USTL – Master mention Informatique – M1

Construction d'applications réparties

VII. Web Services

Lionel.Seinturier@lifl.fr

Plan

1. Caractéristiques

Besoins

Communiquer en environnement réparti & hétérogène (OS, lang)

	Protocole	Format représentation données
• DCOM	RPC DCE	binaire
• CORBA	IIOP	binaire (CDR)
• RMI	JRMP/IIOP	binaire (Java Serialization)

• quel est le protocole le + utilisé sur Internet ?
• quel est le format de données universel ?

• HTTP : simple, sans état, toutes les chances de traverser *firewalls*
• XML : ASCII (pas binaire), *parsing* facile, standard W3C

1. Caractéristiques

Eléments de base : HTTP

• Protocole applicatif (niveau 7)
• Utilise TCP (niveau 4) ⇨ garantie d'un transport **fiable** (sans erreur)
• **Pas** de notion de **connexion** HTTP
• Tous les commandes HTTP sont émises en **mode texte** (ASCII)

⇨ Protocole simple, facilement implantable

Version actuelle HTTP 1.1 (RFC 2067) depuis janvier 1997
Apport principal : **connexions TCP persistantes**

Raison : pour les "petits" fichiers (< 10 Ko, 80 % des documents Web)
le coût de l'ouverture de cx TCP est **non négligeable** / coût du transfert

⇨ gain de temps important

1. Caractéristiques

Eléments de base : HTTP

3 commandes principales (présentes dans HTTP/1.0 et 1.1)

GET	demande d'un document
HEAD	demande de l'en-tête (HTTP) d'un document
POST	dépose d'information sur le serveur

En-têtes client

• informations sur le client	From, Host, User-Agent
• information sur la page contenant le lien	Referer
• *login*, mot de passe	Authorization
• préférences pour le document demandé	Accept …
• conditions sur le document demandé	If …

1. Caractéristiques

Eléments de base : HTTP

commande URL version HTTP comprise par le client
en-tête 1 HTTP: valeur
....
en-tête n HTTP: valeur

informations envoyées par le client

```
GET /index.html HTTP/1.1
Accept-language: fr
User-Agent: Mozilla/4.0
If-modified-since: Tue, 23 Jul 1997 10:24:05
```

```
POST /index.php HTTP/1.1
Accept-language: fr

nom=Seinturier&prenom=Lionel
```


1. Caractéristiques

Eléments de base : HTTP

En-têtes client

• informations sur le client	From, Host, User-Agent
• information sur la page contenant le lien	Referer
• *login*, mot de passe	Authorization
• préférences pour le document demandé	Accept …
• conditions sur le document demandé	If …

Accept: liste de types MIME
Accept-Charset, Accept-Encoding, Aspect-Language
If-Modified-Since, If-Unmodified-Since

1. Caractéristiques

Eléments de base : HTTP

Réponse du serveur

version HTTP du serveur code retour commentaire
en-tête 1 HTTP: valeur
....
en-tête n HTTP: valeur

document texte (HTML, …) ou binaire (GIF, JPG, …)

```
HTTP/1.1 200 OK
Content-Length: 9872
Content-Type: text/html

<HTML>
....
</HTML>
```

