

Web Services

Reda Bendraou

Plan

- WS: Définition et Principes
- XML
- SOAP
- WSDL
- UDDI
- Mise en œuvre avec AXIS 2 et JAX-WS
- Conclusion et Lectures

Définition

- Les Services Web sont des « applications modulaires qui exécutent des tâches spécifiques indépendamment des plates-formes et des langages sur lesquelles elles reposent
- Par exemple:
 - un client demande le prix d'un article en envoyant un message sur le Web.
 - Ce message contient la référence de l'article.
 - Le Web Service va recevoir la référence, effectuer le traitement du service et renvoyer le prix au client via un autre message.

Principes

Pourquoi un nouveau middleware ?

Limitations des middleware étudiés (objet, composant)

Passage à large échelle : Web

- Protocoles hétérogènes
 - IIOP, RMI, DCOM
 - Pare-feu (ex: un objet CORBA sous XP)
- Pas d'ouverture des services
 - Notion de moteur de recherche inexistante
- Trop de contraintes sur le client !
 - Doit posséder les souches
 - Difficulté de le construire dynamiquement

Limitations des middleware

Inconvénients Intrinsèques

- Complexité
 - CORBA : IDL, MapEcho, ...
 - EJB : Container, JNDI, ...
- Pérennité : remise en question
 - CORBA, EJB, .Net, ...
- Prix
 - Plates-formes
 - Compétences

Solutions existantes

- Modification du Protocole
 - RMI / IIOP
- Passerelles
 - CORBA vers DCOM
- Portage d'applications existantes difficile
- Solutions non standards

Extension vers les services

- WSOA : Web Services Oriented Architecture
- Deux préoccupations
 - **ÉCHELLE** : Augmenter la productivité des entreprises à travers des partenariats => nécessité d'ouvrir un parc applicatif à l'extérieur sous forme de services offerts
 - Granularité variable : fonctions, composants, applications, processus métier
 - **INTEROPÉRABILITÉ** des applications (intra- ou inter-entreprises)
 - Comment faire fonctionner un applicatif (ou un composant) fondé sur une technologie CORBA avec un autre fondé sur .Net ?

Approche Envisagée

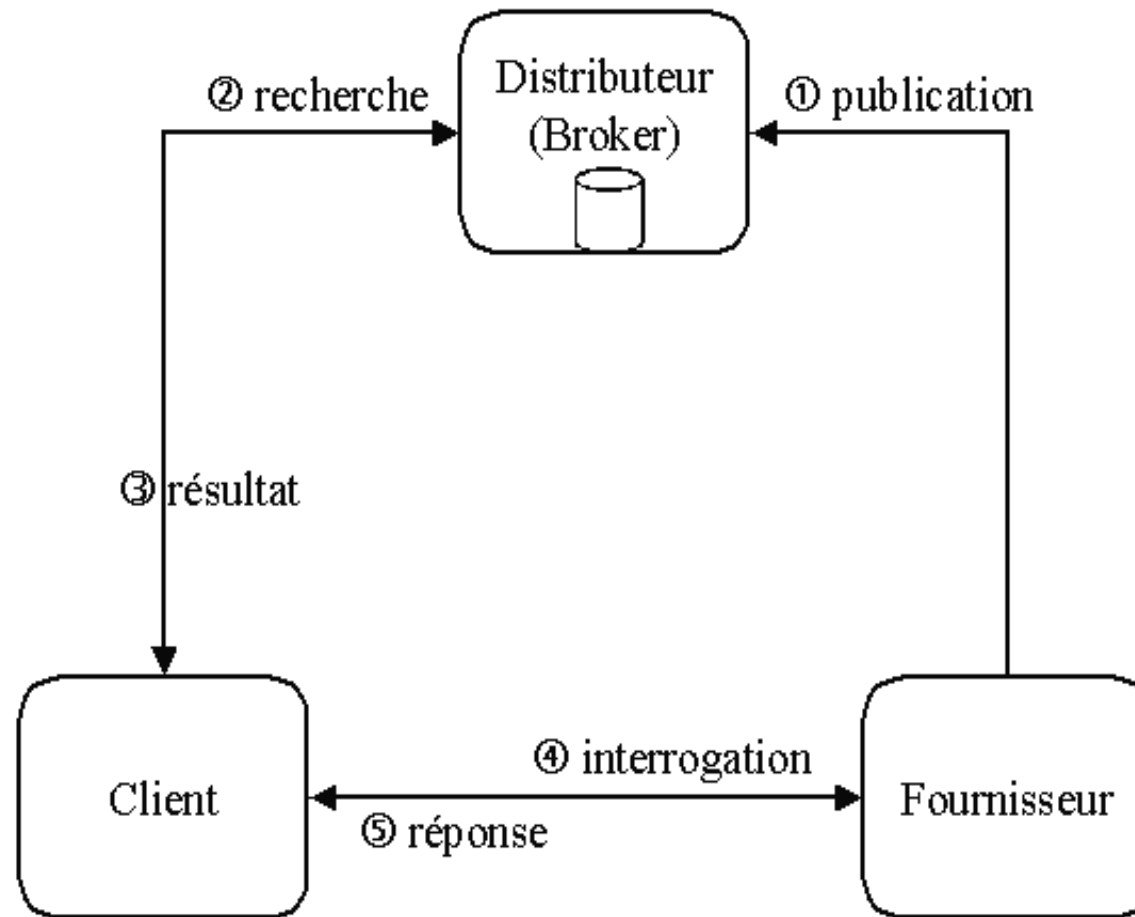
- Un nouveau Protocole : SOAP
 - Basé sur XML
 - Portabilité, Hétérogénéité
 - Porté sur des protocoles large échelle existants
 - HTTP, SMTP, ...
- Paradigme orienté service : WSDL
 - Définition de services offerts (en XML)
- Découverte automatique des services (dynamicité) : UDDI
 - Référentiel de Web Service (Pages Jaunes, Blanches, Vertes)

