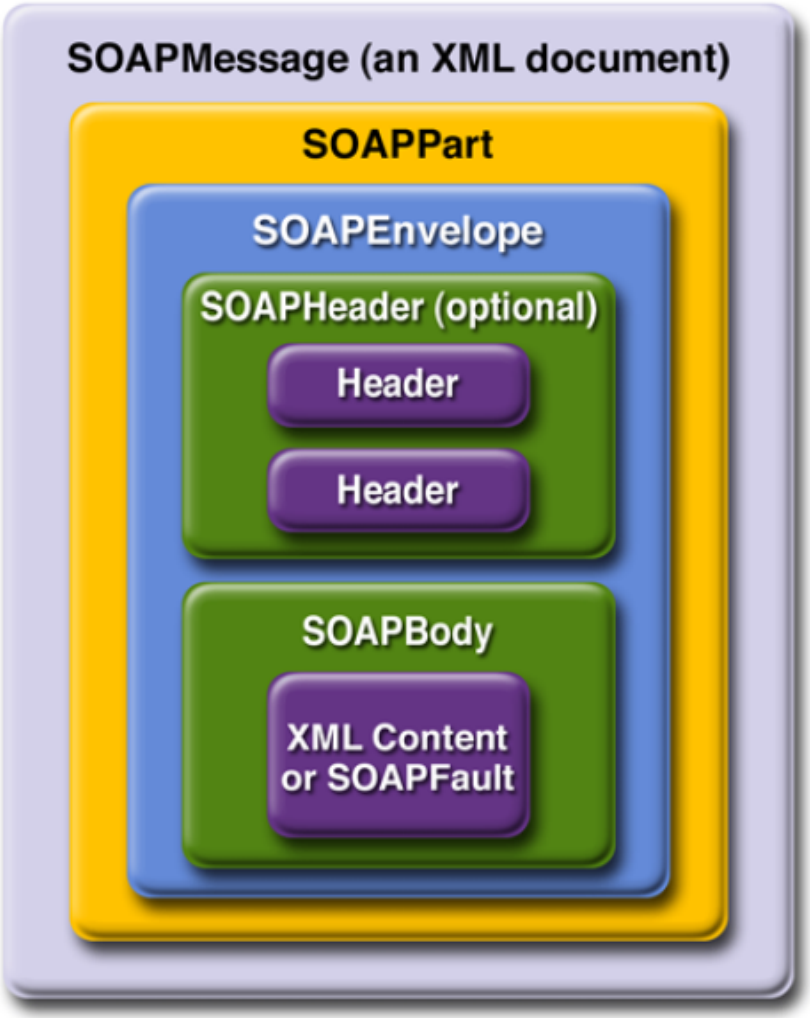
- Protocole d'échange de messages
  - Message ASCII en langage **XML**
    - Pas un format binaire
  - Transport quelconque
    - En pratique : **HTTP**
  - Paradigme essentiellement RPC
    - Mais pas seulement
  - Messages unidirectionnels

Message SOAP

- Un document XML
  - Espace de nommage SOAP
  - Espace de nommage de l'application
- Document auto-descriptif (avec le DTD ou Schema)
- Messages « lisibles »
- Doivent être générés puis parsés
  - Protocole coûteux
  - Messages volumineux
    - Chaque paramètre est en format texte avec une étiquette

Message SOAP

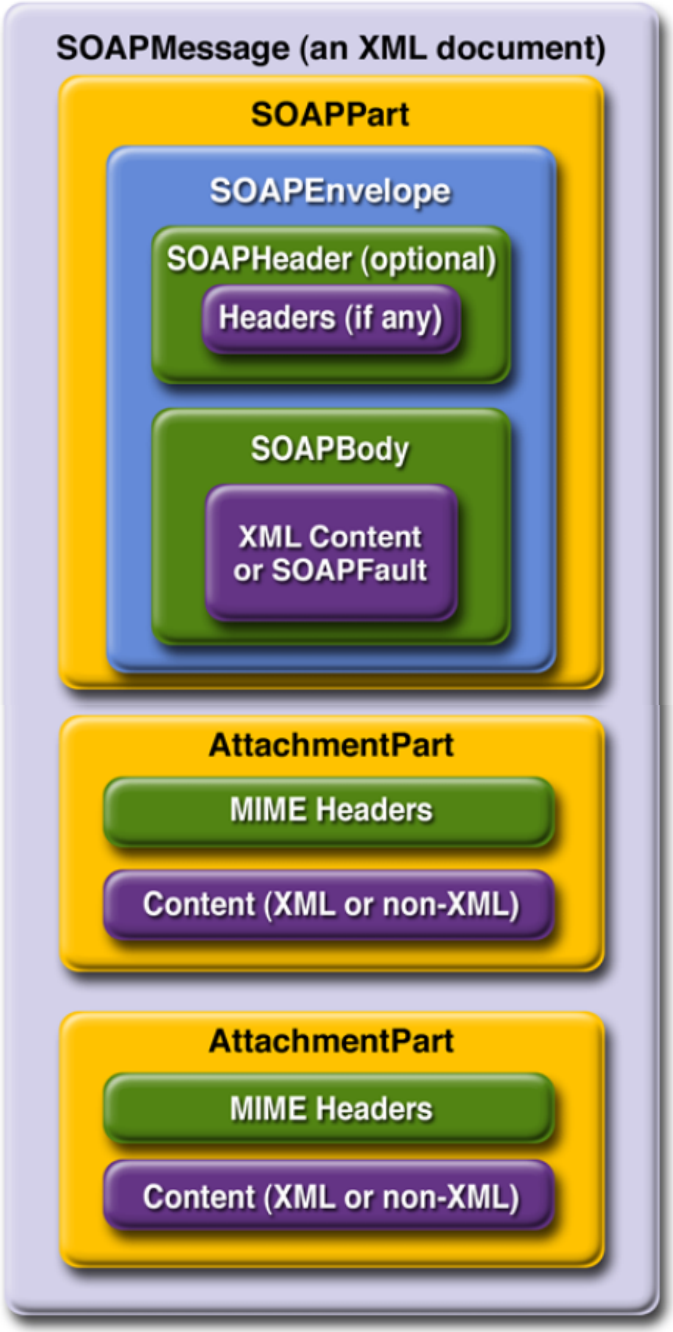
- Enveloppe (*envelope*)
  - Codage, namespace, etc.
- En-tête (*header*)
  - Facultatif
  - Authentification, facturation, etc.
- Corps du message (*body*)
  - Méthodes, paramètres
- Erreurs (*fault*)
  - Retour d'erreur