1. Caractéristiques

Eléments de base : HTTP

code retour : renseigne sur le succès (200) ou l'échec (4xx) de la requête

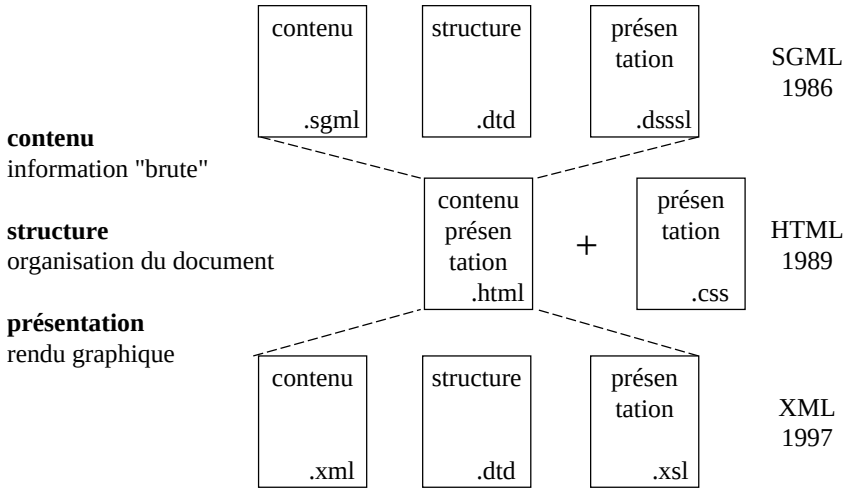
- 200 : ok
- 404 : document inconnu
- 401 : authentification nécessaire
- 500 : erreur du serveur HTTP dans le traitement requête (servlet, PHP, …)
- 503 : serveur temporairement surchargé
- …

en-têtes HTTP : informations transmises par le serveur sur le document envoyé

- Content-Length : taille du document
- Last-Modified : date de dernière modification du document
- Server : nom du logiciel serveur
- Expire : date d'expiration du document
- Content-Type : type (MIME) du document
- … **nombreux** autres en-têtes possibles

1. Caractéristiques

Eléments de base : XML



contenu
information "brute"

structure
organisation du document

présentation
rendu graphique

1. Caractéristiques

Eléments de base : XML

1. DTD
2. XML Schema
3. XML Namespace

Déclaration version XML utilisée
DTD utilisée pour ce document
Corps du document

```
<?xml version="1.0" ?>
<!DOCTYPE individu SYSTEM "individu.dtd" >
<individu>
    <nom>Seinturier</nom>
    <prenom>Lionel</prenom>
</individu>
```

- valide syntaxiquement
- conforme à sa DTD

1. Caractéristiques

Eléments de base : XML

DTD grammaire (balises) du document

1. Définition des balises autorisées `<!ELEMENT ... >`
2. Définition de leurs attributs `<!ATTLIST ... >`

```
<balise attribut="type attr" ...> type balise </balise>

<!ELEMENT graphe (noeud|arc)* >
<!ELEMENT noeud EMPTY >
<!ELEMENT arc EMPTY >
<!ATTLIST noeud numero ID #REQUIRED >
<!ATTLIST arc source IDREF #REQUIRED >
<!ATTLIST arc destin IDREF #REQUIRED >

<graphe>
<noeud numero="1"/> <noeud numero="2"/>
<arc source="2" destin="1"></arc>
</graphe>
```

1. Caractéristiques

Eléments de base : XML

XML Schema

Document XML pour la définition de DTD
⇨ les DTD XML sont définies comme des docs XML λ

Avantages

- 1 seul et même langage pour les docs et la déf. de leurs DTD
- types de données + évolués (entier, réel, chaîne, date, liste, ...)
- grammaires définissables potentiellement + évoluées

1. Caractéristiques

Eléments de base : XML

XML Schema

```
<!ELEMENT promotion (individu)+ >
<!ELEMENT individu ( nom , prenom ) >
<!ELEMENT nom (#PCDATA)> <!ELEMENT prenom (#PCDATA)>
<!ATTLIST individu noSecuriteSociale ID #REQUIRED >
```

promotion.dtd

```
<?xml version="1.0" ?>
<element name="promotion" type="PromotionType" />
<complexType name="PromotionType">
  <element name="individu" type="IndividuType"
           minOccurs="1" maxOccurs="unbounded" />
  <attribute name="noSecuriteSociale"
             type="ID" use="required" />
</complexType>
<complexType name="IndividuType">
<sequence> <element name="nom" type="string">
           <element name="prenom" type="string">
</sequence> </complexType>
```

