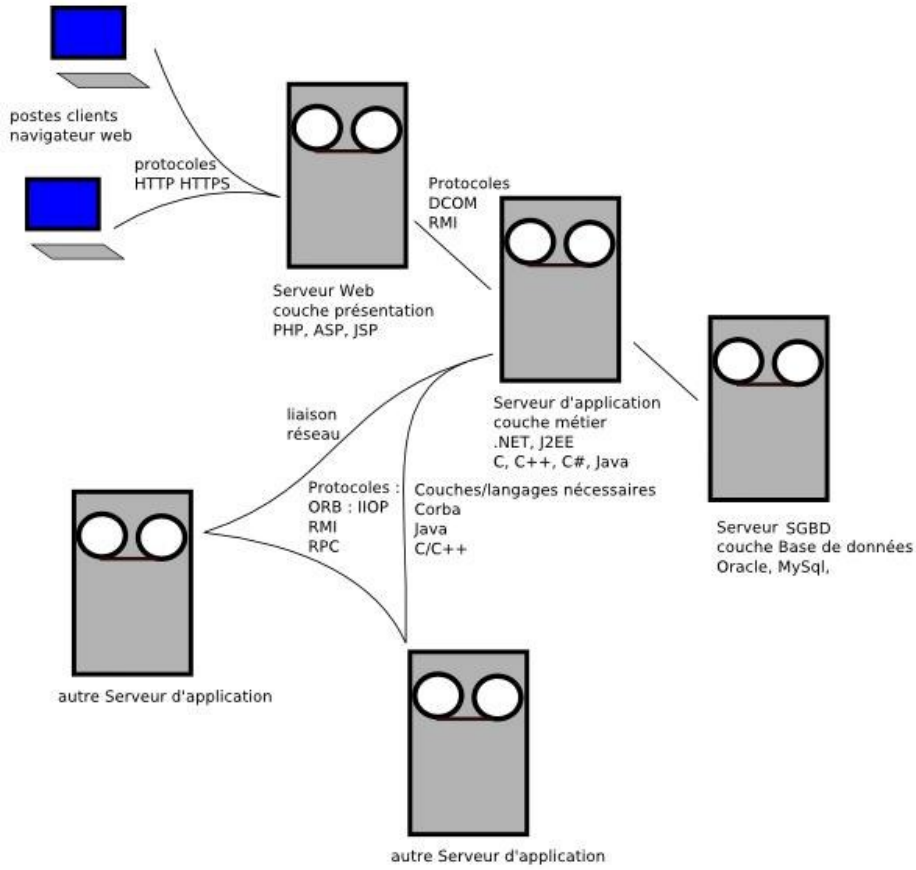
Architecture des Systèmes Informatiques

Didier FERMENT - Université de Picardie
http://www.u-picardie.fr/ferment/java/sommaire3.html

---

les Web Services : introduction

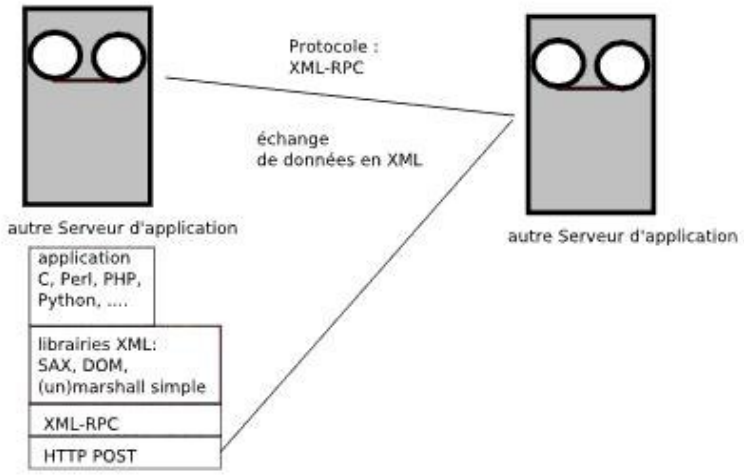
Architecture 3-tiers ... n-tiers



- L'architecture 3 tiers comprend une couche de présentation, une couche métier et une couche d'accès aux données. Chaque couche peut être implémentée sur un ou des serveurs indépendants, ou se trouver sur un même serveur.
    - Postes Clients
      Les postes "clients" légers n'ont besoin que d'un navigateur, donc permet de mixer des postes PC sous WINDOWS ou sous LINUX, des Macintosh, …
    - La couche Présentation
      est articulée autour d'un serveur Web qui génère l'interface graphique de l'application. Les grands standards en la matière sont JSP, PHP ou ASP.
    - La couche Applicatif ou Métier
        - contient les traitements représentant les règles métier . Par exemple, dans le cadre d'une application bancaire : créer un compte, créditer un compte, effectuer un virement, rechercher un client, calculer un amortissement, ...
        - Cette couche est écrite dans des langages et systèmes divers : IBM AS 400 avec COBOL pour d'anciennes applications qui tournent encore, jusqu'aux Framework à base d'objet comme J2EE de SUN et .NET de Microsoft, pour les plus modernes.
    - La couche d'accès aux données
      permet de masquer entièrement l'opération de création et de manipulation de tables

(SGBDR) ou autre types de stockages spécifiques (Fichiers XML, ...), et fonctionner avec n'importe quelles bases de données relationnelles du marché au choix : ORACLE, SYBASE, SQL Server, DB2...

- Quid de la communication entre serveur ?
  - RPC (Remote Procedure Call) : simple, vieux voire simplet
    - protocole d'appels de procédures sur un ordinateur distant
    - la description des données échangées se fait en IDL (Interface Definition Language) proche du langage C
    - de nombreuses versions non compatible ...
  - Des protocoles sophistiqués :
    - Java RMI : multiplate-forme, un seul langage, licence semi-ouverte Oracle
    - CORBA/IIOP : multiplate-forme, multi-langage, cher, version open source
    - DCOM : plateforme Microsoft, multi-langage, propriétaire
  - Puis XML-RPC, ancêtre de SOAP
    - XML (une petite partie !) comme langage d'échange
    - une conversion de types simples
      - les types de bases : int, float, ... date
      - et des types complexes construits avec table ou structure
    - un protocole de type RPC
      - une couche de transport HTTP (Post)