Source : Sun

Message SOAP avec pièces jointes

- Utilisation de XML Attachment
  - Pour des paramètres binaires
  - Pour la performance
    - Éviter les sérialisation / désérialisation en encodage XML
- Autodescription du typage à l'aide du système multipart MIME



Exemple SOAP/HTTP - requête

- En-tête HTTP
- Message XML (dont la requête)

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

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:getLastTradePrice xmlns:m="SomeURI">
      <symbol>SomeCompany </symbol>
    </m:getLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemple SOAP/HTTP - réponse

- En-tête HTTP
- Message XML (dont la réponse)

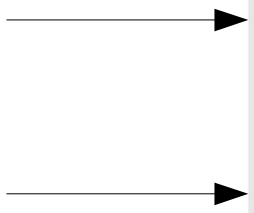
```
HTTP/1.1 200 OK
Content-Type: text/xml: charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:getLastrTradePriceResponse xmlns:m="SomeURI">
      < price>34.5 </price>
    </m:getLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP – codage des données

- Sérialisation : génération de XML
- Désérialisation : parsing de XML
- Typage via XML Schema

Schema

```
<element name="price" type="float/>

<element name="color">
  <simpleType base="xsd:string">
    <enumeration value ="Green"/>
    <enumeration value ="Red"/>
    <enumeration value ="Blue"/>
  </simpleType>
</element>
```



Valeur

```
<price>23.5</price>

<color>Red</color>
```

WSDL – Web Service Description Language

- Description d'un service
  - Format des messages échangés
  - Relations entre messages (requête-réponse)
- Exprimé en XML
- Définit
  - L'interface (~IDL)
    - Types, messages, opérations, ports
  - Les points d'entrée (*endpoints*)
- En pratique
  - Très peu lisible, difficile à écrire pour l'utilisateur
  - Généré automatiquement à l'aide d'outils

WSDL - syntaxe

- `<wsdl:definitions>` racine du document XML décrivant un service web
- `<wsdl:type>` définition des types de données utilisées
- `<wsdl:message>` description du type des messages
- `<wsdl:operation>` description d'un type de requête
- `<wsdl:part>` type des paramètres
- `<wsdl:portType>` type de port, décrivant l'ensemble des opérations du service
- `<wsdl:binding>` liaison décrivant le protocole de transport, et les types des messages associés aux opérations du service.
- `<wsdl:service>` une collection de ports
- `<wsdl:port>` port associant une liaison et une adresse réseau explicite (endpoint)

WSDL - exemple

