Services WEB

(SOAP / WSDL , REST , JAX-WS , JAX-RS, CXF, ...)

Table des matières

I - Services WEB (présentation, concepts).....6

1. Principes et objectifs des Services WEB.....6
   1.1. Interopérabilité 6
2. Deux grands types de WS (REST et SOAP).....7
3. WebService WS-* (XML / SOAP).....9
   3.1. Accès lointains (autres entreprises, ...).....9
   3.2. Modes synchrones et asynchrones.....9
   3.3. Utilisation possible des services web (scénarios).....9
4. Techniques et protocoles: XML, SOAP, WSDL, UDDI.....10

II - SOAP et WSDL (essentiel).....11

1. Protocole SOAP 11

1.1. SOAP on HTTP en mode synchrone (RPC)....11
1.2. SOAP on JMS (ou SMTP) en mode asynchrone 11
2. Eléments du protocole SOAP....11
3. WSDL (Web Service Description Langage)....14
4. "Basic profile" pour services "web"....15
5. "Style/use" SOAP (rpc/literal , document/literal)....16
5.1. "rpc/encoded" et "rpc/literal"....17
5.2. "document/literal" et variantes ("wrapped" , "bared" , ...)....17
5.3. Versions et évolutions de SOAP/WSDL....19
6. Détails sur WSDL....20
6.1. Structure générale d'un fichier WSDL....20
6.2. Exemple de fichier WSDL 20
6.3. WDSL abstraits et concrets....22

III - Web service "REST" (pas SOAP)....24
1. Web Services "R.E.S.T."....24
1.1. REST....24
2. Format "J.S.O.N."....25
2.1. Exemple (équivalence JSON / XML):....26

IV - API java pour services WEB (présentation)....27
1. Api fondamentales (liées à SOAP,WSDL)....27
1.1. Api liées à J2EE 1.4 (déconseillées avec JEE5)....27
1.2. Nouvelles API (jdk 1.5 / 1.6, JEE 5 , ...)....27
2. API complémentaires....28
2.1. API Standards officialisées par Java/Sun....28
2.2. API non officielles - standards de fait....28
3. Technologies "java" pour la mise en oeuvre....28

V - Essentiel JAX-WS (API et Mise en oeuvre)....29
1. Présentation de JAX-WS....29
2. Mise en oeuvre de JAX-WS (coté serveur)....29
2.1. Implémentation sous forme d'EJB3 (pour serveur JEE5 tel que JBoss4.2 ou 5 , Glassfish de Sun , Geronimo 2 d'apache , Jonas d'OW2 ou ...)....32
2.2. Test sans serveur et wsgen (jdk 1.6)....32
2.3. Lancement mini serveur sans Container Web pour Tests....32
3. Tests via SOAPUI....33
Utilisation de JAX-WS coté client....34
3.1. Mode opératoire général (depuis WSDL)....34
3.2. Génération du proxy jax-ws avec wsimport du jdk >=1.6....35
4. Redéfinition de l'URL d'un service à invoquer 35

4.1. Redéfinition de l'URL "SOAP"....35
4.2. Redéfinition de l'URL WSDL (et indirectement SOAP)....35
5. Client JAX-WS sans wsimport et directement basé sur l'interface java....36
5.1. avec le jdk 1.6 (sans CXF ni Spring)....36
6. CXF....37
6.1. Présentation de CXF (apache)....37
6.2. implémentation avec CXF (et Spring) au sein d'une application Web (pour Tomcat 5.5, 6 ou 7)....38
6.3. Client JAX-WS sans wsimport avec CXF et Spring....39
6.4. avec CXF sans Spring....40

VI - JAX-RS (Api Java pour WS REST)....41
1. API java pour REST (JAX-RS)....41
1.1. Exemple JAX-RS....41
1.2. Mise en oeuvre de JAX-RS avec CXF....42
1.2.a. REST Service....45
Single user REST output....45

VII - Types complexes et exceptions SOAP....46
1. Types de données élémentaires bien supportés....46
2. Types de données composés (bien gérés)....46
3. Tableaux et collections (interopérabilité à tester)....47
4. Utilitaire "tcp-monitor" pour intercepter et tracer les messages SOAP....47
5. Transfert des exceptions via SOAP (Fault)....49

VIII - Sécurité pour Services WEB....52
1. Gestion de la sécurité / Services Web 52
1.1. Eléments de sécurité nécessaires sur les services WEB....52
1.2. différents types d'authentifications (WS)....53
1.3. authentification élémentaire / fonctionnelle via "jeton"....53
1.4. Authentification véhiculée via l'entête (header) "HTTP"....54
1.5. Authentification véhiculée via l'entête (header) "soap"....54
1.6. Aspects techniques gérés par intercepteurs....55
1.7. Problématique (sécurité avancée)....55
1.8. Web Service Security (WS-S)....56
1.9. Aspects avancés liés à la sécurité....56
2. Authentification basique http avec JAX-WS....56
2.1. Coté client....56
2.2. Coté serveur (avec CXF)....57

IX - Annuaires de services (UDDI , ...)....60
1. Utilité d'un annuaire de "service WEB"....60

2. Annuaires UDDI (présentation)....61
3. Annuaires publics et annuaires privés....62
3.1. Principaux annuaires publics....62
3.2. Quelques implémentations d'annuaires UDDI....62
4. Structure d'un annuaire UDDI....63
4.1. Infrastructure physique d'un annuaire UDDI....63
4.2. Structure logique d'un annuaire UDDI....64
5. taxonomies....67
5.1. UNSPSC....67
5.2. ISO 3166 Geographic Taxonomy....69

X - Schémas XML (XSD)....71
1. Présentation des schémas W3C....71
2. Lien entre un document XML et un schéma....72
3. Structure d'un schéma....73
3.1. Terminologie (types simples et complexes)....73
3.2. Structure globale d'un schéma (.xsd)....73
3.3. Exemple complet (pour vue d'ensemble)....74
3.4. Types simples prédéfinis....75
3.4.a. Principaux types prédéfinis:....75
3.4.b. Autres types prédéfinis:....75
3.5. Types simples personnalisés....76
3.6. Types complexes....77
3.6.a. Syntaxe générale (élément composé de sous élément(s) et d'attribut(s))....77
3.6.b. Détails sur les attributs....77
3.7. Syntaxes possibles mais un peu moins lisibles....78
3.7.a. Type complexe sans nom et décrit d'une façon imbriquée....78
3.7.b. Référence vers un élément définit au niveau global....78
3.8. Types complexes particuliers....79
3.8.a. Dériver un type complexe d'un type simple 79
3.8.b. type complexe pour élément mixte....79
3.8.c. type complexe pour élément vide....79
4. Aspects avancés des schémas....80
4.1. Prise en compte des namespaces au sein d'un schéma....80
4.2. 1.1.1 Notion de groupe....82
4.3. 1.1.1 Inclusion ou importation d'un schéma dans un autre....83
4.4. Structures de données avancées....83
4.4.a. Héritage entre types complexes (extensions)....83
4.4.b. Références (liaisons)....83

XI - JAXB2....86
1. Principes - JAXB....86
2. Versions 1 et 2 (JAXB)....87
3. Mise en oeuvre (JAXB2)....87

4. Exemple de binding (JAXB2).....87
5. Structure de l'api JAXB.....90
6. Exemple de code (JAXB2).....91

I - Services WEB (présentation, concepts)

1. Principes et objectifs des Services WEB

Services WEB

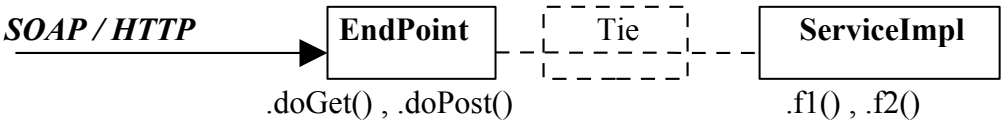
- **L'invocation d'un service** *(connexion vers son point d'accès, protocole d'appel)* **est indépendante de son implémentation** (*ex: code C++, EJB, JavaBean coté Web , .NET, ...*).
==> bonne **interopérabilité** .

Un service WEB est un service qui est invoqué via les technologies du WEB :

- HTTP sert généralement de protocole de transport.
- XML permet d'encoder les requêtes et les réponses.
- Les éventuelles pièces jointes ont des types "MIME".

Service Endpoint (point d'accès)

La classe d'implémentation effective des fonctions du service (ex: EJB , JavaBean coté web , ...) est en fait distincte de son point d'accès sur le réseau TCP/IP (ex: servlet générique).



1.1. Interopérabilité

Enfin une technologie permettant de faire communiquer facilement les mondes suivants:

- Microsoft (COM+ , .Net , ...)
- PHP / LAMP
- Java (RMI over IIOP , ...)
- C/C++ d'origine Unix
- Mac

Car tous ces mondes sont capables de gérer le protocole HTTP et de générer/interpréter des requêtes au format XML.

2. Deux grands types de WS (REST et SOAP)

2 grands types de services WEB: SOAP/XML et REST

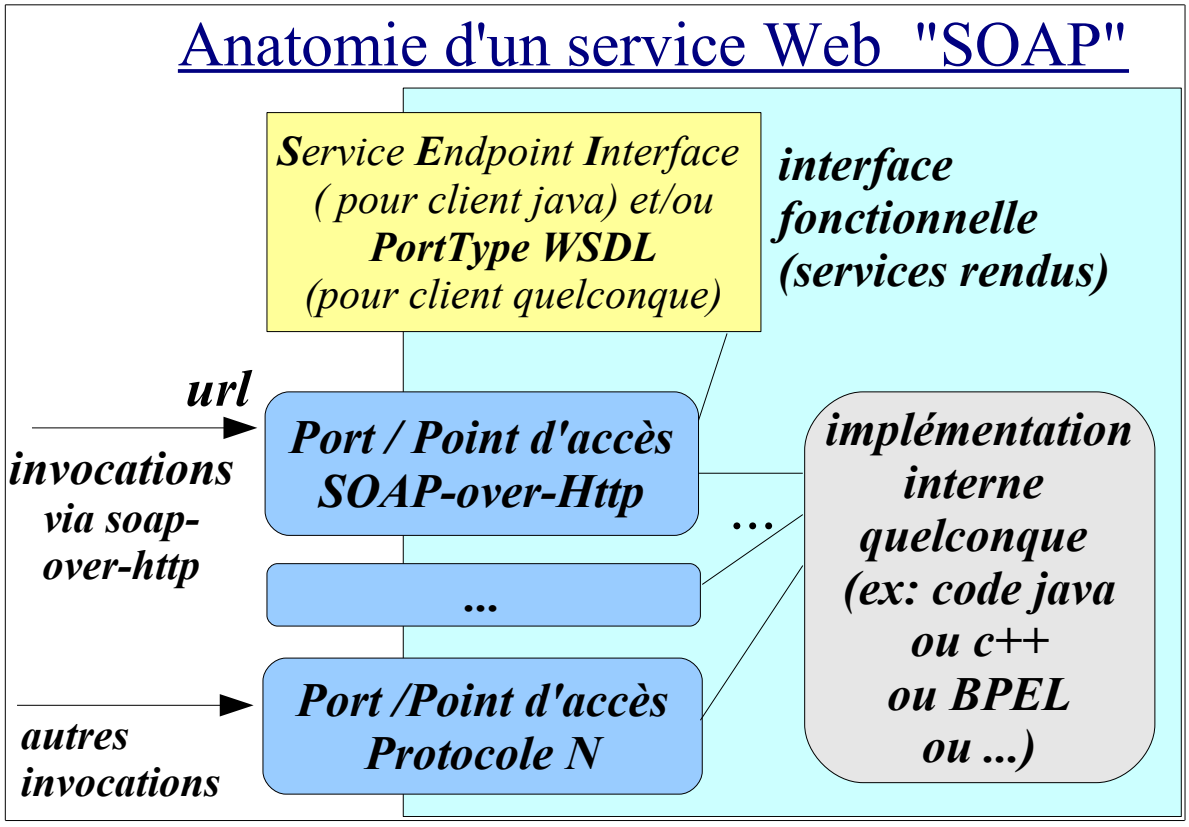
WS-* (SOAP / XML)

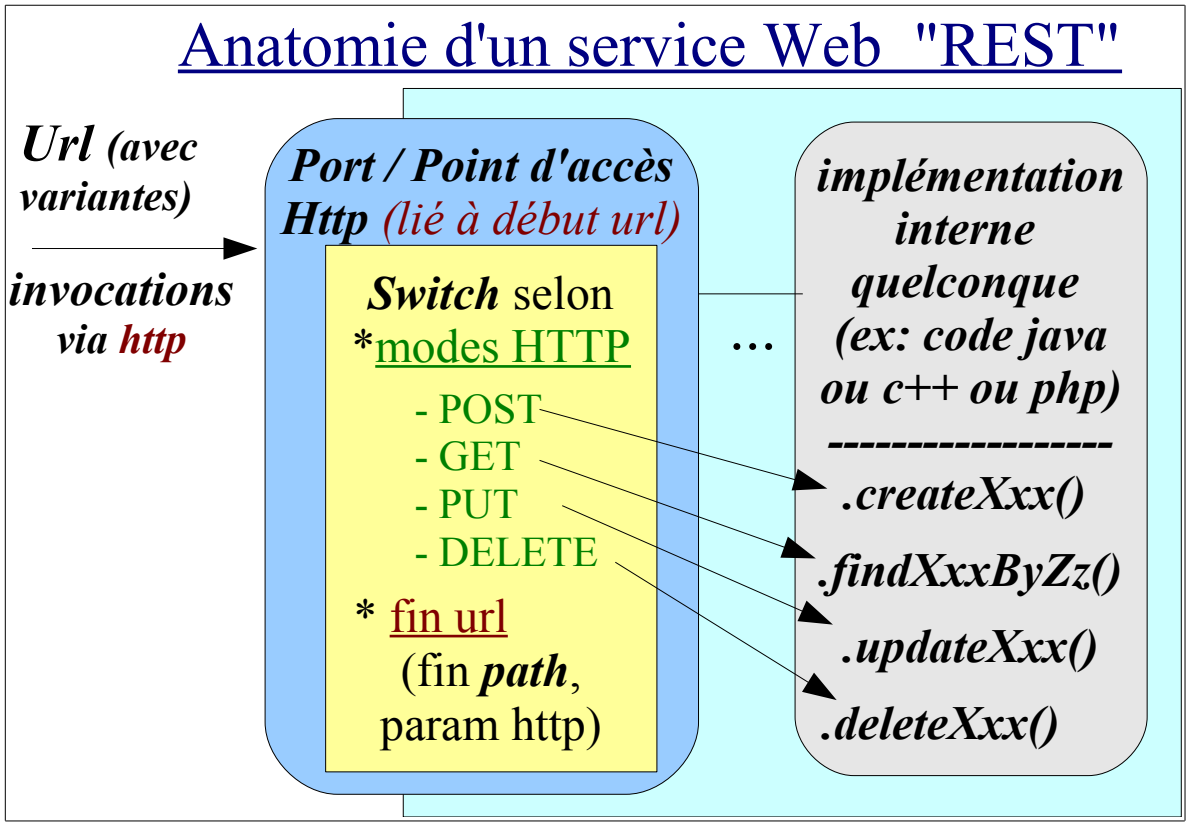
- "**Payload**" systématiquement en **XML** *(sauf pièces attachées / HTTP)*
- **Enveloppe SOAP** en XML *(header facultatif pour extensions)*
- Protocole de transport au choix (HTTP, JMS, ...)
- Sémantique quelconque , **description WSDL**
- Plutôt orienté SOA / proche "back office"

REST (HTTP)

- **"Payload" au choix (XML , HTML , JSON, ...)**
- Pas d'enveloppe imposée
- **Protocole de transport = toujours HTTP.**
- Sémantique "CRUD" **(*modes http* PUT,GET,POST,DELETE)**
- Plutôt orienté Web/Web2 et proche "front office"

Anatomie d'un service Web "SOAP"




Anatomie d'un service Web "REST"
Url (avec variantes)
invocations via http
Port / Point d'accès Http (lié à début url)
Switch selon *modes HTTP
- POST
- GET
- PUT
- DELETE
* fin url (fin path, param http)
...
implémentation interne quelconque (ex: code java ou c++ ou php)
.createXxx()
.findXxxByZz()
.updateXxx()
.deleteXxx()

3. WebService WS-* (XML / SOAP)

3.1. Accès lointains (autres entreprises, ...)

Reposant sur les technologies aujourd'hui bien éprouvées du WEB , le protocole SOAP permet d'établir facilement des communications à longues distances (vers d'autres entreprises ou ....) car les requêtes HTTP sont par défaut laissées passer par les "proxy HTTP" et les firewalls .

3.2. Modes synchrones et asynchrones

Reposant sur une logique de:
- ***traitement de messages entrants***
  et
- d'***éventuels envois de messages de réponses***

le protocole **SOAP** peut être utilisé
- en ***mode synchrone*** (attente immédiate d'un message de retour au sein d'un canal de communication bidirectionnel . ***Ex***: *appel de procédure distante*)
  ou bien
- en ***mode asynchrone*** (on poste un message dans une file d'attente et l'on traitera une éventuelle suite plus tard).

3.3. Utilisation possible des services web (scénarios)

L'utilisation d'un **service Web** revient à **déléguer une partie des traitements** vers une autre entité (autre entreprise, autre service, ...) éventuellement très éloignée .
Exemples:
- Site de commerce électronique déléguant une conversion de devise (ex: Euro en Yen) à un service Web d'une société surveillant régulièrement les taux de change.
- Site d'une agence de voyage qui interagit avec des services Web de réservations (transport , hôtel, ...) .

NB: Le résultat d'un service Web doit généralement être retraité par un programme (ex: applet , servlet , ...) de façon à générer une représentation affichable (*ex*: HTML).

4. Techniques et protocoles: XML, SOAP, WSDL, UDDI

Technologies	*Fonctionnalités*
XML	Socle de base pour toutes les autres technologies du WEB (==> W3C)
SOAP	Protocole réseau qui permet d'échanger des messages au format XML via HTTP (ou autre). Un message Xml envoyé via SOAP comporte fréquemment un appel à une méthode/fonction distante avec ses différents paramètres [idem pour le retour]
WSDL	Décrit la structure d'un service WEB (liste des méthodes/fonctions invocables , liste des paramètres avec noms et types , ...) . Cette définition de la structure d'un service est encodée en XML et est ainsi ré-interprétable par n'importe quel langage de programmation récent/évolué (ex: Java,C/C++,Php,C#, ...) -- > équivaut un peu à la notion "IDL" ou d'interface abstraite .
UDDI	Permet d'enregistrer et rechercher au sein d'annuaires UDDI des informations (ex: URL exacte, ...) sur des services Web effectifs/existants qui implémentent à leurs manières des services dont l'interface est décrite par un fichier WSDL abstrait . ==> recherches généralement possibles par métiers , par marques, par lieux géographiques , ...

NB:
SOAP et **WSDL** sont les 2 grandes technologies de base sur les services WEB. Bien les connaître est vraiment fondamental.

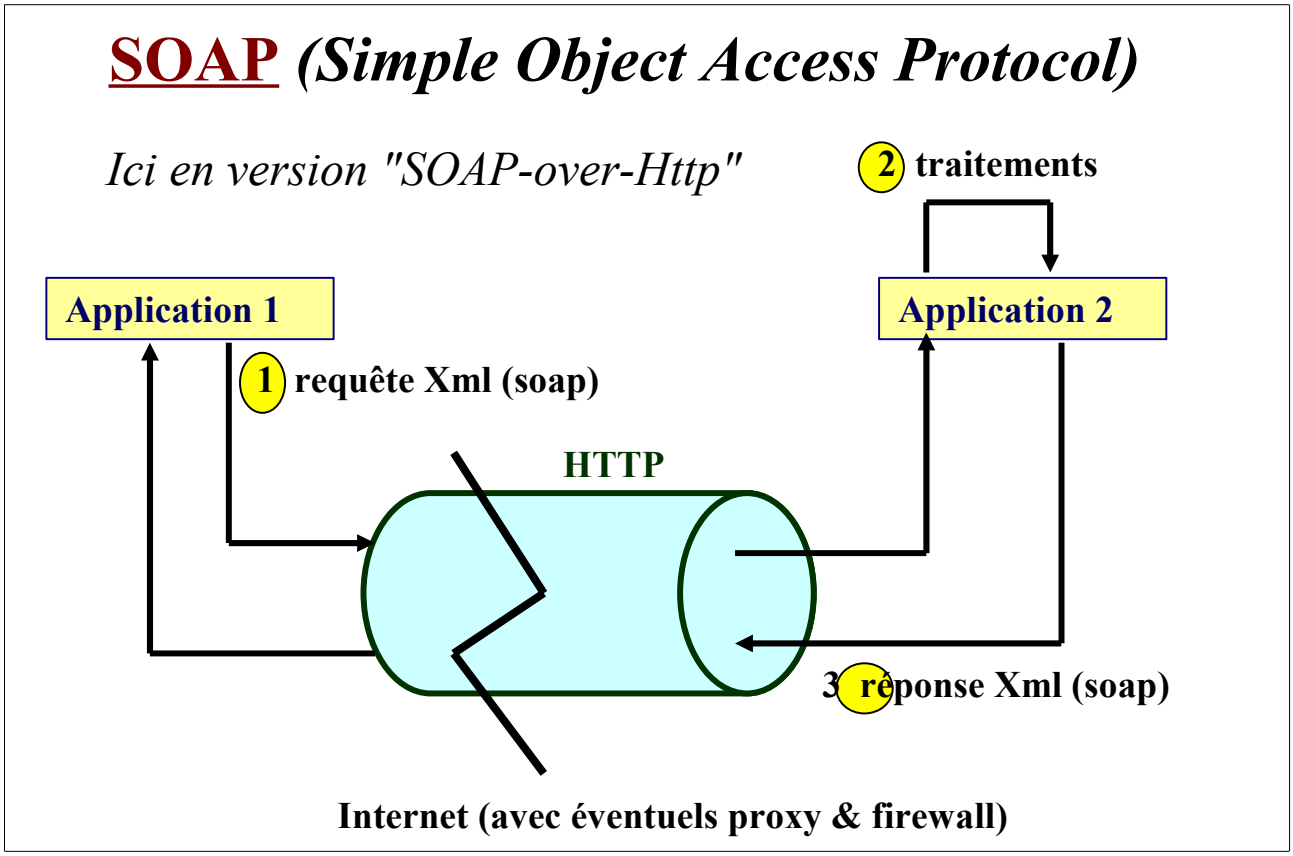
Le protocole **REST** est un concurrent de SOAP qui est plus léger et directement basé sur Http. Bien que moins normalisé , REST est quelquefois utilisé pour obtenir un certain gain en performance ou bien au niveau de la partie "front office". **REST** est très souvent utilisé avec PHP et JavaScript / JSON.

UDDI est une technologie lourde (complexe et difficile à appréhender) . UDDI n'a d'intérêt que dans un environnement large et internationalisé et est bien souvent inutile dans des structures ordinaires.

II - SOAP et WSDL (essentiel)

1. Protocole SOAP

1.1. SOAP on HTTP en mode synchrone (RPC)



1.2. SOAP on JMS (ou SMTP) en mode asynchrone

L'enveloppe SOAP véhiculée est la même qu'avec SOAP over HTTP.
Seul le protocole de transfert est variable ==> SMTP , ...

Un mode asynchrone avec de véritable "files d'attentes" ou ("mail box") peut quelquefois est très pratique.