Service Web : comment ?

- Les services Web s'appuient sur un ensemble de protocoles standards
- **XML** : format utilisé pour décrire et échanger les données
- **SOAP** : protocole d'invocation d'un service Web
- **WSDL** : description XML de l'interface publique d'un service Web
- **UDDI** : centralisation des descriptions de services Web dans un référentiel commun utilisé pour rechercher un service ou pour publier un service

Standards du W3C (World Wide Web Consortium)

Service Web : mise en œuvre



XML

Rappels

XML

- eXtended Markup Language
- Notion de Métalangage
- Un document XML est constitué de deux entités distinctes :
 - Le fond (*le contenu*)
 - La forme (**la structure**), identifiée par des **balises** qui encadrent le contenu typé (les données)

```
<livre>
  <titre> le super livre </titre>
  <chapitre>
    <numero> 1 </numero>
    <titre> titre du chapitre 1 </titre>
    <paragraphe> blabla blabla </paragraphe>
  </chapitre>
  <chapitre>
    </chapitre>
  </chapitre>
</livre>
```

Principes

- D'où viennent les balises ?
 - Ensemble non-prédéfini et non fini de balises (pas comme html)
 - Chaque utilisateur peut définir ses propres balises (sémantique)
- Définir les balises, leur ordre et le typage de leur contenu = définir la grammaire du doc XML
- Grammaires standards :
 - MathML, SVG, XMI, ...

Comment écrit-on une grammaire ?

Langage de grammaire d'un doc XML

- DTD (Document Type Definition)
- Le langage XML Schéma
 - Permet de définir le schéma d'un doc XML
 - Les balises, leur ordre et le typage de leur contenu (= les données encadrées par les balises)
 - Un schéma est un doc XML !! (.xsd = XML Schema Description)
- Un document XML est dit valide lorsqu'il est conforme à une grammaire

Exemple de schéma : livre.xsd

- ```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="livre">
 <xs:complexType><xs:sequence>
 <xs:element name="titre" type="xs:string"/>
 <xs:element name="chapitre">
 <xs:complexType><xs:sequence>
 <xs:element name="numero" type="xs:int"/>
 <xs:element name="titre" type="xs:string"/>
 <xs:element name="paragraphe"
type="xs:string"/>
 </xs:sequence></xs:complexType></xs:element>
 </xs:sequence></xs:complexType></xs:element>
 </xs:schema>
```

**Balise pré-définie dans le langage XML Schema (mot du langage)**

**Nom d'une balise d'un doc XML valide à livre.xsd**

**Type de la balise**



# Autre exemple

- Le schéma `personne.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="personne">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="nom" type="xs:string"/>
 <xs:element name="prenom" type="xs:string"/>
 <xs:element name="date_naissance" type="xs:date"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

- Un doc XML valide: `personne.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<personne xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="personne.xsd">

 <nom>De Latour</nom>
 <prenom>Jean</prenom>
 <date_naissance>1967-08-13</date_naissance>

</personne>
```

Soit un document XML qui inclut des balises issues de différentes grammaires (i.e., +sieurs fichiers xsd)

- Ex : un livre de géométrie qui utilise
  - le schéma `livre.xsd` et
  - le schéma `svg.xsd`



Et si le schéma `livre.xsd` **ET** le schéma `svg.xsd` définissent chacun la balise `auteur` ??