XML *schema* équivalent

1. Caractéristiques

Eléments de base : XML

XML Namespace

Utilisation des balises provenant de **+sieurs DTD** dans un doc. XML

- attribut **réservé** `xmlns` fournissant un nom et l'URL de sa DTD associée
- peut être ajouté à n'importe quelle balise (en général, la 1ère du document)

```
<balise xmlns:nomDEspace="URL associée" ... >
```

```
<html xmlns:m="http://www.w3.org/1998/Math/MathML"
      xmlns:s="http://www.w3.org/2000/svg" >
```

- l'espace de noms reste valide jusqu'à la **balise fermante** (ici `</html>`)
- les balises des ≠ DTD doivent être préxifées par *`nomDEspace:`*

```
<s:svg width="2cm" height="0.6cm">
```


1. Caractéristiques

Eléments de base : XML

XML Namespace

Exemple

Utilisation conjointe de 3 DTD
XHTML, SVG (figures géométriques) et MathML (formules mathématiques)

```
<?xml version="1.0" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:s="http://www.w3.org/2000/svg"
      xmlns:m="http://www.w3.org/1998/Math/MathML">

<head>
<title>Exemple d'utilisation des espaces de noms</title>
</head>

<body>  <h1>Les espaces de noms</h1>
```

1. Caractéristiques

Eléments de base : XML

XML Namespace

```
<p>Un rectangle en SVG</p>
<s:svg width="120" height="35">
    <s:rect width="120" height="35" rx="12"
            fill="blue" stroke="#4C00E5" />
</s:svg>

<p>Une formule de math en MathML</p>
<m:math>
       <m:mroot>x+y</m:mroot>
</m:math>
</body> </html>
```

Visualisation avec Amaya

www.w3.org/Amaya

2. XML-RPC

XML-RPC

Fournir un support simple de communication entre applications

Etre indépendant des technologies d'exécution
- langages de programmation
- plate-forme d'exécution

Mécanisme de base standard pour transporter des requêtes
- basé sur le langage XML pour l'encodage
- basé sur le protocole HTTP pour le transport

Vue d'ensemble

Trois parties

XML-RPC Data Model
- ensemble de types pour encoder les requêtes
- paramètres, valeurs de retour et messages d'erreur

XML-RPC Request Structure
- Requête HTTP Post
- Information : nom de méthode et paramètres

XML-RPC Response Structures
- Réponse HTTP
- Information : valeur de retour ou message d'erreur

XML-RPC Data Model

Définit les structures de base des messages

Six types de base

`int, double, boolean, string, dataTime, base64`

Deux types construits

`array, struct`

Ensemble réduit de structures de données

dénominateur commun de beaucoup de langages

suffisant pour représenter nombre d'informations

Types de base

Eléments XML simples contenant les valeurs

Type	Valeur	Exemple
string	ASCII text	<string>hello</string>
int	Int 32-bit	<int>27</int>
double	Float 64-bit	<double>3.14159</double>
boolean	True/false	<boolean>1</boolean>
dateTime	Date	<dateTime.iso8601>20031130T14:57:16</dateTime.iso8601>
base 64	Info binaire	<base64>SGVcbG8sIFdcvmxklQ==</base64>

Utilisation des types de base

Toujours inclus dans un élément <value>

```
<value><string>hello world</string></value>
<value><int>12345</int></value>
<value><double>-4.25</double></value>
```

Seules les balises <string> peuvent être omises

```
<value>Ceci est une chaîne de caractères</value>
```

Peuvent être combinés pour construire

Tableaux

Structures

Les tableaux

Objectifs

- Structuration d'information séquentielle
- Structuration récursive possible
- Taille non contrainte
- Peut contenir des données hétérogènes
  - Types de base et construits

Utilisation

- Contenu dans un `<value>`
- Définition avec la balise `<array>`
- Valeurs définies dans la partie `<data>`

Utilisation des tableaux

Type de données homogènes

```
  <value>
      <array>
            <data>
                  <value><int>1</int></value>
                  <value><int>2</int></value>
                  <value><int>3</int></value>
            </data>
      </array>
</value>
```