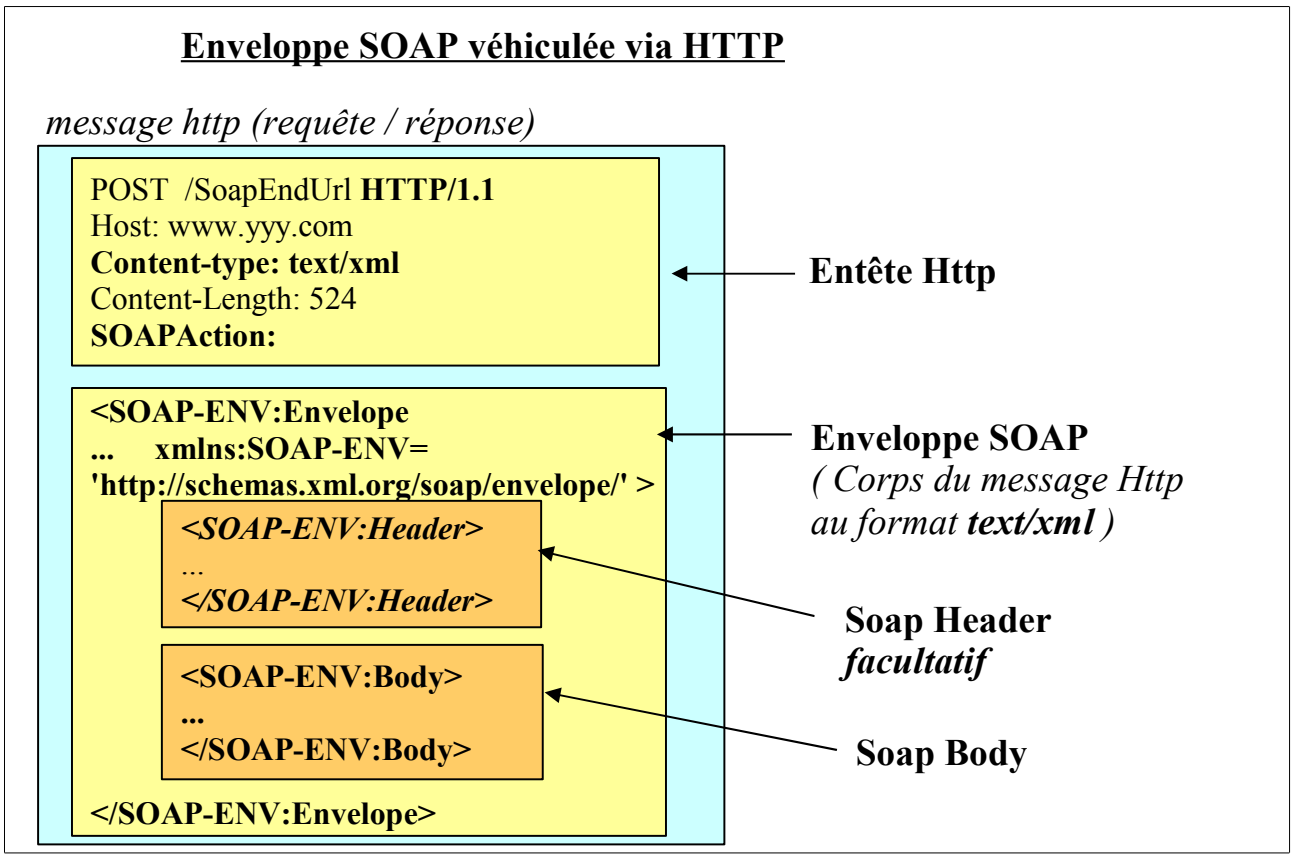
2. Eléments du protocole SOAP

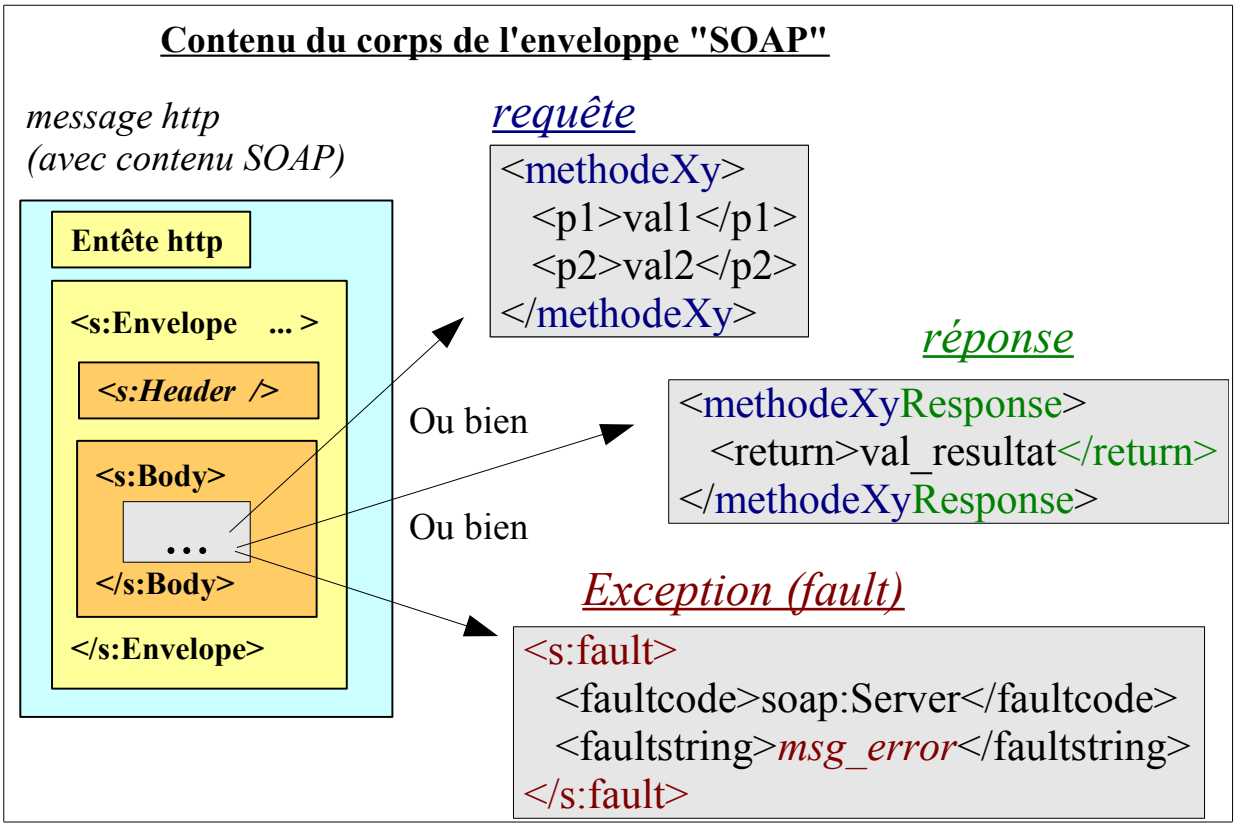
Bien que le sigle **S.O.A.P.** sous entende principalement l'accès distant à un objet de façon à y invoquer des fonctions, le protocole **SOAP** est avant tout basé sur le concept d'**envoi de messages** (message de requête, message de réponse).

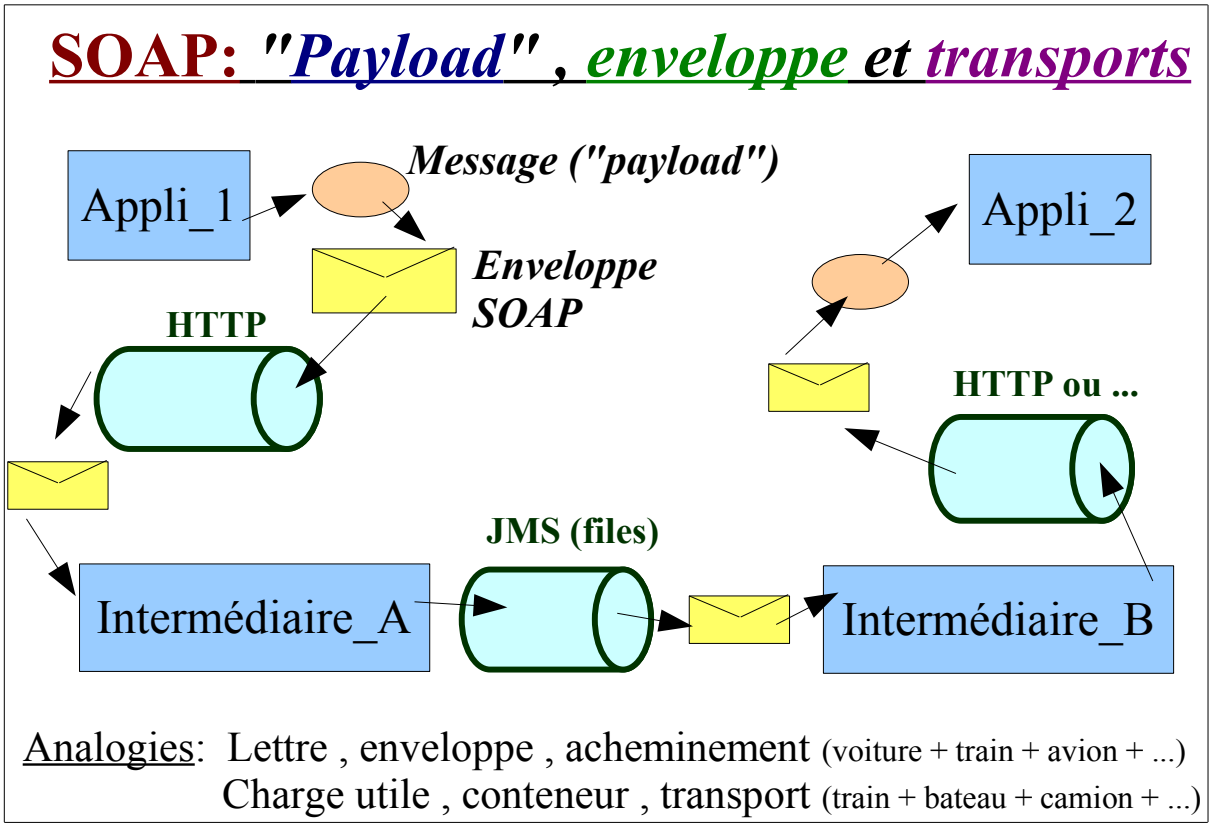
Ces messages peuvent tout aussi bien comporter des appels de fonctions (avec des paramètres valués) que comporter des messages textuels (éventuellement balisés en xml) et eux-même éventuellement accompagnés de pièces jointes.

Le **protocole SOAP** est constitué de **3 grandes parties (aspects complémentaires)**:

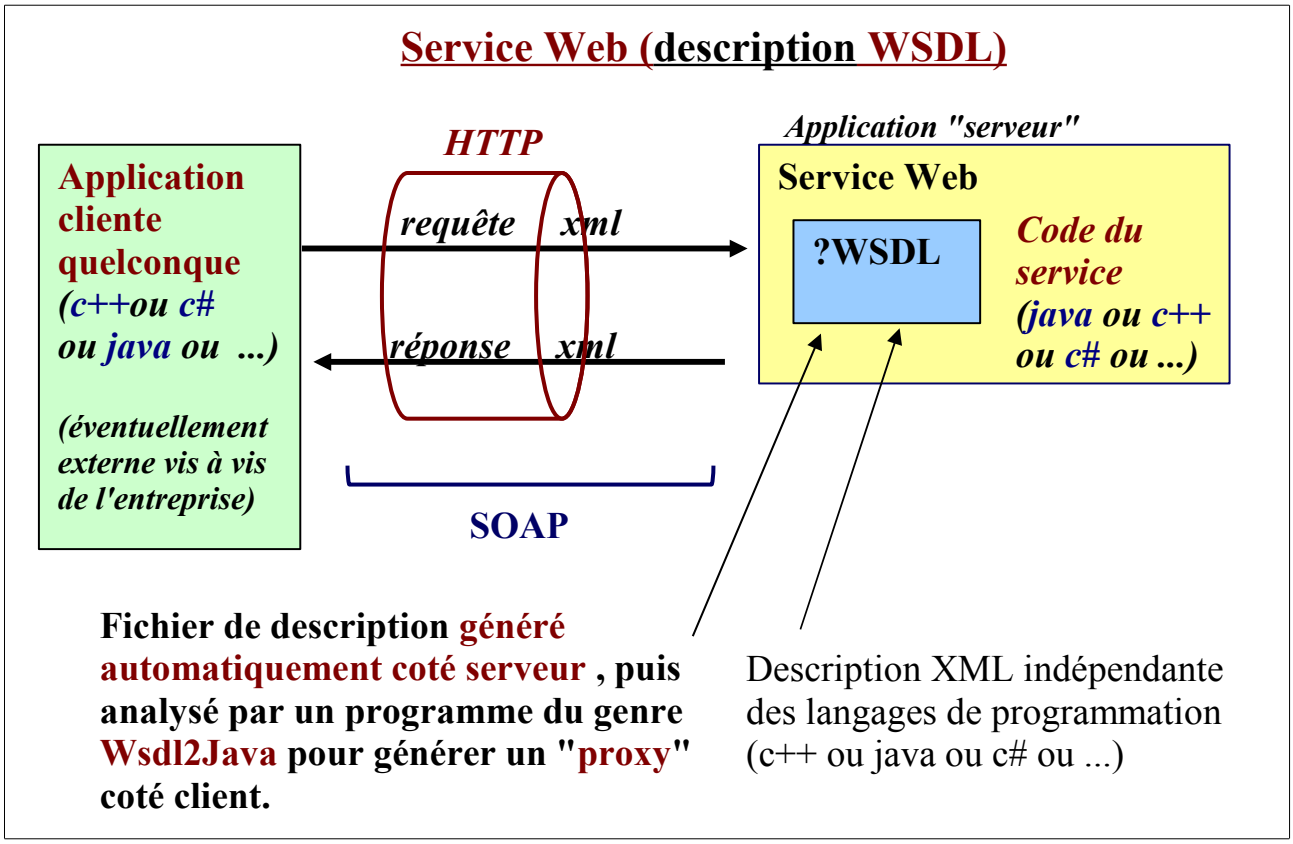
- L' **enveloppe SOAP** précisant le **contenu** du message avec ses parties facultative (header) et obligatoire (body) .
- L' ***ENC****odage* ***SOAP*** (sérialisation et dé-sérialisation des types de données)
  *NB: l'encodage SOAP normalisé au sein de la version 1.1 et correspondant au mode "rpc/encoded" est petit à petit supplanté par l'encodage "**literal**" .*
- La représentation des appels de fonctions (**SOAP-RPC**) et des réponses (ou erreurs).




Contenu du corps de l'enveloppe "SOAP"
message http
(avec contenu SOAP)
Entête http
<s:Envelope ... >
<s:Header />
<s:Body>
...
</s:Body>
</s:Envelope>
requête
<methodeXy>
<p1>val1</p1>
<p2>val2</p2>
</methodeXy>
Ou bien
réponse
<methodeXyResponse>
<return>val_resultat</return>
</methodeXyResponse>
Ou bien
Exception (fault)
<s:fault>
<faultcode>soap:Server</faultcode>
<faultstring>msg_error</faultstring>
</s:fault>


SOAP: "Payload" , enveloppe et transports
Appli_1
Message ("payload")
Enveloppe
SOAP
HTTP
Appli_2
HTTP ou ...
JMS (files)
Intermédiaire_A
Intermédiaire_B
Analogies: Lettre , enveloppe , acheminement (voiture + train + avion + ...)
Charge utile , conteneur , transport (train + bateau + camion + ...)

3. WSDL (Web Service Description Langage)



WSDL est un langage standard basé sur Xml qui a pour objectif de **décrire précisément les prototypes des fonctions activables au niveau d'un certain service web**.

Code coté serveur du *service web* en **C++** (*gSoap*)

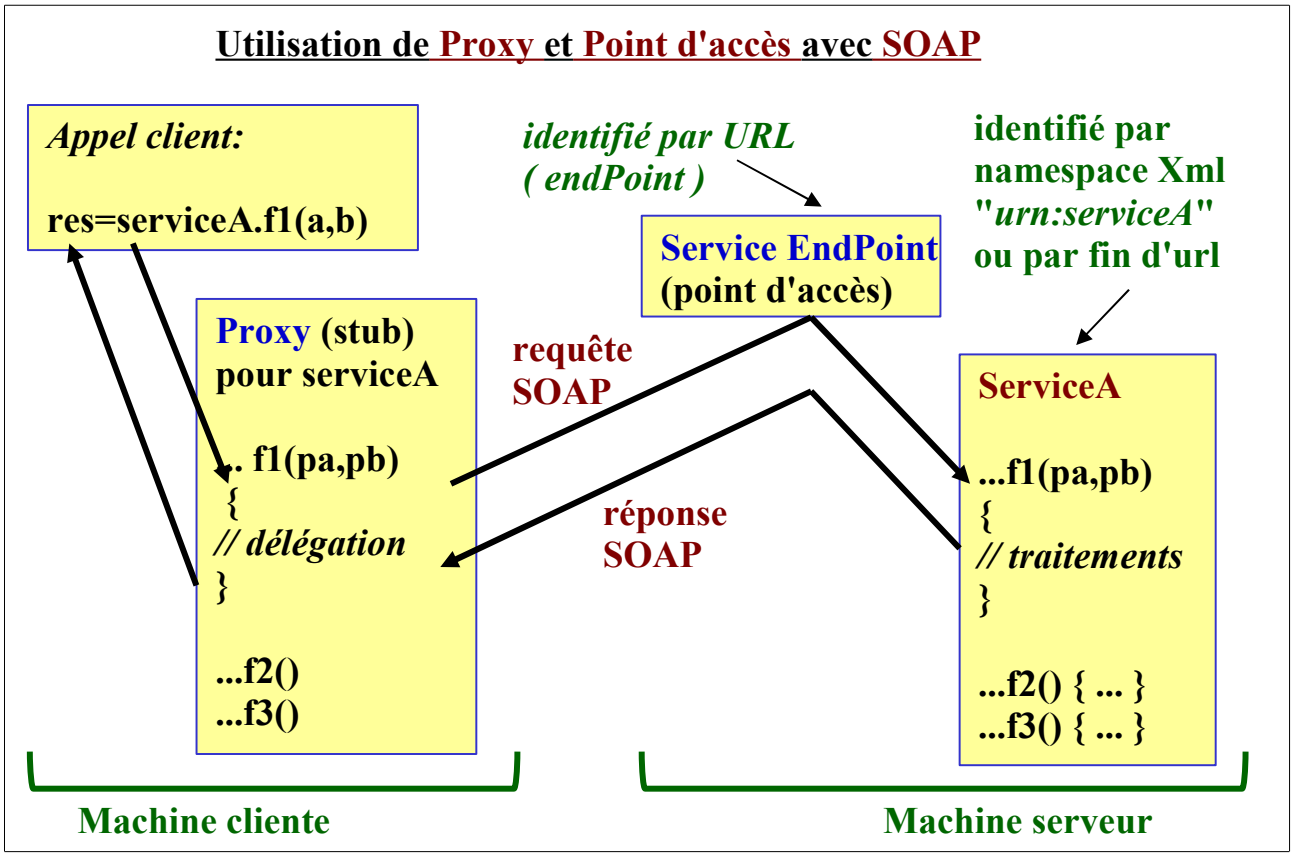
- génération du fichier WSDL via soapcpp2

- génération via ***WSDL2Java***

d'un ***proxy*** pour client **Java**

*Quelques définitions (**WSDL**):*

message	**paramètre(s)** *en entrée* ou bien *en sortie* d'une opération
portType	**Interface** (***ensemble d'opérations abstraites***)
operation	couple (message d'entrée, message de retour) = méthode
binding	associer un portType à un protocole (SOAP, COBBA, DCOM) et à un manespace(un même **URI** qualifiant généralement tout un service web)
service	service Web (avec url) comportant un ensemble de **port(s)** = **endpoint(s)**



4. "Basic profile" pour services "web"

De façon à garantir une bonne interopérabilité entre différentes implémentations des services "Web" (java, c++, .net , ...) qui risqueraient de diverger sur certains détails, l'organisme suivant est censé publier quelques règles assez strictes:

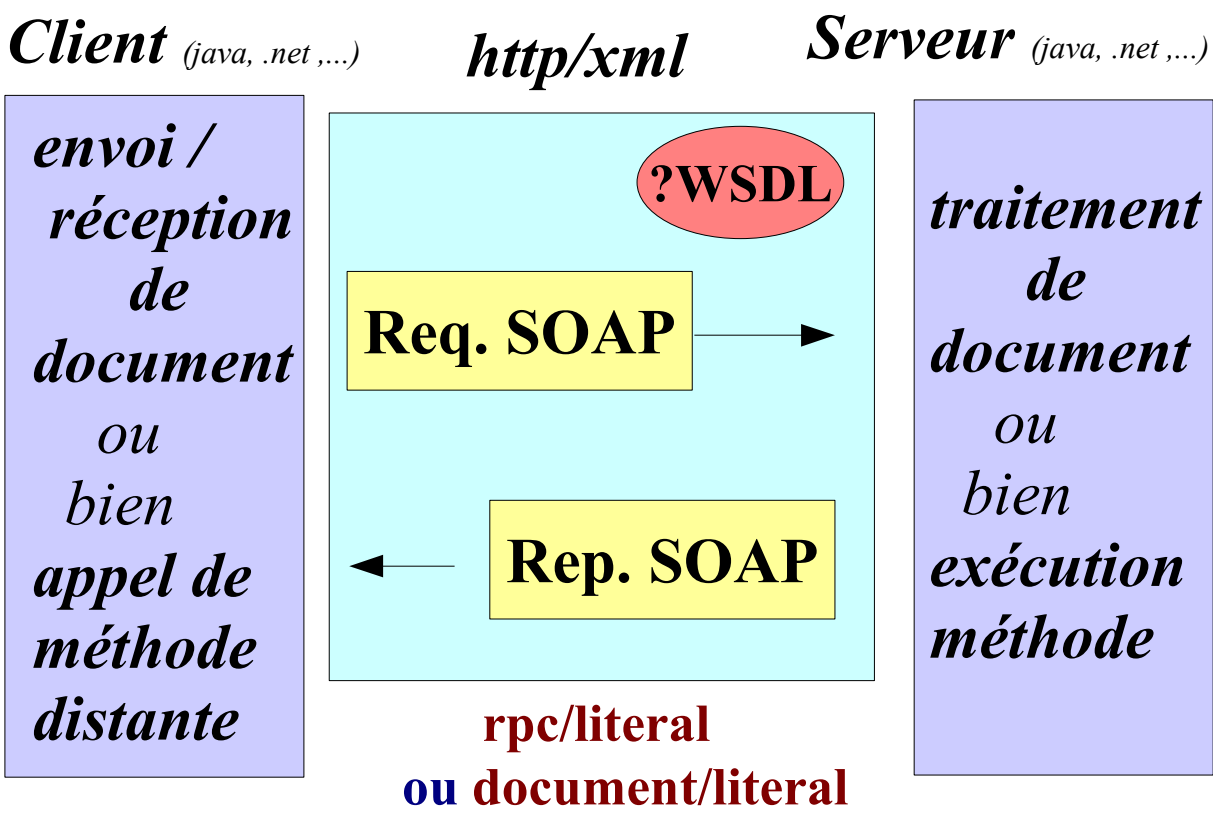
http://www.ws-i.org/ (**W**eb **S**ervices – **I**nteroperability *Organization*)

==> basic profile (v1 , v1.1 , ...) .
confirmations des choix de SOAP et précisions sur quelques points ambigus .

NB: Ceci est essentiellement utile pour IBM, SUN, Microsoft , ... et les autres éditeurs qui doivent mettre en place des outils et des API permettant de gérer simplement SOAP.
Le programmeur de services Web n'a généralement pas à se pré-occuper de ces détails car l'encodage XML est indirectement effectué par des API de hauts niveaux .

NB: Lors du choix d'une technologie liée aux services WEB, la compatibilité WS-I est un critère fondamental .

5. "Style/use" SOAP (rpc/literal , document/literal)



Remarque importante:

Lors du paramétrage d'un service WEB, il faut généralement choisir une combinaison "style/use" qui peut officiellement prendre l'une des valeurs suivantes:

- ***rpc/encoded***
- ***rpc/literal***
- ***document/encoded***
- ***document/literal***

En pratique "***document/encoded***" n'est **jamais utilisé** .