```
-<wsdl:definitions targetNamespace="http://localhost:8080/axis/HelloWorld.jws">
  -<wsdl:message name="testRequest">
     <wsdl:part name="data" type="xsd:string"/>
   </wsdl:message>
  -<wsdl:message name="testResponse">
     <wsdl:part name="testReturn" type="xsd:string"/>
   </wsdl:message>
  -<wsdl:portType name="HelloWorld">
    -<wsdl:operation name="test" parameterOrder="data">
       <wsdl:input message="impl:testRequest" name="testRequest"/>
       <wsdl:output message="impl:testResponse" name="testResponse"/>
     </wsdl:operation>
   </wsdl:portType>
  -<wsdl:binding name="HelloWorldSoapBinding" type="impl:HelloWorld">
     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    -<wsdl:operation name="test">
       <wsdlsoap:operation soapAction=""/>
      -<wsdl:input name="testRequest">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://DefaultNamespace" use="encoded"/>
       </wsdl:input>
      -<wsdl:output name="testResponse">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://localhost:8080/axis/HelloWorld.jws" use="encoded"/>
       </wsdl:output>
     </wsdl:operation>
   </wsdl:binding>
  -<wsdl:service name="HelloWorldService">
    -<wsdl:port binding="impl:HelloWorldSoapBinding" name="HelloWorld">
       <wsdlsoap:address location="http://localhost:8080/axis/HelloWorld.jws"/>
     </wsdl:port>
   </wsdl:service>
 </wsdl:definitions>
```

En-têtes XML omis pour rester concis...

Messages (request, response)

Port Type (operations)

Binding

Service

UDDI – Universal Description, Discovery and Integration

- Annuaire et référentiel pour la description de services web
  - Fourni sous forme de web service
- Pages blanches
  - Annuaire de services, par nom
- Pages jaunes
  - Annuaire de services, par catégorie
- Pages vertes
  - Définition des services en WSDL

Outils pour Web Services

- Seuls les **protocoles** sont normalisés
  - **Pas de modèle de programmation** spécifique !
  - Pas d'outil / API / workflow standard
- Plusieurs approches
  - Compilation de stubs à partir du WSDL
  - Extraction du WSDL à partir d'un code applicatif
  - Annotations dans le code et préprocessing
  - Servlets dans un serveur Web
  - Processus serveur autonome issu de code généré

Middleware pour Web Services

- Apache Axis
  - Framework de référence (maintenant obsolète)
- Apache CXF
  - Framework successeur d'Axis, interopérable CORBA
- JAX-WS - Java API for XML WebServices, JWS, Oracle WebLogic
  - Annotations de code Java
- IBM WebSphere, Redhat JBoss
  - SOAP = une interface parmi d'autres, extraite automatiquement
- gSOAP
  - Stubs et serveur autonome en C/C++
- .NET
  - Classes HttpChannel et SoapFormatter intégrées au SDK

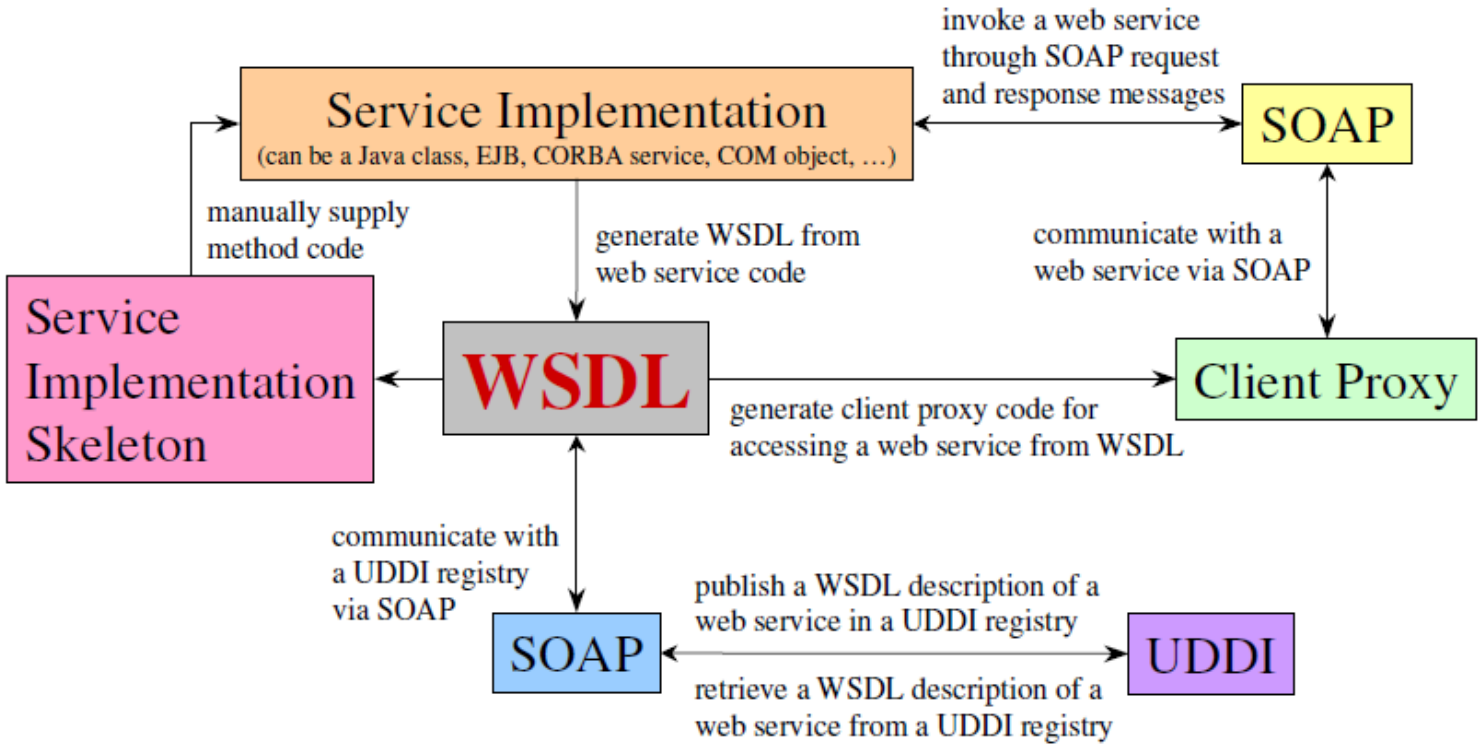
Apache Axis

http://ws.apache.org/axis

- Axis (Apache eXtensible Interaction Service)
  - Un toolkit open source de Web Service basé sur Apache Tomcat
  - Deux styles de déploiement des Web Services
    - Instant Deployment (JWS)
    - Custom Deployment (WSDD)
  - Programmation d'application Java avec JAX-RPC
    - Client statique ou dynamique
  - Outils pour WSDL : WSDL2Java et Java2WSDL
  - Support de EJB : Session Bean accessible comme WS

Apache Axis

- Vue d'ensemble



Source : Mark Volkmann

JWS – Instant Deployement

- Une simple **classe Java** renommée avec l'extension jws
  - Placer le fichier `jws` dans `<tomcat>/webapps/axis/`
  - Traduction automatique en Web Service
  - Déploiement instantané dans `http://<server>:<port>/axis/`
    - Scope Request : un nouvel objet instancié à chaque requête
- Exemple : HelloWorld.jws