```
<livre>
 <auteur>...</auteur>...
<figure><auteur>...</auteur></figure>
...
```

**Prbl de conflit de nommage des balises !!**

# Espaces de noms de balises

- Principe : similaire à celui de package en java ou en UML !
  - Espace de noms : unité de partitionnement qui sert d'espace de désignation
- Utilisation : il suffit d'utiliser le nom qualifié de la balise ! c-à-d de **préfixer** le nom de la balise par le nom logique de l'espace de noms
  - Ex java : un package P1 contient une classe C1 et P2 contient une autre classe qui s'appelle aussi C1 => P1.C1, P2.C1
  - Ex UML : un package P1 contient une classe C1 et P2 contient une autre classe qui s'appelle aussi C1 => P1::C1, P2::C1
  - Ex doc XML : la balise <auteur> de l'espace de noms ayant le nom logique nLivre se note <nLivre:auteur>
- Définition du nom logique : le nom logique d'un espace de noms utilisé dans un doc XML est défini dans n'importe quelle balise par un triplet : l'attribut xmlns, le nom logique et l'URI (identifiant Web du .xsd)  
`<nom_balise xmlns:nom_logique="URI">`

## Ex. d'un doc XML avec 2 espaces de noms

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<commande
 xmlns:produit="http://localhost/XML/produit"
 xmlns:client="http://localhost/XML/client">

 <produit:numero>p49393</produit:numero>
 <produit:nom>JXY Rasierer VC100</produit:nom>
 <produit:quantite>1</produit:quantite>
 <produit:prix>69</produit:prix>

 <client:numero>c2029</client:numero>
 <client:nom>Marius, Escartefigues</client:nom>
 <client:adresseslivraison>Cours Mirabeau 14, 13100 Aix en
 Provence</client:adresseslivraison>

</commande>
```

# Récapitulatif sur le langage XML Schema

- Permet d'écrire des grammaires (= schémas) de doc XML
  - Pré-définit un ensemble de « mots »
    - Des balises ordonnancées avec typage de contenu
  - Pré-définit un ensemble de types simples
    - string, byte, int, long, float, decimal, double, time, boolean, date,...
    - <http://xmlfr.org/w3c/TR/xmlschema-0/#simpleTypesTable>
- Écrire un schéma en langage XML Schéma, c'est
  - Écrire un doc XML
  - En utilisant les balises qui sont les mots du langage XML Schema

```
<xs:schema />
<xs:element />
<xs:complexType>
...

```
- Ces balises appartiennent à l'espace de noms  
<http://www.w3.org/2001/XMLSchema>

[Retour WS => SOAP](#)

# SOAP

Le Protocole de couche Application  
pour invoquer un service web

# SOAP

- Simple Object Access Protocol
- Protocole de transmission de messages pour l'invocation de services Web
  - Définit la structure des messages échangés par les applications via Internet
    - Format des messages défini en XML
  - Achemine les messages (en utilisant des protocoles standards sous-jacents)
    - HTTP pour des appels synchrones
    - SMTP ou bus MOM pour des appels asynchrones

# Structure d'un message SOAP (1)

- Un message SOAP est un document XML de la forme
  - Une déclaration XML (optionnelle), suivie de
  - Une Enveloppe SOAP (l'élément racine) : balise `Envelope`, qui est composée de :
    - Une En-tête SOAP (optionnelle) : balise `header`
      - Pour des info d'authentification ou de gestion de session
    - Un Corps SOAP : balise `body`



# Structure d'un message SOAP (2)

## la balise Envelope

`<soap:Envelope`

`xmlns:soap=«http://.....»`

Identificateur de l'espace de noms utilisé dans le doc (il peut y en avoir plusieurs)

`soap:encodingStyle=«http://.....»>`

Schéma définissant les règles de sérialisation des données typées (facultatif)

`<soap:Header> .....`  
`</soap:Header>`

`<soap:Body>`

`blabla`

`</soap:Body>`

Ex : appel de méthode

`</soap:Envelope>`

## Ex : un service d'addition de 2 entiers

- Soit un service :

```
int add(int a, int b)
```

- L'utilisation de ce service par un client générera l'acheminement sur le réseau de 2 msg SOAP
  - Un pour la requête et un pour la réponse

# Structure du message SOAP : la requête

```
<soapEnv:Envelope
 xmlns:serviceAddition="URImyAddition"
 xmlns:soapEnv="http://schemas.xmlsoap.org/soap/envelope/"
 <soapEnv:Body

 <serviceAddition:add>
 <arg0>5</arg0>
 <arg1>12</arg1>
 </serviceAddition:add>
 </soapEnv:Body>
</soapEnv:Envelope>
```

## Structure du message SOAP : la réponse

```
<soapEnv:Envelope
 xmlns:serviceAddition="URImyAddition"
 xmlns:soapEnv="http://schemas.xmlsoap.org/soap/envelope/"
 >
 <soapEnv:Body
 <serviceAddition:addResponse>
 <return>17</return>
 </serviceAddition:addResponse>
 </soapEnv:Body>
</soapEnv:Envelope>
```