Bien que le style "document" soit plutôt approprié aux échanges de documents et que le style "rpc" soit plutôt approprié aux appels de méthodes à distance, les 2 styles SOAP "rpc" et "document" doivent normalement être replacés dans leurs contextes exacts:

- **Les styles SOAP ("rpc" ou "document") n'ont (en toute rigueur) qu'un impact sur la partie xml (messages SOAP et description WSDL).**

- **Rien n'empêche une application d'utiliser le style "document" pour appeler une méthode à distance.**

5.1. "rpc/encoded" et "rpc/literal"

Détails :

Partie d'une description **WSDL** généré en mode "***rpc/encoded***" ou "***rpc/literal***":

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
```

==> pas de bloc de type "ComplexType" , ...

Partie (un peu simplifiée) d'une requête SOAP en mode "**rpc/encoded**" :

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Partie (un peu simplifiée) d'une requête SOAP en mode "**rpc/literal**" :

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

5.2. "document/literal" et variantes ("wrapped" , "bared" , ...)

Détails:

Partie (un peu simplifiée) d'une requête SOAP en mode "**document/literal**" :

```
<soap:envelope>
    <soap:body>
        <myDoc ou myMethod>
            <x>5</x>
            <y>5.0</y>
        </myDoc ou myMethod>
    </soap:body>
</soap:envelope>
```

(==> pas de différence notable coté "message SOAP" avec le mode "rpc/literal").

La principale nouveauté du mode "**document/literal**" tient dans le fichier **WSDL** décrivant le service WEB :

```
<types>
    <schema>
        <element name="myDoc ou myMethod">
            <complexType>
                <sequence>
                    <element name="x" type="xsd:int"/>
                    <element name="y" type="xsd:float"/>
                </sequence>
            </complexType>
        </element>
        <element name="myMethodResponse">
            <complexType/>
        </element>
    </schema>
</types>
<message name="myMethodRequest">
    <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
    <part name="parameters" element="myMethodResponse"/>
</message>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
```

Le mode "*document/literal*" constitue le meilleur choix dans la majorité des cas.
Ce mode est pris en charge par les technologies récentes (coté ".net" et coté "java") .

Variantes "wrapped" et "bared":

A partir d'une même expression "SOAP/WSDL" lié au mode "document/literal" , on peut envisager 2 variantes au niveau du code (java ou ...) dénommées "**wrapped**" et "**bared**":

- une variante où la méthode distance comporte (et enveloppe) tout un tas de paramètres unitaires (x,y , ...)
- une variante où la méthode distante comporte un seul gros objet de type "message_Requete" comportant lui même les champs élémentaires "x" , "y" , ...

Quelques URL pour approfondir le sujet:

http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/

http://www.awprofessional.com/articles/article.asp?p=169106&seqNum=5&rl=1

http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/index.html

5.3. Versions et évolutions de SOAP/WSDL

Différentes versions (et évolution)

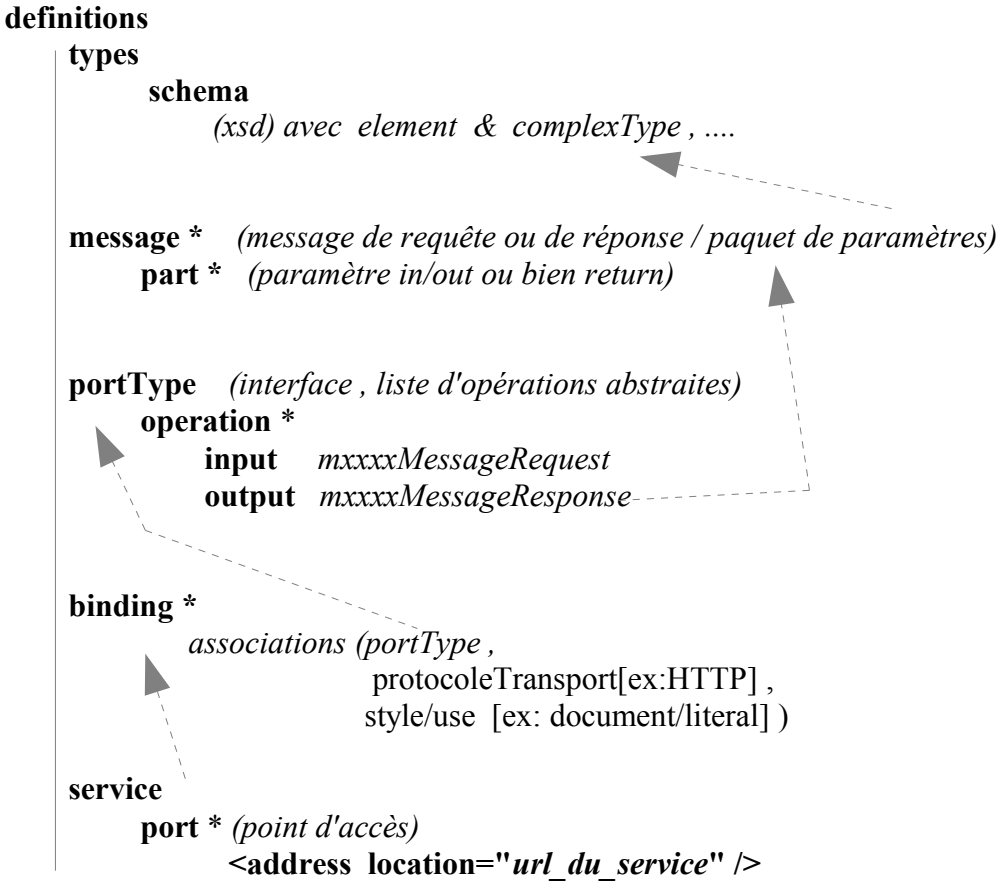
... , **SOAP 1.1** (xmlns="***http://schemas.xml.org/soap/envelope/***")
SOAP 1.2 (xmlns="...2004..")
Attention: SOAP 1.2 n'est pas pris en charge par toutes les technologies "WebService" et la version 1.1 suffit en général.

... , WSDL **1.1** (xmlns="***http://schemas.xmlsoap.org/wsdl/***")
WSDL 2 (xmlns=".....")
Attention: WSDL 2 n'est pas pris en charge par toutes les technologies "WebService" et la version 1.1 suffit en général. Cependant, certaines extensions WSDL2 (MEP) sont utilisées par certains produits (ex: ESB)

Pour SOAP, **style/use** = ~~*rpc/encoded* (complexe et "has been")~~
puis ***rpc/literal*** (plus performant, plus simple)
puis ***document/literal*** (avec schéma "xsd" pour valider)

6. Détails sur WSDL

6.1. Structure générale d'un fichier WSDL



NB: Le contenu de la partie <xsd:schema> (sous la branche "wsdl:types") peut éventuellement correspondre à un sous fichier annexe "***xxx.xsd***" relié au fichier "***xxx.wsdl***" via un "**xsd:*import***" .

6.2. Exemple de fichier WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions  targetNamespace="http://calcul"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://calcul"
xmlns:intf="http://calcul" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48 PDT)-->

 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://calcul"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="addition">
    <complexType>
```

```
    <sequence>
     <element name="a" type="xsd:int"/>
     <element name="b" type="xsd:int"/>
    </sequence>
   </complexType>
  </element>
  <element name="additionResponse">
   <complexType>
    <sequence>
     <element name="additionReturn" type="xsd:int"/>
    </sequence>
   </complexType>
  </element>
 </schema>
</wsdl:types>

  <wsdl:message name="additionResponse"
    <wsdl:part element="impl:additionResponse" name="parameters"/>
  </wsdl:message>

  <wsdl:message name="additionRequest">
    <wsdl:part element="impl:addition" name="parameters"/>
  </wsdl:message>

  <wsdl:portType name="Calculator">
    <wsdl:operation name="addition">
      <wsdl:input message="impl:additionRequest" name="additionRequest"/>
      <wsdl:output message="impl:additionResponse" name="additionResponse"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
    <wsdlsoap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="addition">
      <wsdlsoap:operation soapAction=""/>

      <wsdl:input name="additionRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>

      <wsdl:output name="additionResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="CalculatorService">
    <wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
      <wsdlsoap:address
      location="http://localhost:8080/axis2-webapp/services/Calculator"/>
    </wsdl:port>
```

```
  </wsdl:service>
</wsdl:definitions>
```

6.3. WDSL abstraits et concrets

Structure WSDL (*abstrait* ou *concret*)

WSDL abstrait *(interface)*

```
<definitions  ...>
   <types>
      <schema>
         <complexType ..../> ...
         <element ..../> ...
      </schema>
   </types>
   <message>...</message>
   <message>...</message>
   <portType>
      <operation  ...>
         <input .../> <output .../>
      </operation>
      <operation>...</operation>
   </portType>
</definitions>
```

WSDL concret/complet *(impl)*

```
<definitions>
   <types>
      <schema  ... />
   </types>
   <message>...</message> ...
   <portType>
      <operation .../ > ...
   </portType>
   <binding>
      ... HTTP/SOAP style="document"
      .... use="literal" ....
   </binding>
   <service>
      <port><address location="...url..." />
      </port>  </service>
</definitions>
```

Partie abstraite

Structure et sémantique WSDL (partie abstraite)

- Un ***même "WSDL abstrait"*** (idéalement issu d'une norme) pourra être physiquement implémenté par ***plusieurs services distincts*** (avec des ***points d'accès (et URL) différents***).
- Un ***portType*** représente (en XML) l'***interface fonctionnelle*** qui sera accessible depuis un futur "port" *(alias "endPoint")*
- Chaque opération d'un "portType" est associée à des ***messages*** (structures des ***requêtes[input]*** et ***réponses[output]***).
- En mode "*document/literal*", ces ***messages*** sont définis comme des ***références*** vers des "***element***"s (***xml/xsd***) qui ont la structure XML précise définie dans des "***complexType***" .

Structure et sémantique WSDL (partie concrète)

• Un fichier ***"WSDL concret"*** comporte toute la partie abstraite précédente plus tous les paramétrages nécessaires pour pouvoir invoquer ***un point d'accès précis sur le réseau***.

• Un ***port*** *(alias "endPoint")* permet d'atteindre une ***implémentation*** physique d'un service publié sur un serveur. La principale information rattachée est l'**URL** permettant d'invoquer le service via SOAP.

• La partie "**binding**" (référencée par un "port") permet d'***associer/relier*** entre eux les différents éléments suivants:

* ***interface abstraite (portType)*** et ses opérations
* ***protocole de transport*** (*HTTP* + détails ou ....)
* ***point d'accès (port)***

III - Web service "REST" (pas SOAP)

1. Web Services "R.E.S.T."

1.1. REST

Un style d'architecture

REST est l'acronyme de **Representational State Transfert**. C'est un **style d'architecture** qui a été décrit par ***Roy Thomas Fielding*** dans sa thèse *«Architectural Styles and the Design of Network-based Software Architectures»*.

Architectures orientées ressource

L'information de base, dans une architecture REST, est appelée **ressource**. Toute information qui peut être nommée est une ressource: un article d'un journal, une photo, un service ou n'importe quel concept.

Si la valeur d'une ressource peut changer au cours du temps, il est important de noter que la sémantique de la ressource, elle, ne devra jamais changer.

Une ressource est identifiée par un **identificateur de ressource**. Il permet aux composants de l'architecture d'identifier les ressources qu'ils manipulent. Sur le web ces identificateurs sont les **URI** (Uniform Resource Identifier).

Les composants de l'architecture REST manipulent ces ressources en transférant des **représentations de ces ressources**. Sur le web, on trouve aujourd'hui le plus souvent des représentations au format HTML , XML ou JSON.

```
http://exemple.org/maMethode?p1=v1&p2=v2
```

REST s'appuie à fond sur HTTP et n'impose pas un encodage XML (contrairement à SOAP).

Les méthodes HTTP sont utilisées pour indiquer la sémantique des actions demandées :

- **GET** : lecture d'information
- **POST** : envoi d'information
- **PUT** : mise à jour d'information
- **DELETE** : suppression d'information

Par exemple, pour récupérer la liste des adhérents de mon club, j'effectue une requête de type **GET** vers la ressource **http://monsite.com/adherents** . Pour obtenir que les adhérents ayant plus de 20 ans, la requête devient **http://monsite.com/adherents?ageMinimum=20**.
Pour supprimer les adhérents 3 et 4, on peut employer une requête de type **DELETE** telle que **http://monsite.com/adherents/3/4**.
Pour envoyer des informations, on utilise **POST** ou **PUT** en passant les informations dans le corps du message.

Pour décrire les méthodes d'un service REST --> WADL (pas normalisé) ou bien extension de WSDL 2 .

2. Format "J.S.O.N."

JSON = ***JavaScript Object Notation*** .

JSON s'architecture autour du format des objets ECMAScript (JavaScript) .
Les 2 principales caractéristiques sont :

- Le principe de clé / valeur
- L'organisation des données sous forme de tableau

Les types de données valables sont :

- tableau
- objet
- chaîne de caractères
- valeur numérique (entier, double)
- booléen (true/false)
- null

Par exemple, pour récupérer le nom, le prénom et l'âge d'une personne dans un objet :

```
{
    "nom": "xxxx",
    "prenom": "yyyy",
    "age": 25
}
```

Une liste d'articles :

```
[
    {
        "nom": "article a",
        "prix": 3.05,
        "disponible": false,
        "descriptif": "mon article a ..."
    },
    {
        "nom": "article b",
        "prix": 13.05,
        "disponible": true,
        "descriptif": null
    }
]
```

Plus d'informations sur le format JSON sont disponibles sur le site officiel :
http://www.json.org/json-fr.html

2.1. Exemple (équivalence JSON / XML):

en JSON:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

en XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Évaluer en JavaScript une expression JSON pour la transformer en objet se fait alors de la manière suivante :

```
var donnees = JSON.parse(donnees_json);

// Ou version multi-navigateurs :

var donnees2 = typeof JSON !='undefined' ?  JSON.parse(donnees_json) :
eval('('+donnees_json+')');
```

Performances:

Du point de vue du temps de traitement, évaluer une expression JSON en JavaScript est à peu de choses près aussi efficace que traiter son équivalent XML.

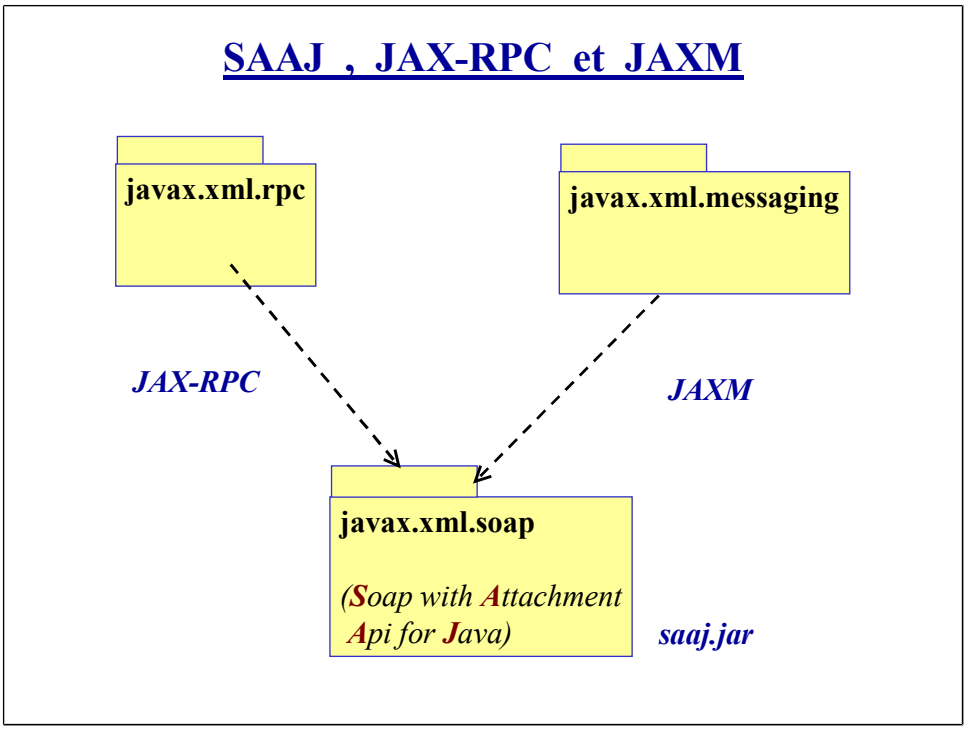
Du point de vue de la taille (stockage, vitesse de transmission…), une expression JSON est légèrement moins volumineuse; la différence n'est cependant sensible que sur de gros volumes, ce qui est rarement le cas pour un usage courant.

http://www.json.org/java/ ==> API Java pour traiter le format JSON . D'autres api existent (performances à comparer !)

IV - API java pour services WEB (présentation)

1. Api fondamentales (liées à SOAP,WSDL)

1.1. Api liées à J2EE 1.4 (déconseillées avec JEE5)



API	*Sigle*	*Description*
SAAJ	Soap with Attachement Api for Java	Partie commune à JAX-RPC et JAXM. Message SOAP , enveloppe SOAP
JAX-RPC	Java Api for Xml RPC	Appels de fonctions distantes
JAXM	Java Api for Xml Messaging	Envoi de messages Xml

1.2. Nouvelles API (jdk 1.5 / 1.6, JEE 5 , ...)

API	*Sigle*	*Description*
JAXB-2	Java Api for Xml Binding	Lien entre structure Xml et classe Java
JAX-WS	Java Api for Xml based Web Services	mieux que JAX-RPC avec utilisation interne de JAXB-2 et annotations de Java5 .
StAX	Stream Api for Xml	Parsing (sans arbre en mémoire) , pull ==> bonnes performances

2. API complémentaires

2.1. API Standards officialisées par Java/Sun

API	*Sigle*	*Description*
JAX-R	**J**ava **A**pi for **X**ml **Registry**	Connexions à des annuaires XML (ex: UDDI , ebXml , ...)
...		

2.2. API non officielles - standards de fait

API	*Sigle*	*Description*
WSDL4J	**WSDL** for **J**ava	Traitements des fichiers WSDL
UDDI4J	**UDDI for Java**	Interaction avec un annuaire UDDI (enregistrements, recherches)
....		

3. Technologies "java" pour la mise en oeuvre

Technologie	*Environnement requis*	*Caractéristiques*
Axis (apache)	Conteneur WEB (ex: Tomcat) , jdk 1.3 , 1.4 ou 1.5	Bonne technologie (très simple à mettre en oeuvre) mais l'intégration au sein d'une application J2EE existante reste incomplète.
Axis 2	Conteneur WEB (ex: Tomcat) , jdk 1.4 ou +	Nouvelle version de Axis beaucoup plus modulaire et plus simple à intégrer. ***Mais assez buggé*** Axis2 reste cependant assez propriétaire (Axiom , ...) et n'est pas directement lié à JAX-WS
WSDP (sun)	Conteneur Web , jdk selon version , ...	Ancienne technologie (un peu plus complexe) – Bien pour générer les fichiers "WSDL" et les fichiers requis par J2EE 1.4 (webservices.xml ,...) Intégration délicate.
wstools (jboss)	J2EE 1.4 ou supérieur, JDK 1.4 ou supérieur	Générateur (WSDL, ...) simple à utiliser .Livré dans JBOSS-AS à partir de la version 4.0.4
versions récentes de wstools	***jdk >=1.5 et Serveur JEE5 (ex: Jboss4.2.1)***	bonne prise en charge de ***JAX-WS*** ==> excellent pour Jboss (config simple & perf.)
metro (sun)	**jdk >= 1.5** , Serveur JEE	Nouveau projet de Sun lié aux services WEB basés sur **JAX-WS** (metro remplace maintenant WSDP) ==> utilisé dans ***Glassfish / NetBeans*** .
CXF (Apache)	**jdk >= 1.5 et Tomcat 5.5** ou autre serveur JEE	Projet Apache (différent de Axis2) , **CXF** est une bonne implémentation de **JAX-WS** qui ***peut se configurer via Spring*** et s'intègre très facilement dans Tomcat 5.5 ou un serveur équivalent.

V - Essentiel JAX-WS (API et Mise en oeuvre)

1. Présentation de JAX-WS

JAX-WS signifie ***Java Api*** *for* ***Xml Web Services***

L'api **JAX-WS** est une api utilisable à partir du jdk 1.5 et qui remplace maintenant l'ancienne api JAX-RPC . L'ancienne Api JAX-RPC n'est aujourd'hui intéressante que pour maintenir d'anciennes applications basées sur le jdk 1.4 .

L'api **JAX-WS** ***ne fonctionne qu'avec le jdk 1.5 , 1.6 ou 1.7*** et n'est donc utilisable qu'au sein des serveurs **JEE** récents (ex: *JBoss 4.2* , *Tomcat 5.5 + Apache_CXF* , ...).

Important : **JAX-WS est déjà intégré dans le jdk >=1.6** (*pas de ".jar" à ajouter* mais "update récent du jdk1.6 conseillé pour une version récente et optimisée/performante de JAX-WS) .

Dans le cas du ***jdk1.5*** , **JAX-WS s'ajoute en tant que ".jar"** d'une des implémentations disponibles (metro , ***CXF*** , ...).

Les principales spécificités de **JAX-WS** (vis à vis de JAX-RPC) sont les suivantes:

- paramétrage par **annotations Java5** (ex: @WebService , @WebMethod)
- utilisation interne de **JAXB2**

NB:

- L'utilisation "**coté client**" de JAX-WS ne nécessite que le **jdk >=1.6** (ou des ".jar" additionnels pour le jdk 1.5)
- L'utilisation "**coté serveur**" de JAX-WS nécessite généralement une ***prise en charge évoluée et intégrée*** (ex : **EJB3** ou bien "***CxfServlet***" dans une ***appli web Spring***).

2. Mise en oeuvre de JAX-WS (coté serveur)

La mise en oeuvre d'un service web coté serveur avec **JAX-WS** consiste essentiellement à utiliser les annotations **@WebService** , **@WebParam , @WebMethod , @WebResult, ...** au niveau du code source.

L'interface d'un service WEB nécessite simplement l'annotation ***@WebService*** et n'a pas absolument besoin d'hériter de java.rmi.Remote .

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

@WebService
public interface Calculator_SEI {

        public int addition(int a,int b);
}
```

NB: les noms associés au service (***namespace, PortType***, S***erviceName***) sont par défaut basés sur les noms des éléments java (package , nom_interface et/ou classe d'implémentation) et peuvent éventuellement être précisés/redéfinis via certains attributs facultatifs de l'annotation ***@WebService***

NB: de façon à ce que les noms des paramètres des méthodes d'une interface java soient bien retranscris dans un fichier WSDL, on peut les préciser via **@WebParam(name="*paramName*")**.

```
public int addition(int a,int b); ==> addition(xsd:int arg0,xsd:int arg1) dans WSDL
public int addition(@WebParam(name="a") int a, @WebParam(name="b") int b)
                                    ==> addition(xsd:int a ,xsd:int b) dans WSDL
```

L'annotation facultative **@WebResult** permet entre autre de spécifier le nom de la valeur de retour (dans SOAP et WSDL) . Par défaut le nom du résultat correspond à "**return**" .

NB: le sigle *SEI* signifie ***Service EndPoint Interface*** . il désigne simplement l'interface d'un service WEB et ne fait l'objet d'aucune convention de nom.

Au dessus de la classe d'implémentation , l'annotation @WebService peut éventuellement être sans argument si la classe n'implémente qu'une seule interface. Dans le cas où la classe implémente(ra) plusieurs interfaces, il faut préciser celle qui correspond à l'interface du Service Web:

```
@WebService (endpointInterface="package.Interface_SEI")
```

La classe d'implémentation d'un service Web peut éventuellement préciser via l'annotation *@WebMethod(exclude=true or false)* les méthodes qui seront vues du coté client et qui constituent le service web.

Si aucune annotation @WebMethod n'est mentionnée, alors toutes les méthodes publiques seront exposées.

```
package calcul;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService (endpointInterface="calcul.Calculator_SEI")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class {

  @WebMethod(operationName="add")
  public int addition(int a, int b) .....
}
```

- NB: L'annotation *SOAPBinding* est facultative. Elle permet de choisir (quand c'est possible) un style SOAP (ex: Style.RPC ou Style.DOCUMENT) .
- Le mode par défaut "***document/literal***" convient **très bien** dans presque tous les cas .

NB: Ces annotations ne sont utiles que si elles sont interprétées par un serveur JEE5 ou 6 (ex: GlassFish de Sun ou Jboss 4.2 ou 5) ou bien par une technologie JAX-WS (telle que CXF) ajoutée à un serveur J2EE (ex: Tomcat 5.5).

NB: Selon l'implémentation retenue de JAX-WS (metro , jboss-ws , CXF, ...) , les annotations sont acceptées ou pas lorsqu'elles sont directement placées au dessus de la classe d'implémentation.

Autres paramétrages importants (packages/namespaces):

- Lorsque l'interface et la classe d'implémentation d'un service WEB sont placées dans des packages java différents , ceci se traduit par différents "namespaces" xml dans le fichier WSDL. Préciser l'attribut "**targetNameSpace**" dans l'annotation **@WebService** de l'interface et aussi dans l'annotation **@WebService** de la classe d'implémentation permet de bien contrôler la (ou les) valeurs attendues dans le WSDL .
- Certaines méthodes ont des paramètres (en entrée ou en sortie) qui correspondent à des classes de données (JavaBean/POJO) .
  Placer **@XmlType(namespace="*http://yyy/data*")** permet de bien contrôler le namespace XML qui sera associé à ces classes de données.

Exemple :

```
@XmlType(namespace="http://entity.tp/")
@XmlRootElement(name="stat")
public class Stat {
   private int num_mois; //de 1 à 12
   private double ventes; //+get/set
…
}
```

```
@WebService(targetNamespace="http://tp.myapp.minibank/")
public interface GestionComptes {

        public Compte getCompteByNum(@WebParam(name="numCpt")long numCpt)
                                throws MyServiceException;

        public List<Stat> getStats(@WebParam(name="annee")int annee);

        public void transferer(@WebParam(name="montant")double montant,
                               @WebParam(name="numCptDeb")long numCptDeb,
                               @WebParam(name="numCptCred")long numCptCred)
                                    throws MyServiceException;
}
```

```
@WebService(targetNamespace="http://tp.myapp.minibank/",
            endpointInterface="tp.myapp.minibank.itf.domain.service.GestionComptes")
public class GestionComptesImpl implements GestionComptes {
        …
}
```

2.1. Implémentation sous forme d'EJB3 (pour serveur JEE5 tel que JBoss4.2 ou 5 , Glassfish de Sun , Geronimo 2 d'apache , Jonas d'OW2 ou ...)

- Combiner les annotations précédentes (@WebService , ...) avec l'annotation **@Stateless** au niveau de la classe d'implémentation d'un EJB3 Session sans état.
- Déployer le tout selon les spécifications JEE (.jar , .ear) au sein d'un serveur d'application comportant un conteneur d'EJB3
- Repérer l'url du service WEB et de la description WSDL (selon serveur)
  [ex: http://localhost:8080/xyz/XXXBean?wsdl]

2.2. Test sans serveur et wsgen (jdk 1.6)

NB: à des simples fins de "*tests unitaires*" , il est éventuellement possible de tester un service WEB en lançant une simple application java (non web) compilée avec le jdk 1.6 (*sans CXF*) qui comporte déjà en lui l'api "JAX-WS" et un mini mini conteneur Web appelé "Jetty" .
==> Ceci n'est valable que pour des petits tests (sans serveur) et nécessite tout de même l'invocation d'un script (.bat / .sh ou ant) qui lance la commande bin/**wsgen** du jdk 1.6 .