```
public class HelloWorld
{
  public String test(String data)
  {
    return "Hello World! You sent the string '" + data + "'.";
  }
}
```

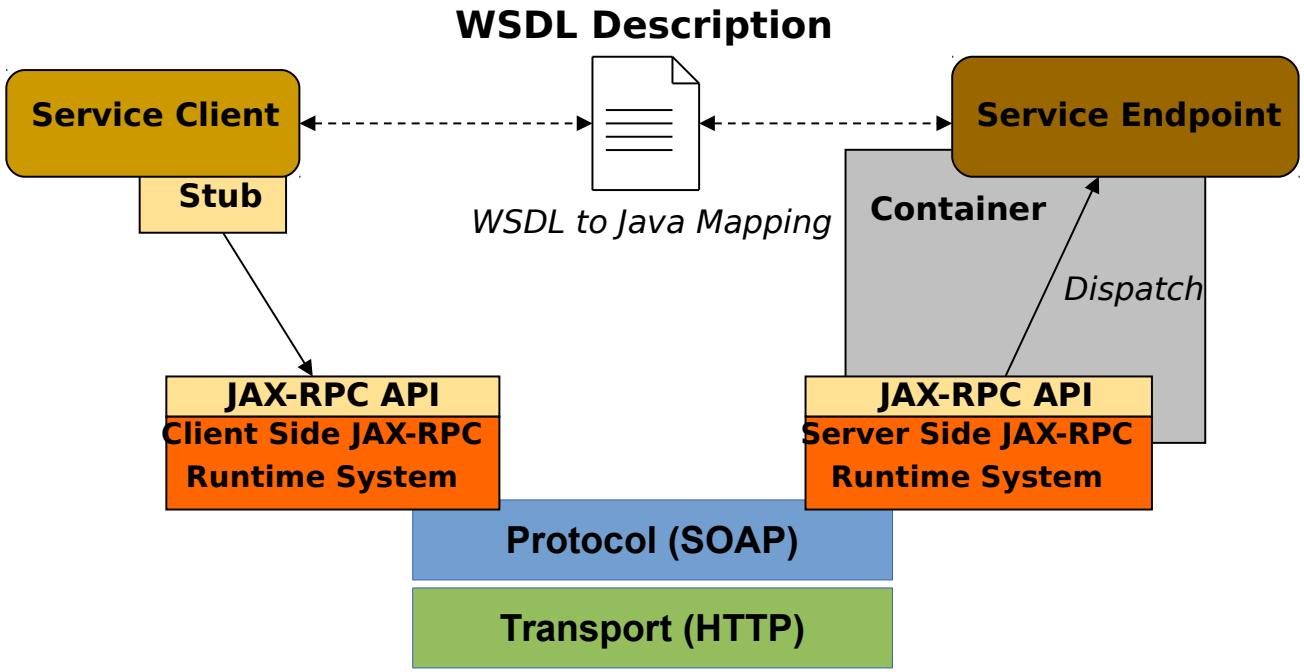
JAX-RPC

- Java API for XML-based RPC (JAX-RPC)
  - Modèle de programmation Sun Java des Web Services (en mode RPC), comparable à RMI
  - Mapping des types Java et WSDL
  - Client statique ou dynamique
    - Génération des stubs avec WSDL2Java...
    - Utilisation des interfaces *Service* et *Call*
  - Version 2.0 de JAX-RPC
    - Java API for XML Web Services (JAX-WS)

Client statique JAX-RPC

- Génération des stubs à partir du WSDL

```
java org.apache.axis.wsdl.WSDL2Java HelloWorld.wsdl
```



Source : Pankaj Kumar

Client statique JAX-RPC

- Exemple