# URImyAddition

```
<xs:schema version="1.0" targetNamespace="URImyAddition">
 <xs:element name="add" type="tns:add"/>
 <xs:element name="addResponse" type="tns:addResponse"/>

 <xs:complexType name="add">
 <xs:sequence>
 <xs:element name="arg0" type="xs:int"/>
 <xs:element name="arg1" type="xs:int"/>
 </xs:sequence>
 </xs:complexType>

 <xs:complexType name="addResponse">
 <xs:sequence>
 <xs:element name="return" type="xs:int"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

# Encodage

- Un message SOAP contient des données
    - Typées et interprétables par le service
- ⇒ Il faut définir un moyen d'encoder ces données qui soit compatible avec XML
- Encodage : représentation XML d'une donnée (valeur)
  - Décodage : reconstitution d'une valeur à partir de sa forme XML
  - La représentation XML d'une donnée est structurée en fonction du type de la donnée
- => Nécessaire de définir le type d'une donnée

# Définition de types

- Par utilisation de schémas (écrits en langage XML Schéma)
  - Pour décrire le type d'une donnée de **type simple**, c'est immédiat puisque le langage XML Schéma propose un système de types (int, boolean, string, ...).

- Voir l'exemple de `URImyAddition`

```
<xs:element name="arg0" type="xs:int"/>
```

```
<arg0>5</arg0>
```

- Pour décrire le type d'une donnée de **type complexe**, utilisation de la balise `<complexType>`

## Acheminement des msg : SOAP avec HTTP

- SOAP peut être facilement porté sur Http.
  - Convient au mode Request/Response de http
  - Le message SOAP est encapsulé dans une requête POST avec un content-type text/xml
  - Définition d'un header http : SOAPAction



# Msg SOAP encapsulé dans HTTP

POST /StockQuote HTTP/1.1

Host: www.stockquoteserver.com

Content-Type: text/xml; charset="utf-8"

Content-Length: nnnn

SOAPAction: "Some-URI"

```
<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="Some-URI">
 <symbol>DIS</symbol>
 </m:GetLastTradePrice>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# Msg SOAP encapsulé dans HTTP

HTTP/1.1 500 Internal Server Error

Content-Type: text/xml; charset="utf-8"

Content-Length: nnnn

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" />
 <SOAP-ENV:Body>
 <SOAP-ENV:Fault>
 < faultactor SOAP-ENV:Server / faultactor>
 < faultstring>Server Error< / faultstring>
 < /SOAP-ENV:Fault>
 < /SOAP-ENV:Body>
< /SOAP-ENV:Envelope>
```

# WSDL

Le langage de définition  
de l'interface d'un service Web

# WSDL

- Web Services Description Language
- Description en XML de l'interface publique d'utilisation des services Web (~ IDL des objets CORBA)
- Séparation entre description de la fonctionnalité abstraite du service et les détails concrets (où et comment la fonctionnalité est offerte)
  - Description abstraite des messages échangés entre un fournisseur de services et un client
  - Mise en relation, liaison (binding) de cette description avec un protocole (et donc le format des messages)

# Présentation

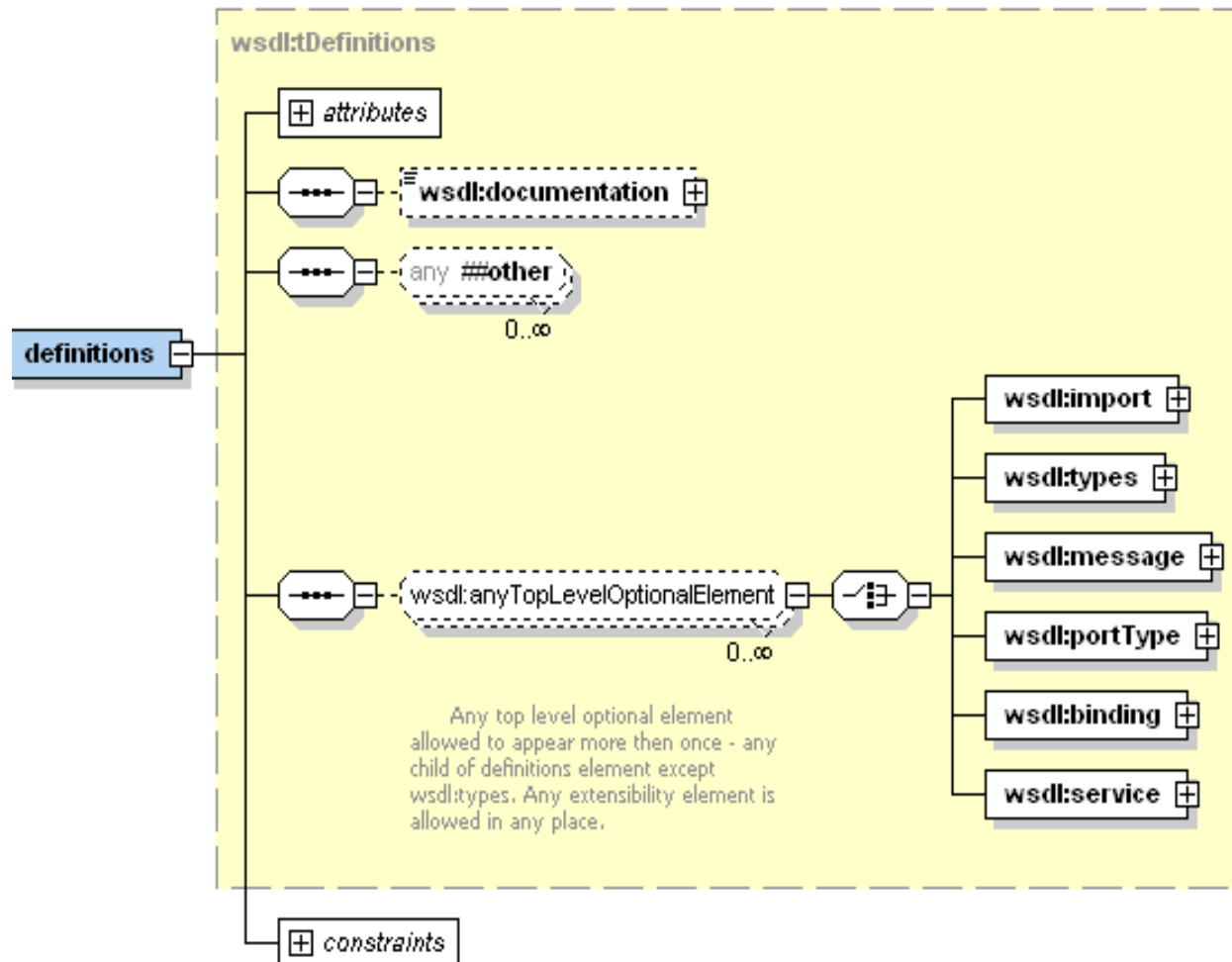
## Une description WSDL :

1. Décrit l'interface d'un service web (méthodes, types des paramètres)  
Cette description peut être comparée à la description IDL CORBA, elle peut servir à générer automatiquement des amorces.
2. Décrit les aspects techniques d'implantation d'un service web (quel est le protocole utilisé, quelle est l'adresse du service)  
Cette description sert à se connecter concrètement à un service web.

# Balises

- Une description WSDL est un document XML qui commence par la balise **definition** et contient les balises suivantes :
  - a – **types**: cette balise décrit les types utilisés
  - b – **message**: cette balise décrit la structure d'un message échangé
  - s –
    - t – **portType**: cette balise décrit abstraitement le service web sous forme d'un ensemble d'opérations (~ interface du service web)
      - a • **operation**: cette balise décrit une opération réalisée par le service web. Une opération reçoit des messages et envoie des messages.
    - i – **binding** : cette balise décrit la liaison entre un protocole (http) et la description abstraite du service (= le portType).
    - t – **service**: cette balise décrit un service comme un ensemble de ports.
      - c • **port**: cette balise décrit un port de communication au travers duquel il est possible d'accéder à un ensemble d'opérations. Un port référence un Binding

# Les balises (graphique XML Spy du wsdl.xsd)



# types

- Exemple de description du type `personne`