```
set JAVA_HOME=C:\.....\java\jdk1.6.....
set WKSP=D:\....\workspace
set PRJ=myProject
set SEI_Impl=basic.SimpleCalculatorImpl
"%JAVA_HOME%\bin\wsgen" -cp %WKSP%\%PRJ%\bin
                                    -d %WKSP%\%PRJ%\bin            %SEI_Impl%
```

NB: ceci est complètement **inutile** (et même quelquefois source d'incompatibilités) avec une utilisation standard de *CXF dans Tomcat* ou bien des *EJB3 dans Jboss* . ===> ***CXF et les EJB3 déclenchent automatiquement un équivalent de wsgen .***

2.3. Lancement mini serveur sans Container Web pour Tests

[==> OK avec jdk >=1.6 ou bien CXF .]

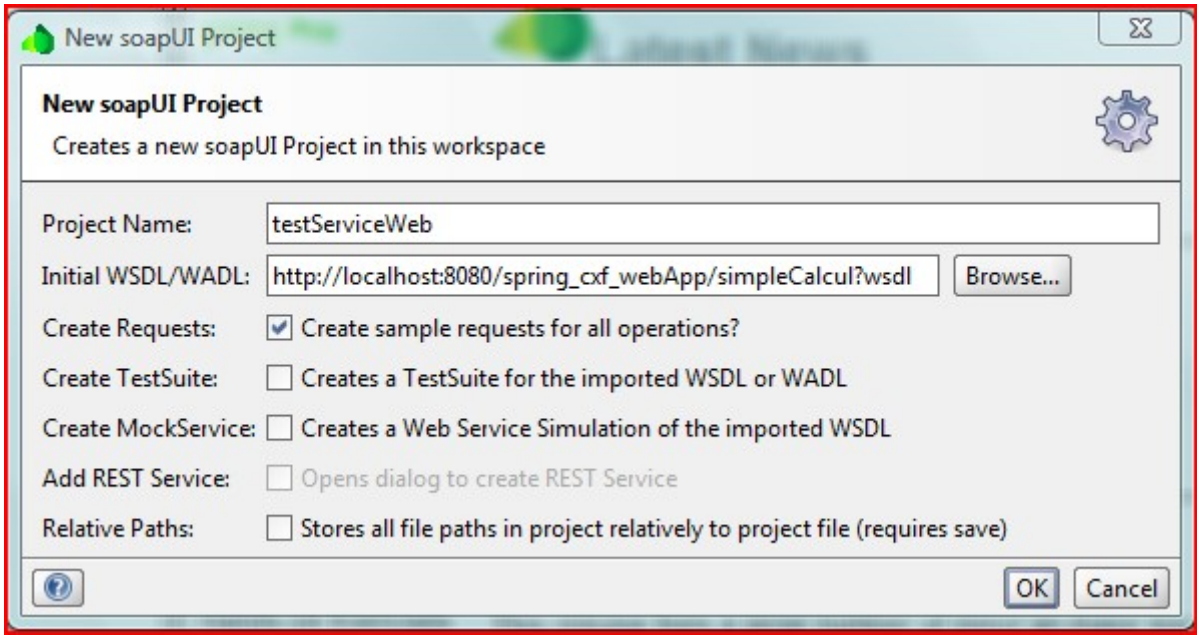
```
...
public class StandaloneServTestApp {
        protected StandaloneServTestApp() throws Exception {
                System.out.println("Starting Server");
                CalculateurImpl implementor = new CalculateurImpl();
                String address ="http://localhost:8080/myApp/services/calculateur";
                Endpoint.publish(address, implementor);
        }
public static void main(String args[]) throws Exception {
                new StandaloneServTestApp();        System.out.println("Server ready...");
                Thread.sleep(15 * 60 * 1000);// 15 minutes
                System.out.println("Server exiting"); System.exit(0);
        }
}
```

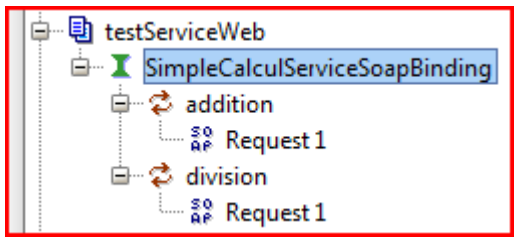
3. Tests via SOAPUI

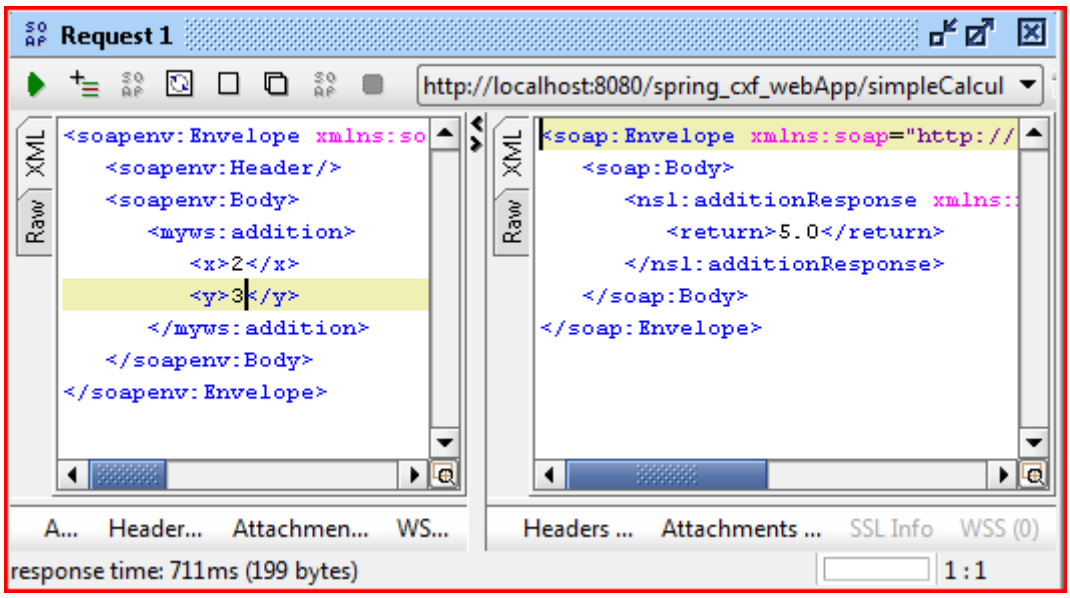
==> pour tester rapidement un service Web , le plus simple est d'utiliser le programme utilitaire "**soapui**" (dont une version gratuite est disponible) .

Mode opératoire:

- Télécharger et installer soapui .
- Lancer le produit via le script ".bat" du sous répertoire bin.
- Nouveau projet (de test) avec nom à choisir.
- Préciser l'URL (se terminant généralement par ***"?wsdl"***) de la description du service à invoquer.
- Sélectionner une des méthodes pour la tester en mode requête / réponse.
- Renseigner quelques valeurs au sein d'une requête.
- Déclencher l'invocation de la méthode via SOAP
- Observer la réponse (ou le message d'erreur)







Utilisation de JAX-WS coté client

3.1. Mode opératoire général (depuis WSDL)

1. **générer (à partir du fichier WSDL)** toutes les classes nécessaires au "***proxy JAX-WS***"
2. écrire le code client utilisant les classes générées:

```
package client;
import calcul.CalculatorSEI;
import calcul.CalculatorService;

public class MyWSClientApp {

      public static void main (String[] args) {
         try {
             CalculatorSEI calc = new CalculatorService().getCalculatorPort ();

             int number1 = 10;  int number2 = 20;
             System.out.printf ("Invoking addition(%d, %d)\n", number1,
                                number2);
             int result = calc.addition (number1, number2);
             System.out.printf ("The result of adding %d and %d is %d.\n\n",
                                number1, number2, result);
         } catch (Exception ex) {
             System.out.printf ("Caught Exception: %s\n", ex.getMessage ());
         }
     }
}
```

Le tout s'interprète très bien avec le **jdk >=1.6** (comportant déjà une implémentation de **JAX-WS**)ou bien avec un **jdk 1.5** augmenté de quelques librairies (**ex: .jar de CXF**).

3.2. Génération du proxy jax-ws avec wsimport du jdk >=1.6

```
set JAVA_HOME=C:\Prog\java\jdk\jdk1.6
cd /d "%~dp0"
"%JAVA_HOME%/bin/wsimport"  [- ....]  http://localhost:8080/...../serviceXY?wsdl
pause
```

sans l'option -keep ==> proxy au format compilé seulement

avec l'option ***-keep*** ==> code source du proxy également

l'option **-d** permet d'indiquer le répertoire *destination* (ou le proxy sera généré).

D'autres options existent (à lister via ***-help***)

4. Redéfinition de l'URL d'un service à invoquer

4.1. Redéfinition de l'URL "SOAP"

Si l'on souhaite ***invoquer un service Web localisé à une autre URL "SOAP" que celle qui est précisée dans le fichier WSDL*** (ayant servi à générer le proxy d'appel) , on peut utiliser l'interface cachée "***BindingProvider***" de la façon suivante:

```
Calculateur calculateur = serviceCalculateur.getCalculateurServicePort();
javax.xml.ws.BindingProvider bp = (javax.xml.ws.BindingProvider) calculateur;
Map<String,Object> context = bp.getRequestContext();
context.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                              "http://localhost:1234/myWebApp/services/calculateur");
System.out.println(calculateur.getTva(200, 19.6));
```

4.2. Redéfinition de l'URL WSDL (et indirectement SOAP)

Il est souvent plus simple (et efficace) de redéfinir l'URL du fichier WSDL qui indirectement précise l'URL SOAP à utiliser :

exemple :

```
URL urlWsdl = new URL("http://localhost:8080/convertisseur-web/services/convertisseur?wsdl");
QName sQname = new QName("http://conversion/","ConvertisseurImplService");
ConvertisseurImplService s = new ConvertisseurImplService(urlWsdl,sQname);
Convertisseur proxyConv = s.getConvertisseurImplPort(); ….
```

5. Client JAX-WS sans wsimport et directement basé sur l'interface java

Dans le cas (plus ou moins rares) où les cotés "client" et "serveur" sont tous les deux en Java , il est possible de se passer de la description WSDL et l'on peut générer un client d'appel (proxy/business delegate) en se basant directement sur l'interface java du service web *(SEI : Service EndPoint Interface)* .

5.1. avec le jdk 1.6 (sans CXF ni Spring)

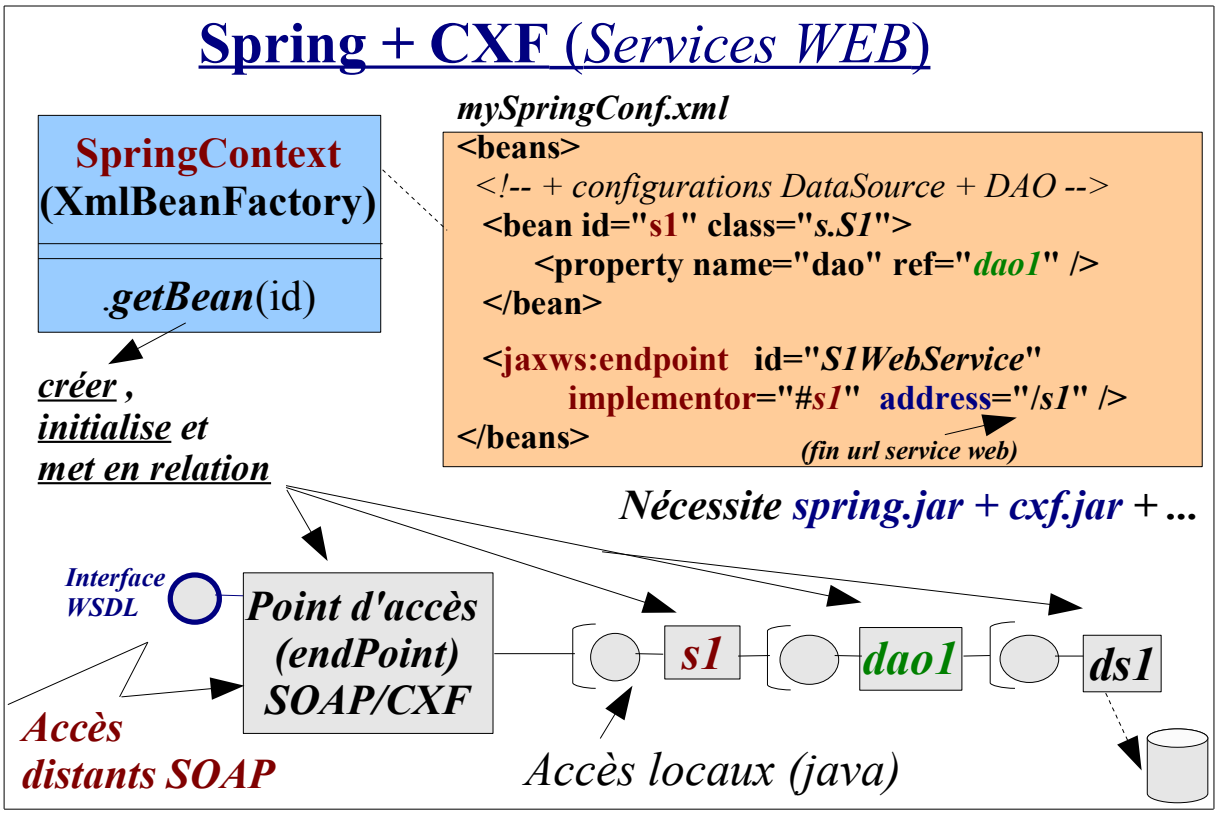
```
private static void testCalculateurServiceWithoutWsImport() {


    QName SERVICE_NAME = new QName("http://myws/", "CalculateurService");
    QName PORT_NAME = new QName("http://myws/", "CalculateurServicePort");

    Service service = Service.create(SERVICE_NAME); //javax.xml.ws.Service
    // Endpoint Address
    String endpointAddress =
              "http://localhost:8080/spring_cxf_webApp/services/calculateur";

    // Add a port to the Service , javax.xml.ws.soap.SOAPBinding
    service.addPort(PORT_NAME, SOAPBinding.SOAP11HTTP_BINDING,
                    endpointAddress);

     /* ou bien (éventuelle variante précisant une URL WSDL connue)
      String wdlUrl =
      "http://localhost:8080/spring_cxf_webApp/services/calculateur?wsdl";
      URL wsdlDocumentLocation=null;
      try {wsdlDocumentLocation = new URL(wdlUrl);
       } catch (MalformedURLException e) {      e.printStackTrace();}
      //avec import javax.xml.ws.Service;
      Service service = Service.create(wsdlDocumentLocation, SERVICE_NAME);
      */


      Calculateur caculateurService = (Calculateur)
           service.getPort(PORT_NAME, Calculateur.class);

            double tauxTvaReduitPct =
                              caculateurService.getTauxTva("TauxReduit");
            System.out.println("TauxReduit=" + tauxTvaReduitPct);
            double tva = caculateurService.getTva(200.0,tauxTvaReduitPct);
            System.out.println("Tva (pour ht=200) = " +  tva);
            Stats s = caculateurService.getStats(2005);
            System.out.print(s.getAnnee() +":");
            for(Infos infos : s.getInfos_par_mois())
                  System.out.print(infos.getRef()+" - ");
      }


...
```

6. CXF

6.1. Présentation de CXF (apache)

- **CXF est** une technologie facilitant le développement et la construction de **Web Services** en se basant sur l'API **JAX-WS**.
- Le framework **CXF** (de la fondation Apache) permet de mettre en place des **points d'accès (SOAP)** vers les **services métiers** d'une application.
- Les points d'accès (endpoints) sont pris en charge par le servlet prédéfini "**CxfServlet**" qu'il suffit de paramétrer et d'intégrer dans une application java web *(sans nécessiter d'EJB)*.

L'intégration de CXF avec Spring est simple :



- Une description WSDL du service web sera alors automatiquement générée et le service métier Spring sera non seulement accessible localement en java mais sera accessible à distance en étant vu comme un service Web que l'on peut invoquer par une URL http .
- Il faudra penser à déclarer le servlet de CXF dans WEB-INF/web.xml et placer tous les ".jar" de CXF dans WEB-INF/lib .
- Le paramétrage fin du service Web s'effectue en ajoutant les annotations standards de JAX-WS (@WebService , @WebParam , ...) dans l'interface et la classe d'implémentation du service métier .

Remarque importante: CXF (et CxfServlet) peut à la fois prendre en charge *SOAP* (via JAX-WS) et aussi *REST* (via JAX-RS) .

6.2. implémentation avec CXF (et Spring) au sein d'une application Web (pour Tomcat 5.5, 6 ou 7)

- Rapatrier les **".jar" de CXF** au sein du répertoire **WEB-INF/lib** d'un projet Web.
  (ou bien configurer convenablement les dépendances "maven").
- Ajouter la configuration suivante au sein de **WEB-INF/web.xml**:

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>WEB-INF/beans.xml</param-value> <!-- ou .... -->
</context-param>
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
<servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>  <!-- ou autre que services/* -->
</servlet-mapping>
```

- Ajouter la configuration Spring suivante dans WEB-INF ou bien dans
  [src ou src/main/resources ==> WEB-INF/classes] :

beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:jaxws="http://cxf.apache.org/jaxws"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
                http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
        <import resource="classpath:META-INF/cxf/cxf.xml" />
        <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
        <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
        <jaxws:endpoint       id="calculateurService"
                implementor="service.CalculateurService"   address="/Calculateur" />
</beans>
```

variante plus pratique si plusieurs niveaux d'injections (ex: DAO , DataSource, ...):

```
<bean id="calculateurService_impl" class="service.CalculateurService" >
    <property name="...."  ref="...." />
 </bean>

<jaxws:endpoint
        id="calculateurService"
        implementor="#calculateurService_impl"
        address="/Calculateur" />
```

NB:

- La syntaxe est "**#idComposantSpring**"
- Dans l'exemple ci dessus "*service.CalculateurService*" correspond à la classe d'implémentation du service Web (avec @WebService) et "*/Calculateur*" est la partie finale de l'url menant au service web pris en charge par Spring+CXF .
- Après avoir déployé l'application et démarré le serveur d'application (ex: run as/ run on serveur),il suffit de préciser une URL du type http://localhost:8080/myWebApp/services pour obtenir la liste des services Web pris en charge par CXF. En suivant ensuite les liens hypertextes on arrive alors à la description WSDL .

==> c'est tout simple , modulaire via Spring et ça fonctionne avec un simple Tomcat !!!

6.3. Client JAX-WS sans wsimport avec CXF et Spring

Rappel: Dans le cas (plus ou moins rares) où les cotés "client" et "serveur" sont tous les deux en Java , il est possible de se passer de la description WSDL et l'on peut générer un client d'appel (proxy/business delegate) en se basant directement sur l'interface java du service web *(SEI : Service EndPoint Interface)*

La configuration Spring suivante permet de générer automatiquement un "business_deleguate" (dont l'id est ici "client" et qui est basé sur l'interface "service.Calculateur" du service Web).

Ce "business deleguate" delèguera automatiquement les appels au service Web distant dont l'url est précisé par la propriété "address".

src/**bean.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:jaxws="http://cxf.apache.org/jaxws"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                http://cxf.apache.org/jaxws http://cxf.apache.org/schema/jaxws.xsd">
        <bean id="clientFactory" class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean">
            <property name="serviceClass" value="service.Calculateur"/> <!-- interface_SEI -->
```

```
        <property name="address" value="http://localhost:8080/serveur_cxf/Calculateur"/>
    </bean>
     <bean id="client" class="service.Calculateur" factory-bean="clientFactory"
                factory-method="create"/>
</beans>
```

il ne reste alors plus qu'à utiliser ce "business_deleguate" au sein du code client basé sur Spring:

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
...
public class WsTestApp {

      public static void main(String[] args) {
            ClassPathXmlApplicationContext context = new
                  ClassPathXmlApplicationContext(new String[] { "beans.xml" });

            Calculateur calculateur = (Calculateur) context.getBean("client");
            System.out.println(calculateur.getTva(200, 19.6));
            }
      }
}
```

NB: il faut penser à rapatrier les ".jar" de cxf et le code source de l'interface SEI (içi service.Calculateur) .

6.4. avec CXF sans Spring

```
private static void testCalculateurServiceWithoutSpring() {


    QName SERVICE_NAME = new QName("http://myws/", "CalculateurService");
    QName PORT_NAME = new QName("http://myws/", "CalculateurServicePort");

    Service service = Service.create(SERVICE_NAME); //javax.xml.ws.Service
    // Endpoint Address
    String endpointAddress =
              "http://localhost:8080/spring_cxf_webApp/calculateur";

    // Add a port to the Service , javax.xml.ws.soap.SOAPBinding
    service.addPort(PORT_NAME, SOAPBinding.SOAP11HTTP_BINDING,
                    endpointAddress);

      Calculateur caculateurService = (Calculateur)
           service.getPort(PORT_NAME, Calculateur.class);

            double tauxTvaReduitPct =
                              caculateurService.getTauxTva("TauxReduit");
            System.out.println("TauxReduit=" + tauxTvaReduitPct);
            double tva = caculateurService.getTva(200.0,tauxTvaReduitPct);
            System.out.println("Tva (pour ht=200) = " +  tva);
            Stats s = caculateurService.getStats(2005);
            System.out.print(s.getAnnee() +":");
            for(Infos infos : s.getInfos_par_mois())
                  System.out.print(infos.getRef()+" - ");
      }
```

VI - JAX-RS (Api Java pour WS REST)

1. API java pour REST (JAX-RS)

JAX-RS est une API java très récente (depuis les spécifications JEE 6) qui permet de mettre en œuvre des services REST en JAVA :

Une classe java avec annotations JAX-RS sera exposée comme web service REST.

JAX-RS se limite à l'implémentation serveur, la spécification ne propose rien du coté client

Pour implémenter les services REST, on utilise principalement les annotations JAX-RS suivantes:

- **@Path** : définit le chemin de la ressource. Cette annotation se place sur la classe et/ou sur la méthode implémentant le service.
- **@GET, @PUT, @POST, @DELETE** : définit le verbe implémenté par le service
- **@Produces** spécifie le ou les Types MIME de la réponse du service
- **@Consumes** : spécifie le ou les Types MIME acceptés en entré du service

1.1. Exemple JAX-RS

Voici en exemple d'implémentation de quelques services REST avec JAX-RS :

```
...
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.WebApplicationException;


@Path("articles")// pour URL de type "http://server/.../articles/..."
public class Articles {
    @GET
    @Produces({MediaType.APPLICATION_JSON})
    public List<Article> findAllArticles(){
        List<Article> articles = articleDao.findAll();
        return articles;
    }

    @GET
    @Produces({MediaType.APPLICATION_JSON,MediaType.APPLICATION_XML})
    @Path("/{article}")
    public Article findArticle(@PathParam("article") String id){
        Article article = articleDao.findById(id);
     if(article ==null){
          throw new WebApplicationException(Status.NOT_FOUND);
     }
        return article;
    }


    @DELETE
    @Path("/{article}")
    public void deleteArticle(@PathParam("article") String id){
        articleDao.remove(id);
    }
}
```

L'utilisation de l'annotation @Produces permet de spécifier le format de la réponse en application/json.

Ainsi, GET http://server/articles

Retourne :

```
[
  {"id":"art1","title":"article1","url":"http://server/articles/art1"},
 {"id":"art2","title":"article2","url":"http://server/articles/art1"}
]
```

Pour lire l'article "art1", il faut exécuter la méthode GET sur l'URI http://server/articles/art1. Comme @Produces définit 2 types MIME de retour, le retour sera fonction du Header "Accept" de la requête http. Selon l'implémentation JAX-RS, il est aussi possible de se baser sur l'extension pour choisir le format. Par exemple http://server/articles/art1.json permet d'obtenir la réponse json.

Pour supprimer l'article "art1", il faut exécuter la méthode DELETE sur l'URI http://server/articles/art1.

Dans cet exemple, les méthodes java retournent des objets et ne s'occupent pas de la conversion en json ou xml. JAX-RS utilise pour cela JAXB pour le marshalling des objets en XML, et des extensions pour les conversions dans les autres formats.

Les principales implémentations de JAX-RS sont :

- Jersey, implémentation de référence de SUN
- Resteasy, l'implementation jboss
- CXF
- Restlet

1.2. Mise en oeuvre de JAX-RS avec CXF

NB: les versions récentes de CXF (2.2 , 2.3 , ...) implémentent maintenant JAX-RS (en plus de JAX-WS)

WEB-INF/**web.xml** (pour spring + cxf):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
...
   <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/classes/deploy-context.xml</param-value>
   </context-param>

   <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
   </listener>

   <servlet>
      <servlet-name>CXFServlet</servlet-name>
      <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Configuration "Spring/CXF" pour JAX-RS:

deploy-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxrs="http://cxf.apache.org/jaxrs"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                  http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/jaxrs
                  http://cxf.apache.org/schemas/jaxrs.xsd"
  default-lazy-init="false">

  <import resource="classpath:META-INF/cxf/cxf.xml" />
  <import resource="classpath:META-INF/cxf/cxf-extension-jaxrs-binding.xml" />
  <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

  <!-- url complete de type "http://localhost:8080/mywebapp/services/rest/myservice/xxx"
   avec "services" associe à l'url-pattern de CxfServlet dans web.xml
   et myservices/xxx associe aux valeurs de @Path() de la classe java et des méthodes
  -->

  <jaxrs:server id="myRestServices" address="/rest">
    <jaxrs:serviceBeans>
      <ref bean="serviceImpl" />
      <!--  <ref bean="service2Impl" /> -->
    </jaxrs:serviceBeans>
    <jaxrs:extensionMappings>
      <entry key="xml" value="application/xml" />          <entry key="json" value="application/json" />
    </jaxrs:extensionMappings>
  </jaxrs:server>

  <bean id="serviceImpl" class="service.ServiceImpl" />
  <bean id="service2Impl" class="service.Service2Impl" />
</beans>
```

User pojo:

```
package pojo;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "user") /* pour préciser  balise xml englobante */
public class User {
   private Integer id;    // +get/set
   private String  name;  // +get/set

   @Override
```

```
    public String toString() {
        return String.format("{id=%s,name=%s}", id, name);
    }
}
```

classe d'implémentation du service:

```
package service;

import java.util.HashMap;  import java.util.Map;

import javax.ws.rs.GET;            import javax.ws.rs.Path;
import javax.ws.rs.PathParam;   import javax.ws.rs.QueryParam;
import javax.ws.rs.Produces;     import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

import pojo.User;

@Path("/myservice/")
@Produces("application/xml") //par défaut pour toutes les méthodes de la classe
public class ServiceImpl  /* implements ServiceDefn  */{

    private static Map<Integer,User> users = new HashMap<Integer,User>();
    static {
        users.put(1, new User(1, "foo"));        users.put(2, new User(2, "bar"));
    }
    public ServiceImpl() {
    }

    @GET
    @Path("/users")
    public Collection<User> getUsers() {
        return users.values();
    }

    @GET
    @Path("/user/{id}")
    // pour URL = http://localhost:8080/mywebapp/services/rest/myservice/user/2
    public User getUser(@PathParam("id") Integer id) {
        return users.get(id);
    }

    @GET
    @Path("/utilisateur")
    // pour URL = http://localhost:8080/mywebapp/services/rest/myservice/utilisateur?id=2
    // et quelquefois ...?p1=val1&p2=val2&p3=val3
    public User getUserV2(@QueryParam("id") Integer id) {
        return users.get(id);
    }

    @GET
    @Path("/users/bad")
    public Response getBadRequest() {
        return Response.status(Status.BAD_REQUEST).build();
    }

    @POST  // ou @GET   (bien que REST conseille POST pour des ajouts )
    @Path("/users/add")
    // pour action URL = http://localhost:8080/mywebapp/services/rest/myservice/users/add
    // dans form avec <input name="id" /> et <input name="name" /> et method="POST"
    public Response addNewUser(@FormParam("id") Integer id,@FormParam("name") String  name) {
```

```
        users.put(id,new User(id,name)) ;
        return Response.status(Status.OK).build();
    }

}
```

1.2.a. REST Service

The following urls should now show the expected REST output.