Utilisation des tableaux

Type de données hétérogènes

```
  <value>
    <array>
      <data>
         <value><int>1</int></value>
         <value><string>poisson</string></value>
         <value><dateTime.iso8601>20051130T14:57:19
         </dateTime.iso8601></value>
         <value><boolean>1</boolean></value>
      </data>
    </array>
</value>
```


Utilisation des tableaux

Tableaux à plusieurs dimensions

```
<value>
  <array>
     <data>
        <value>
           <array>
             <data>
                <value><int>00</int></value>
                <value><int>01</int></value>
             </data>
           </array>
        </value>
        <value>
           <array>
             <data>
                <value><int>10</int></value>
                <value><int>11</int></value>
             </data>
           </array>
        </value>
     </data>
  </array>
</value>
```


Les structures

Objectifs
- définir des listes de paires nom/valeur
- proche des tables de hashage/propriétés
- contient des éléments non ordonnés
- peut contenir tout type de données

Utilisation
- contenu dans un <value>
- définition avec la balise <struct>
- une paire est définie par la balise <member>
- chaque est défini par <name> et <value>

Utilisation des structures

Une structure simple à deux entrées

```
<value>
  <struct>
    <member>
       <name>nom</name>
       <value><string>Seinturier</string></value>
    </member>
    <member>
       <name>bureau</name>
       <value><int>233</int></value>
    </member>
  </struct>
</value>
```


Utilisation des structures

Une structure plus complexe : nom, année, modules

```
<value>
     <struct>
        <member>
           <name>nom</name> <value><string>Alexandre</string></value>
        </member>
        <member>
           <name>annee</name> <value><int>5</int></value>
        </member>
        <member>
           <name>modules</name>
           <value><array>
                 <data>
                   <value><string>CAR</string></value>
                   <value><string>COO</string></value>
                   <value><string>C++</string></value>
                   <value><string>Réseaux</string></value>
                 </data>
           </array></value>
        </member>
   </struct>
</value>
```


XML-RPC Request Structures

Combinaison d'information XML et de requêtes HTTP

- Utilise le modèle de données XML-RPC
  - Spécifie la procédure appelée et ses arguments
- Exploite l'en-tête du protocole HTTP
  - Encapsule les requêtes sur le réseau

Une requête = un document XML

- Racine du document: `<methodCall>`
  - Nom de la procédure : `<methodName>`
  - Liste ordonnée des paramètres : `<params>`
  - Chaque paramètre identifié par un `<param>`

Exemple de requête XML-RPC

Invocation d'une procédure de tri d'une tableau d'entiers

```
<?xml version="1.0" ?>
<methodCall>
   <methodName>tri</methodName>
   <params>
     <param>
        <value><array><data>
                <value><int>123</int></value>
                <value><int>675</int></value>
                <value><int>389</int></value>
        </value></array></data>
     </param>
   </params>
</methodCall>
```

Exemple de requête XML-RPC

En-tête HTTP d'une requête POST

- Reflète l'émetteur (`myXMLRPCClient`) de la requête et le contenu
- Serveur XML-RPC à l'adresse `/xmlrpc`

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 134.206.11.23
Content-Type: text/xml
Content-Length: 249
```


Structure des réponses

Similaire aux requêtes

Si exécution correcte

- document XML avec racine <methodResponse>

Un seul paramètre

- Type simple ou construit
- si procédure sans retour
  - Une valeur doit quand même être retournée
    - Exemple : un booléan avec la valeur vraie (1)

En-tête des réponses XML-RPC

Réponses HTTP

- HTTP 200 OK
- Précise le serveur HTTP ayant traité la requête
- Content-type : text/xml
- Connexion fermée après chaque réponse

En-tête des réponses XML-RPC

```
HTTP/1.0 200 OK
Data: Mon Nov 24 03:15:11 CET 2005
Server: APACHE.2.0.1(Linux)
Content-Type: text/xml
Content-Length: 147