```
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import localhost.axis.HelloWorld_jws.*;

public class HelloWorldStaticClient
{
  public static void main(String[ ] args)
  throws ServiceException, RemoteException
  {
    HelloWorldService locator = new HelloWorldServiceLocator();
    HelloWorld stub = locator.getHelloWorld();
    String returnValue = stub.test("toto");
    System.out.println(returnValue);
  }
}
```

import des classes stub

localisation du endpoint

récupération du stub

interface du service HelloWorld

Projection des types

- Projection des tyes WSDL/XSD/SOAP en Java

xsd:base64Binary	byte[]
xsd:boolean	boolean
xsd:byte	byte
xsd:dateTime	java.util.Calendar
xsd:decimal	java.math.BigDecimal
xsd:double	double
xsd:float	float
xsd:hexBinary	byte[]
xsd:int	int
xsd:integer	java.math.BigInteger
xsd:long	long
xsd:QName	javax.xml.namespace.QName
xsd:short	short
xsd:string	java.lang.String

Source : http://ws.apache.org/axis/java/user-guide.html

Client dynamique JAX-RPC

- Construction dynamique des requêtes (DII)
  - N'utilise pas le WSDL
  - Utilisation des interfaces *Service* et *Call*
    - `javax.xml.rpc.Service`
    - `javax.xml.rpc.Call`
  - Pas de vérification du type des paramètres à la compilation
    - Passage d'un tableau d'Object.

Client dynamique JAX-RPC

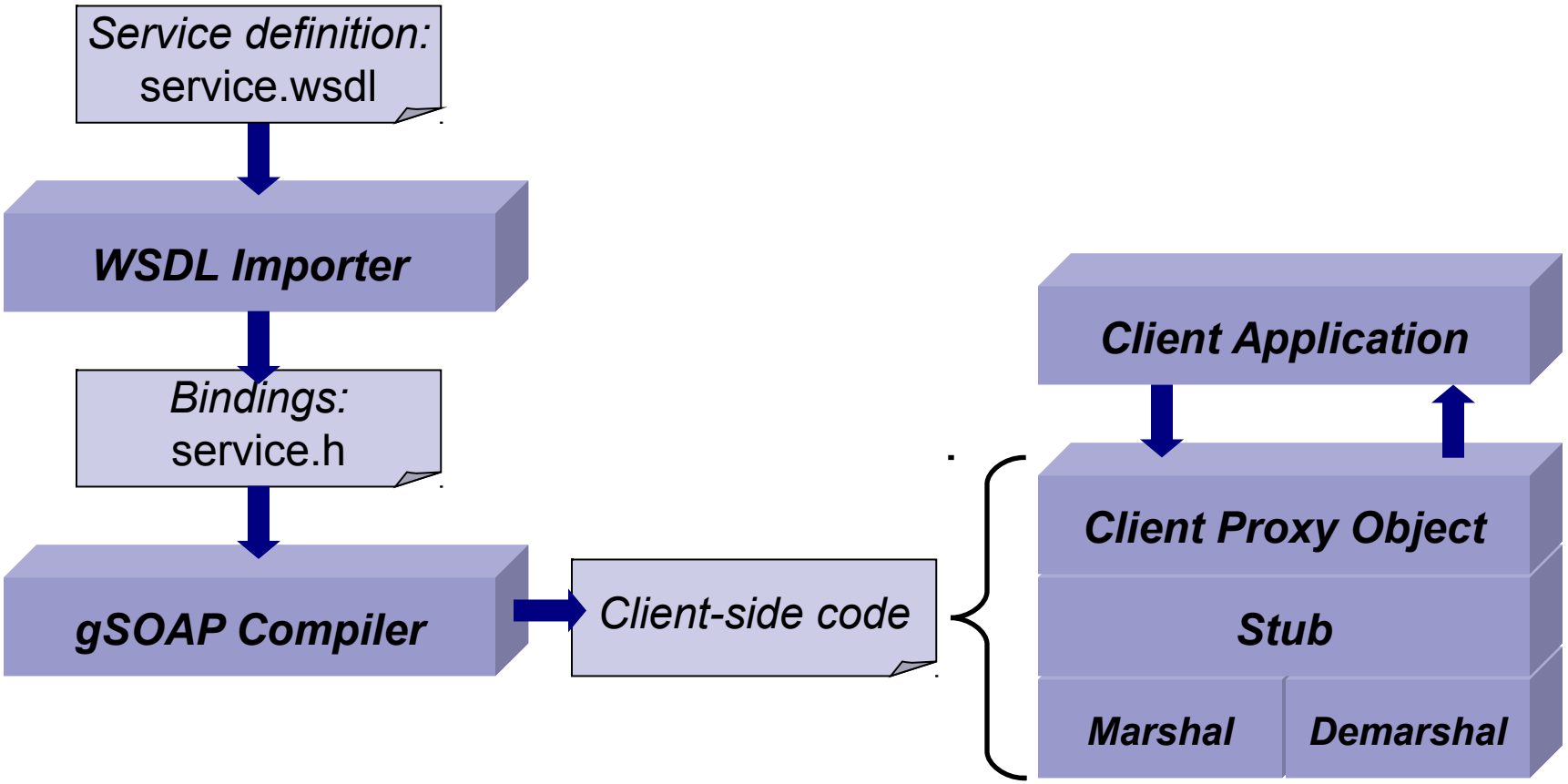
- Exemple

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

public class HelloWorldClient
{
  private static final String ENDPOINT = "http://localhost:8080/axis/HelloWorld.jws";
  private static final String NAMESPACE = "http://soapinterop.org/";
  private static final String OPERATION = "test";

  public static void main(String[ ] args)
    throws ServiceException, MalformedURLException, RemoteException
  {
    Service service = new Service();
    Call call = (Call)service.createCall();
    call.setTargetEndpointAddress(new URL(ENDPOINT));
    call.setOperationName(new QName(NAMESPACE, OPERATION));
    String returnValue = (String)call.invoke(new Object[ ]{"toto"});
    System.out.println(returnValue);
  }
}
```