```
<wsdl:types>
 <xs:schema
 targetNamespace="http://www.exemple.fr/personne.xsd"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="personne">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="nom" type="xs:string" />
 <xs:element name="prenom" type="xs:string" />
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 </xs:schema>
</wsdl:types>
```



## types (suite)

- Pas obligé de définir les types au sein même du `.wsdl`, on peut en importer

```
<types>
 <xsd:schema>
 <xsd:import namespace="http://session.mpg/"
 schemaLocation="http://localhost:8080/location
DuXSD"/>
 </xsd:schema>
</types>
```

# message

- Les messages sont envoyés entre le service et son client (et réciproquement !)
  - ex: une opération reçoit des messages et envoie des messages
- Un message peut avoir plusieurs paramètres appelés **parts**
- Les paramètres sont typés

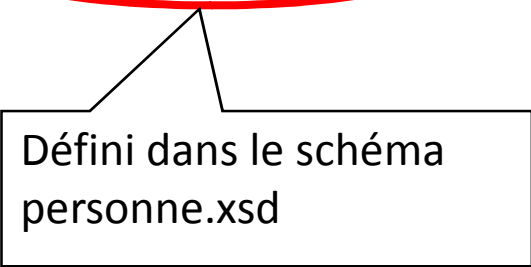
# Paramètres de message

- Paramètre de type simple

```
<wsdl:message name="personneMsg">
 <wsdl:part name="nom" type="xsd:string" />
 <wsdl:part name="prenom" type="xsd:string" />
</wsdl:message>
```

- Paramètre de type complexe (type défini dans un schéma)

```
<wsdl:message name="personneMsg">
 <wsdl:part name="personne" element="exemple:personne" />
</wsdl:message>
```



Défini dans le schéma  
personne.xsd

# portType

- Description abstraite du service Web sous forme d'un ensemble d'opérations
- Un portType a un nom (attribut `name` de la balise `portType`)

