‘WS : Web Services’ Services sur la toile

Gérard Florin
Laboratoire CEDRIC
CNAM Paris

Plan de l'exposé

Introduction

I SOAP ' Simple Object Access Protocol'

II XML schéma représentation des données.

III WSDL 'Web Services Definition language'

IV UDDI 'Universal Description Discovery and Integration'.

Conclusion

Introduction

Définition (très large)

- ***Le besoin :*** *des communications de programme à programme utilisant comme support la toile mondiale.*
- ***Service toile****: une application informatique quelconque (plutôt de gestion et même de commerce électronique).*
  - *modulaire,*
  - *basée sur la toile,*
  - *utilisant des services et protocoles standards.*
- ***Fournisseur de services*** *sur la toile: (ASP 'Application Service Providers').*

Exemples d'applications visées

- Commerce électronique : Simulation de crédits
- Commerce électronique : Paiement sécurisé
- Commerce électronique : Calcul de frais de port
- Commerce électronique : Suivi de commandes
- Offre de services : Conversion de devises
- Offre de services : Tout service comportant un accès à des données (température, trafic, ...) ou une algorithmique plus ou moins complexe.

Organisation actuelle des standards

UDDI	Ws-Security	WS-Transaction	Bpel4WS, ..
Services d'annuaire	Services de sécurité	Services de transaction	Services de synchronisation

SOAP, WSDL :	Interactions de communication
HTTP, SMTP, MOM(JMS) :	Transmission effective

SOAP ('Simple Object Access Protocol')

- *Le protocole de **communication** de base (définissant la principale interaction de communication).*
- *Une approche de mode **message** mais aussi un protocole d'appel de procédure distante (**RPC**).*
- *Utilise **XML** pour représenter les données.*
- *Utilise des protocoles applicatifs Internet pour acheminer ses messages:*
  - ***HTTP** (pour sa généralité, son mode synchrone)*
  - ***SMTP** , **MOM** (plutôt pour le mode asynchrone).*

WSDL
(‘Web Services Definition Language’)

- *Un standard de **description d’interfaces** (analogue de IDL).*
- *Permet de décharger les utilisateurs des **détails techniques** de réalisation d’un appel.*
- *Basé sur **XML, XML Schéma**.*
- *Règles de **sérialisation des données** échangées.*

UDDI ('Universal Description, Discovery, and Integration')

- *Le standard **d'annuaire réparti** pour la description des services Web.*
- *Très orienté **affaires** (ventes, prestations)*
- ***Accessible au moyen de Soap**.*
- *Permet d enregistrer des informations variées via la notion de **tmodel** (modèle technique).*

Organismes de standardisation

- **Consortiums / organismes**
  - W3C World Wide Web Consortium
  - Oasis
  - WS-I Web Services - Interoperability
  - Nations Unies (UN-CEFACT)
  - EbXML
  - Rosetta net : Web services en électronique
- **Editeurs de logiciels**
  - Microsoft, IBM, BEA
- **Logiciel libre**
  - Apache Axis

Exemples de produits

- **Microsoft** .NET
- **IBM** Web Services Architecture (Websphere)
- **SUN** One
- **BEA** Web Services Architecture (Weblogic)
- **Iona**, **CapeClear**, **SilverStream**, **Systinet**

- Logiciel libre **Apache** Axis (WSIF ‘ Web Services Invocation Framework ’)

Chapitre I

SOAP 'Simple Object Access Protocol'

Introduction : Communications en objets répartis pour le Web

- *Besoin de **communication** des applications WEB: **unification en RPC/Objets**.*
- *Les outils existants (CORBA, COM+, RMI, ..).*
- *Problèmes des outils existants => nouveaux outils.*

Quelques problèmes des outils existants

- ***Problème 1:*** *Les outils existants (approches objets répartis) ne sont pas* ***interopérables****.*

- ***Problème 2:*** *Les outils existants présentent des difficultés relatives à la* ***sécurité*** *(murs anti feux).*

- ***Problème 3****: Les outils existants sont* ***complexes****.*

Base de départ : utiliser HTTP en objets répartis

- **Encapsulation :** de données échangées dans des protocoles d'applications WEB.
- **Interopérabilité:** HTTP est le protocole **le plus utilisé** (supporté par tous les serveurs et navigateurs WEB).
- **Acheminement: HTTP** est toujours transmis par **les murs anti feux**.

Première possibilité: Utiliser HTTP pour les systèmes d'objets

- *Faire passer les messages des systèmes d'objets répartis (exemple CORBA IIOP) dans un tunnel HTTP.*
- *Solution au problème d'acheminement: déjà essayée.*
- *Solution qui ne* ***résout pas les problèmes d'interopérabilité*** *entre systèmes d'objets répartis* **CORBA, DCOM, RMI***.*

Seconde possibilité: Améliorer HTTP pour en faire un RPC

- *Idée relative à HTTP : **HTTP peut être adapté** pour devenir un protocole de **RPC** **(**HTTP est un protocole client serveur qui présente certaines caractéristiques d'un RPC**)***
- *Idée apparue avec XML : **utiliser XML** comme format de présentation des données **dans les messages**.*

Historique des RPC avec XML

Première génération

XML-RPC.

Seconde génération

SOAP ('Simple Object Access Protocol').

Proposition IBM, Microsoft, Ariba.

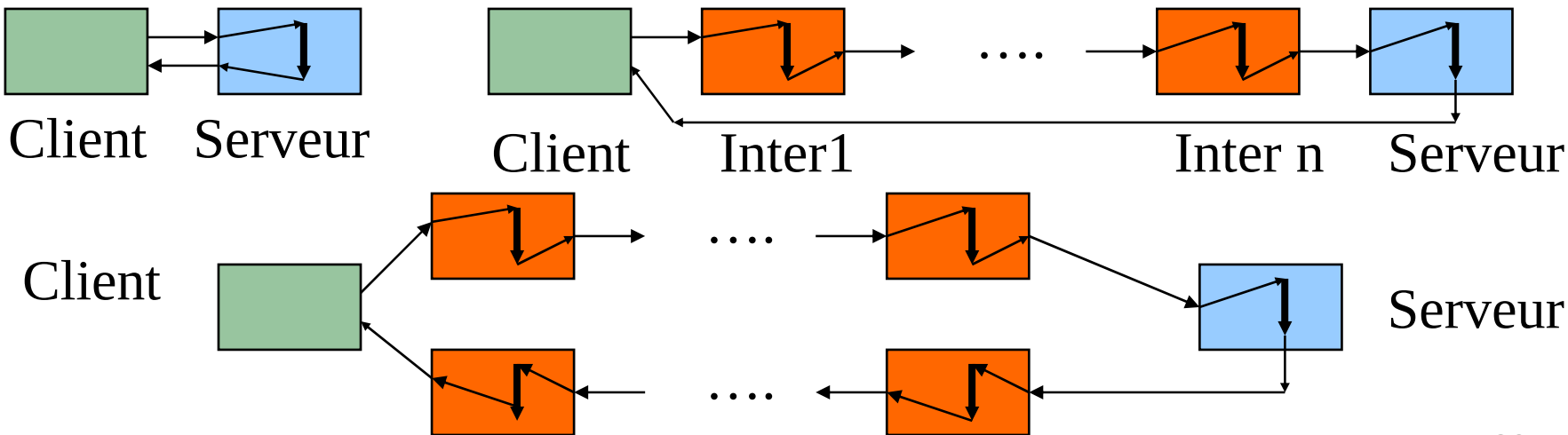
Le protocole SOAP: un protocole client-serveur ‘léger’