<?xml version="1.0" ?>
<methodResponse>
   <params>
     <param>
        <value><array><data>
                <value><int>123</int></value>
                <value><int>389</int></value>
                <value><int>675</int></value>
        </value></array></data>
     </param>
   <params>
</methodResponse>
```

Structure des réponses : échec

Si problème dans l'exécution de la procédure
- Elément XML <methodResponse>
- Contient un élément XML fault
  - contient une seule valeur reflétant l'erreur
  - Codes d'erreurs non standardisés (de niveau applicatif)

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
      <value><string>
      Methode inconnue : tri
      </string></value>
  </fault>
</methodResponse>
```


Mise en œuvre de XML-RPC

Mise en œuvre
- Utilisation d'une librairie XML-RPC
- Appels de procédure via la librairie
- Ecriture d' "encapsuleurs" pour réutiliser le code

Illustration avec un serveur : 1 convertisseur
- Utilisation de la librairie Java XML-RPC du projet Apache
- http://xml.apache.org/xmlrpc/

Mise en œuvre du service

Classe Java implantant une méthode de conversion
- Transforme une valeur en euros vers une valeur en francs

```
package fr.lifl.cmlrpc.exemple;

public class ConvertisseurHandler {
  public double euro2franc(double valeur){
      return valeur * 6.55975;
  }
}
```


Mise à disposition du service

Un serveur doit rendre cette classe accessible via HTTP
La procédure doit être enregistrée auprès du package XML-RPC

```
package fr.lifl.xmlrpc.exemple;
import org.apache.xmlrpc.WebServer;

public class Server {
   public static void main (String[] args) throws Exception {

   // démarrage du serveur sur le port 12345
   WebServer server=new WebServer(12345);

   // enregistrement du service CONVERTIR
   server.addHandler(
        "CONVERTIR",
        new ConvertisseurHandler() );
   }
}
```

Réalisation d'un client

```
package fr.lifl.xmlrpc.exemple;
import org.apache.xmlrpc.XmlRpcClient;

public class Client {
   public static void main (String[] args)
        throws Exception {

   XmlRpcClient client =
        new XmlRpcClient("http://localhost:12345/CONVERTIR");

   java.util.Vector params = new java.util.Vector();
   params.addElement (new Double(15.25));

   Object resultat = client.execute("euro2francs",params);
}  }
```


Requête XML-RPC de l'exemple

```
POST /HTTP 1.1
Content-Length: 188
Content-Type: text/xml

<?xml version="1.0" encoding="ISO-8859-1" ?>
<methodCall>
  <methodName>CONVERTIR.euro2franc</methodName>
  <params>
      <param>
            <value><double>15.25></double></value>
      </param>
  </params>
</methodCall>
```


Réponse XML-RPC de l'exemple

```
HTTP/1.0 200 OK
Date: Mon Nov 24 03:15:11 CET 2003
Server: APACHE: 1.3.20(Linux)
Content-Type: text/xml
Content-Length: 143

<?xml version="1.0" encoding="ISO-8859-1" ?>
<methodResponse>
   <params>
        <param>
              <value>
                    <double>100.03619></double>
              </value>
        </param>
   </params>
</methodCall>
```


Conclusion XML-RPC

Concept simple

- Approche par plus petit dénominateur commun
- Facile à implanter
- Permet la coopération d'applications hétérogènes

Mais

- Ne supporte pas la gestion de session
- Pas de contrat entre client et serveur

3. SOAP

SOAP

(proposition IBM + Microsoft)

3 points de vues sur SOAP

- protocole d'échange de message
- format d'échange de documents

- indépendant des langages
- indépendant des OS
- indépendant des protocoles de communication (mais HTTP ++)

But : invoquer un service distant

Specifications SOAP

1. *SOAP envelope specification* quelle méthode ? paramètre ? retour ? erreur ?
2. *Data encoding rules* règles de représentation des types de données