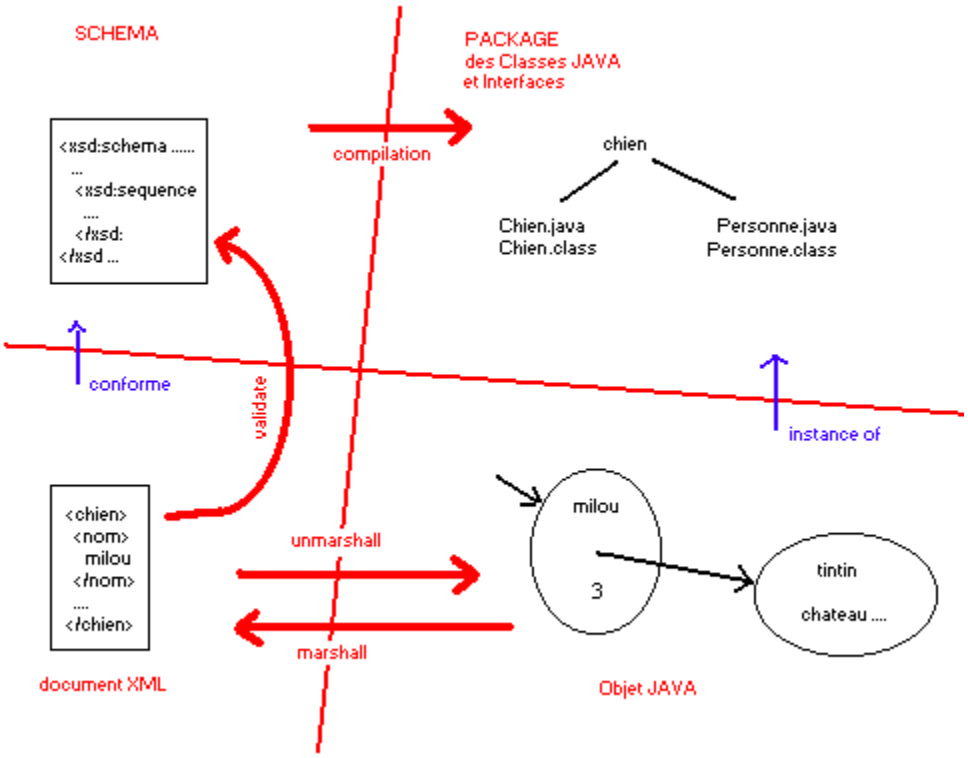


  - Enfin :
    - SOAP :
      - protocole « surcouche »
      - multiplate-forme, multi-langage
      - format universel d'échange : XML
      - protocole(s) de transport simple : HTTP, SMTP, ...
      - non propriétaire : W3C
      - associé à WSDL (doc à distance) et UDDI (annuaire de services) pour former l'architecture SOA (archi orienté service)
      - sophistiqué mais lourd
    - REST :
      - style d'architecture donc non propriétaire
      - basé sur HTTP et l'identification unifiée des ressources en URI
      - tout format de données : souvent XML et JSON
      - ne nécessite aucun protocole ou outils
      - très simple (trop?)
    - REST prend le dessus sur SOAP mais utilise principalement le format de données XML et le concept de WSDL (doc à distance)

XML Schema et Marshalling

Marshalling en XML

Pour pouvoir utiliser le XML comme format d'échange entre les serveurs, il faut un mécanisme de transformation entre les types de données de langages d'applications (C, Java, Python, PHP, Perl, ...) et XML :



- Marshalling et unmarshalling
  - agrégation en XML et l'inverse
  - en Java : basé sur API JAXB (java XML Binding)
  - La description des données en XML se fait grâce à Schéma :
    - XSD un langage XML de description de données XML
    - grammaire plus puissante que la DTD

- Source de Chien.java

```
package chien;
public class Chien {
  public String nom;
  public Personne maitre;
  public int nombrePuces;
  public Chien() { }
  public Chien(String n, Personne p, int i) {
    nom = n; maitre = p; nombrePuces = i ;
  }
}
```

Source de Personne.java

```
package chien;
public class Personne {
   public String nom;
   public String adresse;
   public Personne() { }
   public Personne(String n, String a) {
      nom = n; adresse = a ;
   }
}
```

- Marshall et unmarshall :
  - A l'objet instance de Chien :

        - de nom milou,
        - de maître la personne :
            - de nom tintin
            - et d'adresse château de moulinsart
        - et de nombre de puces 3
    - correspond le document xml :

```
<tns:chien xmlns=tns:"urn:MeuteService">
  <tns:nom>milou</tns:nom>
  <tns:maitre>
    <tns:nom>Tintin</tns:nom>
    <tns:adresse>chateau de Moulinsart</tns:adresse>
  </tns:maitre>
  <tns:nombrePuces>3</tns:nombrePuces>
</tns:chien>
```

- son schéma est :

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns=tns:"urn:MeuteService"
        targetNamespace="urn:MeuteService">
  <complexType name="Personne">
    <sequence>
      <element name="nom" nillable="true" type="xsd:string"/>
      <element name="adresse" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
  <element name="Chien">
    <complexType">
      <sequence>
        <element name="nom" nillable="true" type="xsd:string"/>
        <element name="maitre" nillable="true" type="tns:Personne"/>
        <element name="nombrePuces" type="xsd:int"/>
      </sequence>
    </complexType>
  <element>
</schema>
```

XML Schema

- langage de définition de document XML plus puissant que la DTD :
    - définit des éléments lexicaux, des structures syntaxiques et des types.
    - permet d'exprimer la "grammaire" :
        - des langages à structures XML (balises, emboîtement, ID, IDRef, ..)
        - des données des langages de programmation (type complexes, types dérivés, héritage, ...)
        - des données de tables de bases de données
- Pour conformer un document XML à un schéma :

```
<racine
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://.../monSchema.xsd">
 .......... le document XML ..............
</racine>
```

- Comparaison aux DTDs :
    - une DTD peut s'exprimer en schema mais l'inverse est plus rare

- Schema est écrit en XML et non en EBNF comme la DTD
- Schéma contient :
  - des nombreux types simples
  - un mécanisme de type complexe : sequence, choice, ...
  - un mécanisme d'extension de type
  - une forme de spécialisation
- norme : http://www.w3.org/XML/Schema
- un exemple de schema : celui de SOAP

Les constructions de Schema

- Fichier schema

  MonSchema.xsd :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://.../monSchema.xsd">
     .... déclarations ....
</xsd:schema>
```

  - spécification du **Namespace**

- Déclaration d'élément

```
<xsd:element name="livre">
    ... description de l'élément  .....
</xsd:element>
```

  - le **type** est içi **anonyme**

```
<xsd:element  name="livre" type="TypeLivre"/>
```

  - déclaration à partir d'un type simple ou dérivé.
  - le type est içi **nommé**

- Types complexes

  - permet de définir un nouveau type

```
<xsd:complexType name=TypeLivre">
   <xsd:sequence>
     <xsd:element name="titre"   type="xsd:string"/>
     <xsd:element name="auteur" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="isbn" type="IsbnType"/>
<xsd:complexType/>
```

  - plusieurs variantes : **sequence choice all** (peuvent apparaître ou non et dans n'importe quel ordre)
  - remarquons la **déclaration d'un attribut** dans l'exemple précédent

```
<xsd:complexType name=TypeLivre">
   <xsd:sequence>
```

```
      <xsd:element name="titre"   type="xsd:string"/>
      <xsd:element name="auteur" type="xsd:string"
                   minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

- des attributs précise les définitions : **default, fixed, minOccurs**, ...
- ici, le nombre d'occurrence de l'élément

- Type simple

  - boolean, string,
  - integer, int, positiveInteger, ... , long, unsignedInt, ...
  - decimal, float, double,
  - time,date, duration, ...
  - ID, IDREF, ENTITY, NOTATION, ...

- Dérivation de type par restriction

```
<xsd:simpleType name="centPremierEntier">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="99"/>
  </xsd:restriction>
</xsd:simpleType>
```

  - est obtenue grace à 2 **facettes** : minInclusive et maxInclusive
  - Ceci est une **définition de type simple**
- la restriction est possible sur une définition de type complex

```
<xsd:simpleType name="TypeNumeroVoiture">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{4} [A-Z]{2} \d{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

  - obtenu grâce à la **facette** pattern.