```
<wsdl:portType name="gererPersonnes" >
 <wsdl:operation name="getPersonne" >
 ...
 </wsdl:operation>
 <wsdl:operation name="setPersonne" >
 ...
 </wsdl:operation>
</wsdl:portType>
```

# operation

- WSDL définit 4 catégories d'opération :
  - One-Way : une opération qui reçoit des messages mais n'en émet pas
  - Request-response : une opération qui reçoit des messages puis renvoie des messages
  - Solicit-response : une opération qui envoie des messages puis en reçoit
  - Notification : une opération qui envoie des messages mais n'en reçoit pas

## operation

- Quelque soit la catégorie d'opérations, la définition est sensiblement la même :
- Une opération :
  - Reçoit des messages : `<wsdl:input ...>`
  - Envoie des messages : `<wsdl:output ...>` ou `<wsdl:fault ...>`
- La catégorie de l'opération détermine la présence et l'ordre des input/outputs/fault

# operation

```
<wsdl:operation name="operation_name">
 <wsdl:input name="nom_optionel" message="nom_message" />
</wsdl:operation>
```

```
<wsdl:operation name="operation_name">
 <wsdl:input name="nom_optionel" message="nom_message" />
 <wsdl:output name="nom_optionel" message="nom_message" />
 <wsdl:fault name="nom_optionel" message="nom_message" />*
</wsdl:operation>
```

```
<wsdl:operation name="operation_name">
 <wsdl:output name="nom_optionel" message="nom_message" />
 <wsdl:input name="nom_optionel" message="nom_message" />
 <wsdl:fault name="nom_optionel" message="nom_message" />*
</wsdl:operation>
```

# binding

- Liaison d'une description abstraite (portType) à un protocole
- Chaque opération d'un portType peut être liée de manière différente (à un protocole distinct)
- Protocoles standardisés pour les liaisons
  - SOAP
  - HTTP
  - MIME



# binding

- Une liaison a
  - Un nom : attribut (optionnel) `name` de la balise `binding`
  - Un type : attribut `type` de la balise `binding`
    - Le type identifie le `portType` (i.e., le nom du type de la liaison est le nom du `portType`)

```
<wsdl:binding name="binding_name"
 type="nom du portType" >
```

...

```
</wsdl:binding>
```

## Le cas du binding SOAP

- Pour préciser que la liaison est de type SOAP, il faut inclure la balise suivante :

```
<soap:binding transport="uri" style="soap_style" />
```

- L'attribut *transport* de la balise définit le protocole qui encapsule les messages SOAP
  - Ex : pour HTTP, la valeur de l'attribut `transport` est `http://schemas.xmlsoap.org/soap/http`
- L'attribut *style* identifie le mode de traduction d'une liaison en messages SOAP (comment créer les messages SOAP à partir des définitions WSDL des opérations ?)  
Deux valeurs possibles :
  - `rpc`
  - `document`

## binding SOAP

- Pour chaque opération du portType :
  - Il faut préciser l'URI de l'opération : `soapAction`
  - On peut (pas obligatoire) repréciser la façon dont sont créés les messages SOAP : `style`
- Pour chaque message input/output/fault de chaque opération, il faut définir le mode d'usage (encodage d'un message WSDL en un message SOAP)  
Deux valeurs possibles de l'attribut `use` de la balise `body`
  - `encoded`
  - `literal`

Pour aller plus loin...

`http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/`

# service

- Un service est accessible à un port
- Un port : point d'accès au service
  - Référence une liaison (attribut binding dont la valeur doit être le nom de la liaison)
  - A une adresse (qui correspond à l'adresse http)

```
<wsdl:service name="MonService">
 <wsdl:port binding="intf:MonServiceSoapBinding">
 <soap:address
 location="http://mda.lip6.fr:8080/axis/services/MonService"/>
 </wsdl:port>
</wsdl:service>
```

# Résumé du vocabulaire WSDL

- Modèle logique (abstrait)
  - Un **message** = description abstraite des données transmises lors d'un échange
  - Une **opération** = groupement logique de messages
  - Un **portType** = une collection d'opérations
- Correspondance logique-physique : **binding**
  - description concrète, i.e., protocole et format de données qui implantent un portType
- Modèle physique
  - Un **port** = point d'accès au service défini par une liaison et une adresse réseau
  - Un **service** = un ensemble de ports

# Example

```
<?xml version="1.0" ?>
 <definitions name="CurrencyExchangeService"
 targetNamespace="http://www.xmethods.net/sd/CurrencyExchangeService
 .wsdl" »
 xmlns:tns="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl
 »
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" »
 xmlns="http://schemas.xmlsoap.org/wsdl/">
 <message name="getRateRequest">
 <part name="country1" type="xsd:string" />
 <part name="country2" type="xsd:string" />
 </message>

 <message name="getRateResponse">
 <part name="Result" type="xsd:float" />
 </message>

 <portType name="CurrencyExchangePortType">
 <operation name="getRate">
 <input message="tns:getRateRequest" />
 <output message="tns:getRateResponse" />
 </operation>
 </portType>
```

# Exemple (suite)

```
<binding name="CurrencyExchangeBinding" type="tns:CurrencyExchangePortType">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
 <operation name="getRate">
 <soap:operation soapAction="" />
 <input>
 <soap:body use="encoded"
 namespace="urn:xmethods-CurrencyExchange" >
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
 </input>
 <output>
 <soap:body use="encoded"
 namespace="urn:xmethods-CurrencyExchange" >
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
 </output>
 </operation>
</binding>