3. SOAP

SOAP

v1.1 SOAP = Simple Object Access Protocol
depuis v1.2 SOAP = SOAP

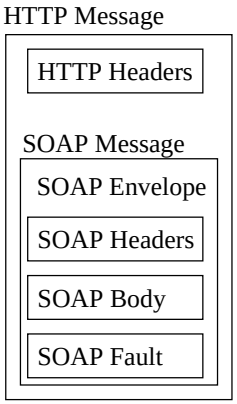
- SOAP n'a **aucune** notion orientée objet
  - pas de référence d'objet
  - pas d'instanciation
  - pas de cycle de vie
  - pas de GC
  - pas de variables d'instances
  - pas d'état "conversationnel" i.e. session (+/- = EJB *stateless bean*)

SOAP : technologie centrée sur document XML

3. SOAP

Structure message SOAP

- 1 invocation par message
- la réponse a la même structure



Envelope & Body : obligatoire
Headers & Fault : facultatif

3.1 Envelope

Enveloppe

- les informations du messages
- document XML

- balise racine `<Envelope>`
- 2 balises principales : `<Header>` et `<Body>`

- balise utilisateur (méthode & param) : ex `<euroToDollar> <value>`

Invocation du service `euroToDollar` avec la valeur `12.34`

```
<Envelope>
  <Body>
        <euroToDollar>
          <value>12.34</value>
        </euroToDollar>
  </Body>
</Envelope>
```

3.1 Envelope

Enveloppe

Risque conflits balises SOAP & utilisateur de noms utilisation de namespace

• SOAP-ENV namespace XML pour SOAP
`http://schemas.xmlsoap.org/soap/envelope`

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-Env:Body>
       <euroToDollar>
         <value>12.34</value>
       </euroToDollar>
  </SOAP-Env:Body>
</SOAP-ENV:Envelope>
```


3.1 Enveloppe

Enveloppe

Exemple HTTP

```
POST /convertisseur HTTP/1.1
Content-Type: text/xml
Content-Length: 999
SOAPAction: http://some.host/SOAPServer/convertisseur

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-Env:Body>
       <euroToDollar>
         <value>12.34</value>
       </euroToDollar>
  </SOAP-Env:Body>
</SOAP-ENV:Envelope>
```

`SOAPAction` (URI du service web invoqué) doit être présent
mais peut-être vide ⇨ URL POST suffisante pour identifier le service

3.1 Enveloppe

Enveloppe

Exemple HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 999

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Body>
       <euroToDollarResponse>
         <return>10.07</return>
       </euroToDollarResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


3.2 Header

En-tête

• informations supplémentaires sur le message : balises
• non liées à l'invocation proprement dite
• en relation avec son contexte d'exécution

Exemple

```
  <SOAP-ENV:Header>
       <Transaction SOAP-ENV:mustUnderstand="1">
         5
       </Transaction>
  </SOAP-ENV:Header>
```

2 attributs facultatifs
• `mustUnderstand="1"` le serveur doit être capable de traiter le message
• `actor="`*`url`*`"` destinataire du header en cas d'appels en cascade

3.3 Fault

Erreur

• signale une erreur d'exécution (message de retour)
• balise `<Fault>`

4 sous-balises

`<faultcode>`	type d'erreur
`<faultstring>`	message d'erreur pour l'utilisateur
`<faultactor>`	émetteur de l'erreur en cas d'appels en cascade
`<detail>`	message détaillé pour l'application (ex. *stack trace*)

4 valeurs principalespossibles pour `<faultCode>`

`Client`	erreur provenant de la requête du client
`Server`	erreur provenant du serveur
`MustUnderstand`	incapacité à traiter un header mustUnderstand
`VersionMismatch`	namespace de l'enveloppe incorrect

mais valeurs extensibles (même esprit que les code 4xx pour HTTP)
ex : `Client.Authentication`

3.3 Fault

Erreur

Exemple

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Fault>
     <SOAP-ENV:faultcode>Client</SOAP-ENV:faultcode>
     <SOAP-ENV:faultstring>Méthode inexistante</SOAP-ENV:faultstring>
  </SOAP-ENV:Fault>
</SOAP-ENV:Envelope>
```