- Dérivation de type par extension :

```
<xsd:complexType name="TypeELivre">
   <xsd:complexContent>
     <xsd:extension base="TypeLivre"/>
       <xsd:sequence>
         <xsd:element name="url" type="xsd:anyURI"/>
      </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

- Limitation :

  - il aurait été possible d'empêcher la dérivation précédente ainsi :

```
<xsd:complexType name=TypeLivre" final="extension">
```

```
                    ..........
<xsd:complexType/>
```

- Type Abstract :
    - a l'inverse, il est possible de déclarer un type abstract

```
<xsd:complexType name="TypeDocument" abstract="true"/>

<xsd:complexType name="TypeLivre">
  <xsd:complexContent>
   <xsd:extension base="target:TypeDocument"/>
  </xsd:complexContent>
...
```

- Importer/Inclure d'autres schemas :

```
<xsd:include schemaLocation="http://.../autre_schema.xsd"/>
```

- et le reste :
    - unicité, key, ...
    - list, enumeration, union,....
    - namespace, ...
    - any, ...
    - substitutiongroup, ..

SOAP

Simple Object Access Protocol : protocole de RPC utilisant le format de données XML

un premier dialogue SOAP

- la requête :
  URI: api.google.com/search/beta2
  SOAPACtion: urn:GoogleSearchAction

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:doSpellingSuggestion xmlns:ns1="urn:GoogleSearch"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<key>cmYE.....Xqlf</key>
<phrase>gemany</phrase>
</ns1:doSpellingSuggestion>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

- la réponse :

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:string">germany</return>
</ns1:doSpellingSuggestionResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

un dialogue SOAP sur transport HTTP

- la requête :

```
POST /soap/servlet/rpcrouter HTTP/1.1
Host: services.xmethods.net
Connection: Keep-Alive, TE
TE: trailers, deflate, gzip, compress
User-Agent: RPT-HTTPClient/0.3-3
SOAPAction: ""
Accept-Encoding: deflate, gzip, x-gzip, compress, x-compress
Content-type: text/xml; charset=utf-8
Content-length: 435

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<tns:getTemp xmlns:tns="urn:xmethods-Temperature"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<zipcode>55406</zipcode>
</tns:getTemp>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

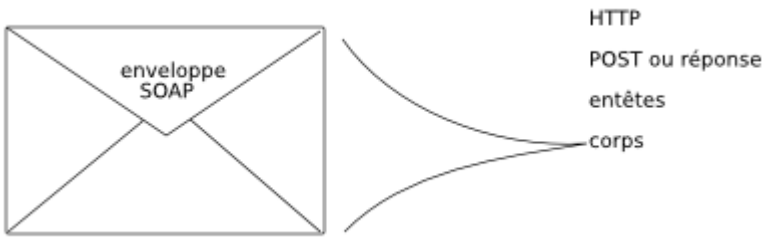
- la réponse Soap dans/via HTTP :

```
HTTP/1.1 200 OK
Date: Wed, 04 Dec 2002 12:07:33 GMT
Server: Enhydra-MultiServer/3.1.1b1
Status: 200
Content-Type: text/xml; charset=utf-8
Servlet-Engine:  Enhydra Applic... java.vendor=Sun Microsystems Inc.)
Content-Length: 465
Set-Cookie: JSESSIONID=GMTbfqqlWnQzSgGhQI3liOiS;Path=/soap
X-Cache: MISS from www.xmethods.net
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:float">19.0</return>
</ns1:getTempResponse>
</SOAP-ENV:Body>
```

</SOAP-ENV:Envelope>

Principe de l'enveloppe :



- protocole de transmission de messages : unidirectionnel ou en dialogue requête-réponse du type RPC (Remote Procedure call)
- exemple : pour un cabinet de gestion de portefeuille, obtenir auprès de la Bourse le prix de telle action.
  l'application du cabinet appellera la méthode double getPrice(typeAction action) du serveur de la Bourse
- Client --->SoapRequest--->Serveur--->SoapResponse--->Client

le Web service :

- c'est un ensemble de méthodes qui sont appelées via TCP/IP à distance
- l'information est transportée dans une enveloppe SOAP
- il fournit des données XML à une autre ordinateur en fonction des paramètres d'appel, contrairement aux serveurs de pages dynamiques qui fournissent une information formatée pour un utilisateur humain

Norme SOAP W3C:

- http://www.w3.org/TR/SOAP/

- **Structure du message SOAP**
  - une enveloppe
    - racine du document XML
    - spécifiant le Namespace SOAP-ENV http://schemas.xmlsoap.org/soap/envelope/
    - souvent la sérialisation attribut encodingStyle est celle de Soap http://schemas.xmlsoap.org/soap/encoding/
  - une en-tête optionnelle : Header
  - un corps Body
- **Header optionnel**
  - indications comme : transactions, session, ...
  - l'attribut mustUnderstand :
    - si égal à 1, indique que le header est à prendre en compte au niveau de l'application qui traite le contenu Soap
    - sinon absent ou = à 0
- **Body**
  - contient les données pour l'application réceptrice
  - utilise l'encodage des types selon l'encodingStyle et le Schema xsd "http://www.w3.org/1999/XMLSchema
  - peut contenir 1 ou 0 élément FAULT
    - spécifie les codes d'erreurs, voire plus !
    - en particulier : en réponse à une requête mal formulée
    - exemple :
      <SOAP-ENV:Body>
      <tns:getTemp xmlns:tns="urn:xmethods-Temperature"

SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<zipcode>55406</zip>
</tns:getTemp>
</SOAP-ENV:Body>
--- PRODUIT LA REPONSE :-----
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:Client</faultcode>
<faultstring>parsing error: org.xml.sax.SAXParseException: The element type
"zipcode" must be terminated by the matching end-tag
"</zipcode>".</faultstring>
<faultactor>/soap/servlet/rpcrouter</faultactor>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

- **EncodingStyle**
  - le (un)marshalling des types de données :
    - types simples :
      int, float, string, ...
    - types composés :
      - struct exemple :
        <paireDentier>
        <premier>2</premier>
        <second>3</second>
        </paireDentier>
      - array exemple :
        <paireDentier SOAP-ENC:arrayType="xsd:int[2]">
        <number>2</number>
        <number>3</number>
        </paireDentier>
- **RPC encapsulé dans Soap**
  - l'appel de la méthode est un struct
    - son nom peut être celui de la méthode
    - chaque "accessor" du struct est un paramètre (in ou in/out) apparaissant dans le même ordre
      que la signature de la méthode.
      - le nom de l'accessor correspond au nom du paramètre
      - correspondance des types
  - le résultat de l'appel de la méthode est un struct
    - son nom peut être celui de la méthode suivi de "Response"
    - le premier accessor est la valeur retournée (de nom quelconque)
    - les autres accessors"du struct sont les paramètres "in/out" de la méthodes :
      - ils apparaissant dans le même ordre que la signature de la méthode.
      - le nom de l'accessor correspond au nom du paramètre
    - correspondance des types des accessors
- **SOAP encapsulé dans HTTP**
  - rappels sur HTTP
    - protocole requête-réponse de données caractère
      - 1ère ligne :

        - requête : Action URI versionProtocol
            - action : POST GET ou HEAD
            - versionProtocol HTTP/1.0 ou HTTP/1.1
        - réponse: versionProtocol Etat Phrase
            - état : 200, 402 ou ...
            - phrase : OK, Unauthorized, ...
    - suivies de plusieurs lignes d'en-tête contenant chacune une clef : sa valeur
    - suivi d'une ligne vide
    - et enfin le contenu textuel
- LE protocole de l'Internet
    - simple, peu gourmand
    - disponible sur toutes les plates-formes
    - protocole sans connexion
    - est rarement filtré par les firewalls
- sécurité : HTTPS