- *SOAP définit **un cadre***
  - *pour des interactions à messages et à RPC,*
  - *pour structurer des messages codés en XML ,*
  - *pour utiliser des protocoles applicatifs de l’Internet,*
  - *pour faire de la désignation (méthodes, adresses).*
- *SOAP ne définit **pas de modèle applicatif***
  - *pas de modèle de programmation,*
  - *pas de modèle d'objets,*
  - *pas de notion de cycle de vie (déploiement, …*
  - *pas de modèle de pannes (pour le RPC)*

Soap: les deux modes

- *SOAP : **un mode de communication par message asynchrone avec des continuations***

- *SOAP : **un mode appel de procédure distante** (un mode requête réponse à deux messages sous produit du mode message précédent).*



Un premier exemple simple: Soap en mode RPC sur HTTP

```
POST /chemin/serveurcommercialsoap HTTP/1.1
Host: www.portailcommercial.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 352
SOAPAction: "/URI/code/a/exécuter/pour/la/requête"

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:ObtenirPrix xmlns:m="/URI/définition/des/données/messages">
      <IdentProduit>vvvvvv</IdentProduit >
</m:ObtenirPrix >
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemple de base
La réponse

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 305

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:ObtenirPrixResponse xmlns:m="/URI/définition/des/données/messages">
    <Prix>24.0</Prix>
</m:ObtenirPrixResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Soap: trois chapitres principaux

- ***1) Enveloppe : définir la structure des messages****, qui les traite, de façon optionnelle ou obligatoire.*

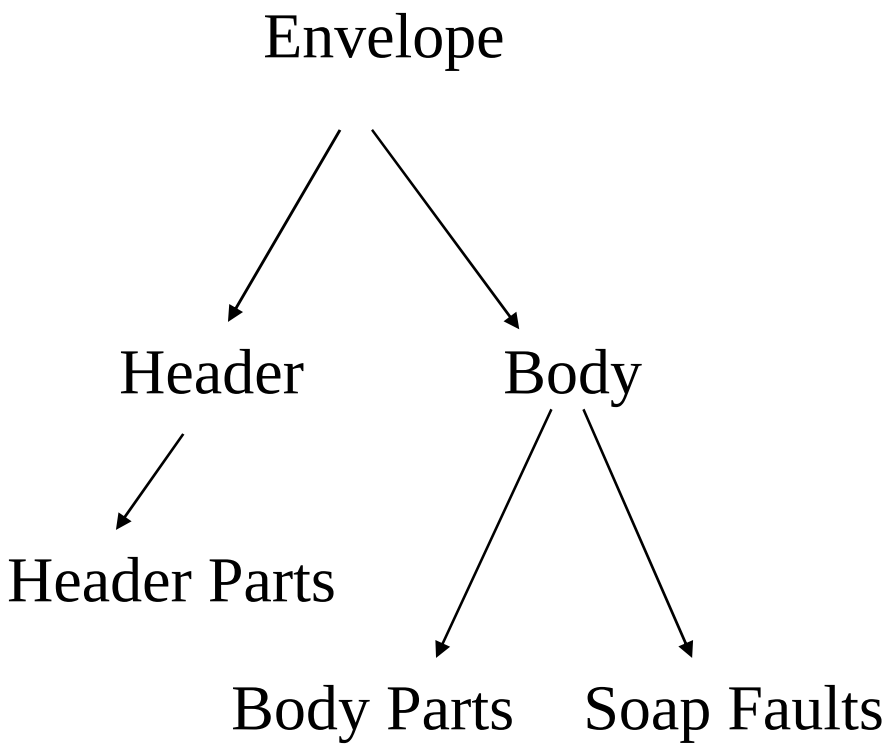
- ***2) Règles d'encodage :*** *les règles* ***pour coder les instances de types de données*** *échangées par les programmes (syntaxe de transfert).*

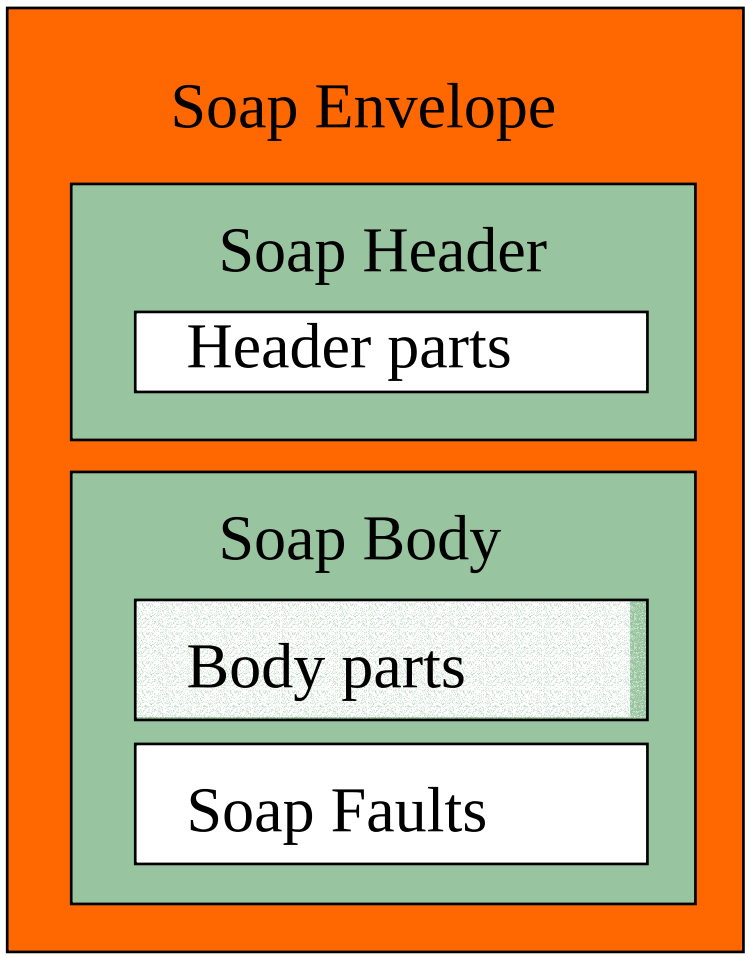
- ***3) Représentation du RPC :*** *La convention pour* ***représenter les appels de procédures distantes*** *(requête réponse) dans le cadre d'un protocole en mode message. La norme montre l'utilisation de HTTP (notion de « binding » HTTP).*

1 La structure des messages L'enveloppe SOAP

- *C'est l'élément racine d'un document XML définissant le contenu d'un message.*
- *La structure de l'enveloppe est spécifiée par "http://schemas.xmlsoap.org/soap/envelope/".*
- *Pas de gestion des versions successives: référence à une URI définissant le format utilisé.*

Eléments principaux d'une enveloppe SOAP





Exemple d'enveloppe avec entête (' header ')