3.4 Data encoding rules

Règles de représentation des types de données

But : typer les données échangées

• types simples : string, int, double, boolean, date, time, enum, tableaux d'octets
• types composés : structures, tableaux

positionner l'attribut
`encodingStyle http://schemas.xmlsoap.org/soap/encoding`
(aussi utilisé comme namespace)

• typage en référençant un XML Schema
• typage directement avec les données transmises
  attribut `type` ⇨ namespace `xsi`
  typage XML Schema ⇨ namespace `xsd`

```
xsi   http://www.w3.org/2001/XMLSchema-instance
xsd   http://www.w3.org/2001/XMLSchema
```


3.4 Data encoding rules

Types simples

Exemple de message avec données typées

```
<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>
  <euroToDollarResponse
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
         <return xsi:type="xsd:double">10.07</return>
  </euroToDollarResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3.4 Data encoding rules

Types simples

Exemple de message avec typage externe

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Body>
  <euroToDollarResponse
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
         <foo:return xmlns:foo="uri du schema">10.07</foo:return>
  </euroToDollarResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Schéma

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="return" type="xsd:double" />
</xsd:schema>
```


3.4 Data encoding rules

Types simples

Enumération

• cas particulier de chaîne avec une liste de valeurs autorisées

Schéma

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="return">
    <simpleType base="xsd:string">
      <enumeration value="mineur" />
      <enumeration value="majeur" />
    </simpleType>
  </xsd:element>
</xsd:schema>
```

Message

```
<foo:return xmlns:foo="uri du schema">majeur</foo:return>
```


3.4 Data encoding rules

Types simples

Tableau d'octets

• type `base64`

```
<picture xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:SOAP-ENC="http://www.w3.org/soap/encoding"
         xsi:type="SOAP-ENC:base64" >
   aG93IG5vDyBicm73biBjb3cNCg==
</picture>
```


3.4 Data encoding rules

Types composés

Structures

```
struct Personne { string nom; int age; }
```

```
<foo:Personne xmlns:foo="uri du schema"
  <foo:nom>Bob</foo:nom>
  <foo:age>45</foo:age>
</foo:Personne>
```

Schéma

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Personne" >
    <xsd:complexType base="xsd:string">
      <xsd:element name="nom" type="xsd:string" />
      <xsd:element name="age" type="xsd:int" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

3.4 Data encoding rules

Règles de représentation des types de données

Tableaux

• type `arrayType`

```
<myFavoriteNumbers
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENC="http://www.w3.org/2001/06/soap-encoding"
  SOAP-ENC:arrayType="xsd:int[2]" >
  <number>3</number>
  <number>4</number>
</myFavoriteNumbers>
```

• éléments de types non homogènes,
• éléments de types structure ou tableaux
• tableaux multi-dimensionnels
• creux (seuls certains éléments sont transmis)

3.5 Implantation

Axis `ws.apache.org/axis/`

Architecture

• serveur HTTP
• Tomcat
• servlet RPC Router
  - "comprend" SOAP
  - aiguille les requêtes

Services SOAP en. Java + descripteur de service XML

```
public class EssaiDeWS {
   public double euroToDollar(double euro) { return euro/1.12; }
}
```

Client SOAP en Java avec API `org.apache.soap`

3.6 Conclusion SOAP

Conclusion

• protocole simple, facilement implantable
• sécurité basée sur la sécurité du protocole sous-jacent (ex. HTTPS)
• indépendant langages, OS

mais

• pas de passage d'objets par référence
• pas d'activation à la demande
• pas de gestion de l'exécution des requêtes
• pas de ramasse-miettes
• pas de transaction entre +sieurs invocations (voir extensions)
• fiabilité essentiellement basée sur TCP

⇨ un protocole de transport d'invocation de services

4. WSDL

WSDL (Web Service Description Language)

Description de services web

• contrat pour l'utilisation du service
• description XML

4 informations spécifiées

• interface du service (méthodes disponibles)
• types de données
• mode de connexion
• localisation du service

4. WSDL

Balises WSDL

<definitions> Racine
  <types> les types de données échangées
  <message> les messages transmis
  <portType> des regroupements d'opérations
  <operation> des opérations
  <binding> les modes de transmissions des messages
  <service> la localisation des services

4. WSDL

Types

Définis sous forme de XML schema