- pour SOAP dans HTTP, obligatoire :
    - le type MIME doit être text/xml
    - pour le requêtes doivent être présent :
        - l'URI de l'intent
        - SOAPAction pour le filtrage des firewalls entre_autres
          si valeur "", alors la SOAPAction est par défaut l'URI
- exemple :

```
POST /soap/servlet/rpcrouter  HTTP/1.1
Host: services.xmethods.net
Connection: Keep-Alive, TE
TE: trailers, deflate, gzip, compress
User-Agent: RPT-HTTPClient/0.3-3
SOAPAction: ""
Accept-Encoding: deflate, gzip, x-gzip, compress, x-compress
Content-type: text/xml; charset=utf-8
Content-length: 435
```

WSDL

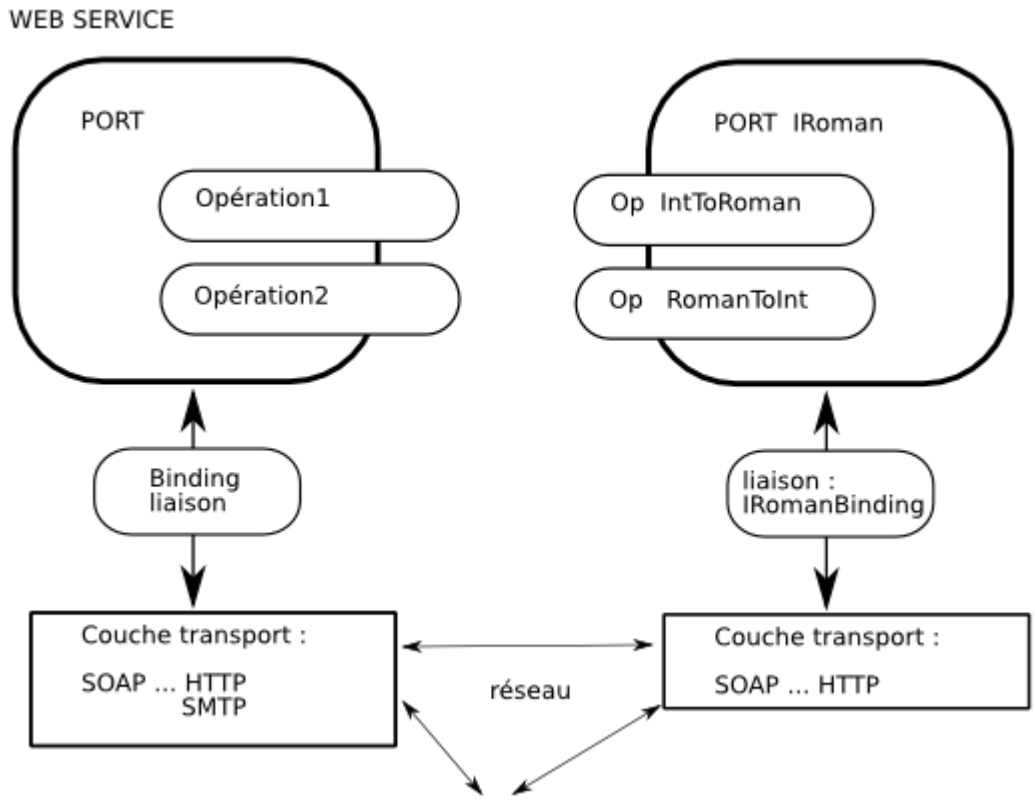
Web Services Description Language

- décrit et localise les webs services
  documente les méthodes à distance
  écrit en XML
  standard W3C avec norme 1.1 et la 2.0

Structure d"un Fichier WSDL

- le document Wsdl décrivant un web service :

```
<definitions>
   <types>
      définition des types
      les types de données des valeurs transmises
      exprimés en XSD la syntaxe XML Schema
      ........
   </types>
   <message>
      définition des messages
      noms et types des paramètres in, ou et in-out
      ....
   </message>
   <portType>
      définition des ports/portTypes
      noms des opérations/méthodes et leurs messages
      .......
   </portType>
   <binding>
      définition des liaisons (binding)
      type de transport pour chaque portType
      ....
   </binding>
   <service>
      définition des ports
      <port ..../> associe un "endpoint" et une liaison
      <port ..../>
      ......
   </service>
</definitions>
```



- Dans un service RPC, portType-message-types définit la signature d'une méthode distante, alors

que le binding définit l'implémentation pour le transport de l'appel.

```
<message name="IntToRoman0Request">
      <part name="Int" type="xs:int"/>
   </message>
   <message name="IntToRoman0Response">
      <part name="return" type="xs:string"/>
   </message>
   <message name="RomanToInt1Request">
      <part name="Rom" type="xs:string"/>
   </message>
   <message name="RomanToInt1Response">
      <part name="return" type="xs:int"/>
   </message>
   <portType name="IRoman">
      <operation name="IntToRoman">
         <input message="tns:IntToRoman0Request"/>
         <output message="tns:IntToRoman0Response"/>
      </operation>
      <operation name="RomanToInt">
         <input message="tns:RomanToInt1Request"/>
         <output message="tns:RomanToInt1Response"/>
      </operation>
   </portType>
```

signifie :
string IntToRoman(int Int)
int RomanToInt(string Rom)

```
   <binding name="IRomanbinding" type="tns:IRoman">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="IntToRoman">
         <soap:operation soapAction="urn:Roman-IRoman#IntToRoman" style="rpc"/>
         <input message="tns:IntToRoman0Request">
            <soap:body use="encoded" encodingStyle=.... />
         </input>
         <output message="tns:IntToRoman0Response">
            <soap:body use="encoded" encodingStyle=..../>
         </output>
      </operation>
      <operation name="RomanToInt">
         <soap:operation soapAction="urn:Roman-IRoman#RomanToInt" style="rpc"/>
         <input message="tns:RomanToInt1Request">
            <soap:body use="encoded" encodingStyle=.... />
         </input>
         <output message="tns:RomanToInt1Response">
            <soap:body use="encoded" encodingStyle=..../>
         </output>
      </operation>
   </binding>
   <service name="IRomanservice">
      <port name="IRomanPort" binding="tns:IRomanbinding">
         <soap:address location="http://www.ebob42.com/cgi-bin/Romulan.exe/soap/IRoman"/>
      </port>
   </service>
```