- `http://localhost:8080/mywebapp/services/rest/myservice/users/`
- `http://localhost:8080/mywebapp/services/rest/myservice/user/1`

Users REST output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<userCollection>
   <users>
      <user><id>1</id><name>foo</name></user>
      <user> <id>2</id> <name>bar</name> </user>
   </users>
</userCollection>
```

Single user REST output

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user>
   <id>1</id>
   <name>foo</name>
</user>
```

Quelques remarques :

```
@GET
@Path("/euroToFranc/{s}")
@Produces("text/plain")
public double euroToFranc(@PathParam("s")double s){
        return s*6.55957 ;
}
```

….

VII - Types complexes et exceptions SOAP

Les types de données des paramètres des méthodes d'un service WEB sont très importants car ils conditionnent les éléments suivants:

- interopérabilité (java , .net , ...)
- simplicité de la mise en oeuvre , incompatibilité éventuelle , ...
- ...

1. Types de données élémentaires bien supportés

types "xml/xsd"	*types "java" correspondants*
xsd:int	**int**
xsd:string	**java.lang.String**
xsd:double	**double**
xsd:float	**float**
xsd:long	**long**
xsd:short	**short**
xsd:boolean	**boolean**
xsd:byte	byte
xsd:hexBinary , xsd:base64Binary	byte[]
xsd:dateTime	java.util.Calendar
xsd:integer	java.math.BigInteger
xsd:decimal	java.math.BigDecimal

2. Types de données composés (bien gérés)

Un type de donné composé (xsd:ComplexType) est bien supporté coté SOAP/WSDL/Xml , il sera associé à une structure en C/C++ et sera associé à une classe/JavaBean en Java (avec get/set) .

package data;

```
public class MyDataBean implements java.io.Serializable {
    private String p1;    private int p2;
    private double p3;    //+get/set
...       }
```

==> type complexe généré au sein du fichier WSDL ==>

....

```
<wsdl:types> <schema targetNamespace="data" xmlns="http://www.w3.org/2001/XMLSchema">
  ...    <complexType name="MyDataBean">
          <sequence>
               <element name="p1" nillable="true" type="xsd:string" />
               <element name="p2" type="xsd:int" />     <element name="p3" type="xsd:double" />
          </sequence>
   </complexType>
 </schema> ...
```

3. Tableaux et collections (interopérabilité à tester)

- Les tableaux de types élémentaires (tableau d'entiers , de réels , de chaînes de caractères, ...) sont en règle général bien supportés (sauf avec une technologie anormalement défaillante).
- Les tableaux d'objets et collections d'objets sont bien gérés par les technologies récentes JAVA (CXF , EJB3, …) et .NET de Microsoft .
  Si anciennes technologies ou si autre langage ==> à tester .

4. Utilitaire "tcp-monitor" pour intercepter et tracer les messages SOAP

L'utilitaire "**tcp-monitor**" est un petit programme qui permet d'intercepter les messages SOAP (requêtes et réponses) de façon à afficher le contenu XML.

L'utilitaire **tcp-monitor** était livré avec Axis 1 (apache).
Aujourd'hui , avec Axis2 , **tcp-monitor** correspond à un **projet séparé (à télécharger à part)**.

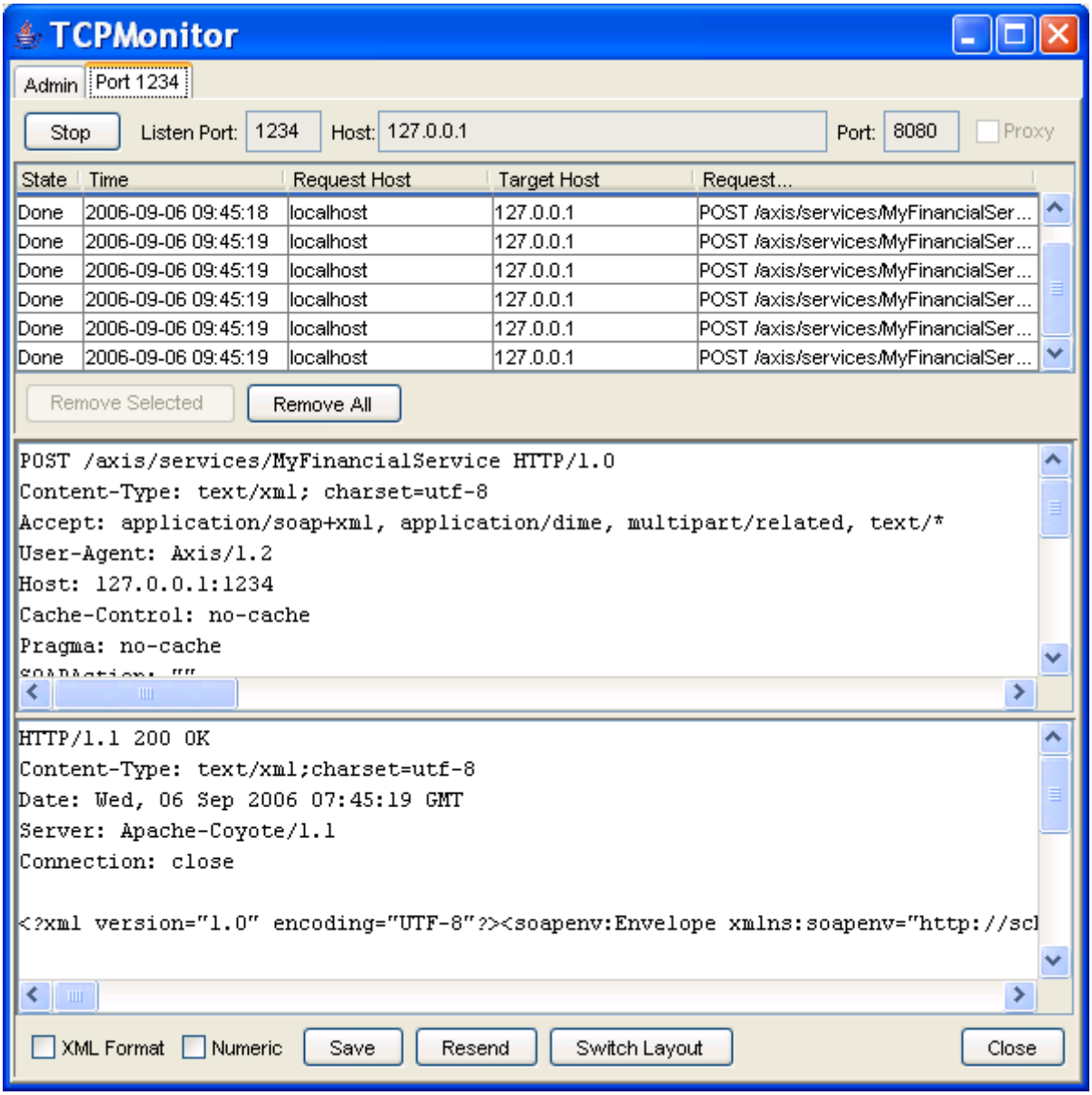
On peut le lancer comme un programme java:

```
java org.apache.ws.commons.tcpmon.Tcpmon [listenPort ...]
```

```
Le principal paramétrage correspond au numéro de port (ex: listenPort=1234) de
l'utilitaire .

Client ====>  SOAP/Http vers xxx:1234

                Interception (affichage requête)

                                    =====> dialogue SOAP/Http avec le
                                           serveur  (:80 , :8080)

                                    <====

                Interception (affichage réponse)

    <====

Client
```

lancerTcpMonitor.bat

```
set CLASSPATH=C:\Prog\JAVA\tcpmon-1.0-bin\build\tcpmon-1.0.jar
REM java org.apache.ws.commons.tcpmon.Tcpmon [listenPort targetHost targetPort]
java org.apache.ws.commons.tcpmon.TCPMon 1234 127.0.0.1 8080
```



NB: cocher "*Xml Format*" pour une présentation lisible avec "saut de ligne" .

5. Transfert des exceptions via SOAP (Fault)

Exemple:

```
public class CalculException extends Exception {

	private static final long serialVersionUID = 1L;

	public static enum ErrorCodeEnum  { UNKNOWN , DIV_BY_ZERO ,
                                        ERROR2 , ERROR3 };
	private ErrorCodeEnum errorCode = ErrorCodeEnum.UNKNOWN;
	private String details="";


	public ErrorCodeEnum getErrorCode() {     return errorCode; }

	public void setErrorCode(ErrorCodeEnum errorCode) {
		this.errorCode = errorCode;
	 }
	public String getDetails() {  return details;   }
	public void setDetails(String details) {  this.details = details; }
	public CalculException() {super();  }
	public CalculException(String msg, Throwable cause) {
		super(msg, cause);
	 }
	public CalculException(String msg) {super(msg);}
	public CalculException(Throwable cause) {super(cause);}
}
```

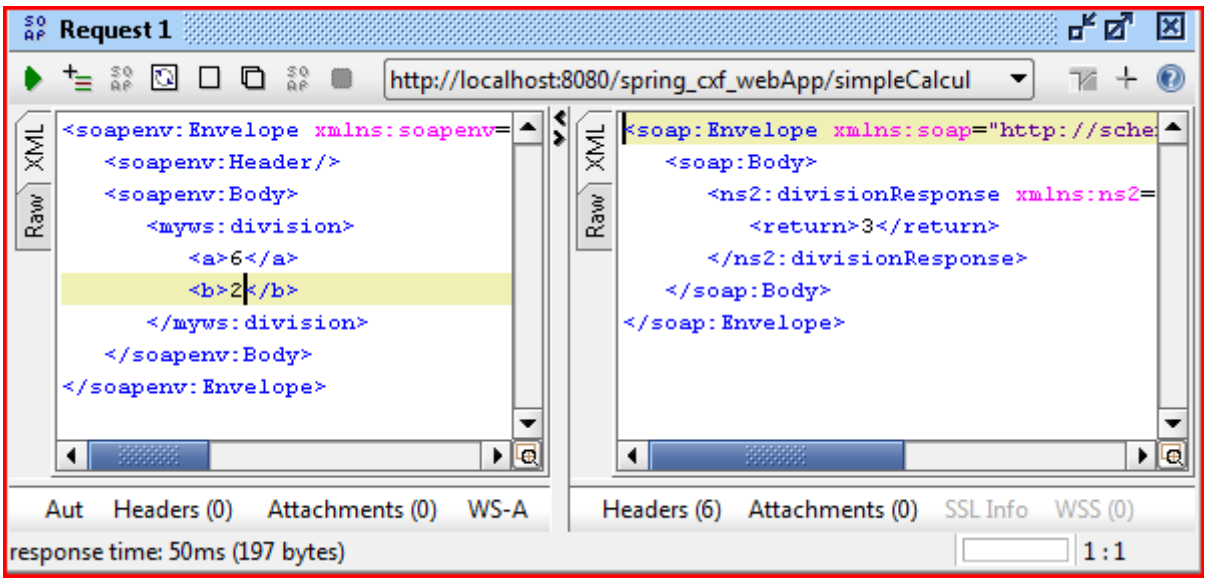
```
@WebService
public interface SimpleCalcul {
	...
	public int division(@WebParam(name="a")int a,
                            @WebParam(name="b")int b)
                throws CalculException;
}
```

```
@WebService(endpointInterface="myws.SimpleCalcul",
          serviceName="SimpleCalculService" ,
          portName="SimpleCalculServicePort")
public class SimpleCalculImpl implements SimpleCalcul {

	public int division(int a, int b) throws CalculException {
		int res=0;
		if(b==0){
		 CalculException e = new CalculException("division par 0 interdite");
		 e.setErrorCode(CalculException.ErrorCodeEnum.DIV_BY_ZERO);
		 e.setDetails("a="+a+" div by b="+b);
		 throw e;
		}
		 else res=a/b;
		 return res;
	}

}
```

description WSDL avec Exception / Fault SOAP

```
<wsdl:definitions name="SimpleCalculService" targetNamespace="http://myws/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://myws/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:types>
      <xs:schema attributeFormDefault="unqualified"
elementFormDefault="unqualified" targetNamespace="http://myws/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
         ...
         <xs:simpleType name="errorCodeEnum">
            <xs:restriction base="xs:string">
               <xs:enumeration value="UNKNOWN"/>
               <xs:enumeration value="DIV_BY_ZERO"/>
               <xs:enumeration value="ERROR2"/>
               <xs:enumeration value="ERROR3"/>
            </xs:restriction>
         </xs:simpleType>
         <xs:element name="CalculException" type="tns:CalculException"/>
         <xs:complexType name="CalculException">
            <xs:sequence>
               <xs:element name="errorCode" nillable="true"
                           type="tns:errorCodeEnum"/>
               <xs:element name="details" nillable="true" type="xs:string"/>
            </xs:sequence>
         </xs:complexType>
      </xs:schema>
   </wsdl:types>
      ...
   <wsdl:message name="CalculException">
      <wsdl:part element="tns:CalculException" name="CalculException"/>
   </wsdl:message>
   <wsdl:portType name="SimpleCalcul">
      <wsdl:operation name="division">
         <wsdl:input message="tns:division" name="division"/>
         <wsdl:output message="tns:divisionResponse" name="divisionResponse"/>
         <wsdl:fault message="tns:CalculException" name="CalculException"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="SimpleCalculServiceSoapBinding" type="tns:SimpleCalcul">
      <soap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="division">
         <soap:operation soapAction="" style="document"/>
         <wsdl:input name="division">
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="divisionResponse">
            <soap:body use="literal"/>
         </wsdl:output>
         <wsdl:fault name="CalculException">
            <soap:fault name="CalculException" use="literal"/>
         </wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="SimpleCalculService">
      <wsdl:port binding="tns:SimpleCalculServiceSoapBinding"
                 name="SimpleCalculServicePort">
         <soap:address
           location="http://localhost:8080/spring_cxf_webApp/simpleCalcul"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

test avec b différent de zéro ==> aucune exception :



retour SOAP en cas d'erreur (b=0) :

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>division par 0 interdite</faultstring>
         <detail>
            <ns1:CalculException xmlns:ns1="http://myws/">
               <errorCode xsi:type="ns2:errorCodeEnum"
                          xmlns:ns2="http://myws/"
                  xmlns:xsi=
                  "http://www.w3.org/2001/XMLSchema-instance">
                     DIV_BY_ZERO</errorCode>
               <details xmlns:ns2="http://myws/">
                     a=6 div by b=0</details>
            </ns1:CalculException>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

VIII - Sécurité pour Services WEB

1. Gestion de la sécurité / Services Web

1.1. Eléments de sécurité nécessaires sur les services WEB

Sécurité liée aux ***services "WEB"***

- ***Authentification du client*** via ***(username,password)*** ou autre pour contrôler les accès aux services
  --> "infos auth" dans ***entête Http*** *ou* ***entête SOAP*** *ou ... .*

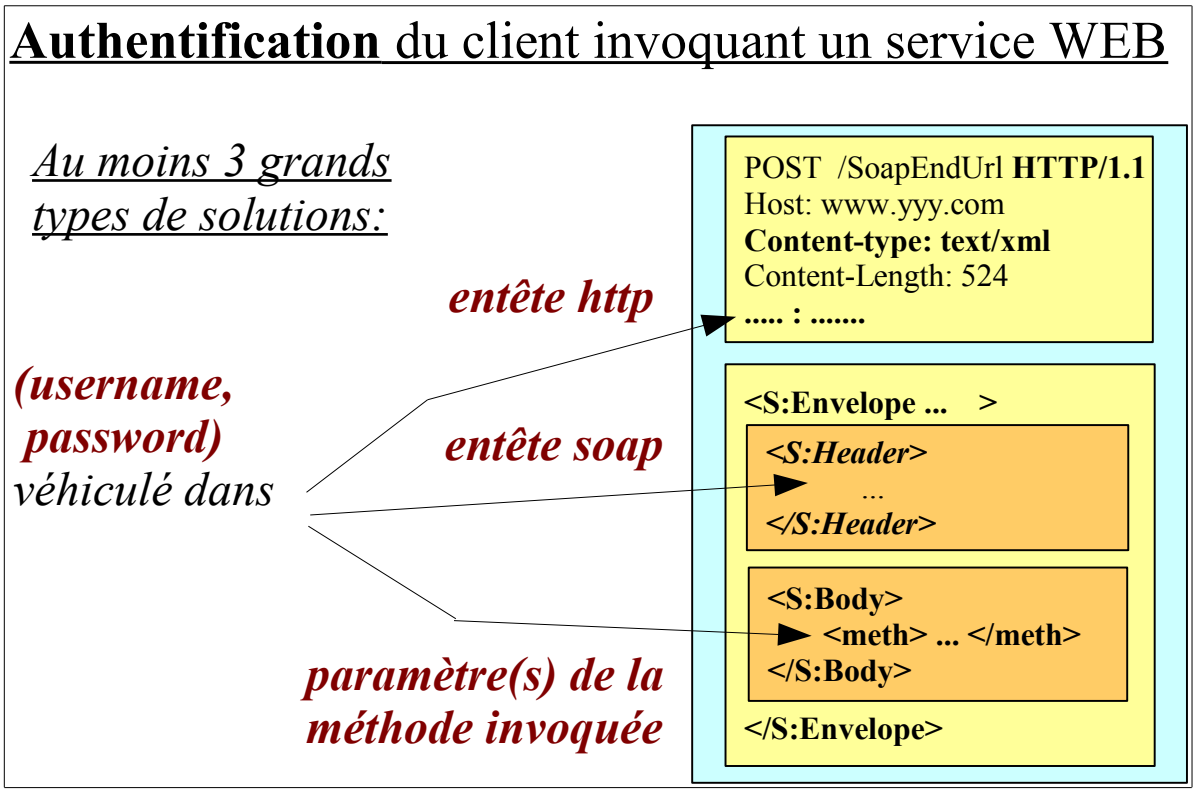
- ***Cryptage / confidentialité***
  --> via ***HTTPS***/SSL et/ou cryptage XML/SOAP

- **Signature électronique des messages** (certificats)

- ***Intégrité*** des messages ***(vérif. non interception/modification)***
  --> via signature d'une empreinte d'un message

- ...

1.2. différents types d'authentifications (WS)



1.3. authentification élémentaire / fonctionnelle via "jeton"

La technique élémentaire d'authentification associée aux services WEB correspond à celle qui est utilisée au niveau de UDDI à savoir:

1) appel d'une méthode de type
   String sess*Token* = ***getSessionToken***(String **userName**,String **password**)
   qui renvoie un jeton de sécurité lié à une session après avoir vérifier la validité du couple (username,password) d'une façon ou d'une autre (ex: via ldap).
2) passage du jeton de sécurité "sess*Token*" en tant qu'argument supplémentaire de toutes les autres méthodes du service Web : ***methodeXy***(String sess***Token***, String param2,.....) ; Une simple vérification de la validité du jeton servira à décider si le service accepte ou pas de répondre à la requête formulée.

Variantes classiques:

- **repasser le jeton de sécurité/session au travers d'un argument plus large généralement appelé "contexte"** (au sens "contexte technique" et non métier). Ce contexte pourra ainsi véhiculer d'autres informations techniques jugées pertinentes (ex: transactions , référence xy, ...).
- utiliser en plus **SOAP over HTTPS** pour crypter si besoin certaines informations échangées (ex: [username, password] ou plus).

---> gros inconvénient de la méthode "élémentaire/fonctionnelle" : chaque méthode appelée et sécurisée comporte au moins un paramètre supplémentaire .

Autrement dit , la sécurité se voit dans les signatures des méthodes à invoquer.

1.4. Authentification véhiculée via l'entête (header) "HTTP"

En passant les informations d'authentification (username,password) ou ... dans l'entête HTTP on bénéficie des avantages suivants:

- authentification bien séparée des aspects fonctionnels
- réutilisation d'une fonctionnalité standard d'HTTP "Authentification basique HTTP" (username/password) véhiculés de façon un peu crypté dans un champ de l'entête HTTP.
- paramétrage/codage assez simple à faire via des intercepteurs ou API des technologies "Web Services" *(ex: intercepteur "cxf" , interface "BindingProvider" de JAX-WS)*

Pour encore plus de sécurité , on peut utiliser conjointement SSL/HTTPS .

Cette méthode est très bien dans la plupart des cas.

Limitation:

Dans certains cas pointus , l'enveloppe SOAP est relayée par différents intermédiaires (ex: ESB , ...) . Certains maillons du transport peuvent être effectués par d'autres protocoles que HTTP et les informations de l'entête HTTP risquent alors de ne pas bien être retransmises jusqu'à la destination prévue.

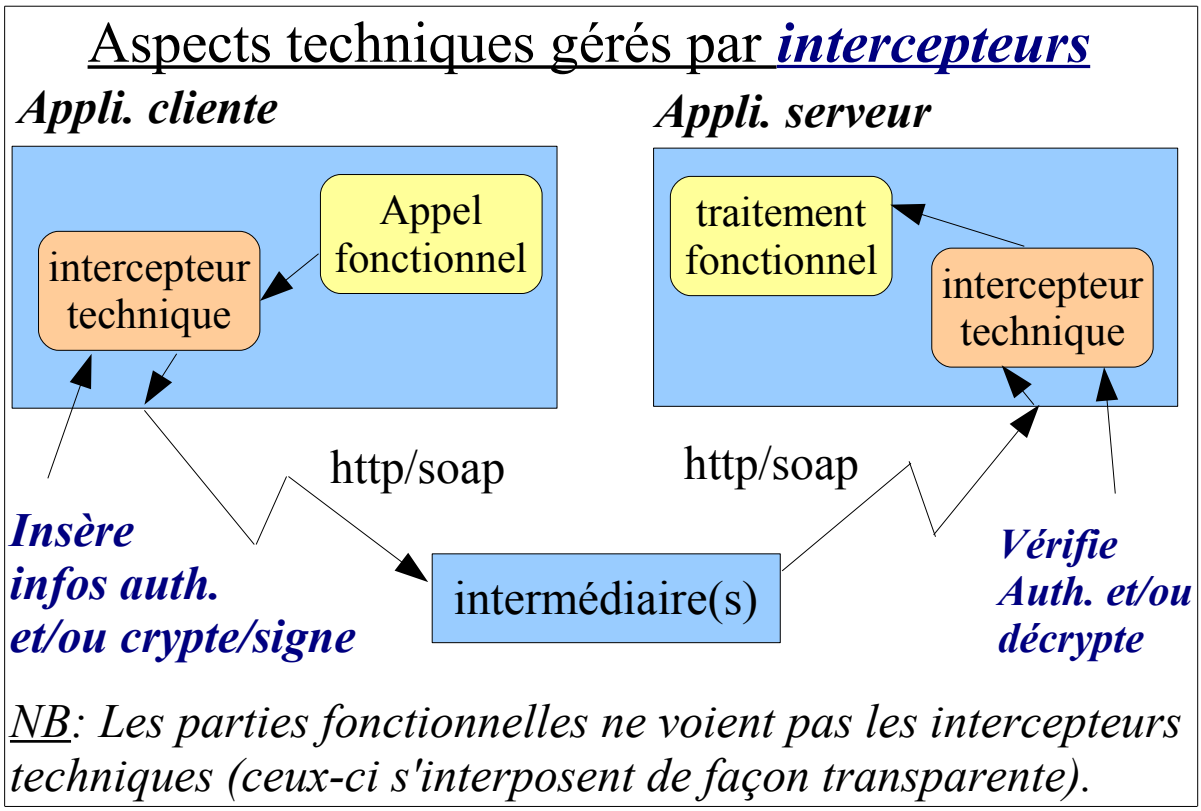
1.5. Authentification véhiculée via l'entête (header) "soap"

Avantage: fonctionne même si plusieurs protocoles de transport (HTTP puis JMS puis ...)

Inconvénients :

- API et/ou intercepteur(s) plus complexe(s)
- Les technologies "client" et "serveur" doivent toutes les 2 être compatibles sur la gestion des entêtes SOAP .

1.6. Aspects techniques gérés par intercepteurs



1.7. Problématique (sécurité avancée)

Bien que permettant globalement une interaction entre 2 parties (client et serveur) , la technologie des services WEB (XML sur HTTP) fait intervenir de multiples intermédiaires à travers le réseau internet.
Le contexte de sécurité SOAP (idéalement géré de bout en bout mais pas toujours pour des raisons de coûts) doit être suffisamment perfectionné pour supporter des éventuelles attaques ou menaces (interception des messages , altération de ceux-ci , ...).

D'autre part , tout service n'est pas forcément gratuit et un accord préalable (abonnement, ...) peut déboucher sur la génération d'un compte utilisateur (username, password) permettant d'accorder finement des droits d'accès au service Web.

Recommandation du ws-i en terme de sécurité:

==> utiliser **HTTPS** (avec SSL 3.0 ou TLS 1.0) et non pas HTTP pour véhiculer l'enveloppe SOAP.
Cette recommandation est toutefois nuancée et insiste sur le compromis **sécurité/sur-coûts** qui doit être évalué au cas par cas.
Dans certains cas SSL (prévu pour un mode point à point) ne suffit pas .
Il faut alors utiliser quelques unes des technologies présentées ci-après.

1.8. Web Service Security (WS-S)

Développé par l'organisme OASIS, **WS-S** est une **extension de SOAP** permettant de contrôler de bout en bout les éléments suivants sur certains messages:

- intégrité (via *Xml-Signature* et *Security token* (ex: *certificat X-509* ou *ticket kerberos* encapsulés dans des éléments XML))
- confidentialité (via *Xml-Encrytion* + *Security token*)
- authentification

NB:

WS-S indique comment représenter en XML des jetons de sécurité mais n'impose pas un type précis de certificat .
Ces jetons (certificats) servent essentiellement à s'assurer que les messages SOAP n'ont pas été interceptés ni modifiés.
Un peu à la façon de SSL , WS-S agit de façon transparente en ajoutant automatiquement des éléments de sécurité sur les messages "SOAP" .

Exemple (partiel):