```
<wsdl:types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="PersonneType">
      <xsd:complexType>
        <xsd:element name="nom" type="xsd:string" />
        <xsd:element name="age" type="xsd:int" />
      </xsd:complexType>
  </xsd:schema>

  ...
</wsdl:types>
```


4. WSDL

Message

- défini un profil de paramètres
- un message est composé de `part`
  - élément de type simple
  - référence un `types` précédemment défini

```
<wsdl:message name="AddPersonneRequest">
  <wsdl:part name="nom" type="xsd:string" />
  <wsdl:part name="age" type="xsd:int" />
</wsdl:message>

<wsdl:message name="AddPersonneResponse">
</wsdl:message>

<wsdl:message name="RemovePersonneRequest">
  <wsdl:part name="nom" element="PersonneType" />
</wsdl:message>
```


4. WSDL

Port

- regroupement d'opérations
- a.k.a. interface

```
<wsdl:portType name="PersonPortType">
  <wsdl:operation name="addPersonne">
  ...
  </wsdl:operation>
  <wsdl:operation name="removePersonne">
  ...
  </wsdl:operation>
</wsdl:message>
```

4. WSDL

Opération

• un **message** invocable (`input`)
• et/ou un **message** émis (`output`) ou un **message** d'erreur (`fault`)

• input
  - one-way 1 seul message en input
  - request-response 1 message en input + 1 en output
• output
  - solicit-response 1 seul message en output + 1 en input
  - notification 1 message en output

```
<wsdl:operation name="addPersonne">
  <wsdl:input message="AddPersonneRequest" />
  <wsdl:output message="AddPersonneResponse" />
  <wsdl:fault message="AddPersonneFault" />
</wsdl:operation>
```


4. WSDL

Binding

• spécifie la liaison entre un port et un protocole
• précise le
  - type de transport (ex. HTTP)
  - la façon dont sont créés les messages (`style`)

```
<wsdl:binding name="PersonneBinding" type="PersonPortType">
  <soap:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="rpc" />
  ...
</wsdl:binding>
```


4. WSDL

Binding

• pour chaque opération l'URI associée (voir en-tête HTTP SOAPAction)
• pour chaque message le style d'encodage

```
<wsdl:operation name="addPersonne">
  <soap:operation soapAction="http://some.host/SOAPServer/conv" />
  <wsdl:input>
    <soap:body
      use="encoded"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding" />
  </wsdl:input>
  <wsdl:output>
    ...
  </wsdl:output>
  <wsdl:fault>
    ...
  </wsdl:fault>
</wsdl:operation>
```


4. WSDL

Service

• un ensemble de liaisons (i.e. de ports)
• pour chaque liaison, sa localisation

```
<wsdl:service name="ServicePersonne">
  <wsdl:port binding="PersonneBinding">
    <soap:location="http://localhost:8080/soap/servlet/rpcrouter" />
  </wsdl:port>
  ...
</wsdl:service>
```

4. WSDL

Conclusion WSDL

- langage de définition d'interfaces de services web
- type ⊂ message ⊂ operation ⊂ port ⊂ liaison ⊂ service
- fourni le lien entre ces éléments et SOAP

- XML !!
- génération automatique de WSDL à partir de Java, EJB, ...

- invocation dynamique d'un service à partir de sa réprésentation WSDL
  voir `http://www.alphaworks.ibm.com/tech/wsif/`