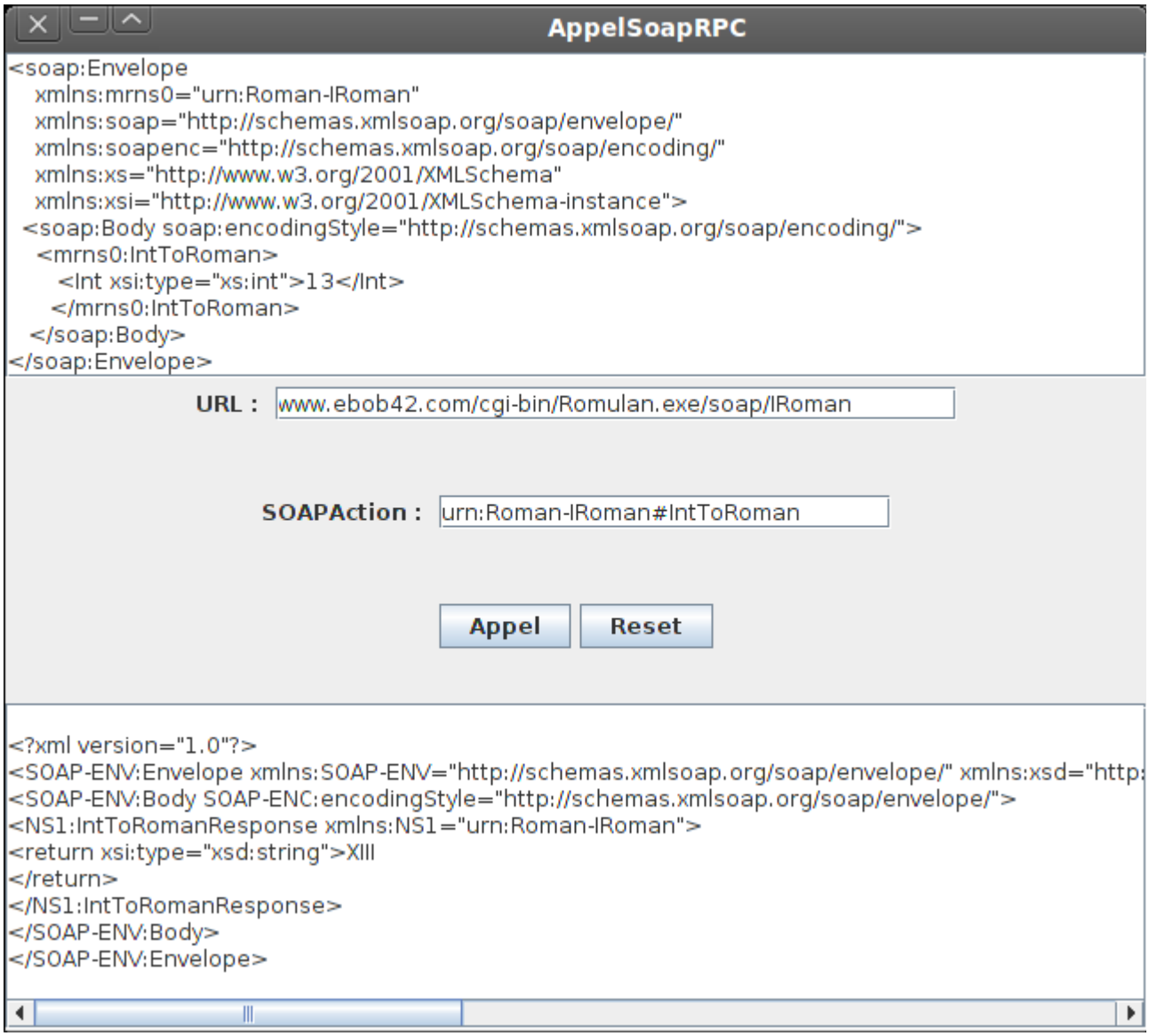
La liaison et le service définissent que :
le port d'"endpoint" http://www.ebob42.com/cgi-bin/Romulan.exe/soap/IRoman est transporté en utilisant SOAP sur HTTP selon un mode RPC et offre les 2 méthodes précisées avant.

- 4 types d'opération :
    - One-way émission sans requete
    - Request-response requete avec réponse synchrone

    - Solicit-response requête avec réponse asynchrone
    - Notification envoi simple
- 2 styles de transmissions : RPC et DOC
- La Sécurité des transactions est basées sur plusieurs normes dont :
  XML Encryption : mécanisme d'encryptage et de décryptage des documents XML.
  XML Signature : implémentation des signatures digitales dans les documents XML.

envois de message Soap à des services dont on connait la signature WSDL

- Récupérez la classe AppelSoapRPC.java
    - La classe envoie un document SOAP via HTTP
        - l'URL est celle du EndPoint du service
        - L'UrnSOAPACTION est inscrite dans l'entête HTTP (souvent vide "")
        - Le SoapContentXMLFile est le fichier XML qui contient de document SOAP



- La requête est construite en utilisant la description WSDL d'un service SOAP
    - Romulan Numbers XLII :
      trouvé sur le site des Web services : **xmethods** Ce qui signifie :
        - service soap

- type RPC
- transport HTTP
- url http://www.ebob42.com/cgi-bin/Romulan.exe/soap/IRoman
- soapAction "urn:Roman-IRoman#IntToRoman"
- nom de la méthode IntToRoman
- namespace utilisé http://eBob42.org/
- parametre donnée : Int de type int
- resultat : return de type string
- ci-joint l'enveloppe

Exercices :

- En utilisant la classe JAVA SoapHttpConnect ci-dessus,
  - Réaliser l'appel de la méthode RomanToInt sur la valeur XIII
  - Construire et envoyer quelques requêtes.
    En découvrant des web services de **xmethods** , par exemple :
    - get Curan verse
    - Magic square
    - ....

AXIS

Apache AXIS

- http://ws.apache.org/axis2
  - Un moteur qui implémente de protocole SOAP :
    - sur transport HTTP
    - qui fonctionne seul
    - en pluggin de TOMCAT
    - implémente WSDL
    - des outils WSDL2Java et Java2WSDL
    - ....

- Téléchargez le serveur TOMCAT d'APACHE :
  - http://tomcat.apache.org/ download ... binary distrib ... core .... zip
  - intallez Tomcat 5.5.34
    - dézippez-le
    - Ajoutez des variables d'environnement :
      `JAVA_HOME`
      `CATALINA_HOME chemin vers le répertoire d'installation de TOMCAT`
      - Voici un fichier ad-hoc setenv.sh
        `JAVA_HOME=/usr/lib/jvm/java-6-sun; export JAVA_HOME`
        `CATALINA_HOME=$HOME/chemin_a_preciser/apache-tomcat-5.5.34 ; export CATALI`

```
PATH=.:$PATH ; export PATH
```

- le rendre exécutable : `chmod u+x setenv.sh`
- A exécuter ainsi à chaque fois que vous ouvrez une console terminal ! ... ne fermez pas votre console !
  `. setenv.sh`
  Le point indique d'exécuter dans le shell même et non dans un shell processus fils

- pour vérifier l'état de vos variables d'environnement : la commande env
- rendez exécutable les .sh du répertoire bin :
  `chmod u+x $CATALINA_HOME/bin/*.sh`

- Démarrez TOMCAT :
  $CATALINA_HOME/bin/startup.sh
- testez TOMCAT :
  Dans un Navigateur, essayez http://localhost:8080/
  puis vérifiez que les JSPs proposés fonctionnent

- Installez et pluggez AXIS2 à TOMCAT :
  - http://axis.apache.org/axis2/java/core/
  - Télécharger axis2 1.5 le war :
    download ... realease ... 1.5.6 ..... WAR Distribution
  - dézippez axis2-1.5.6-war.zip
  - copiez le dossier axis2.war dans $CATALINA_HOME/webapps : cp axis2.war $CATALINA_HOME/webapps
  - redémarrez Tomcat :
    $CATALINA_HOME/bin/shutdown.sh
    $CATALINA_HOME/bin/startup.sh
  - Testez AXIS :
    - Dans un Navigateur, essayez http://localhost:8080/axis2
    - puis vérifiez que les services proposés fonctionnent
      - validate
- Installez le client_AXIS2 :
  - Télécharger axis2 1.5 le zip
    download ... realease ... 1.5.6 ..... Binary Distribution zip
  - dézippez axis2-1.5.6-bin.zip
  - Préparez les variables pour les clients :
    - Ajoutez une variable d'environnement : AXIS2_HOME
    - Modifiez le CLASSPATH pour prendre en compte les jars d'AXIS et le PATH pour ses utilitaires :
      dans le fichier setenv.sh, ajoutez les lignes :
      AXIS2_HOME=$HOME/chemin_a_preciser/axis2-1.5.6
      export AXIS2_HOME
      `PATH=$PATH:$AXIS2_HOME/bin ; export PATH`
      `CLASSPATH=$CLASSPATH:$AXIS2_HOME/lib/* ; export CLASSPATH`
    - re-exécutez `: . setenv.sh`

Un premier web service

- Le service :
  Source de TwiceSoapService.java

```
package service.twice;
public class TwiceSoapService  {
```

```
  public int doubler (int nombre) {
     return 2 * nombre;
  }
}
```

Minimal comme service !