```
<SOAP:Envelope xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP:Header>
    <wsse:Security
              xmlns:wsse='http://schemas.xmlsoap.org/ws/2002/07/secext'>
      <Signature xmlns='http://www.w3.org/2000/09/xmldsig#'>
        ...defined below...
      </Signature>
    </wsse:Security>
  </SOAP:Header>
  <SOAP:Body id='Body'>
      ...message body...
  </SOAP:Body>
</SOAP:Envelope>
```

1.9. Aspects avancés liés à la sécurité

==> Approfondir si besoin WS-Security (Xml-Signature, Xml-encryption, ...) pour les cas "pointus" .

2. Authentification basique http avec JAX-WS

L'authentification basique Http (la plus classique) consiste à véhiculer les informations d'authentification (basiquement cryptées) dans l'entête Http des requêtes Http véhiculant l'enveloppe SOAP .

2.1. Coté client

L'interface cachée "***BindingProvider***" peut servir à renseigner les paramètres d'authentification coté client:

```
Calculateur calculateur = serviceCalculateur.getCalculateurServicePort();
```

```
// ou calculateur = (Calculateur) service.getPort(PORT_NAME, Calculateur.class);
javax.xml.ws.BindingProvider bp = (javax.xml.ws.BindingProvider) calculateur;
Map<String,Object> context = bp.getRequestContext();
context.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                        "http://localhost:1234/myWebApp/services/calculateur");
context.put(BindingProvider.USERNAME_PROPERTY,"userNameSiBasicHttpAuth");
context.put(BindingProvider.PASSWORD_PROPERTY, "pwdSiBasicHttpAuth");
System.out.println(calculateur.getTva(200, 19.6));
```

2.2. Coté serveur (avec CXF)

Récupérer le code de ***BasicAuthAuthorizationInterceptor.java*** dans **src .**

Ajouter cette configuration spring:

```
...
<bean id="securityInterceptor" class="BasicAuthAuthorizationInterceptor">
    <property name="users">
      <map>
        <entry key="user1" value="pwd1"/>
        <entry key="user2" value="pwd2"/>
      </map>
    </property>
  </bean>

  <jaxws:endpoint id="calculateurEndPoint"
                    implementor="#calculateurService_impl" address="/calculateur" >
              <jaxws:inInterceptors>
                    <ref bean="securityInterceptor"/>
              </jaxws:inInterceptors>
  </jaxws:endpoint>
```

code de la classe ***BasicAuthAuthorizationInterceptor.java***

```
import java.io.IOException;                          import java.io.OutputStream;
import java.net.HttpURLConnection;                   import java.util.Arrays;
import java.util.List;                               import java.util.Map;
import org.apache.cxf.binding.soap.interceptor.SoapHeaderInterceptor;
import org.apache.cxf.configuration.security.AuthorizationPolicy;
import org.apache.cxf.endpoint.Endpoint;             import org.apache.cxf.interceptor.Fault;
import org.apache.cxf.message.Exchange;              import org.apache.cxf.message.Message;
import org.apache.cxf.transport.Conduit;
```

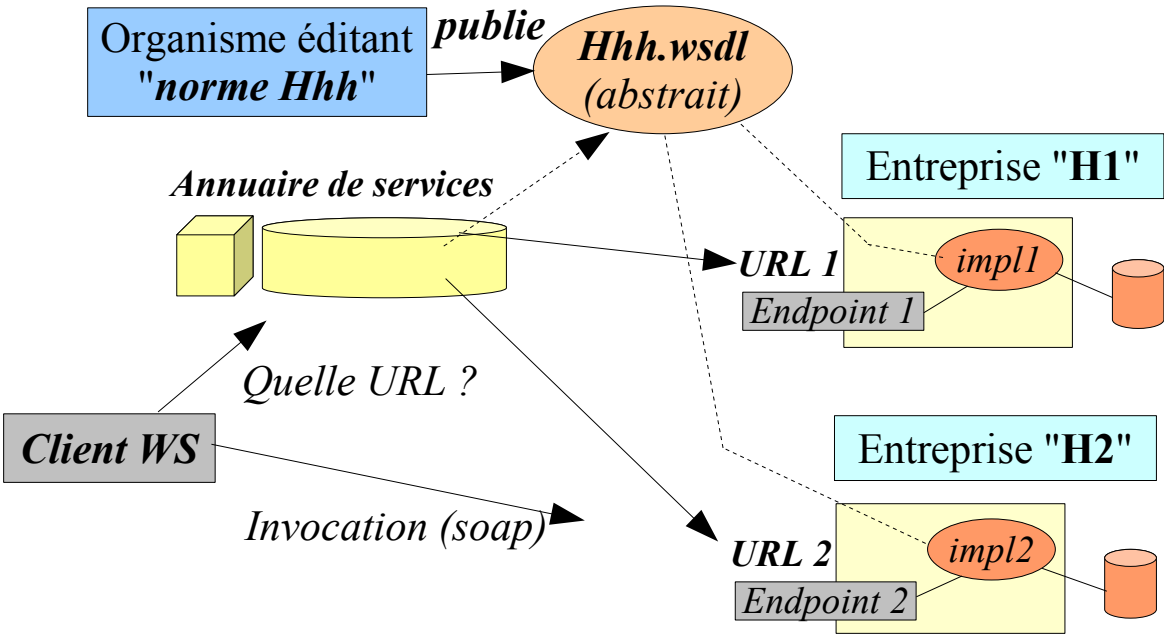
```
import org.apache.cxf.ws.addressing.EndpointReferenceType;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Required;

/**
 * CXF Interceptor that provides HTTP Basic Authentication validation. Based on the concepts outline
 *  here: http://chrisdail.com/2008/03/31/apache-cxf-with-http-basic-authentication
 * @author CDail
 */
public class BasicAuthAuthorizationInterceptor extends SoapHeaderInterceptor {
    protected Logger log = Logger.getLogger(getClass());

    /** Map of allowed users to this system with their corresponding passwords. */
    private Map<String,String> users;

    @Required
    public void setUsers(Map<String, String> users) {
        this.users = users;
    }

    @Override public void handleMessage(Message message) throws Fault {
        // This is set by CXF
        AuthorizationPolicy policy = message.get(AuthorizationPolicy.class);

        // If the policy is not set, the user did not specify credentials
        // A 401 is sent to the client to indicate that authentication is required
        if (policy == null) {
            if (log.isDebugEnabled()) {
                log.debug("User attempted to log in with no credentials");
            }
            sendErrorResponse(message, HttpURLConnection.HTTP_UNAUTHORIZED);
            return;
        }
        if (log.isDebugEnabled()) {
            log.debug("Logging in use: " + policy.getUserName());
        }

        // Verify the password
        String realPassword = users.get(policy.getUserName());
        if (realPassword == null || !realPassword.equals(policy.getPassword())) {
            log.warn("Invalid username or password for user: " + policy.getUserName());
            sendErrorResponse(message, HttpURLConnection.HTTP_FORBIDDEN);
        }
    }

    private void sendErrorResponse(Message message, int responseCode) {
        Message outMessage = getOutMessage(message);
        outMessage.put(Message.RESPONSE_CODE, responseCode);

        // Set the response headers
        Map<String, List<String>> responseHeaders =
            (Map<String, List<String>>)message.get(Message.PROTOCOL_HEADERS);
```

```
        if (responseHeaders != null) {
            responseHeaders.put("WWW-Authenticate", Arrays.asList(new String[]{"Basic realm=realm"}));
            responseHeaders.put("Content-Length", Arrays.asList(new String[]{"0"}));
        }
        message.getInterceptorChain().abort();
        try {
            getConduit(message).prepare(outMessage);
            close(outMessage);
        } catch (IOException e) {   log.warn(e.getMessage(), e);
          }
    }

    private Message getOutMessage(Message inMessage) {
        Exchange exchange = inMessage.getExchange();
        Message outMessage = exchange.getOutMessage();
        if (outMessage == null) {
            Endpoint endpoint = exchange.get(Endpoint.class);
            outMessage = endpoint.getBinding().createMessage();
            exchange.setOutMessage(outMessage);
        }
        outMessage.putAll(inMessage);       return outMessage;
    }

    private Conduit getConduit(Message inMessage) throws IOException {
        Exchange exchange = inMessage.getExchange();
        EndpointReferenceType target = exchange.get(EndpointReferenceType.class);
        Conduit conduit =  exchange.getDestination().getBackChannel(inMessage, null, target);
        exchange.setConduit(conduit);
        return conduit;
    }

    private void close(Message outMessage) throws IOException {
        OutputStream os = outMessage.getContent(OutputStream.class);
        os.flush();        os.close();
    }
}
```

IX - Annuaires de services (UDDI , ...)

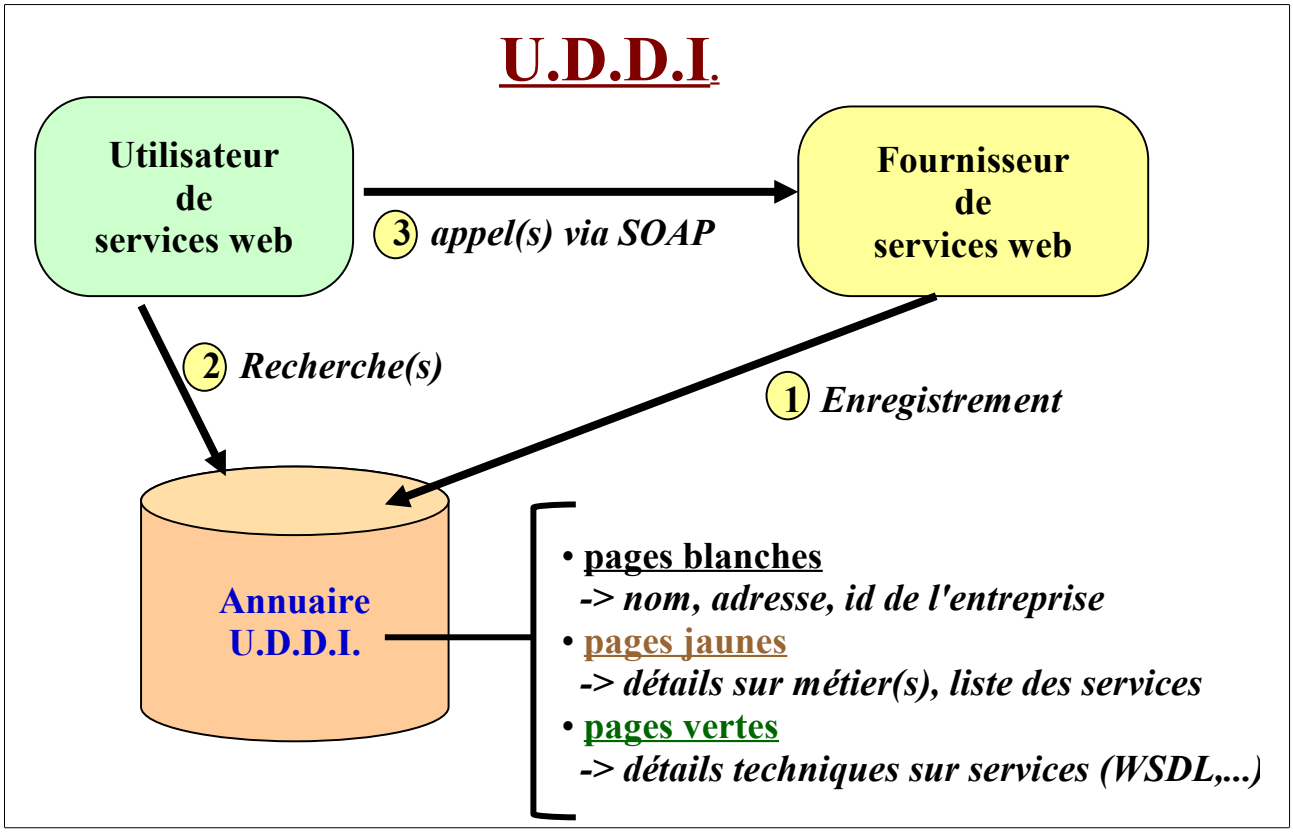
1. Utilité d'un annuaire de "service WEB"

Un **service** fonctionnel *(avec WSDL abstrait)* peut avoir **plusieurs implémentations** avec **URL(s) à découvrir** via un **annuaire**.



2. Annuaires UDDI (présentation)

UDDI signifie *Universal **D**escription, **D**iscovery & **I**ntegration*



Universal Description, Discovery & Integration ***(http://www.uddi.org)***

UDDI est un **service d'annuaire basé sur Xml** essentiellement prévu pour **référencer des services "web"**. Le noyau du projet UDDI est l'***UDDI Business Registry***, annuaire contenant des informations de trois types, décrites au format XML :

- **Pages blanches** : noms, adresses, contacts, identifiants,… des entreprises enregistrées. Ces informations sont décrites dans des entités de type ***Business Entity***. Cette description inclut des informations de catégorisation permettant de faire des recherches spécifiques dépendant du métier de l'entreprise ;
- **Pages jaunes** : *détails sur le métier de l'entreprise, les services qu'elle propose*. Ces informations sont décrites dans des entités de type ***Business Service*** ;
- **Pages vertes** : *informations techniques* sur les services proposés. Les pages vertes incluent des références vers les spécifications des services Web, et les détails nécessaires à l'utilisation de ces services : interfaces implémentées, information sur les contacts pour un processus particulier, description du processus en plusieurs langages, catégorisation des processus, pointeurs vers les spécifications décrivant chaque API. Ces informations sont décrites dans deux documents : un *Binding Template*, et un *Technology Model (tModel)*.

3. Annuaires publics et annuaires privés

3.1. Principaux annuaires publics

NB: La plupart des annuaires UDDI publics (mise en place par IBM et Microsoft) ne sont plus maintenus.

Raisons prétextées:

- Technologie au point , tests terminés
- Ce n'est pas au éditeurs de logiciels (IBM ou Microsoft , ...) de maintenir des annuaires publics (ce n'est pas leurs métiers)

Conséquences:

La technologie UDDI à un niveau privé perd un peu de son intérêt mais il faut accepter cet état de fait (tant qu'un organisme international indépendant ne prendra pas à sa charge un annuaire UDDI véritablement public).

3.2. Quelques implémentations d'annuaires UDDI

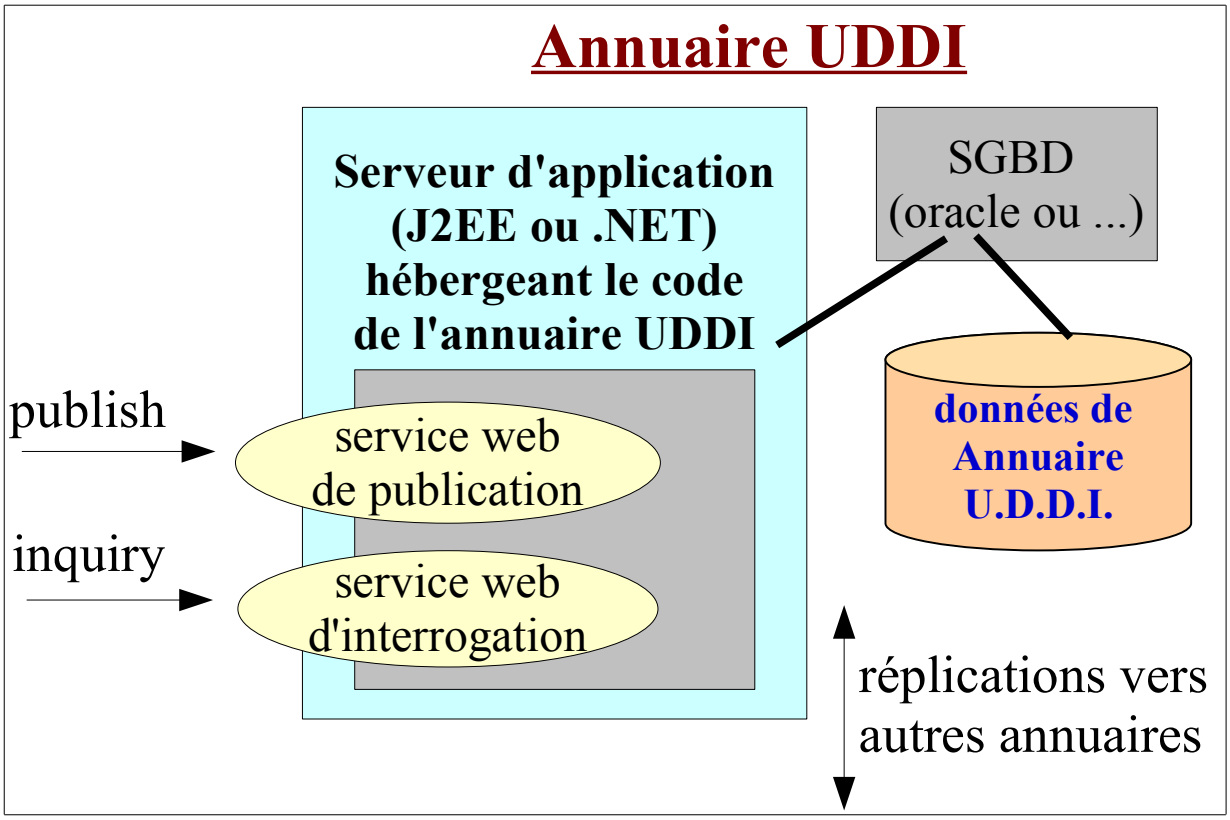
IBM/WebSphere/AppServer/InstallableApps/**uddi.ear** à installer dans **WebSphere 6**

JUDDI (***http://ws.apache.org/juddi***) ===> **juddi.ear** (ou **juddi.war**) = produit open source à installer dans n'importe quel conteneur Web récent (ex: Tomcat 5) ou serveur J2EE (ex: JBoss) .

NB: L'application J2EE correspondant à un annuaire UDDI nécessite généralement la mise en place d'une base de données relationnelle ainsi qu'un pool de connexions JDBC. ==> Lire la documentation de JUDDI ou autre pour connaître la procédure d'installation.

4. Structure d'un annuaire UDDI

4.1. Infrastructure physique d'un annuaire UDDI



Connexion à l'annuaire avec authentification :

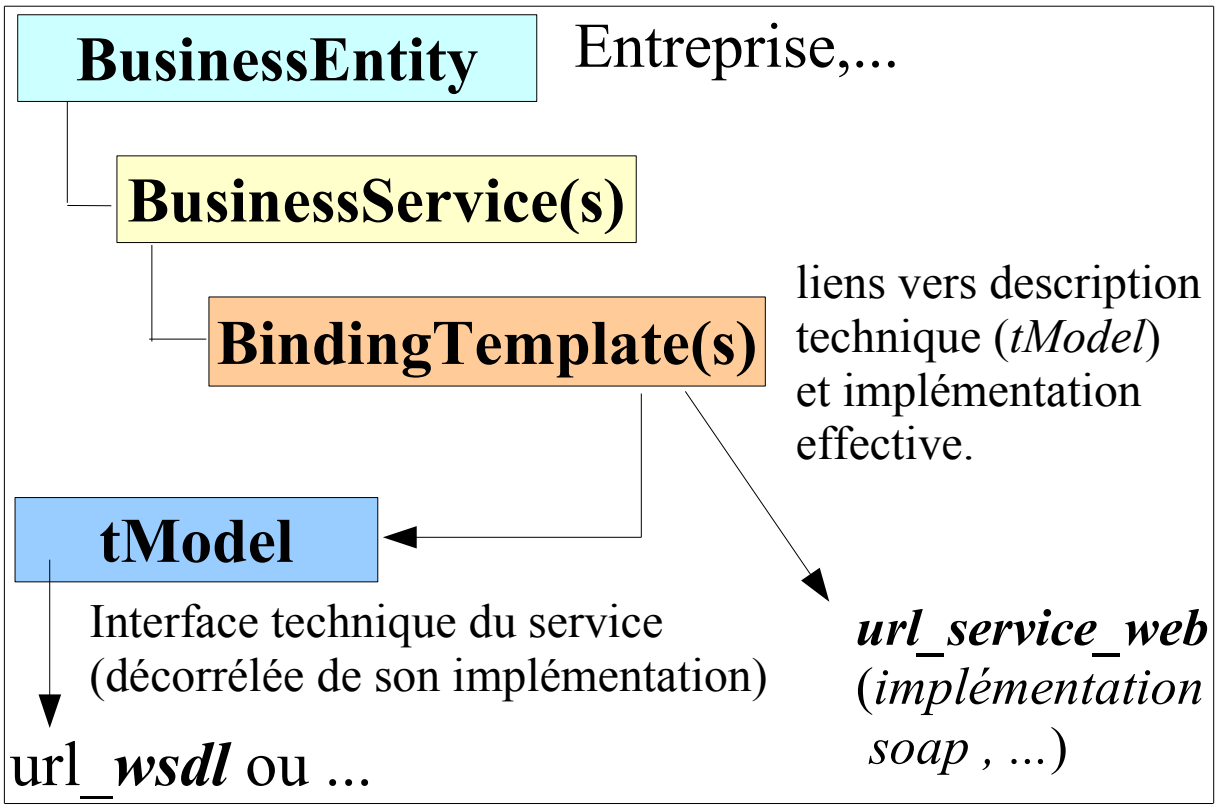
===> connexion UDDI (username , password)

<=== authentificationToken (authTokenString)

===> requêtes_ultérieures_uddi (authTokenString , autresParamètres)

NB: Pour publier des informations dans un annuaire UDDI , il faut préalablement s'inscrire de façon à disposer d'un compte (username,password).

4.2. Structure logique d'un annuaire UDDI



NB:

- La branche "**BusinessEntity / BusinessService / BindingTemplate**" correspond à des informations qui décrivent le fournisseur du service (sa mise en oeuvre concrète).

- La branche "**tModel**" correspond à une description technologique mais abstraite du service. Un enregistrement "tModel" (et le WSDL associé) est censé décrire une interface d'appel , une éventuelle partie d'une norme B2B ou d'un protocole inter-entreprise.

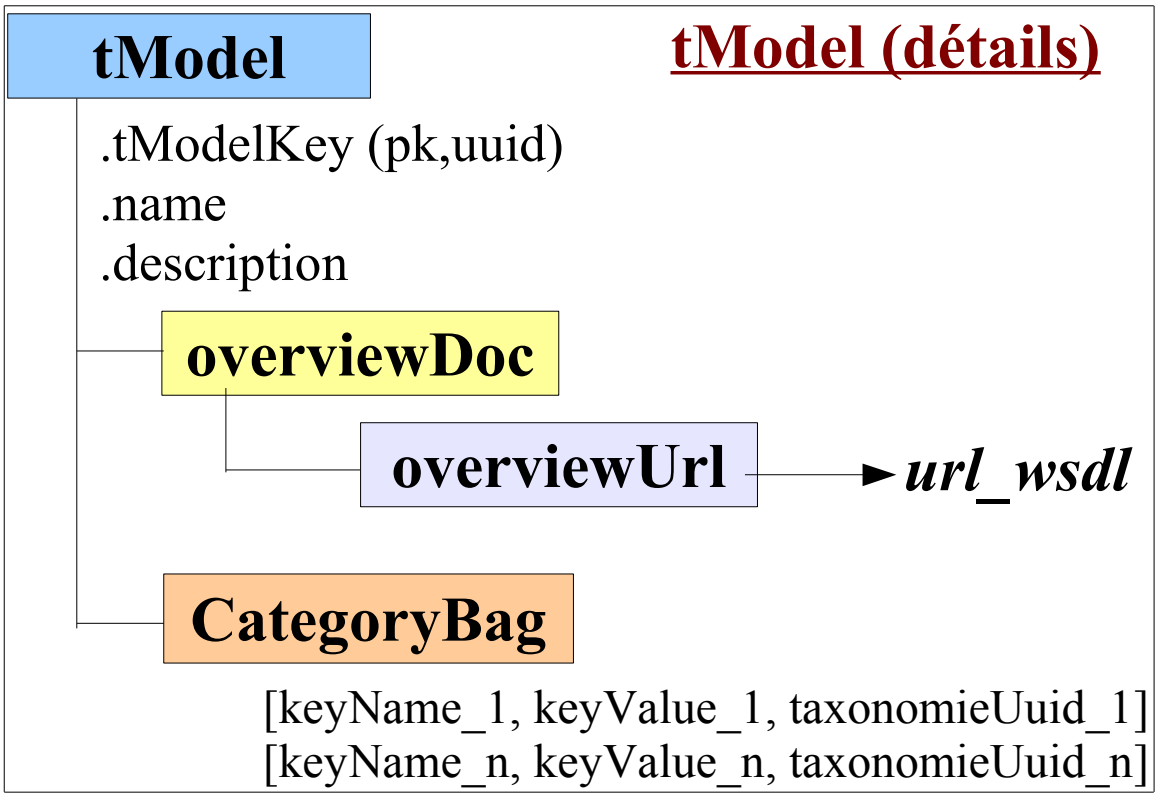
Ces deux branches sont à bien dissocier.

Remarque: le fait qu'un fichier WSDL comporte souvent (mais pas obligatoirement) l'url concrète d'un service WEB apporte quelquefois un peu de confusion.

Remarque générale:

La plupart des données qui sont enregistrées dans un annuaire UDDI (businessEntity , tModel , ...) sont identifiées par des ***clefs primaires*** au format ***uuid*** (*universal unique id*) à l'image des *uuid* qui sont utilisés dans la base de registre de Windows (chaîne hexadécimal très longue et unique de par l'algorithme de génération tenant compte de la date, de l'heure , de l'ordinateur , ...).

Il est fondamental de récupérer/retenir ou bien rechercher la valeur d'un uuid généré lors d'un enregistrement de façon à ne pas enregistrer plusieurs fois les mêmes informations (lors des mises à jour ultérieures au premier enregistrement) .



- Un ***tModel*** devrait idéalement avoir un ***nom*** de type URI (ex: "*MonEntreprise_com_ou_org:MyWebServiceInterface:v1*") et devra pointer sur l'emplacement du fichier WSDL.

- ***L'url du fichier WSDL*** doit être renseignée comme valeur de la propriété "***overviewUrl***" de la sous partie "***overviewDoc***".

- Si le tModel fait référence à un fichier wsdl (ce qui est souvent le cas) , on doit normalement faire apparaître l'utilisation de la norme WSDL dans l'une des entrées de CategoryBag :

  KeyedReference kr = new KeyedReference("***uddi-org:types***", "***wsdlSpec***");

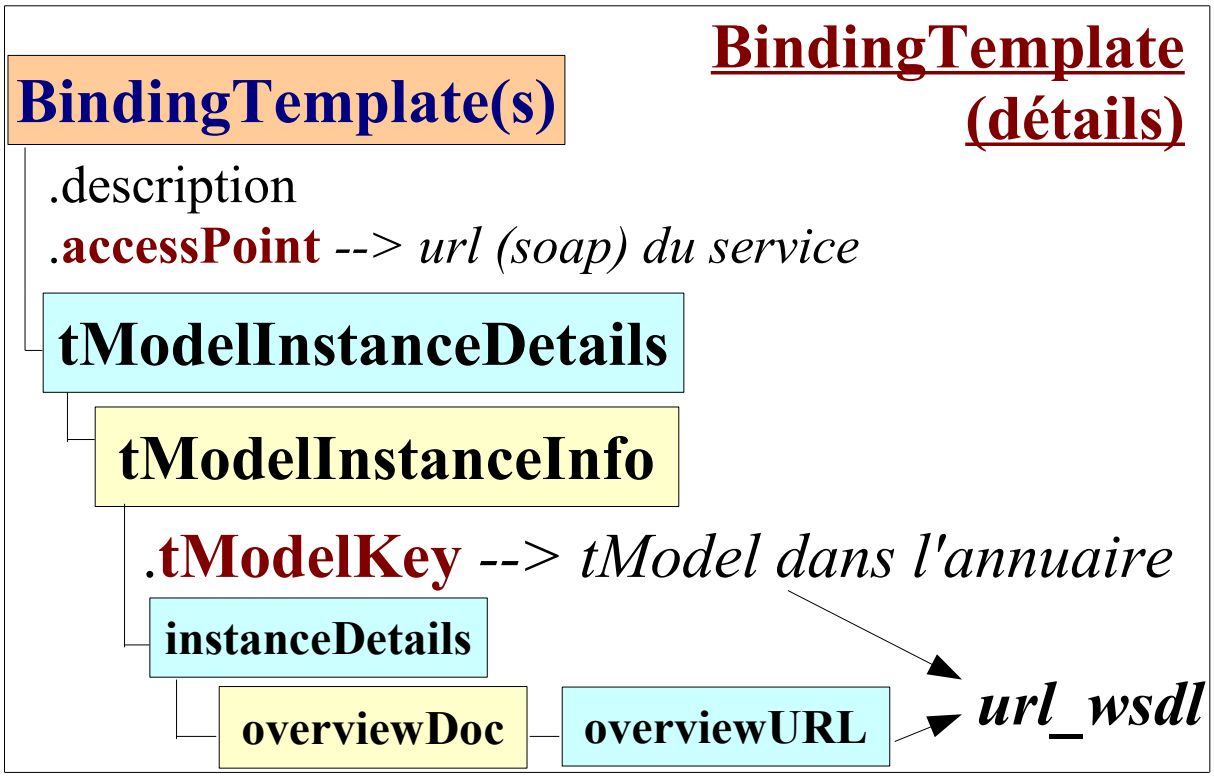
  kr.setTModelKey("***UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4***");

  krList.add(kr);

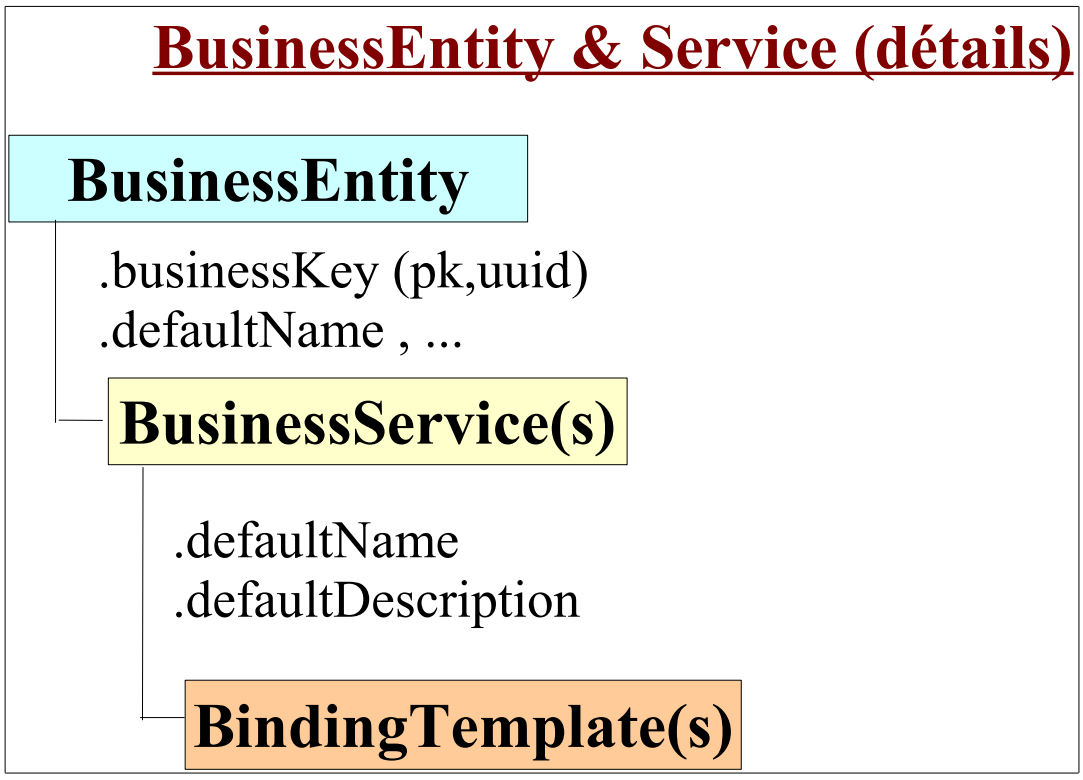
- Les autres éléments de "***CategoryBag***" correspondent à ***éléments de catégorisation*** (appelés "*taxonomies*"). Ces taxonomies (qui seront ultérieurement approfondies en fin de chapitre) permettent d'associer des ***critères géographiques ou professionnels (ex: banche métier) qui seront utiles lors des recherches.***

NB:

- il est important de récupérer le tModelKey (clef primaire de type "UUID:.....") qui a été affecté au tModel lors de son enregistrement. Cet identifiant de tModel devra être ultérieurement renseigné au sein d'un "BindingTemplate".
- Un tModel générique (associé à différentes implémentations d'un service abstrait) peut quelquefois être enregistré par un organisme de normalisation (consortium , ...).



Bien qu'un peu technique (sur le plan de sa composition en sous éléments), un "***BindingTemplate***" peut être simplement considéré comme un ***double lien*** (un lien externe "***accessPoint***" vers une implémentation concrète de Service WEB – url soap , et un autre lien vers un "***tModel***" référençant lui même un fichier externe WSDL décrivant l'interface abstraite du service).



==> Description de l'entreprise et ses services. Attention au businessKey : pas de doublon

5. taxonomies

Il existe plusieurs sortes de taxonomies (UNSPSC , ...) et de nouvelles peuvent éventuellement apparaître. Une entrée de taxonomie comporte les 3 éléments suivants:

- l'***identifiant (au format UUID) de la taxonomie*** (lié à un organisme et une version)
- le ***nom*** de l'entrée = *valeur en clair* (ex: "France" , "Accounting Software" , ...)
- la ***valeur*** de l'entrée = *valeur codée* (ex: "Fr" , "43.23.16.01")

Le nom est pratique pour un affichage et une compréhension humaine.

La valeur est pratique et bien souvent nécessaire pour effectuer une recherche précise.

NB: Certaines taxonomies ont un attribut "Checked" à "true". Ceci signifie que la valeur sera vérifiée .

5.1. UNSPSC

The **United Nations Standard Products and Services Code** is a hierarchical convention that is used to classify all products and services.

Each level in the hierarchy has its own unique number:

XX Segment

[The logical aggregation of families for analytical purposes]

XX Family

[A commonly recognized group of inter-related commodity categories]

XX Class

[A group of commodities sharing common characteristics]

XX Commodity

[A group of substitutable products or services]

XX Business Function (optional)

[The function performed by an organization in support of the commodity]

All UNSPSC entities are further identified with an 8-digit structured numeric code which both

indicates its location in the taxonomy and uniquely classifies it. An additional 2-digit suffix indicates the business function identifier.

A structural view of the code set would look as follows:

Hierarchy Category Number and NameSegment:

43 *Information Technology Broadcasting and Telecommunications Communications Devices and Accessories*

Family :

20 *Components for information technology or broadcasting or telecommunications Computer Equipment and Accessories*

Class:

15 *Computers Computer accessories*

Commodity:

01 *Computer switch boxes Docking stations*

Business Function:

14 *Retail*

==========> ***43.20.15.01.14***

Autre exemple: 43.23.16.01 , Accounting Sofware

Name: unspsc-org:unspsc

Description: Product and Services Taxonomy: UNSPSC (Version 7.3 remplaçant 3.1)

tModel UUID: **uuid:CD153257-086A-4237-B336-6BDCBDCC6634**

Categorization: categorization

Checked: Yes

ex:

```
<categoryBag>
    <keyedReference keyName="Domestic Air Cargo Transport"
    keyValue="78.10.15.01.00"
    tModelKey = "uuid:CD153257-086A-4237-B336-6BDCBDCC6634"/>
 </categoryBag>
```

5.2. ISO_3166 Geographic_Taxonomy

tModels:

Name: uddi-org:iso-ch:3166-1999

Description: ISO 3166-1:1997 and 3166-2:1998. Codes for names of countries and their subdivisions. Part 1: Country codes. Part 2: Country subdivision codes. Update newsletters include ISO 3166-1 V-1 (1998-02-05), V-2 (1999-10-01), ISO 3166-2 I-1 (1998).

tModel UUID: **uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88**

Categorization: categorization

Checked: Yes

---

http://www.uddi.org/taxonomies/iso3166-1999-utf8.txt

---

exemple : valeur (codée) , nom (en clair)

ex1 (région): ***FR-J*** , Ile-De-France

ex2 (département): ***FR-75*** , Paris

ex3 (pays): ***FR*** , France

```
<categoryBag>
         <!-- Categorize this businessEntity as serving California using the ISO 3166
               taxonomy -->
         <keyedReference keyName="California, USA"
         keyValue="US-CA"
         tModelKey="uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88"/>
    </categoryBag>
```

ANNEXES

X - Schémas XML (XSD)

1. Présentation des schémas W3C

Les **schémas W3C** constituent une **alternative aux DTD**.
Cette technologie proposée à l'origine par Microsoft , a été normalisée (après de nombreux ajustements) par l'organisme W3C en 2001 (soit assez longtemps après les DTD).
Beaucoup de logiciels récents utilisent aujourd'hui les schémas (*ex*: J2EE 1.4 , JEE 5 & 6) alors que d'anciens programmes ou d'anciennes versions utilisaient les DTD (*ex*: J2EE 1.3).

Les schémas W3C offrent les principaux avantages suivants:

- **syntaxe homogène vis à vis d'XML** (un schéma est codé comme un cas particulier de fichier XML)
- On peut attacher des **types de données très précis** (string,integer,date,...) aux éléments et aux attributs
- Certains aspects des schémas sont assez proches des concepts objets .

Schéma XML (.xsd) plus évolués que DTD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name='pays' type='PaysType' />
  <xsd:complexType name="PaysType">
          <xsd:sequence>
               <xsd:element name="region" type="RegionType" maxOccurs='22' />
          </xsd:sequence>
          <xsd:attribute name="nom" type="xsd:string"/>
          <xsd:attribute name="capitale" type="xsd:string"/>
  </xsd:complexType>
 <xsd:complexType name="RegionType">...</xsd:complexType>
  <xsd:complexType name="DepartementType">
          <xsd:attribute name="nom" type="xsd:string"/>
          <xsd:attribute name="num" type="xsd:positiveInteger"/> ....
  </xsd:complexType>
</xsd:schema>
```

france.xsd

Types précis

```
<pays xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="france.xsd"
        nom='France' capitale='Paris' >
   <region nom='Haute-Normandie' >   <departement nom='Eure' num='27' ... />
...</region> ... </pays>
```

2. Lien entre un document XML et un schéma

Un **schéma** correspond à un fichier ayant l'extension **xsd** (*Xml Schema Definition*).

Un tel fichier (.xsd) décrit la structure d'une **classe** de documents .

Un ***fichier XML conforme au schéma*** sera considéré comme une **instance** .

De façon à associer un schéma à un document XML, on aura recours à l'une des deux syntaxes suivantes:

```
<rootElement  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:noNamespaceSchemaLocation="schema_yyy.xsd" >
...
<rootElement>
```

ou bien

```
<p:rootElement  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.yyy.com/namespaceYYY      schema_yyy.xsd"
                    xmlns:p="http://www.yyy.com/namespaceYYY">
...
<p:rootElement>
```

La localisation d'un schéma (.xsd) s'introduit donc via de nouveaux attributs que l'on rattache à la balise de premier niveau .

L'attribut "***schemaLocation***" (ou bien "***noNamespaceSchemaLocation***") sera pris en compte par **parseur en mode validant** (qui de plus doit être *paramétré pour tenir compte des schémas W3C*).

Pour effectuer rapidement quelques tests de conformité, on pourra utiliser un des outils suivants:

- **XML-Spy** (sous Windows) *(bon produit commercial avec version d'évaluation)*
- **XSD** (Xml Schema Validator) *(freeware en mode texte)*
- **ValidApp** (petit programme utilisant JAXP à écrire en Java)
- **Plugin eclipse**
- ...

3. Structure d'un schéma

3.1. Terminologie (types simples et complexes)

Au sein de la définition d'un schéma, on distingue:

- les **types complexes** correspondant à des éléments qui ont ***au moins un sous élément ou bien un attribut*** .
- les **types simples** correspondant à des ***valeurs atomiques*** (qui ne se décomposent pas).

Beaucoup de types simples sont prédéfinis (*ex*: **xsd:string** , **xsd:integer** , **xsd:decimal** , ...).
Ces *types simples* seront utilisés pour caractériser des *attributs* ou des *balises feuilles (terminales)* du genre "titre" , "prenom" , "age" , ...
On peut en définir de nouveaux en précisant de nouvelles contraintes (motif à respecter pour une chaîne de caractères , plage de valeurs obligatoire ***[*** *ex*: *type_age = positiveInteger* ayant *150 comme valeur maximale]*)

Au sein d'un document XML , toute ***balise xxx qui comporte des sous balises*** doit clairement être associée à un **type complexe** (souvent nommé *xxxType*) . A un arbre XML comportant *N niveaux de profondeur* , correspond généralement un schéma comportant *N types complexes principaux* .

Petite subtilité:

<valeur>1200</valeur> est considérée comme étant de ***type simple***

<**prix** ***monnaie="euro"***>1200</**prix**> est considérée comme étant de **type complexe**

3.2. Structure globale d'un schéma (.xsd)

1. Un bloc ***<annotation>*** fait généralement office de commentaire et permet d'indiquer la raison d'être du schéma.

2. Figure ensuite la définition d'**un ou plusieurs éléments** (balises) que l'on peut potentiellement trouver en tant que **point d'entrée du document**.
   On trouve également à ce **niveau global** des définitions de sous éléments récurrents qui seront plus tard référencés via *<xsd:element* ***ref****="nom_element_global" />* .

3. La plus grande partie du schéma est ensuite structurée par une série de définitions de types complexes (chaînés entre eux via *<xsd:element* .... ***type****="type_sous_élément"/>*).

3.3. Exemple complet (pour vue d'ensemble)

personne.xml

```
<?xml version='1.0'?>
<personne age="35"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="personne.xsd">
        <nom>Defrance</nom>
        <prenom>Didier</prenom>
        <adresse>
            <rue>10 rue Elle</rue>
            <cp>75000</cp>
            <ville>Paris</ville>
        </adresse>
</personne>
```

est conforme au schéma ci-après

personne.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
                <xsd:documentation>Schéma pour personne avec adresse</xsd:documentation>
  </xsd:annotation>

  <xsd:element   name="personne"   type="persType" />

  <xsd:complexType name="persType">
                <xsd:sequence>
                        <xsd:element  name="nom"   type="xsd:string"/>
                        <xsd:element  name="prenom"  type="xsd:string" maxOccurs="3" />
                        <xsd:element  name="adresse"  type="adrType" />
                </xsd:sequence>
                <xsd:attribute name="age" type="myLittleInteger"/>
  </xsd:complexType>

  <xsd:complexType name="adrType">
                <xsd:sequence>
                        <xsd:element name="rue" type="xsd:string"/>
                        <xsd:element name="cp" type="xsd:string"  />
                        <xsd:element name="ville" type="xsd:string" />
                        <xsd:element name="pays" type="xsd:string" minOccurs="0" />
                </xsd:sequence>
  </xsd:complexType>

<xsd:simpleType name="myLittleInteger">
     <xsd:restriction base="xsd:positiveInteger">
       <xsd:maxInclusive value="149"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

3.4. Types simples prédéfinis

3.4.a. Principaux types prédéfinis:

xsd:**string**	Chaîne de caractères quelconque
xsd:**integer**	Nombre entier (*ex*: -126789, -1, 0, 1, 126789)
xsd:**positiveInteger**	Nombre entier positif ou nul (*ex*: 0, 1, 126789)
xsd:**decimal**	Nombre décimal (à virgule éventuelle) (*ex*: -1.23, 0, 123.4, 1000.00)
xsd:**date**	Date au format international / US (aaaa-mm-dd) (*ex*: 1999-05-31)
xsd:**boolean**	Booléen (*ex*: true , false , 0 , 1)
xsd:**hexBinary**	Valeur binaire codé en hexa (base 16) (*ex*: 0BF7)
xsd:**long**	Entier long
xsd:**float** , xsd:**double**	Nombre réel en simple précision (32bits) et double précision (64 bits) (*ex*: -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN)
xsd:**time**	heure au format HH:MM:SS.mmm (*ex*: 13:20:00.000)
xsd:**anyURI**	URI (ou URL) quelconque (*ex*: http://www.yyy.com/page1.html#p1)
xsd:**language**	Code de langue (*ex*: en-GB , en-US , fr , ...)
xsd:**ID**	Identificateur unique au sein d'un document
xsd:**IDREF**	Référence (renvoi) vers un ID

3.4.b. Autres types prédéfinis:

***(xsd:)* token , byte , normalizedString, unsignedByte , base64Binary, negativeInteger, short, int, unsignedInt, unsignedLong , unsignedShort , dateTime, duration, gMonth , gYear , gDay , Name , QName , NCName , ENTITY , NOTATION , ...**

==> consulter la documentation de référence pour approfondir le sujet .

3.5. Types simples personnalisés

Ex1: (entier restreint à l'intervalle [1,99])

```
<xsd:simpleType name="myInteger">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="1"/>
      <xsd:maxInclusive value="99"/>
  </xsd:restriction>
</xsd:simpleType>
```

***Ex2*:(chaîne de caractères devant se conformée à un certain format / expression régulière)**

Format içi imposé : 2 chiffres , suivis d'un tiret , lui même suivi de 3 lettres majuscules.

```
<xsd:simpleType name="RefProdType">
 <xsd:restriction base="xsd:string">
       <xsd:pattern value="\d{2}-[A-Z]{3}"/>
 </xsd:restriction>
</xsd:simpleType>
```

***Ex3*: (Enumération)**

```
<xsd:simpleType name="Mois">
  <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Janvier"/>
      <xsd:enumeration value="Février"/>
      <xsd:enumeration value="Mars"/>
       <!-- and so on ... -->
  </xsd:restriction>
</xsd:simpleType>
```

Ex4: (TypeSimple "liste de ...")

```
<xsd:simpleType name="listOfMyIntType">
     <xsd:list   itemType="myInteger"/>
</xsd:simpleType>
```

validant un élément XML du type suivant:

<listOfMyInt> 20003 15037 95977 95945*</listOfMyInt>*

Voir la documentation de référence pour plus de détails.

3.6. Types complexes

3.6.a. Syntaxe générale (élément composé de sous élément(s) et d'attribut(s))

```
<xsd:complexType name="persType">
        <xsd:sequence>
            <xsd:element  name="nom"   type="xsd:string"/>
            <xsd:element  name="prenom"  type="xsd:string" maxOccurs="3" />
            <xsd:element  name="adresse"  type="adrType" />
        </xsd:sequence>
        <xsd:attribute  name="age" type="xsd:positiveInteger"/>
</xsd:complexType>
```

La **liste des sous éléments** doit absolument (si elle existe) être englobée par l'une des trois balises suivantes:

<xsd:**sequence**> .....</xsd:**sequence**> impose un **ordre** pour les sous éléments.

<xsd:**all**> ... </xsd:**all**> signifie tous les sous éléments dans **n'importe quel ordre** .

<xsd:**choice**> ..... </xsd:**choice**> signifie **un** sous élément **parmi les n** possibilités *(XOR)*

Les attributs **minOccurs** (*ex*: ="***0***") et **maxOccurs** (*ex*: ="***unbounded***")
permettent d'indiquer le nombre d'exemplaires d'un sous élément (**0 à n** , ...).

Par défaut, ***minOccurs*** et ***maxOccurs*** sont fixés à "***1***" .

3.6.b. Détails sur les attributs

Outre le fait de comporter un nom et un type , un attribut peut être caractérisé par:

- une valeur par défaut (à préciser via **default**="*valeur_defaut*" dans le schéma)
- une présence éventuellement obligatoire (à indiquer via **use**="*required*")

exemples:

```
<xsd:attribute   name="prix"   type="xsd:decimal"   use="required" />
```

```
<xsd:attribute   name="couleur"   type="xsd:string"   use="optional"   default="noir" />
```

Remarques:

- Par défaut la valeur de **use** est fixée à "*optional*". ***Tout attribut est donc par défaut facultatif***.
- Une application analysant un document XML via un parseur (*ex*: Jaxp/DOM) pourra récupérer

une valeur par défaut dans le cas où un attribut n'a pas été renseigné.

3.7. Syntaxes possibles mais un peu moins lisibles

Les éléments de syntaxe suivants peuvent parfois être pratiques.
Une utilisation abusive est toutefois fortement déconseillée.