```
<soapenv:Envelope
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
    <authentication SOAP-ENV:Soap-Actor= " URI serveur authentification" >
            <userIdentifier>Nom d'utilisateur</userIdentifier>
            <credential>Valeur de l'authentifiant</credential>
    </authentication>
</soapenv:Header>
<soapenv:Body>
     <EnvoiFacture>
            <DonneeFacture>   xxxxx  </DonneeFacture >
       </EnvoiFacture>
</soapenv:Body>
</soapenv:Envelope>
```

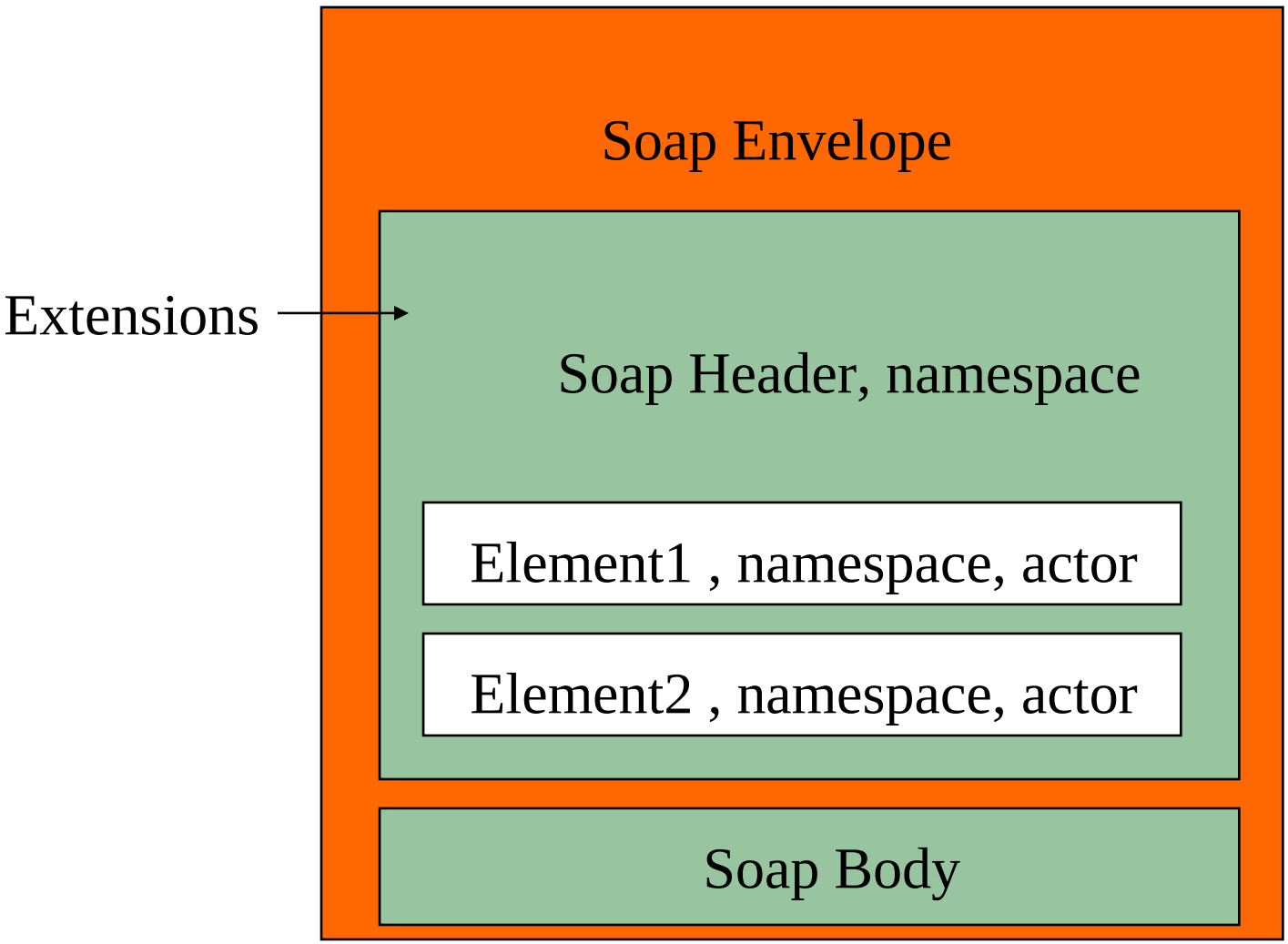
Types d'enveloppes

- **Catégories de messages échangés:**
  - **Document**: Le corps du message Soap peut contenir n'importe quel document XML.
  - **Rpc**: Le corps du document contient obligatoirement le nom de la méthode invoquée et les données correspondent aux arguments de cette méthode.
- **Nature de la sérialisation des données:**
  - **Encoded:** Règles particulières d'encodage définies par SOAP.
  - **Literal:** Pas de règles particulières d'encodage (la donnée échangée est considérée comme un document XML).
- **Deux exemples typiques d'association:**
  - **Rpc/Encoded Document/Literal**

Soap en mode document/literal

```
<soapenv:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body><facture xmlns="http://schemas.entreprise.com/facture">
    <numeroFacture>12-ABC-45</numeroFacture>
    <dateFacture>2001-06-12</dateFacture>
    <sousTotal>300.00</sousTotal>
    <tva>62.50</tva>
    <total>362.50</total>
    <delai>30</delai>
    <contact>NNNNN YYYYY</contact>
    <itemsfacture><item>
        <nomenclature>AK47</nomenclature>
        <quantite>5</quantite><prix>60</prix>
        <total>300</total>
    </item></ itemsfacture >
</facture></soapenv:Body></soapenv:Envelope>
```

Entête Soap ‘SOAP Header’



Approfondissement: Entête Soap 'SOAP Header'

- *Le premier fils de l'élément enveloppe est l'élément Header*
- *Chaque fils de Header décrit une entrée d'entête qui a:*
  - *Un attribut de qualification par un espace de nommage obligatoire*
  - *Un attribut de définition de codage optionnel (encodingstyle).*
  - *Un attribut mustunderstand : traitement obligatoire 1 ou optionnel 0*
  - *Un attribut Soap-Actor : URI du code de traitement.*