- Déploiement :
  - le mettre dans un répertoire serviceDoubler
    `mkdir serviceDoubler`
    `cd serviceDoubler`
  - Compilez
    `javac -d . TwiceSoapService.java`

  - Ajoutez un fichier de description du service (web service descriptor)
    Source de META-INF/services.xml -> donc un fichier service.xml dans le répertoire META-INF dans serviceDoubler

```
<service name="TwiceService" scope="application">
    <description>
        Twice Service
    </description>
    <messageReceivers>
        <messageReceiver
            mep="http://www.w3.org/2004/08/wsdl/in-only"
            class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
        <messageReceiver
            mep="http://www.w3.org/2004/08/wsdl/in-out"
            class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
    <parameter name="ServiceClass">
        service.twice.TwiceSoapService
    </parameter>
</service>
```

  - Créez le fichier archive du Web service :
    un ensemble de fichier avec un répertoire META-INF contenant le services.xml
    dans le répertoire contenant le package java et META-INF :
    `jar cvf TwiceService.aar *`
  - Déployez :
    - arréter Tomcat
    - copier le fichier archive du Web service dans
      $CATALINA_HOME/webapps/axis2/WEB-INF/services
      `cp TwiceService.aar`
      `$CATALINA_HOME/webapps/axis2/WEB-INF/services`
    - re-démarrez le server
  - Testez le déploiement :
    - Dans un Navigateur, essayez http://localhost:8080/axis2
      pour voir si le service est proposé
    - puis http://localhost:8080/axis2/services/TwiceService?wsdl
      pour découvrir le fichier WSDL du service généré par le serveur
    - puis http://localhost:8080/axis2/services/TwiceService/doubler?args0=45
      pour faire un test de fonctionnement
    - puis http://localhost:8080/axis2/services/TwiceService?xsd
      pour avoir le schéma des données

Un premier client RPC

- Un client :
  - A partir du fichier wsdl du service, identifiez :
    - le endpoint du service
    - le nom de la méthode avec son namespace
    - les paramètres données et leur type
    - le type de résultat
  - Source de TwiceRPCClient.java -> le sauvegarder dans le répertoire clientDoubler1
    `package rpcclient;` -> donc un fichier service.xml dans le répertoire META-INF dans serviceDoubler

```
import javax.xml.namespace.QName;
import org.apache.axis2.AxisFault;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.rpc.client.RPCServiceClient;
public class TwiceRPCClient {
  public static void main(String[] args)  {
    try {
      int i = Integer.valueOf(args[0]);
      RPCServiceClient serviceClient = new RPCServiceClient();
      Options options = serviceClient.getOptions();
      EndpointReference targetEPR =
           new EndpointReference("http://localhost:8080/axis2/services/TwiceService")
      options.setTo(targetEPR);
      QName opDoubler = new QName("http://twice.service", "doubler");
      Object[] TabValeurs = new Object[] { i };
      Class[] returnTypes = new Class[] { Integer.class };
      Object[] response = serviceClient.invokeBlocking(opDoubler, TabValeurs, returnTy
      int result = (Integer) response[0];
      if (response == null) {
        System.out.println("PB ");
       return;
      }
      System.out.println("Doubler ( " +i+")="+result);
    } catch (AxisFault af) {System.out.println("AxisFault " +af.getMessage());}
  }
}
```

  - **les lignes de la classe :**
    - classe Options :
      - comment le client communique au service : transport, password, timeOut ...
      - méthode setTo(EndpointReference) : indique le Webservice cible
    - classe QName : pour définir des noms qualifiés XML (avec leur nameSpace)
    - classe RPCServiceClient :
      - méthode invokeBlocking() : lance la communication SOAP et attend la réponse
    - classe AxisFault :
      - hérite d'Exception
      - représente les erreurs SOAP
- EXECUTION

```
$ cd clientDoubler1
$ javac -d . TwiceRPCClient.java
$ java rpcclient.TwiceRPCClient 4
log4j:WARN No appenders could be found for logger (org.apache.axis2.util.Loader).
log4j:WARN Please initialize the log4j system properly.
```

```
Doubler ( 4)=8
```

pas de log branché : c'est pas grave !

- Trace de l'échange capturé par Wireshark : c'est bien du SOAP !

```
POST /axis2/services/TwiceService HTTP/1.1
Content-Type: text/xml; charset=UTF-8
SOAPAction: "urn:anonOutInOp"
User-Agent: Axis2
Host: localhost:8080
Transfer-Encoding: chunked
e9
<?xml version='1.0' encoding='UTF-8'?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlso
<soapenv:Body><doubler xmlns="http://twice.service"><arg0 xmlns="">4</arg0></doubler></soap
0
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 05 Sep 2010 08:42:09 GMT
103
<?xml version='1.0' encoding='UTF-8'?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlso
<soapenv:Body><ns:doublerResponse xmlns:ns="http://twice.service"><ns:return>8</ns:return><
</soapenv:Body></soapenv:Envelope>
0
```

Un second client généré semi-automatiquement

- Utilisation de l'outil wsdl2java pour générer les classes nécessaires à l'appel au web service :
  `wsdl2java.sh -uri http://localhost:8080/axis2/services/TwiceService?wsdl -o clientDoubler2`
  L'outil wsdl2java de Axis génère des classes java cliente du service en fonction du wsdl de ce service :
  - Ces classes sont le stub.
  - il est possible aussi de générer un squelette su service en java : utile si on dispose du même service dans un autre langage et qu'on souhaite le porter sur java
- Compilez les classes et générez la doc :
  `cd clientDoubler2`
  `javac -d . src/service/twice/*.java`
- Creez la doc du stub :
  `javadoc -d doc src/service/twice/*.java`
  Consultez la doc du stub
- Compilez et Executez :
  - Source de ClientStub.java -> le mettre dans clientDoubler2

```
import service.twice.TwiceServiceStub;
import service.twice.TwiceServiceStub.Doubler;
import service.twice.TwiceServiceStub.DoublerResponse;

public class ClientStub {
     public static void main(String[] args) throws Exception {
         service.twice.TwiceServiceStub stub = new service.twice.TwiceServiceStub();
         //Create the request
         service.twice.TwiceServiceStub.Doubler request =
              new service.twice.TwiceServiceStub.Doubler();
        request.setArgs0(45);
         //Invoke the service
```

```
            service.twice.TwiceServiceStub.DoublerResponse response
                = stub.doubler(request);
            System.out.println("Response : " + response.get_return());
        }
    }
```

Web service avec des données complexes

- **le service**
  - Source du service Cac40Service.java -> le mettre dans un répertoire serviceCac

```
package cacservice;
import java.util.*;
public class Cac40Service {
    private HashMap<String, Double> map = new HashMap<String, Double>();
    public String initialisation() {
      map.put("Alstom", new Double(340.0));
      map.put("Pernod-Ricard", new Double(560.0));
      map.put("AXA", new Double(834.0));
      map.put("UPJV", new Double(666.0));
      return map.toString();
    }
    public double getPrix(String symbol) {
      Double prix = (Double) map.get(symbol);
      if (prix != null)
         return prix.doubleValue();
      else
        return -1.00;
    }
    public void miseAJour(String symbol, double prix) {
      map.put(symbol, new Double(prix));
    }
    public String[] listCorbeille() {
      Set<String> corbeille = map.keySet();
      return  corbeille.toArray(new String[0]);
    }
}
```