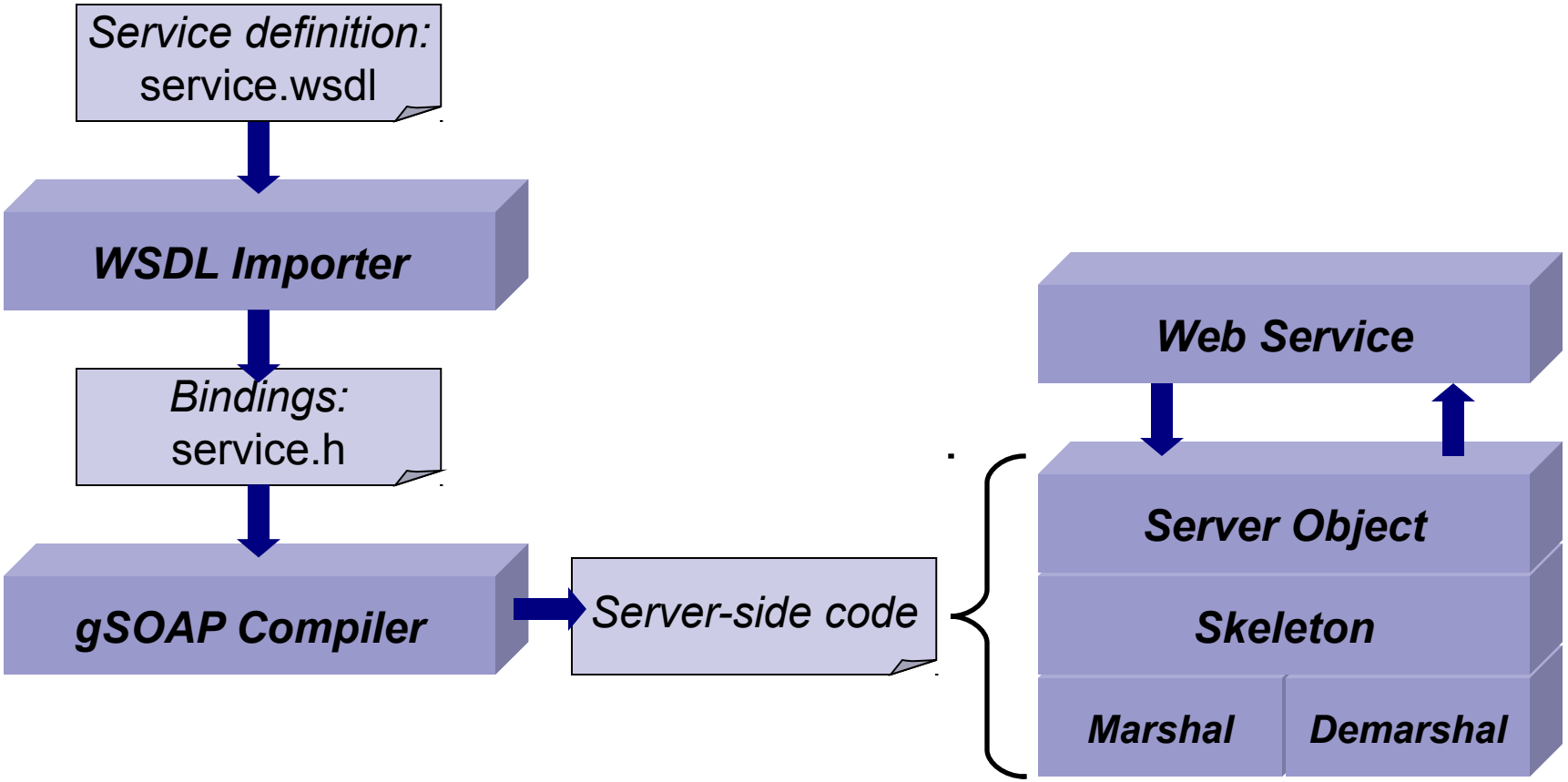
gSOAP

- Implémentation Web Service en C/C++
  - Issu de Florida State University
  - Stubs et squelettes compilés statiquement
  - Sérialisation/désérialisation XML optimisée
  - Serveur http intégré ou utilisable en CGI
  - Projection SOAP <-> C/C++ spécifique
    - Reconnaissance de types
    - Directives `//gsoap`