3.7.a. Type complexe sans nom et décrit d'une façon imbriquée

Lorsqu'une balise isolée est un type bien particulier qui a très peu de chance d'être ré-utilisé, on peut éventuellement définir le type (simple ou complexe) de celle ci de manière imbriquée:

```
<element    name="quantite">
    <xsd:simpleType>
        <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="1"/>
                <xsd:maxInclusive value="99"/>
        </xsd:restriction>
   </xsd:simpleType>
</element>
```

3.7.b. Référence vers un élément définit au niveau global

```
<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
                <xsd:documentation>Schéma pour livre, chapitre et paragraphe</xsd:documentation>
 </xsd:annotation>

 <xsd:element    name="livre"    type="livreType" />
 <xsd:element    name="titre"    type="xsd:string" />

 <xsd:complexType name="livreType">
                <xsd:sequence>
                        <xsd:element  ref="titre" />
                        <xsd:element  name="chapitre"  type="chapType" maxOccurs="unbounded" />
                </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="chapType">
                <xsd:sequence>
                        <xsd:element  ref="titre" />
                        <xsd:element name="paragraphe" type="paraType"  maxOccurs="unbounded"/>
                </xsd:sequence>
 </xsd:complexType>
...
</xsd:schema>
```

3.8. Types complexes particuliers

3.8.a. Dériver un type complexe d'un type simple ...

... en lui ajoutant un attribut :

```
<xsd:element name="prix" type="prixInternational"/>

<xsd:complexType    name="prixInternational">
    <xsd:simpleContent>
        <xsd:extension base="xsd:decimal">
            <xsd:attribute name="monnaie" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

==> **<prix monnaie="*Euro*" >*12.5*</prix>**

3.8.b. type complexe pour élément mixte

En précisant **mixed="true"** au niveau de la définition d'un type complexe , les éléments de ce type pourront comporter des blocs de texte à coté des sous éléments.

```
<xsd:element name="paragraphe" type="paraType"/>

<xsd:complexType    name="paraType"  mixed="true">
    <xsd:all>
        <xsd:element name="note" type="noteType" minOccurs="0" maxOccurs="unbounded" />
    </xsd:all>
</xsd:complexType>
```

==>

```
<paragraphe>
    Debut texte
    <note comment="important"> Texte de la note </note>
    Fin texte
</paragraphe>
```

3.8.c. type complexe pour élément vide

Un élément vide (sans texte ni sous balise) mais avec des éventuels attributs tel que

```
<param    name="nom_param"    value="valeur_param"   />
```

... peut se schématiser de l'une des 2 façons suivantes:

```
<xsd:complexType>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="value"  type="xsd:string"/>
</xsd:complexType>
```

```
<xsd:complexType   name="parameterType">
    <xsd:complexContent>
        <xsd:restriction  base="xsd:anyType">
            <xsd:attribute name="name" type="xsd:string"/>
            <xsd:attribute name="value" type="xsd:string"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

4. Aspects avancés des schémas

4.1. Prise en compte des namespaces au sein d'un schéma

Exemple:

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

<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:bi="http://www.yyy.com/my_namespace"
      targetNamespace="http://www.yyy.com/my_namespace"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified" >

   <xsd:annotation>
                <xsd:documentation>Base bibliographique avec namespace</xsd:documentation>
   </xsd:annotation>

<xsd:element name="bibliographie" type="bi:bibliographieType"/>

<xsd:element name="titre" type="xsd:string"/>

<xsd:complexType name="bibliographieType">
         <xsd:sequence>
            <xsd:element ref="bi:titre"/>
                <xsd:element name="sujet" type="bi:sujetType" minOccurs="0" maxOccurs="unbounded" />
         </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="sujetType">
         <xsd:sequence>
                <xsd:element ref="bi:titre"/>
                <xsd:element name="livre" type="bi:livreType" minOccurs="0" maxOccurs="unbounded" />
         </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="livreType">
                <xsd:sequence>
                        <xsd:element ref="bi:titre"/>
                        <xsd:element name="auteur" type="xsd:string"/>
                        <xsd:element name="editeur" type="xsd:string"/>
                </xsd:sequence>
                <xsd:attribute name="prix" type="xsd:decimal"/>
                <xsd:attribute name="parution" type="xsd:date"/>
</xsd:complexType>

</xsd:schema>
```

Au sein de l'exemple précédent, l'attribut **targetNamespace**=*"http://www.yyy.com/my_namespace"* étant placé sur la balise englobante ***<xsd:schema>*** , toutes les parties imbriquées (*éléments , types complexes, ...*) qui sont habituellement de niveau global sont içi implicitement liées au namespace précisé par l'attribut ***targetNamespace*** hérité .

==> il n'est donc pas nécessaire d'écrire <complexType name="bi:xxxType"> .... d'autant plus que cette écriture n'est pas autorisée au niveau d'une définition de type.

Par contre, au sein d'une référence vers un type ou un élément, il faut préciser (via un préfixe local) le namespace de l'entité référencée *(ex: **ref**="**bi:**titre").*

NB: Le préfixe local (*ex*: **bi**) utilisé pour encoder le schéma n'a aucune importance, il peut d'ailleurs être différent de celui qui sera utilisé pour encoder le document XML.

Les attributs **elementFormDefault**="***qualified***" et **attributeFormDefault**="***unqualified***" permettent respectivement de préciser que l'on souhaite trouver au sein des documents qui seront validés:

- des balises préfixées (avec des noms qualifiés)
- des attributs non préfixés (pas qualifiés)

Le fichier xml suivant est conforme au schéma précédent :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<b:bibliographie  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.yyy.com/my_namespace    schema_biblio.xsd"
          xmlns:b="http://www.yyy.com/my_namespace" >

        <b:titre> Bibliographie sur certains langages de programmation </b:titre>
        <b:sujet>
                <b:titre>C++</b:titre>
                <b:livre prix="30.5">
                        <b:titre>Mieux développer en C++</b:titre>
                        <b:auteur>auteur francais</b:auteur>
                        <b:editeur>Inter-Editions</b:editeur>
                </b:livre>
                <b:livre prix="50">
                        <b:titre>Numerical recipes in c++</b:titre>
                        <b:auteur>auteurs américains</b:auteur>
                        <b:editeur>????</b:editeur>
                </b:livre>
        </b:sujet>
         ....
</b:bibliographie>
```

Le préfixe local choisi ***b*** (*et non pas bi*) n'a aucune importance. Ce qui compte c'est qu'il soit associé au bon namespace : celui précisé au niveau du schéma via ***targetNamespace*** .

4.2. 1.1.1 Notion de groupe

Un <xsd:**group**> est une ***construction*** constituée de plusieurs éléments (ou attributs) qui servira ultérieurement à définir une ***partie*** d'un (ou plusieurs) élément(s) complexe(s).

Contrairement à un Type complexe , **un groupe ne correspond pas à une basile XML .**

L'intérêt d'un regroupement est la réutilisation multiple d'une partie commune que l'on cherche à factoriser sans vraiment imposer une délimitation via un niveau de balise supplémentaire.

exemple1 (groupe d'éléments):

```
<xsd:complexType   ...>
  <xsd:sequence>
     <xsd:choice>
        <xsd:group   ref="Adresses_livraison_et_facturation"/>
        <xsd:element   name="adresse"   type="AdresseType"/>
     </xsd:choice>
     <xsd:element   name="xxx"  type="..." />
  </xsd:sequence>
</xsd:complexType>
...
<xsd:group name="Adresses_livraison_et_facturation">
 <xsd:sequence>
     <xsd:element name="adresse_livraison" type="AdresseType"/>
     <xsd:element name="adresse_facturation" type="AdresseType"/>
 </xsd:sequence>
</xsd:group>
```

exemple 2 (groupe d'attributs):

```
<xsd:complexType   ...>
        <xsd:sequence>
        ...
        </xsd:sequence>
<!-- attributeGroup replaces individual declarations -->
        <xsd:attributeGroup   ref="Prix_et_Qte"/>
</xsd:complexType>
...

<xsd:attributeGroup   name="Prix_et_Qte">
        <xsd:attribute name="prix" type="xsd:decimal" use="required"/>
        <xsd:attribute name="qte" type="xsd:integer"/>
</xsd:attributeGroup>
```

4.3. 1.1.1 Inclusion ou importation d'un schéma dans un autre

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
        <xsd:documentation>Schema biblio (englobant) </xsd:documentation>
 </xsd:annotation>
<!-- inclusion du "sous-schéma" décrivant la structure d'un livre -->
<!- - à placer absolument dès le début (avant element et complexType) -->
<xsd:include     schemaLocation="livre.xsd" />
...
```

NB: à la place de <xsd:include ..../> on utilise assez souvent

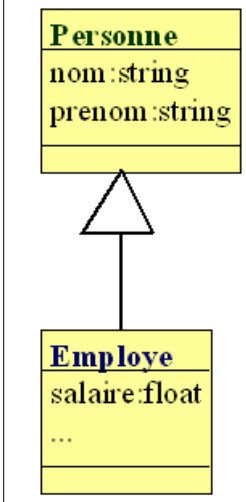
```
<xsd:import   namespace="nomAutreNameSpace"   schemaLocation="sousSchema.xsd" />
```

car <xsd:import /> permet d'importer un sous-schéma qui peut être d'un autre namespace.

4.4. Structures de données avancées

4.4.a. Héritage entre types complexes (extensions)

```
<xsd:complexType name="EmployeType">
   <xsd:complexContent>
      <xsd:extension base="PersonneType">
         <xsd:all>
               <xsd:element  name="salaire"  type="xsd:decimal"/>
               <xsd:element  name="fonction" type="xsd:string"/>
          </xsd:all>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```



<element name="*personne*" type="*EmployeType*" substitutionGroup="*PersonneType*" />

4.4.b. Références (liaisons)

On peut reproduire au sein d'un document XML des liaisons entre éléments qui sont très proches des jointures du modèle relationnel (FK ==>PK).

Afin d'atteindre cet objectif, il faudra:

- Préciser des informations de type "***<xsd:keyref>***" et "***<xsd:key>***" au sein du schéma de façon à encoder les ***liaisons*** "***référence --> clef***" et pour que **la phase de validation** puisse **vérifier les contraintes d'intégrités.**
- Qu'une application tienne compte de ces liaisons (via par exemple des requêtes XPath / XQuery)

Exemple:

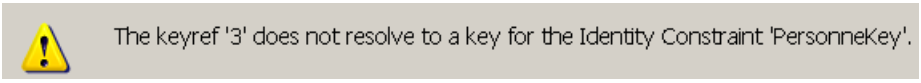
schema_id_ref.xsd

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

<xsd:element name="biblio" type="biblioType">
        <!-- attribut id définit en tant que clef primaire sur personne -->
          <xsd:key    name="PersonneKey">
                    <xsd:selector      xpath="personne" />
                    <xsd:field       xpath="@id" />
          </xsd:key>

          <!-- contrainte d'intégrité : livre/@auteur vers personne/@id -->
          <xsd:keyref   name="LivreToPersonne"   refer="PersonneKey">
                    <xsd:selector   xpath="livre" />
                    <xsd:field   xpath="@auteur" />
          </xsd:keyref>
</xsd:element>

<xsd:complexType name="biblioType">
    <xsd:sequence>
          <xsd:element name="livre" type="livreType" minOccurs="0" maxOccurs="unbounded" />
          <xsd:element name="personne" type="personneType" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="livreType">
          <xsd:sequence>
                    <xsd:element name="titre" type="xsd:string"/>
                    <xsd:element name="editeur" type="xsd:string" />
          </xsd:sequence>
          <xsd:attribute name="auteur" type="xsd:string"  />
</xsd:complexType>

<xsd:complexType name="personneType">
          <xsd:sequence>
                    <xsd:element name="nom" type="xsd:string"/>
                    <xsd:element name="prenom" type="xsd:string" />
          </xsd:sequence>
          <xsd:attribute name="id" type="xsd:string" />
</xsd:complexType>
</xsd:schema>
```

livre_auteur.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="schema_id_ref.xsd">

        <livre auteur="1">
                <titre>Livre A</titre>
                <editeur>Eyrolles</editeur>
        </livre>

         <livre auteur="1">
                <titre>Livre B</titre>
                <editeur>Masson</editeur>
          </livre>

         <livre auteur="2">
                <titre>Livre C</titre>
                <editeur>InterEditions</editeur>
          </livre>

        <personne id="1">
                <nom>durant</nom>
                <prenom>luc</prenom>
        </personne>

        <personne id="2">
                <nom>dupond</nom>
                <prenom>julie</prenom>
        </personne>

</biblio>
```

Si par erreur , ***auteur="3"*** ==>



Remarque:

<xsd:key ...> permet de préciser un champ clef dont la valeur doit être unique pour un type donné (2 personnes ne peuvent pas avoir le même id).

<xsd:ID> (issu des ***DTD***) permet de préciser un identificateur unique au sein d'un document (fichier) complet (une personne ne peut pas avoir le même ID qu'un éditeur) .

XI - JAXB2

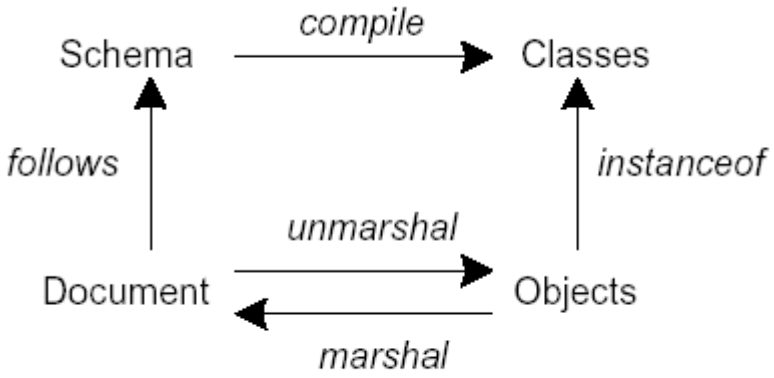
1. Principes - JAXB

JAXB (**J**ava **A**pi for **X**ml **B**inding) est une api assez récente permettant d'établir une **association (correspondance)** entre :

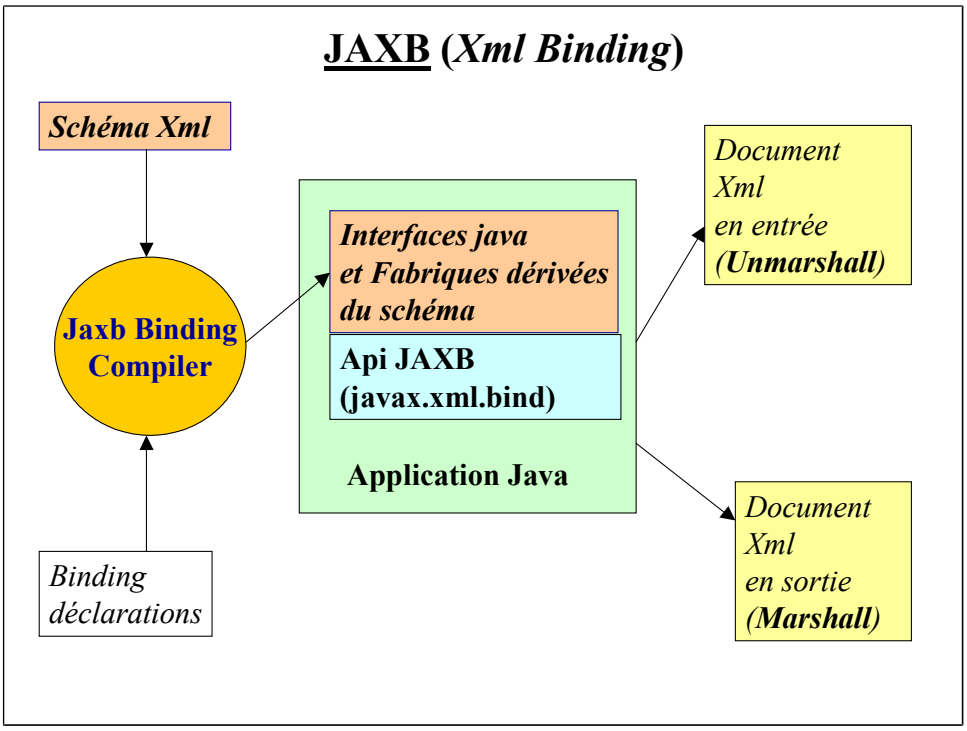
un **document XML** (dont le *type* est décrit par un *schéma XML*)

et

un **ensemble de classes java** (formant ensemble une structure arborescente) .



Une fois que les classes java ont été générées conformément au schéma , on peut alors effectuer très rapidement une "**sérialisation XML**" entre des instances des classes java (comportant certaines valeurs) et un fichier XML (instance du schéma) .



2. Versions 1 et 2 (JAXB)

La première version de JAXB peut fonctionner avec le jdk 1.4 car elle n'utilise pas d'annotations Java5 au niveau de son paramétrage.

La nouvelle version JAXB2 est quant à elle spécifique à Java 5 (et 6) .

La suite de ce chapitre sera illustré avec la version JAXB2.

3. Mise en oeuvre (JAXB2)

Fixer quelques variables d'environnement:

```
set JAVA_HOME=C:\Prog\JAVA_5\jdk1.5.0_09
set JAXB_HOME=C:\Prog\Sun\jwsdp-2.0\jaxb
set JAXB_LIBS=%JAXB_HOME%\lib


set CLASSPATH=%CLASSPATH%;%JAXB_LIBS%\jaxb-api.jar;
              %JAXB_LIBS%\jaxb-xjc.jar;%JAXB_LIBS%\jaxb-impl.jar;
```

Invocation du compilateur de binding xml (générateur de code java en fonction d'un schéma xml):

```
%JAXB_HOME%\bin\xjc.bat -p pack1 po.xsd

  Usage: xjc [-options ...] <schema>
  Options:
    -nv        :  do not perform strict validation of the input schema(s)
    -d <dir>   :  generated files will go into this directory
    -p <pkg>   :  specifies the target package
    -readOnly  :  generated files will be in read-only mode
    -help      :  display this help message
```

NB: ***xjc.bat*** lance en interne la commande java suivante:

```
java -jar %JAXB_HOME%/lib/jaxb-xjc.jar
```

4. Exemple de binding (JAXB2)

Schéma XML (biblio.xsd):

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="bibliographie" type="BibliographieType"/>
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="BibliographieType">
  <xsd:sequence>
    <xsd:element name="titre" type="xsd:string"/>
    <xsd:element name="sujet" type="xsd:string"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="livre" type="Livre"  minOccurs="1"
maxOccurs="unbounded" />
  </xsd:sequence>
```

```
  <xsd:attribute name="date" type="xsd:date"/>
</xsd:complexType>

<xsd:complexType name="Livre">
        <xsd:sequence>
         <xsd:element name="titre" type="xsd:string"/>
         <xsd:element name="auteur" type="xsd:string"/>
         <xsd:element name="editeur" type="xsd:string"/>
        </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Document Xml (biblio.xml = instance possible du schéma)

```
<?xml version="1.0"?>
<bibliographie date="01/01/2002">
 <titre> Bibliographie sur Programmation Xml en java </titre>
 <sujet> Livres evoquant la programmation Xml en langage Java </sujet>
 <livre>
     <titre>XML, langage et applications</titre>
     <auteur>Alain Michard (INRIA)</auteur>
     <editeur>Eyrolles</editeur>
 </livre>
 <livre>
     <titre>Construire une application XML ...</titre>
     <auteur>J.C. Bernadac , F.Knab </auteur>
     <editeur>Eyrolles</editeur>
 </livre>
</bibliographie>
```

%JAXB_HOME%\bin\xjc.bat -p mybiblio biblio.xsd

==> Interfaces java et fabriques générées:

```
parsing a schema...
compiling a schema...
mybiblio\BibliographieType.java
mybiblio\Livre.java
mybiblio\ObjectFactory.java

....
import javax.xml.bind.annotation.AccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;
...
```

BibliographieType.java

```
@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "BibliographieType", propOrder = {
   "titre",
   "sujet",
   "comment",
   "livre"
})
public class BibliographieType {

   protected String titre;
   protected String sujet;
   protected String comment;
```

```
    protected List<Livre> livre;
    @XmlAttribute
    protected XMLGregorianCalendar date;
public String getTitre() {        return titre;
    }

    public void setTitre(String value) {        this.titre = value;
    }

    public String getSujet() {        return sujet;
    }

    public void setSujet(String value) {        this.sujet = value;
    }

    public String getComment() {        return comment;
    }

    public void setComment(String value) {        this.comment = value;
    }

    public List<Livre> getLivre() {
        if (livre == null) {
            livre = new ArrayList<Livre>();
        }
        return this.livre;
    }
    public XMLGregorianCalendar getDate() {        return date;
    }

    public void setDate(XMLGregorianCalendar value) {        this.date = value;
    }

}
```

Livre.java

```
@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "Livre", propOrder = {
    "titre",
    "auteur",
    "editeur"
})
public class Livre {

    protected String titre;
    protected String auteur;
    protected String editeur;

   // + public get/set
}
```

ObjectFactory.java

```
package mybiblio;

import javax.xml.bind.JAXBElement;
import javax.xml.bind.annotation.XmlElementDecl;
import javax.xml.bind.annotation.XmlRegistry;
import javax.xml.namespace.QName;
import mybiblio.BibliographieType;
import mybiblio.Livre;
import mybiblio.ObjectFactory;

@XmlRegistry
public class ObjectFactory {

   private final static QName _Comment_QNAME = new QName("", "comment");
   private final static QName _Bibliographie_QNAME =
                           new QName("", "bibliographie");


   public ObjectFactory() {
   }

   public Livre createLivre() {
      return new Livre();
   }

   public BibliographieType createBibliographieType() {
      return new BibliographieType();
   }

   @XmlElementDecl(namespace = "", name = "comment")
   public JAXBElement<String> createComment(String value) {
      return new JAXBElement<String>(_Comment_QNAME,
                              String.class, null, value);
   }

   @XmlElementDecl(namespace = "", name = "bibliographie")
   public JAXBElement<BibliographieType>
            createBibliographie(BibliographieType value) {
      return new JAXBElement<BibliographieType>(_Bibliographie_QNAME,
                                    BibliographieType.class,
                                    null, value);
   }

}
```

5. Structure de l'api JAXB

Le principal package de **JAXB** est **javax.xml.bind**

La classe abstraite **JAXBContext** est le point d'entrée de l'API.

```
JAXBContext   jctx =
   JAXBContext.newInstance( "com.mycompany.pack1:com.mycompany.pack2" );
```

Le paramètre d'entrée de la méthode statique **newInstance** correspond à la liste de(s) package(s) de classes et d'interfaces java dérivées du schéma xml.

Une fois instancié, le contexte JAXB peut servir à créer les éléments suivants:

```
Unmarshaller um = jctx.createUnmarshaller(); // pour remonter des objets en mémoire
                                             // suite à la lecture d'un flux xml
Marshaller m = jctx.createMarshaller(); // pour générer un flux xml (fichier)
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
```

Dé-sérialisation d'un flux xml:

```
JAXBElement<XxxType> xElement =
            (JAXBElement<XxxType>)  um.unmarshal( new File( "xxx.xml" ) );
XxxType xxxObj = (XxxType) xElement.getValue();
```

Sérialisation vers un flux xml:

```
JAXBElement<YyyType> yyyElement = objectFactory.createYyyy(yyyObj);
m.marshal( yyyElement, myOutputStream );
```

Création d'une nouvelle instance via la fabrique générée au sein du package (fruit de la compilation du schéma Xml):

```
com.mycompany.pack1.ObjectFactory objectFactory
                        = new com.mycompany.pack1.ObjectFactory();
com.mycompany.pack1.Xxx po =  objectFactory.createXxx();
```

6. Exemple de code (JAXB2)

```
package myprog;

import javax.xml.bind.*; // JAXB
import javax.xml.datatype.*;
import mybiblio.*; // Result of XML Binding
import java.io.*;



public class MyJaxbApp {
```

```
   private static XMLGregorianCalendar getDateAsXmlGreogorianCalendar() {
            try {
                return DatatypeFactory.newInstance()
                        .newXMLGregorianCalendar(new GregorianCalendar());
            } catch (DatatypeConfigurationException e) {
                throw new Error(e);
            }
        }

      public static void main(String[] args) {

      try {
      JAXBElement<BibliographieType> biblioElement = null;
      JAXBContext jctx = JAXBContext.newInstance("mybiblio");
      Unmarshaller um = jctx.createUnmarshaller();
      Marshaller m = jctx.createMarshaller();
      m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);


      biblioElement=(JAXBElement<BibliographieType>)
                 um.unmarshal( new File( "biblio.xml" ) );
      BibliographieType biblioObj = (BibliographieType)biblioElement.getValue();
      // ....
      // create an element for marshalling
      biblioElement = objectFactory.createBibliographie(biblioObj);
      m.marshal( biblioElement, System.out );

      ObjectFactory objectFactory = new ObjectFactory();
      BibliographieType biblioObj3 = objectFactory.createBibliographieType();
      biblioObj3.setDate( getDateAsXmlGreogorianCalendar());
      biblioObj3.setTitre("titre biblio");
      biblioObj3.setSujet("sujet qui va bien");
      biblioObj3.setComment("mon commentaire qui ne sert a rien");

      Livre l1 = objectFactory.createLivre();
      l1.setAuteur("Auteur qui ecrit bien");
      l1.setEditeur("Eyrolles");
      l1.setTitre("Titre du livre");

      Livre l2 = objectFactory.createLivre();
      l2.setAuteur("Auteur fou");
      l2.setEditeur("Masson");
      l2.setTitre("Titre2 du livre2");

      biblioObj3.getLivre().add(l1);
      biblioObj3.getLivre().add(l2);

      /*  Validator ==> in JAXB1 but no more in JAXB2
            Validator v= jctx.createValidator();
            v.validate(biblioObj3);
            */

      // create an element for marshalling
       biblioElement = objectFactory.createBibliographie(biblioObj3);
      // marshalling
      m.marshal( biblioElement, new FileOutputStream("biblio3.xml") );

      }
      catch(Exception ex) { ex.printStackTrace(); }

      }
}
```