  - Source du fichier "service descriptor" services.xml dans le répertoire META-INF dans le répertoire serviceCac

```
<service name="Cac40Service" scope="application">
    <description>
        Cac 40 Service
    </description>
    <messageReceivers>
        <messageReceiver
            mep="http://www.w3.org/2004/08/wsdl/in-only"
    class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
        <messageReceiver
            mep="http://www.w3.org/2004/08/wsdl/in-out"
    class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
    <parameter name="ServiceClass">
        cacservice.Cac40Service
    </parameter>
</service>
```

  - déployez :
    - `cd serviceCac`

        - $CATALINA_HOME/bin/shutdown.sh
        - javac -d . src/Cac40Service.java
        - jar cvf Cac40Service.aar *
        - cp Cac40Service.aar $CATALINA_HOME/webapps/axis2/WEB-INF/services/
        - $CATALINA_HOME/bin/startup.sh
    - testez :

```
http://localhost:8080/axis2/services/Cac40Service?wsdl
...
http://localhost:8080/axis2/services/Cac40Service?xsd
....
http://localhost:8080/axis2/services/Cac40Service/listCorbeille
http://localhost:8080/axis2/services/Cac40Service/initialisation
http://localhost:8080/axis2/services/Cac40Service/listCorbeille
<ns:listCorbeilleResponse>
<ns:return>AXA</ns:return>
<ns:return>Pernod-Ricard</ns:return>
<ns:return>Alstom</ns:return>
<ns:return>UPJV</ns:return>
</ns:listCorbeilleResponse>
http://localhost:8080/axis2/services/Cac40Service/getPrix?args0=AXA
<ns:getPrixResponse>
<ns:return>834.0</ns:return>
</ns:getPrixResponse>
http://localhost:8080/axis2/services/Cac40Service/getPrix?args0=Bonux
<ns:getPrixResponse>
<ns:return>-1.0</ns:return>
</ns:getPrixResponse>
http://localhost:8080/axis2/services/Cac40Service/miseAJour?args0=Bonux&args1=100
http://localhost:8080/axis2/services/Cac40Service/getPrix?args0=Bonux
<ns:getPrixResponse>
<ns:return>100.0</ns:return>
</ns:getPrixResponse>
```

- **le client RPC :**
    - Source du client Cac40RPCClient.java ... le mettre dans clientCac1

```
package rpcclient;
import javax.xml.namespace.QName;
import org.apache.axis2.AxisFault;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.rpc.client.RPCServiceClient;
public class Cac40RPCClient {
    public static void main(String[] args) throws AxisFault {
        RPCServiceClient serviceClient = new RPCServiceClient();
        Options options = serviceClient.getOptions();
        EndpointReference targetEPR =
           new EndpointReference("http://localhost:8080/axis2/services/Cac40Service")
        options.setTo(targetEPR);
        QName op = new QName("http://cacservice", "getPrix");
        Class[] returnTypes = new Class[] { Double.class };
        Object[] response = serviceClient.invokeBlocking(op, new Object[] {"AXA" }, r
        Double result = (Double) response[0];
        if (result == null) {
            System.out.println("didn't initialize!");
            return;
        }
        System.out.println("getPrix(AXA)  = " + result.toString());

        op = new QName("http://cacservice", "miseAJour");
```

```
            serviceClient.invokeRobust(op, new Object[] {"Bonux", new Double(100.0)});
            op = new QName("http://cacservice", "listCorbeille");
            String[] tabResult = {};
            returnTypes = new Class[] { tabResult.getClass() };
            response = serviceClient.invokeBlocking(op, new Object[] {}, returnTypes);
            tabResult = (String[]) response[0];
            if (tabResult == null) {
                System.out.println("tabResult didn't initialize!");
                return;
            }
            System.out.println("listCorbeille : ");
            for (int i=0; i<tabResult.length; i++)
              System.out.println("["+i+"] = " + tabResult[i].toString());
        }
    }
```

- la méthode invokeRobust() permet l'appel de méthodes In-Only MEP (message exchange pattern)
- passage de 2 arguments

- EXECUTION

```
$ cd clientCac1
$ javac -d . src/Cac40RPCClient.java
$ java rpcclient.Cac40RPCClient
log4j:WARN No appenders could be found for logger (org.apache.axis2.util.Loader).
log4j:WARN Please initialize the log4j system properly.
getPrix(AXA)  = 834.0
listCorbeille :
[0] = Bonux
[1] = AXA
[2] = Pernod-Ricard
[3] = Alstom
[4] = UPJV
```

- **le client généré par wsdl2java :**
  -
```
$ wsdl2java.sh -uri http://localhost:8080/axis2/services/Cac40Service?wsdl -o
clientCac2
 Using AXIS2_HOME:   /home/ferment/upjv/axis2/axis2-1.5.1
 Using JAVA_HOME:       /usr/lib/jvm/java-6-sun
Retrieving document at 'http://localhost:8080/axis2/services/Cac40Service?wsdl'.
$ cd clientCac2
$ javac -d . src/cacservice/*.java
Note: src/cacservice/Cac40ServiceStub.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
$ javadoc -d doc src/cacservice/*.java
```
  - Source du service ClientStub.java -> le mettre dans clientCac2

```
import cacservice.Cac40ServiceStub;
import cacservice.Cac40ServiceStub.GetPrix ;
import cacservice.Cac40ServiceStub.GetPrixResponse;
import cacservice.Cac40ServiceStub.ListCorbeilleResponse;

public class ClientStub {
     public static void main(String[] args) throws Exception {
         Cac40ServiceStub stub = new Cac40ServiceStub();
         //Create the request
         Cac40ServiceStub.GetPrix  request = new Cac40ServiceStub.GetPrix();
        request.setArgs0("AXA");
         //Invoke the service
        Cac40ServiceStub.GetPrixResponse response = stub.getPrix(request);
        System.out.println("getPrix(AXA) : " + response.get_return());
         //Invoke the service
```

```
            Cac40ServiceStub.ListCorbeilleResponse rep2 = stub.listCorbeille();
            String[] tabResult = rep2.get_return();
            System.out.println("listCorbeille : ");
            for (int i=0; i<tabResult.length; i++)
              System.out.println("["+i+"] = " + tabResult[i].toString());
          }
    }
```

- EXECUTION

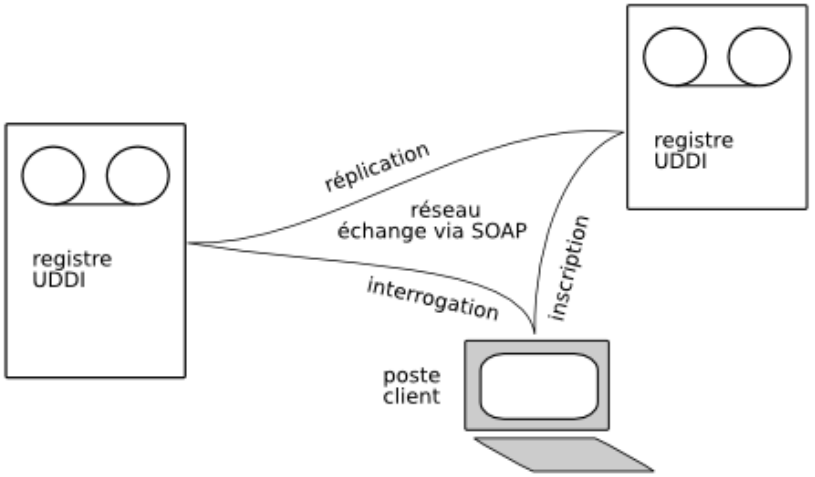
```
$ javac -d . src/ClientStub.java
$ java ClientStub
log4j:WARN No appenders could be found for logger (org.apache.axis2.description.AxisS
log4j:WARN Please initialize the log4j system properly.
getPrix(AXA) : 834.0
listCorbeille :
[0] = Bonux
[1] = AXA
[2] = Pernod-Ricard
[3] = Alstom
[4] = UPJV
```