<service name="CurrencyExchangeService">
 <documentation>Returns the exchange rate between the two
 currencies</documentation>
 <port name="CurrencyExchangePort" binding="tns:CurrencyExchangeBinding">
 <soap:address location="http://services.xmethods.net:80/soap"
 />
 </port>
</service>
</definitions>
```



# UDDI

## L'annuaire des Web Services

# UDDI

- Universal Description, Discovery and Integration
- UDDI offre des services sous forme de WS
- Rôles:
  - Pages Blanches: informations sur les fournisseurs de services
  - Pages Jaunes: critères et options de catégorisation de services
  - Pages Vertes: les WSDL

# UDDI

- L'ambition d'avoir un annuaire universel n'a malheureusement pas pris!
- On lui préfère plutôt la notion d'annuaire privés

# WS: Mise en œuvre

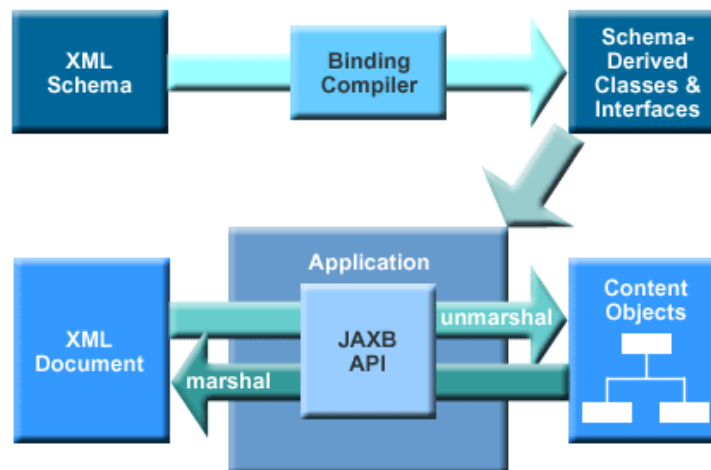
## Axis 2 et JAX-WS

## Axis: Principes

- Implémentation Apache de SOAP
- Support de la spécification JAX-WS
- Servlet coté serveur qui écoute les messages SOAP
- Lors du déploiement, les méthodes offertes par l'objet sous forme de web service sont déclarées à la servlet AxisServlet
- API coté client pour échanger les messages SOAP sur HTTP ou bien SMTP
  - Coté serveur, le démon reçoit les messages SMTP et les renvoie sur HTTP à AxisServlet

# JAX-WS

- Java API for XML Web Services, partie de JEE
- Utilise les annotations afin de simplifier le développement de services web et leur déploiement coté client
- Se base sur JAXB (Java Architecture for XML Binding) pour le mapping des types Java vers les types XML Schema



## JAX-WS: Annotations

- Développer un service web avec JAX-WS consiste à prendre un POJO( Plain Old Java Object) ou un EJB3 et à le décorer par l'une des annotations suivantes:

@WebService

@WebMethod

@OneWay

@WebParam

@WebResult

Etc.

## @WebMethod, @WebParam & @WebResult

- @WebMethod: Expose une méthode comme opération du service web
- @WebParam: Personnalise le mapping d'un paramètre d'un message Web Service (i.e., part)
- @WebResult: Personnalise le mapping entre le retour d'une opération du service web et l'élément lui correspondant dans WSDL généré



# Exemple: Exposer un WS à partir d'un EJB

```
package org.lip6.fr.ws;
import javax.ejb.Remote;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

@Remote
@WebService(name="Calculatrice",targetNamespace="http://org.lip6.fr/calculatrice")
public interface CalculatriceServiceRemote {

 @WebMethod(action="urn:additionner",operationName="add")
 public @WebResult(name="Resultat")
 double additionner(@WebParam(name="Operande1")
 double operande1,
 @WebParam(name="Operande2")
 double operande2);
 @WebMethod(action="urn:multiplier",operationName="multiplier")
 public @WebResult(name="Resultat")
 double multiplier(@WebParam(name="Operande1") double operande1,
 @WebParam(name="Operande2") double operande2);
}
```

# @WebService

- Sert à définir le contrat de service
- Peut être associée à l'interface ainsi qu'à l'implémentation

```
package org.lip6.fr.ws;
import javax.ejb.Stateless;
import javax.jws.WebService;
```

```
@Stateless
@WebService(targetNamespace="http://org.lip6.fr/calculatrice", endpointInterface=
"org.lip6.fr.ws.CalculatriceServiceRemote")
```