```
<SOAP-ENV:Header>
  <t:Transaction xmlns:t="URI"  SOAP-ENV:mustUnderstand="1" SOAP-
   ENV:Soap-Actor= "URI gestionnaire transaction" >
        hierarchical
  </t:Transaction>
</SOAP-ENV:Header>
```

2 Syntaxe de transfert: règles d'encodage des paramètres

- **Définition des règles de codage et de sérialisation** utilisées pour le transport des paramètres.
- Un **ensemble de règles est** défini par SOAP par l'URI: *http://schemas.xmlsoap.org/soap/encoding.* C'est un espace de nommage à utiliser pour la partie encodage).
- **Ces règles d'encodage ne sont pas obligatoires** : on peut créer ses propres règles de codage.

- Pour des raisons **d'efficacité** (grande influence sur les performances).
- Pour **réutiliser d'autres modes** Exemple : **XMI** 'XML Metadata Interchange' ou même ASN1/BER.

3 Le RPC Soap

- *Les ingrédients nécessaires pour réaliser un RPC:*
  - *L'URI de l'objet cible.*
  - *Le nom de la méthode à exécuter*
  - *La signature de la méthode (optionnel).*
  - *Les paramètres de la méthode.*
  - *Les informations d'entête (optionnelles).*

Règles de définition des structures de données

- ***Invocation de méthode*** *=> type article (un élément XML) nommé et typé comme la méthode.*
- ***Les paramètres [in] ou [in/out]*** *=> un nom et un type identique au nom et au type du paramètre dans la signature de la méthode (un sous élément XML).*
- ***Transmis dans le même ordre*** *que dans la signature.*
- ***Les paramètres [out] ou [in/out]*** *=> modélisés de la même façon dans la réponse (le nom de la structure de donnée réponse =nom de la méthode suivi de "Response").*
- ***Les cas d'erreurs*** *=> éléments SOAP Fault.*

4 Soap avec HTTP

POST */comptabilite/service_facturation/serveur.asp HTTP/1.1*
SOAPAction: /comptabilite/service_facturation/facture.exe
Content-Type: text/xml
Accept-Language: en-us
Content-Length: xxxxx
*Accept: */**
User-Agent: Mozilla/4.0
Host: localhost
Connection: Keep-Alive

<soapenv:Envelope>
xxxxx
</soapenv:Envelope>

Erreurs Soap

- *Les erreurs sont définies dans l'élément Fault.*

- *Sous Élément Description*

<faultcode> Code identifiant un erreur.

<faultstring> Message en clair décrivant l'erreur.

<faultactor> Acteur ayant causé l'erreur.

<detail> Détails complémentaires spécifiques.

Les codes d'erreurs

- **VersionMismatch**

  L'espace de nommage pour l'enveloppe est invalide.

- **MustUnderstand**

  Un élément mustUnderstand = "1" n'a pas été compris.

- **Client**

  Le message est mal construit, les données sont incorrectes.

- **Server**

  Le serveur n'a pas pu traiter le message.

Exemple de message d'erreur

```
HTTP/1.1 500 Internal Server Error
Content-Type texte/xml; charset="utf-8"
Content-Length: nnnnn

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body> <SOAP-ENV:Fault>
        <faultcode> SOAP-ENV:Server </faultcode>
        <faultstring> Erreur du serveur </faultstring>
        <detail> <e:DetailsFautesServeurs xmlns:e= "une URI xxx">
                        <message> Connexion impossible </message>
                        <erreurcode> 3 </erreurcode>
        <e:DetailsFautesServeurs></detail>
    </SOAP-ENV:Fault> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

Exemple récapitulatif: la requête

```
POST /ibm-soap/rpcrouter.jsp HTTP/1.1
Host: localhost
Content-Type: text/xml; charset="utf-8"
Content-Length: 476
SOAPAction: "http://www.psol.com/soap/reverse"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<ns1:reverse xmlns:ns1="http://www.psol.com/soap/reverse"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
soft
</ns1:reverse> </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemple: la réponse

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"      Content-Length: 508
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<ns1:reverseResponse xmlns:ns1="http://www.psol.com/soap/reverse"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    tfos
</ns1:reverseResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemple: le client

```
import java.net.*; import java.util.*; import com.ibm.soap.*; import com.ibm.cs.xml.*;
import com.ibm.soap.rpc.*; import com.ibm.soap.encoding.*;
import com.ibm.soap.encoding.soapenc.*;
public class DoReverse
{ public final static void main(String args[])
   throws MalformedURLException, SOAPException
   { Call call = new Call();
      call.setTargetObjectURI("http://www.psol.com/soap/reverse");
      call.setMethodName("reverse");
      call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
      Vector params = new Vector();
      params.addElement(new Parameter("st",String.class, args[0],null));
      call.setParams(params);
```

Exemple: fin du client et serveur distant

```
URL url = new URL("http://localhost:8080/ibm-soap/rpcrouter.jsp");
      Response resp =  call.invoke(url,"http://www.psol.com/soap/reverse");
       if (!resp.generatedFault()) { Parameter ret = resp.getReturnValue();
            System.out.println(ret.getValue()); }
       else { Fault fault = resp.getFault();
            System.err.println(fault.getFaultCode() + " / " +
   fault.getFaultString());
 } } }
```

Le serveur distant