Exercices

- Appelez le service Twice d'un autre ordinateur (avec son IP)
- Ajoutez des fonctionnalités au Cac40 :
  - méthode enlever(entreprise) du cac
  - méthode liste des prix de toutes les actions du CAC
  - créez un client rpc et un client généré par wsdl2java pour tester ces nouvelles fonctions.
- créer un service ToutouService :
  - utilisez Chien7.java
  - le service dispose d'une liste de chiens
  - mettre en service la fonction initialisation de quelque chiens et la fonction getList() qui donne la liste de tous les noms des chiens
    - tester
    - créez un client rpc et un client généré par wsdl2java pour tester la fonction getList().
  - ajouter une méthode boolean existe(Chien7) qui renvoie vrai si le chien existe dans le tableau de chiens du Toutou Service
    - testez et écrivez un client ad-hoc (Rpc ou généré par wsdl2java)
  - ajouter une méthode add(Chien7)
    - testez et écrivez un client ad-hoc (Rpc ou généré par wsdl2java)
  - ajouter une méthode find(nom) qui renvoie le Chien7 de ce nom s'il existe
    - testez et écrivez un client ad-hoc (Rpc ou généré par wsdl2java)

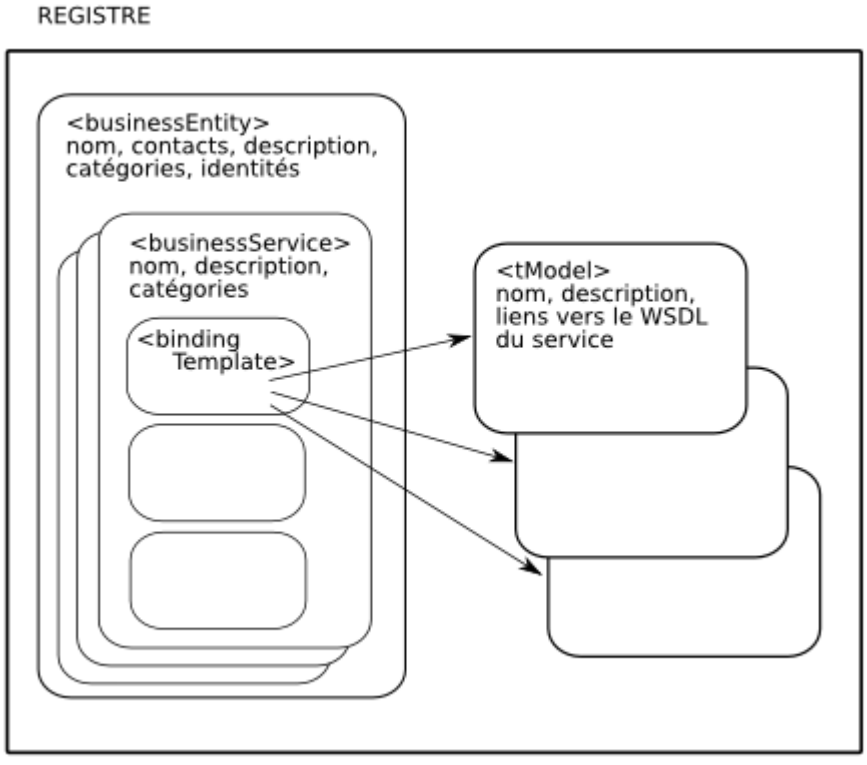
UDDI - SOA

UDDI



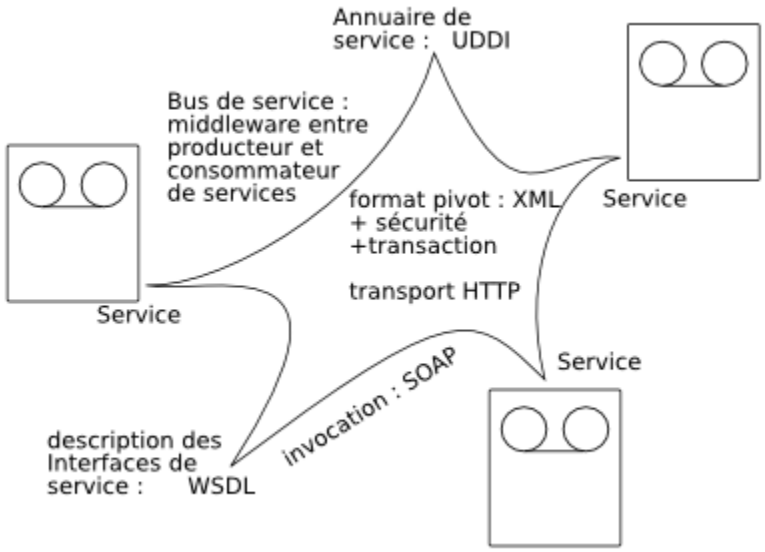
- Universal Description, Discovery and Integration
  - annuaire des web services et interface d'interrogation des informations.
    - les "pages blanches" = liste des entreprises : nom de l'entreprise, coordonnées, description, des identifiants ;
    - les "pages jaunes" = liste des services web de chaque entreprise en format WSDL;
    - les "pages vertes" = informations techniques sur les services fournis.
  - annuaires privés pour le B2B (échange Business to Business)
  - norme édictée par l'OASIS.
  - utilise XML pour les descriptions
  - protocole de transport SOAP.
  - l'annuaire est "universel" mais physiquement réparti : constitué d'un réseau d'annuaire UDDI et utilisant des mécanismes de réplication, comme le DNS.

- Le contenu XML d'un registre UDDI :



- Impossible d'essayer des appels à un UDDI car ils sont propriétaires !

SOA



- Service Oriented Architecture
    - format interne strict/fort : XML
    - couplage externe faible : loosely-coupled
      le protocole d'échange entre machines communicantes exige le moins de contrainte possible.
    - solution souple, simple, interchangeable pour le B2B
- composé de :
    - annuaire de services UDDI
    - bus de service :
        - middleware entre le consommateur et le producteur du service
        - réalise le couplage lâche basé sur SOAP
          donc un format des données échangées XML
        - transport des données avec le protocole : HTTP
        - sécurité basé sur :
            - SAML (Security Assertion Markup Language),
            - XML Signature, XML Encryption,
            - XKMS (XML Key Management Specification),
            - SSL (Secure Sockets Layer)
        - gestion transactionnelle par le "two-phase commit" :
          pour mettre à jour plusieurs bases de données réparties entre plusieurs services, la transaction attend l'acquittement (commit) des différents serveurs
            - WS-Transaction,
            - XAML (Transaction Authority Markup Language),
            - BTP (Business Transaction Protocol).
    - Le service :
        - Interfaces
          décrite par WSDL
        - Localisation :
          s'obtient par UDDI
          et la section binding de WSDL
        - invocation :
            - par un couplage faible basé sur SOAP
            - mode Synchrone requête avec attente de la réponse
            - Asynchrone requête sans attente de la réponse mais mise en place d'un handler qui traitera la réponse quand elle arrivera
        - l' orchestration ou chorégraphie des services pour constituer des processus métier
            - BPEL4WS devenu WS-BPEL(Business Process Execution Language for Web Services) langage d'orchestration