gSOAP - client

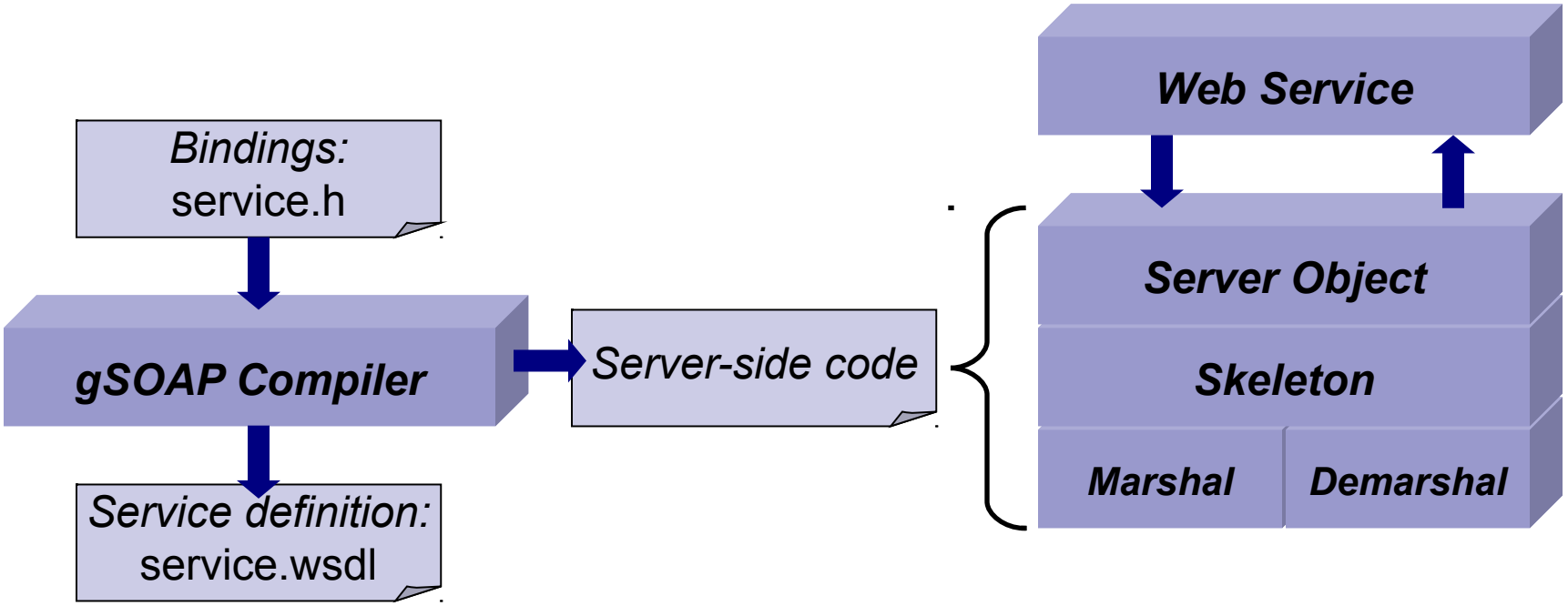


gSOAP - serveur



gSOAP – extraction du WSDL

- Extraction automatique du WSDL à partir du code C/C++



gSOAP – génération d'un parseur XML

- Parseur XML spécifique pour le Schema donné


```
<complexType name="List">
  <complexContent>
    <sequence>
      <element name="item"
       type="xsd:string"
       maxOccurs="unbounded"/>
    </sequence>
  </complexContent>
</complexType>
```

```
class ns__List
{ std::vector<char*> item;
  int in(char* tag);
  int out(char *tag);
};
```

```
int ns__List::in(char* tag)
{ if (begin_element(tag) != OK)
    return TAG_MISMATCH;
  in_vectorOfstring(item, "item");
  end_element(tag);
}
```

gSOAP - Exemple

- Définition de l'interface en C

Namespace (ici : **ns**)

```
ns__getQuote(char *symbol, float &result);
```

- Génération du WSDL, souches, squelettes :

```
soapcpp2 -ptest -c test.h
```

- Sortie :
  - Interface WSDL : `ns.wsdl`
  - Schema XML pour le typage : `ns.xsd`
  - Stubs, squelettes : `test*.c`
- Invocation :

```
main()
    { float q;
      if (soap_call_ns__getQuote("URL", "", "AOL", q) == 0)
        cout << "AOL: " << q << endl;
    }
```

gSOAP - Exemple

- XSD généré

```
<?xml version="1.0" encoding="UTF-8"?>
 <schema targetNamespace="http://tempuri.org/ns.xsd"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="http://tempuri.org/ns.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <!-- operation request element -->
  <element name="getQuote">
   <complexType>
    <sequence>
     <element name="symbol" type="xsd:string" minOccurs="0" maxOccurs="1" nillable="true"/>
     <!-- ns__getQuote::symbol -->
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="getQuoteResponse">
   <complexType>
    <sequence>
     <element name="result" type="xsd:float" minOccurs="1" maxOccurs="1"/><!-- ns__getQuote::result -->
    </sequence>
   </complexType>
  </element>
 </schema>
```

gSOAP - Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Service"
 targetNamespace="http://tempuri.org/ns.xsd/Service.wsdl"
 xmlns:tns="http://tempuri.org/ns.xsd/Service.wsdl"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="http://tempuri.org/ns.xsd"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:HTTP="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
 <schema targetNamespace="http://tempuri.org/ns.xsd"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="http://tempuri.org/ns.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <!-- operation request element -->
  <element name="getQuote">
   <complexType>
    <sequence>
     <element name="symbol" type="xsd:string" minOccurs="0" maxOccurs="1"
nillable="true"/><!-- ns__getQuote::symbol -->
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="getQuoteResponse">
   <complexType>
    <sequence>
     <element name="result" type="xsd:float" minOccurs="1" maxOccurs="1"/><!--
ns__getQuote::result -->
    </sequence>
   </complexType>
  </element>
 </schema>
</types>
```

- WSDL généré...