```
public class Reverse
{public String reverse(String st)
    { return new StringBuffer(st).reverse().toString(); }
 }
```

Conclusion Soap

- ***Avantages***
  - *Communication en univers **hétérogène.***
  - *Assez léger **simple** et facile à déployer**.***
  - ***Extensible**.*
  - ***Ouvert.***
- ***Inconvénients***
  - ***Peu de fonctionnalités** offertes pour pouvoir concurrencer la richesse des solutions existantes.*
  - ***Problèmes de performances.***

Bibliographie Soap

- *W3Schools.com A SOAP tutorial for beginners.*
- *Another small SOAP tutorial Benoît Marchal/Gamelan java journal*
- *Beginners introduction to SOAP : http://www.4guysfromrolla.com/webtech/070300-2.shtml*
- *SOAP primer Michael Deem*
- *Simple Object Access Protocol (SOAP) 1.1 W3C Note 08 May 2000 http://www.w3.org/TR/2000/NOTE-SOAP-20000508*
- *Yasser Shohoud , Chapter 3 SOAP: Invoking Web Services http://www.devxpert.com/shohoudy*

Chapitre II

Le typage XML avec ‘XML schéma’

Plan de l'exposé

- Introduction
- Les structures
- Les types de données
- Conclusion

Le typage avec les DTD

- XML est un outil d'avenir pour l'échange, le stockage et l'affichage des données sur Internet.
- Un problème majeur de XML 1.0 (dans sa définition de base) concerne son approche du typage => DTD 'Document Type Definition'.
- L'objectif n'est pas atteint pour des applications informatiques: la définition des types par les DTD est trop orientée documents textuels.

Problèmes des DTD

- 1) Les DTD offrent des possibilités de typage des données très limitées.
- 2) Les DTD ne sont pas définies comme des documents XML: deux syntaxes différentes.
- 3) Les DTD ne sont pas extensibles.
- 4) Les DTD ne prennent pas en compte les espaces de nom (‘namespaces’).
- 5) Les DTD sont (un peu) difficiles à lire.

Nombreuses propositions pour améliorer le typage XML

- **DCD** Document Content Description for XML (DCD), http://www.w3.org/TR/NOTE-dcd
- **DDML** Document Definition Markup Language. http://www.w3.org/TR/NOTE-ddml
- **SOX**-2 Schema for Object-oriented XML, Version 2.0, http://www.w3.org/TR/NOTE-SOX/
- **XDR** XML-Data Reduced,

http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm

- **XML-Data** http://www.w3.org/TR/1998/NOTE-XML-data-0105/
- **XSchema** XSchema Specification. http://www.simonstl.com/xschema/spec/xscspecv4.htm .
- **DSD** (Document Structure Description),
- **TREX** (Tree Regular Expression for XML),

Processus de définition des schémas XML

- Décision complexe au sein du consortium WEB (W3C) pour arbitrer entre les différentes propositions.
- Unification autour d'un projet de recommandation, adopté définitivement en mai 2001: trois documents
- XML Schema Primer
- XML Schema partie 1: Structures (pour décrire la structure et contraindre le contenu de documents XML).
- XML Schema partie 2: Datatypes (ensemble de types prédéfinis et techniques de construction de types).
- Existence de critiques => toujours de nouvelles propositions.

Objectifs généraux des schémas

- Besoin de pouvoir définir précisément la structure, les types possibles des données d'un document.
- XML schéma est l'analogue:
  - D'un langage de définition de données DDL ('Data Definition Language') des bases de données et un document est l'analogue d'un article d 'une base de données.
  - D'un langage de syntaxe abstraite des réseaux comme ASN1 (' Abstract Syntax Notation') et un document est l'analogue du contenu en syntaxe de transfert (BER ' Basic Encoding Rules').
  - D'un langage évolué dans sa partie définition de données comme C++ ou Java. Un schéma définit l'analogue d 'une classe et un document est une instance de cette classe.
- Un document XML doit pouvoir être validé relativement à son schéma.

Un schéma : une syntaxe abstraite

- Un document instance d’un schéma XML ne possède aucune méthode.
- L'interpréteur d’un document instance d'un certain schéma définit la sémantique du langage associé au schéma.

Objectifs précis (1)
- Structures des schémas -

- 1. Définir la structure et les contenus des documents.
- 2. Définir des relations d'héritage.
- 3. Définir des URI qui référencent les standards de sémantique des constructions.
- 4. Définir une documentation interne.
- 5. Définir des descriptions et des contraintes spécifiques des applications.
- 6. Supporter l'évolution des schémas.
- 7. Intégrer aux définitions structurelles des types prédéfinis.

Objectifs précis (2)
- Typage des données -

- 1. Fournir un ensemble de types primitifs.
- 2. Définir un système de typage suffisamment riche pour importer/exporter des données d'une base de données.
- 3. Distinguer les aspects reliés à la représentation lexicale des données de ceux gouvernant les données.
- 4. Permettre de créer des types de données usagers dérivés de types existants en contraignant certaines propriétés (domaine, précision, longueur, format).

Objectifs précis (3)
- Vérification de conformité -

- 1. Décrire les responsabilités des processeurs de conformité.
- 2. Définir les relations entre schémas et documents.
- 3. Définir les relations entre la validité au sens des schémas et la validité XML.
- 4. Définir les relations entre schémas et DTD
- 5. Définir les relations entre les schémas, les espaces de nommage et la validité.
- 6. Définir le schéma xml des schémas xml.

Domaines d'utilisation

- 1. Publication d'informations sur le WEB.
- 2. Commerce électronique.
- 3. Gestion de documents traditionnels.
- 4. Assistance à la formulation et à l'optimisation des requêtes en bases de données.
- 5. Transfert de données entre applications en réseaux.
- 6. Contrôle de supervision et acquisition de données.
- 7. Échange d'informations de niveau méta.

Les structures

Principes généraux des schémas

- **Un schéma XML est un document XML.**
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <xsd:schema
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <!-- Déclaration de deux types d ’éléments -->
- <xsd:element name="nom" type="xsd:string" />
- <xsd:element name= "prenom" type="xsd:string" />
- </xsd:schema>

Exemple d'adresse postale en XML: le document

```
<?xml version="1.0"?>
<Addresse_postale_France pays="France">
   <nom>Mr Jean Dupont</nom>
   <rue>rue Camille Desmoulins</rue>
   <ville>Paris</ville>
   <departement>Seine</departement>
   <code_postal>75600</code_postal>
</Addresse_postale_france >
```

DTD d'une adresse postale

```
<!DOCTYPE une_DTD_adresse
[   <!ELEMENT Addresse_postale_france
          (nom, rue, ville, département, code_postal)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT département (#PCDATA)>
<!ELEMENT code_postal (#PCDATA)>
<!ATTLIST Addresse_postale_france
          pays  NMTOKEN #FIXED 'France' > ]>
```

Typage d'une adresse postale au moyen d'un schéma XML

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:complexType name="Addresse_postale_france" >
        <xsd:sequence>
            <xsd:element name="nom" type="xsd:string" />
            <xsd:element name="rue" type="xsd:string" />
            <xsd:element name="ville" type="xsd:string" />
            <xsd:element name="departement" type="xsd:string" />
            <xsd:element name="code_postal" type="xsd:decimal" />
        </xsd:sequence>
         <xsd:attribute name=  "pays" type="xsd:NMTOKEN"
                use="fixed" value="FR"/>
</xsd:complexType>       </xsd:schema>
```

Les composants primaires

- Un schéma XML est construit par assemblage de différents composants (13 sortes de composants rassemblés en différentes catégories).
- Composants de définition de types
  - Définition de types simples (Simple type).
  - Définition de types complexes (Complex type).
- Composants de déclaration
  - Déclaration d'éléments.
  - Déclaration d'attributs.

Les composants secondaires

- Composants secondaires
  - Définitions de modèles de groupes d'éléments.
  - Définition de groupe d'attributs ('Attribute group').
  - Définition de contraintes d'identifiants ('Identity-constraint').
  - Déclaration de notations.

- Les composants secondaires doivent être nommés.

Les composants d'aides

- **Les composants d'aides "helpers"** définissent des petites parties d'autres composants qui dépendent du contexte.

  Annotations ('Annotations') : Ex Commentaires

  Modèles de groupes ('Model groups') : Ex sequence

  Wildcards ('Wildcards') : Apparier sur la base d'un nom

  Particules ('Particles') : Règles de grammaire pour les contenus d'éléments

  Usage des attributs ('Attribute Uses') : Contenu des attributs (optionnel, obligatoire ...)

Déclaration des éléments

- Un élément XML est déclaré par **la balise 'element'** de XML schéma qui a de nombreux attributs.
- **Les deux principaux attributs** sont:

  - **name** : Le nom de l'élément (de la balise associée).
  - **type** : Le type qui peut être simple ou complexe.

Exemple de base

```
<xsd:element name="code_postal" type="xsd:decimal"/>
```

Autres attributs d'un élément

- **targetNamespace** : L'élément déclaré l'est au titre d'un espace de nommage spécifié par une URI. (optionnel).
- **scope** : Définition d'une portée: soit globale soit locale celle d'une définition d'un type complexe.
- **value** : Contrainte de valeur définie par un couple d'une chaîne et de 'default ' ou 'fixed' (optionnel).
- **annotation** : Une annotation cad un commentaire (optionnel).
- .....

Déclaration des attributs

Un attribut est une valeur nommée et typée associée à un élément.