```
public class CalculatriceService implements CalculatriceServiceRemote {

 public double additionner(double operande1, double operande2) {

 return operande1 + operande2; }

 public double multiplier(double operande1, double operande2) {

 return operande1 * operande2; }
}
```

# Les approches de Développement

- Principalement, 3 démarches de développement:
  - 1) Partir du code Java et générer le WSDL (i.e. Code First)
    - Utilisation de l'utilitaire **wsgen** par exemple
  - 2) Générer le code Java à partir d'un WSDL fourni (i.e., Contract first)
  - 3) L'approche « meet in the middle »

Le choix de la bonne approche dépendra bien sûr des besoins de l'application

## Code First et l'utilitaire wsgen

- Utilitaire fourni dans le JDK
- Génère les artefacts portables nécessaires aux applications JAX-WS lors du démarrage à partir du code Java
- Génère le WSDL à la demande
- Génère toute classe requise pour assembler/ désassembler les contenus d'un message

## Contract First et l'utilitaire wsimport

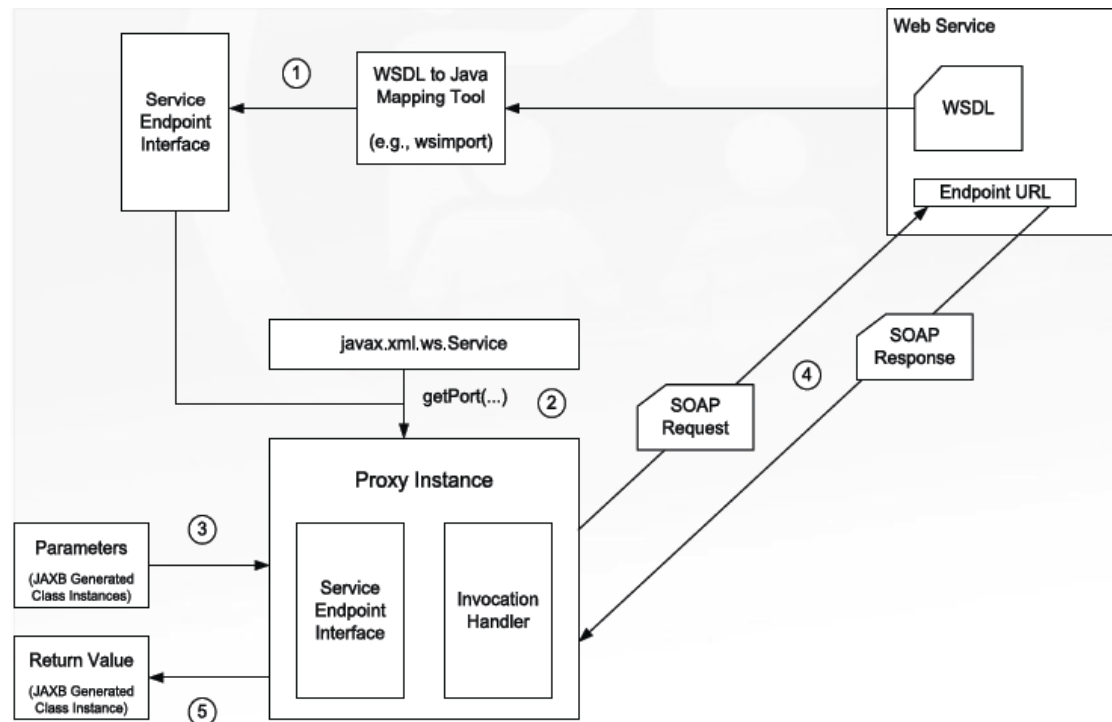
- L'utilitaire wsimport prend en entrée le WSDL et vous génère une interface pour chaque portType
- Vous n'aurez par la suite qu'à créer une classe qui implémente cette /ces interface (s) pour y fournir votre logique métier
- Déployez l'implémentation de votre service web dans un conteneur JAX-WS

## Meet in the Middle

- En entrée un WSDL valide et un existant Java (classes, interfaces)
- Objectif: trouver un compromis entre les deux. Souvent on a recours à des design patterns, type adaptateur

# Implémentation du Client

- L'implémentation standard de JAX-WS fourni avec l'utilitaire wsimport un moyen pour la génération du code Java nécessaire à la communication avec un Web Service



## Problème de Mapping avec JAXB

- JAXB offre un mapping par défaut des types Java en types XML schéma
- En parle de marshaling, unmarshaling pour l'encodage / décodage des objets java / xml
- Peut être source de problèmes
- JAXB fournit des annotations pour définir un mapping personnalisé



## Conclusions

- Les WS sont devenus incontournables dans les applications d'aujourd'hui
- Tout le monde autour de nous dit que les WS c'est facile mais pas si évident que ça en réalité
  - Si ça ne marche pas du premier coup, difficile d'identifier le problème
- Très bien outillé, documenté
- Et REST dans tout ça ??

# Lectures

- Java Web Services: Up and Running de Martin Kalin,  
Editeur : O'Reilly Media, Inc, USA; Édition : 1 (27 février 2009),  
ISBN-10: 059652112X
- Service Design Patterns: Fundamental Design Solutions for  
SOAP/WSDL and RESTful Web Services de Robert Daigneau,  
Editeur : Addison Wesley; Édition : 1 (25 octobre 2011), ISBN-  
10: 032154420X