```
<message name="getQuoteRequest">
 <part name="Body" element="ns:getQuote"/><!-- ns__getQuote::ns__getQuote -->
</message>

<message name="getQuoteResponse">
 <part name="Body" element="ns:getQuoteResponse"/>
</message>

<portType name="ServicePortType">
 <operation name="getQuote">
  <documentation>Service definition of function ns__getQuote</documentation>
  <input message="tns:getQuoteRequest"/>
  <output message="tns:getQuoteResponse"/>
 </operation>
</portType>

<binding name="Service" type="tns:ServicePortType">
 <SOAP:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="getQuote">
  <SOAP:operation soapAction=""/>
  <input>
     <SOAP:body parts="Body" use="literal"/>
  </input>
  <output>
     <SOAP:body parts="Body" use="literal"/>
  </output>
 </operation>
</binding>

<service name="Service">
 <documentation>gSOAP 2.8.18 generated service definition</documentation>
 <port name="Service" binding="tns:Service">
  <SOAP:address location="http://localhost:80"/>
 </port>
</service>

</definitions>
```

gSOAP – projection C

- La projection de SOAP vers le C est codifiée
  - Le séparateur entre namespace et méthode est `__`
  - Une fonction `double foo(double a)` sera définie comme
    - `int ns__foo(double a, double*result)` dans **l'interface**
      - La valeur de retour est pour gSOAP
    - `int ns__foo(struct soap*soap, double a, double*result)` dans **l'implémentation**
      - Ne pas inclure la définition d'interface dans l'implémentation !
    - `soap_call_ns__foo(&soap, &server, &action, a, &result)` dans le client

gSOAP – projection C

- Les types primitifs sont projetés vers leur équivalent en XSD
  - `int, long, double, float`
  - `char*` -> `string`
- Tableau dynamique :

```
struct dynamic_array
{
        Type*__ptr;
        int __size;
};
```

  - *Pas de namespace dans le nom de la structure, sinon ce serait traduit comme une structure XSD et non comme tableau !*
- Consultez la **documentation** pour le détail de la projection des types

Travail à faire

Exemple gSOAP

- Téléchargez l'exemple calc.tar.gz sur la page du cours
- Compilez l'exemple à l'aide du Makefile
- Il s'agit d'un serveur de calcul minimaliste doté des opérations `add` et `sub`
  - Regardez la définition d'interface `calc.h`
  - Regardez le fichier `calc.wsdl` pour voir la projection
- Des patrons de requêtes sont générés automatiquement
  - `calc.*.*.xml`

HTTP à la main

- Lancez `./server`
  - Le serveur écoute alors sur `localhost:18083`
  - Connectez-vous à la main à l'aide de `telnet`
  - Testez une requête HTTP à la main, en vous aidant des patrons `calc.*.req.xml` pour former une requête correcte
    - Antisèche HTTP :

      ```
      POST /calc/ HTTP/1.1
      Content-Length: xxx
      ```

      À vous de calculer la longueur !
- Au bout d'un nombre de tentatives fini, vous devriez obtenir

```
HTTP/1.1 200 OK
Server: gSOAP/2.8
...
<SOAP-ENV:Envelope ...
```

HTTP avec curl

- Lancez des requêtes vers le serveur à l'aide de l'outil `curl`
  - Consultez le manuel de curl si besoin
    - Vous aurez sans doute besoin de : --header (pour forcer le Content-Type), --upload-file, --request
  - Vers quelle URL faut-il pointer ?
  - Quelle opération HTTP faut-il invoquer ?
- Testez plusieurs requêtes, vérifiez que ça calcule juste !

Avec le client gSOAP

- Lancez des requêtes avec le client fourni
- Ajoutez des opérations (mult, div, sqrt, etc.)
  - Dans l'interface
  - Dans l'implémentation du serveur
  - Dans leur utilisation par le client
- Testez la connexion avec le serveur du voisin
  - Attention, le hostname client et serveur sont codés en dur dans le source, pensez à les modifier !

Construire un service web

- Proposez une interface pour un serveur d'annuaire
  - Rechercher une entrée, ajouter une entrée
  - Écrivez l'interface en C gSOAP, extraire le WSDL
- Implémentez le service
- Créez un client de test
- Envoyez le WSDL (et non le .h !) à votre voisin
  - Écrivez un client qui interroge le serveur du voisin

Vers un service externe

- Choisissez un service vers lequel vous connecter
  - Conversion de monnaie : http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
  - Prévisions météo : http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php
  - Génération de code barre : http://www.webservicex.net/genericbarcode.asmx?WSDL
  - Via Michelin : http://dev.viamichelin.com/viamichelin-soap-api.html
  - Microsoft translator : http://api.microsofttranslator.com/V2/Soap.svc
  - Bing search : https://datamarket.azure.com/dataset/bing/search
  - Ebay : http://developer.ebay.com/webservices/latest/eBaySvc.wsdl

Vers un service externe

- Convertissez le WSDL en .h à l'aide de : `wsdl2h -c`
- Écrivez un client utilisant gSOAP pour le service choisi

À vous de jouer !

http://dept-info.labri.fr/~denis/