Le type d'un attribut défini en XML schéma est obligatoirement simple.

```
<xsd:complexType name="TypeRapport">
   <xsd:attribute name= "Date_creation"
    type="xsd:date"/>
      ……
</xsd:complexType>
<xsd:element name="Rapport" type="TypeRapport"/>
```

Autres attributs

- L'élément attribute de XML Schema peut avoir deux attributs optionnels : **use** et **value**.
- On peut ainsi définir des contraintes de présence et de valeur.
- Selon ces deux attributs, la valeur peut:

  être obligatoire ou non

  être définie ou non par défaut.
- Exemple: <xsd:attribute name= "Date_peremption" type="xsd:date" use="default" value= "2005-12-31"/>

Valeurs possibles pour use

- Use = **required** : L'attribut doit apparaître et prendre la valeur fixée si elle est définie.
- Use= **prohibited** : L'attribut ne doit pas apparaître.
- Use = **optional** : L'attribut peut apparaître et prendre une valeur quelconque.
- Use= **default** : Si l'attribut à une valeur définie il la prend sinon il prend la valeur par défaut.
- Use= **fixed** : La valeur de l'attribut est obligatoirement la valeur définie.
- **Exemple :** <xsd:attribute name= "Date_creation" type="xsd:date" use="required"/>

Types simples, types complexes

SimpleType et ComplexType permettent de définir de nouveaux types.

- **‘SimpleType’** permet de définir des éléments ou des attributs non structurés (dérivés d’une chaîne, d’un entier etc)
- **‘ComplexType’** définit des éléments structurés (qui comportent des éléments fils ou des attributs).

Types simples (1)

- **Types simples prédéfinis** au sens de la norme XML Schémas ‘ datatypes’: string, integer, boolean ...

```
<xsd:element name="code_postal " type="xsd:integer"/>
```

- **Types simples définis par dérivation** d'un autre type simple, au moyen de l’élément <xsd:simpleType ...>
- Exemple 1 de dérivation par restriction.

```
<xsd:simpleType name=  "DeuxDecimales">
        <xsd:restriction base="xsd:decimal">
                <xsd:fractionDigits value="2" />
        </xsd:restriction>
</xsd:simpleType>
```

Types simples (2)

- Exemple 2 de dérivation par restriction (énumération de valeurs)

```
<xsd:simpleType name="EuroDollar">
    <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Dollar" />
           <xsd:enumeration value="Euro" />
    </xsd:restriction>
</xsd:simpleType>
```

Types complexes

- Déclarés au moyen de l'élément <xsd:complexType name="... Ils peuvent contenir d'autres éléments, des attributs, etc.

```
<xsd:complexType name= "TypePrix">
        <xsd:simpleContent>
            <xsd:extension base="DeuxDecimales">
                <xsd:attribute name="Unite" type= "FrancEuro" />
            </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

- Trois façons de composer des éléments dans un type complexe: sequence, choice, all.

Types complexes: Sequence

- Un type **sequence** est défini par une suite de sous-éléments qui doivent être présents dans l'ordre donné.
- Le nombre d'**occurences** de chaque sous-élément est défini par les attributs minOccurs et maxOccurs.

```
<xsd:complexType name= "Commande">
    <xsd:sequence>
        <xsd:element name= "Ad_livraison" type="Addresse"/>
        <xsd:element name= "Ad_facturation" type="Addresse"/>
       <xsd:element name=  "texte" type="xsd:string" minOccurs="1" />
        <xsd:element name="items" type="Items" maxOccurs= "30" />
     </xsd:sequence>
</xsd:complexType
```

Types complexes: Choice

- Un seul des éléments listés doit être présent.
- Le nombre d'occurences possible est déterminé par les attributs minOccurs et maxOccurs de l'élément.

```
<xsd:complexType name=  "type_temps">
   <xsd:choice >
    <xsd:element name= "Noire"  type="Note" minOccurs="1"
    maxOccurs="1" />
    <xsd:element name= "Croche"  type="Note" minOccurs="2"
    maxOccurs="2" />
   </xsd:choice>
</xsd:complexType>
```

Types complexes: All

- C’est une composition de type ensembliste. Dans un document conforme, les éléments listés doivent être tous présents au plus une fois. Il peuvent apparaître dans n'importe quel ordre.

```
<xsd:complexType name= "Commande">
   <xsd:all>
      <xsd:element name= "Ad_livraison" type="Addresse"/>
      <xsd:element name= "Ad_facturation" type="Addresse"/>
      <xsd:element name= "texte" type="xsd:string" minOccurs="0" />
      <xsd:element name="items"  type="Items" maxOccurs= "30" />
   </xsd:all>
</xsd:complexType>
```

Types nommés

Un type nommé possède un nom et une définition relative à ce nom (approche classique).

Un élément (ou un attribut) utilise le type au moyen du nom.

```
<xsd:complexType name=”personne">
    <xsd:sequence>
        <xsd:element name=”nom" .../>
        <xsd:element name=”prenom" .../>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name=”auteur" type=”personne"/>
```

Types anonymes d'éléments

- **Une définition de type reste locale à un élément et n'est pas besoin d'être nommée.**
- **Elle n'est pas réutilisable.**

```
<xsd:element name=”auteur">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name=”nom" .../>
            <xsd:element name=”prenom" .../>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Notion de référence

- **Attribut ref : réutilisation d'élément ou d'attribut.**

```
<xsd:element name="titre" type="xsd:string"/>
<xsd:element name="auteur" type="xsd:string"/>
<xsd:attribute name="datePubli" type="xsd:date"/>
<xsd:element name="book">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="titre"/>
         <xsd:element ref="auteur"/>
      </xsd:sequence>
      <xsd:attribute ref="datePubli"/>
   </xsd:complexType>
</xsd:element>
```

Déclarations globales et locales

- **Déclaration globale**: un fils de schéma
- **Déclaration locale**: dans le cadre d ’un type complexe

```
<xsd:schema    .....>
   <!-- Déclaration d’un élément global-->
   <xsd:element name= "patronyme" type= "xsd:string"/>
   <!-- Déclaration d’un type complexe-->
   <xsd:complexType name= "Personne">
     <xsd:sequence>
       <xsd:element name= "nom" ref="patronyme"/>
       <!-- Déclaration de deux éléments locaux-->
       <xsd:element name="prenom" type="xsd:string"/>
       <xsd:element name="dateDeNaissance" type="xsd:date"/>
     </xsd:sequence>
</xsd:complexType>
```

Domaines nominaux

- **Targetnamespace**: espace de nommage du schéma
- **elemFormDefault**: les éléments sont obligatoirement qualifiés dans les instances valides.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.cnam.fr/schemas/s1"
        xmlns:msh="http://www.cnam.fr/schemas/s1"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetnamespace="http://www.cnam.fr/schemas/s1"
        elemFormdDefault="qualified"
        attributeFormDefault="qualified" >
```

Approfondissements concernant les structures

Groupes d'éléments

On peut définir des groupes *d'éléments.*

```
<xsd:group name="elementPrincipauxDocuments">
  <xsd:sequence>
     <xsd:element name="titre" type="xsd:string"/>
     <xsd:element name="auteur" type="xsd:string"/>
   </xsd:sequence>
</xsd:group>
<xsd:complexType name="livre">
<xsd:sequence>
   <xsd:group ref=" elementPrincipauxDocuments"/>
    ...
</xsd:sequence>
```

Groupes d'attributs

- *Les groupes d'attributs sont des ensembles d'attributs définis pour être réutilisés.*

```
<xsd:complexType name="typedocument">
    <xsd:attributeGroup ref="Infos_Coherence"/>
     <!-- Autres définitions à placer ici -->
</xsd:complexType>
<xsd:attributeGroup name="Infos_Coherence">
       <xsd:attribute name= "Der_maj" type="xsd:date"/>
       <xsd:attribute name= "Redacteur" type="xsd:string"/>
 </xsd:attributeGroup>
```

Annotations

- ***Les annotations*** *sont des éléments définis pour permettre la compréhension des schémas.*
  - ***Documentation****: commentaires pour humains.*
  - ***Appinfo****: commentaires pour machines.*

```
<xsd:simpleType fn:note="special">
 <xsd:annotation>
 <xsd:documentation>A type for experts only</xs:documentation>
 <xsd:appinfo>
 <fn:specialHandling>checkForPrimes</fn:specialHandling>
 </xsd:appinfo>
 </xsd:annotation>
```

Types de contenus d'éléments

- **Contenu complexe** (complexContent) peut être mentionné, mais c'est le contenu par défaut.
- **Contenu vide**.

```
<xsd:complexType name="TypeDistant">
    <xsd:attribute name= "URIdedefinition" type="xsd:anyURI" />
</xsd:complexType>
```

- **Contenu simple** (simpleContent) doit être mentionné pour des éléments qui contiennent des données de type simple mais qui ont des attributs.
- **Contenu mixte** (attribut mixed='true' de complexType ou complexContent). La valeur du contenu est mélangé avec des sous éléments.

Exemple: contenu de type mixte

```
<xsd:element name= "formuleDePolitesse">
<xsd:complexType mixed="true">
  <xsd:sequence>
     <xsd:element name= "destinataire" type="xsd:string"/>
     <xsd:element name= "niveau" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

<formuleDe Politesse>
Recevez <destinataire>Monsieur</destinataire> l'expression de mes sentiments <niveau>les meilleurs</niveau>.
</formuleDePolitesse>

Élément ‘unique’

- Pour définir qu’une valeur d’un champ d’un élément doit être unique.

```
<xsd:complexType name=    "aveclistepersonne">
<xsd:unique name="UniciteSecSoc">
    <xsd:selector xpath= "personne"/>
    <xsd:field xpath=  "numeroSecSoc"/>
</xsd:unique>
<xsd:element name= "personne">
....
```

Élément ‘clé’

- Pour définir qu’une valeur d’un champ d’un élément peut servir de clé.

```
<xsd:complexType name=    "aveclistepersonne">
<xsd:key name="UniciteSecSoc">
    <xsd:selector xpath= "personne"/>
    <xsd:field xpath=  "numeroSecSoc"/>
</xsd:key>
```

Modularité, importation de schéma

- Elément import avec un attribut namespace:

<xsd:schema ..... xlmns:chim
="http://chimie.cnam.fr/schemas/schem1 >

<xsd:import
namespace="http://chimie.cnam.fr/schemas/schem"/>

<xsd:element name='formule'> <complextype>

....

<xsd:element name=' hx' type='chim:ethyl'/>

.....

<complextype> <xsd:element>

Modularité, inclusion de schéma

- *<xsd:include schemaLocation= "uri"/>*
- Objectif: Assembler des documents schémas en un schéma unique.
- Contrainte: Avoir la même cible (targetnamespace).

- *<xsd:redefine schemaLocation= "uri"/>*
- Inclure en modifiant les définitions.

Les types de données XML schéma

Objectifs de la définition des types

- 1. Fournir des types primitifs analogues à ceux qui existent en SQL ou en Java.
- 2. Définir un système de typage suffisamment riche pour importer/exporter des données d'une base de données.
- 3. Distinguer les aspects reliés à la représentation lexicale des données de ceux gouvernant les ensembles de données sous-jacents.
- 4. Permettre de créer des types de données usagers dérivés de types existants en contraignant certaines propriétés (domaine, précision, longueur, format).

Système de typage des schémas

Trois composantes:

a) L'ensemble des valeurs du type ('value space')

Ex: type float.

b) L'ensemble des représentations lexicales possibles des valeurs ('lexical space ').

Ex: "10" ou "1.0E1"

c) **L'ensemble des facettes (l'ensemble des propriétés)** qui définit l'ensemble des valeurs (notion de facette fondamentale et de facette de contrainte).

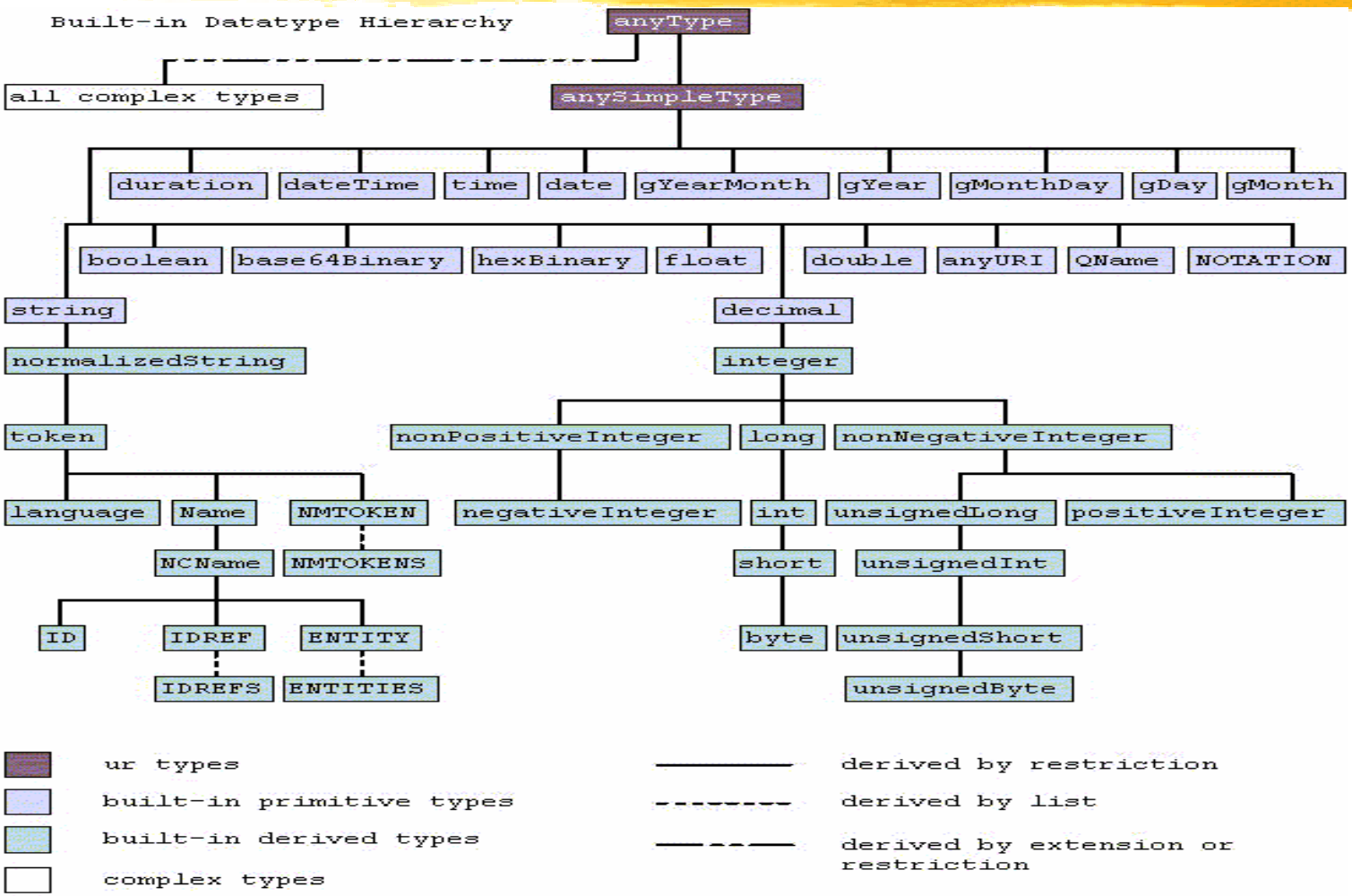
Ex: Le type float est défini par la norme IEEE 754-1985 (c'est un flottant simple précision sur 32-bit).

On peut dériver des types par contraintes.

Définitions relatives aux types

- **Types primitifs** (‘Primitive’) Non défini en référence à d’autres types..
- **Types dérivés** (‘Derived’) Définis par dérivation à partir d’autres types.
- **Types prédéfinis** (‘Built-in’) Définis dans le cadre de la spécification XML Schéma datatypes (primitif ou dérivé).
- **Types usagers** (‘User-derived’) Types construits par les utilisateurs.
- **Types atomiques** (‘Atomic’) Types indivisibles du point de vue de la spécification XML schéma.
- **Types listes** (‘List’) Types dont les valeurs sont des listes de valeurs de types atomiques.
- **Types unions** (‘Union’) Types dont les ensembles de valeur sont la réunion d’ensemble de valeurs d’autres types.

Hiérarchie des types prédéfinis



Quelques types prédéfinis

Type	Forme lexicale
String	**Bonjour**
boolean	{**true**, **false, 1, 0**}
float	**2345E3**
double	**23.456789E3**
decimal	**808.1**
dateTime	**1999-05-31T13:20:00-05:00.**
binary	**0100**
uriReference	**http://www.cnam.fr**
....	

Dérivation de types simples

- *Les quatre formes principales de dérivation d'un type simple sont:*
  - *1) l'extension,*
  - *2) la restriction,*
  - *3) l'union,*
  - *4) la liste.*

1- Dérivation par restriction

- La dérivation par restriction restreint l'ensemble des valeurs d'un type pré existant.
- La restriction est définie par des contraintes de facettes du type de base: valeur min, valeur max ...
- *<xsd:simpleType name= "ChiffresOctaux">*
  <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="0" />
  <xsd:maxInclusive value= 7" />
  </xsd:restriction>
  </xsd:simpleType>

Les contraintes de facettes (1)

- 1) lenght : la longueur d'une donnée.
- 2) minLenght: la longueur minimum.
- 3) maxLenght: la longueur maximum.
- 4) pattern: défini par une expression régulière.
- 5) enumeration: un ensemble discret de valeurs.
- 6) whitespace: contraintes de normalisation des chaînes relativement aux espaces (preserve, replace, collapse).

Les contraintes de facettes (2)

- 1) maxInclusive: une valeur max comprise.
- 2) maxExclusive: une valeur max exclue.
- 3) minInclusive: une valeur min comprise.
- 4) maxInclusive: une valeur min exclue.
- 5) totalDigits: le nombre total de chiffres.
- 6) fractionDigits: le nombre de chiffres dans la partie fractionnaire.

Exemple d'une énumération

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

2 - Dérivation par extension

- Dériver un nouveau type par **extension** consiste à ajouter à un type existant des sous-éléments ou des attributs.
- On obtient inévitablement un type complexe.

```
<xsd:complexType name= "mesure">
  <xsd:simpleContent><xsd:extension base="xsd:Decimal">
      <xsd:attribute name="unite" type="xsd:NMTOKEN"/>
  </xsd:extension></xsd:simpleContent>
</xsd:complexType>
 <xsd:element name= "temperature" type= "mesure"/>
 <temperature unit="Kelvin">230</temperature>
```

3 - Dérivation par union

- Pour créer un nouveau type on effectue l'union ensembliste de toutes les valeurs possibles de différents types existants.

```
<xsd:simpleType name="TransportFormatCaracteres">
<xsd:union memberTypes="base64Binary hexBinary"/>
</xsd:simpleType>
```

Autre exemple d'union

```
<xsd:attribute name= "taille"><xsd:simpleType>
<xsd:union>
<xsd:simpleType><xsd:restriction base="xsd:positiveInteger">
      <xsd:minInclusive value= "35"/>
      <xsd:maxInclusive value= "46"/>
</xsd:restriction></xsd:simpleType>
<xsd:simpleType><xsd:restriction base="xsd:NMTOKEN">
       <xsd:enumeration value= "s"/>
       <xsd:enumeration value="m"/>
        <xsd:enumeration value= "xxl"/>
</xsd:restriction></xsd:simpleType>
</xsd:union></xsd:simpleType></xsd:attribute>
<chemise taille= '46'> Description chemise </chemise>
<maillot taille= 'xxl'> Description maillot </maillot>
```

4 - Dérivation par liste

- Une **liste** permet de définir un nouveau type de sorte qu'une valeur du nouveau type est une liste de valeurs du type pré existant (valeurs séparées par espace).
- <simpleType name= 'DebitsPossibles'>

  <list itemType='nonNegativeInteger'/>

  </simpleType>

  <debitsmodemV90 xsd:type='DebitsPossibles'>

  33600 56000

  </debitsmodemV90>

Conclusion typage avec XML schéma

Un standard très utile en réseaux/systèmes répartis

- Comme outil d'interopérabilité en univers réparti.
  - Entre des applications WEB.
  - Dans des approches objets répartis comme SOAP ('Simple Object Protocol')

  => WSDL ('Web Service Definition Language').
  - Entre des bases de données hétérogènes.

Quelques reproches

- Les performances.
- Les imperfections à découvrir dans les choix de conception du système de typage.

Bibliographie Normes

- ‘XML Schema Part 0: Primer’ W3C
  http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
- ‘XML Schema Part 1: Structures’ W3C
  http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
- ‘XML Schema Part 2: Datatypes’ W3C
  http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
- XML Schema Requirements http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215 (Février 1999)

Bibliographie Documents

- Introduction aux schémas
  http://www.w3schools.com/schema
- Les schémas XML Gregory Chazalon, Joséphine Lemoine
  http://site.voila.fr/xmlschema
- W3C XML Schema, Eric Van Der Vlist,
  http://www.xml.com/pub/a/2000/11/29/schemas/

Chapitre III

WSDL 'Web Services Description Language '

Introduction WSDL

- **Problème:** spécifier un service Web (comment utiliser ce service).
  - Exemple: un site de commerce électronique.
- **Construire** les messages de requête, et analyser les messages de réponse, connaître le mode d'accès réseau.
  - Exemple: Soap sur HTTP en RPC à telle URI pour la cotation d'un produit, d'une valeur boursière.
- **Fonctionner en mode statique et dynamique**
  - Fabriquer des souches
  - Générer des requêtes en exécution.

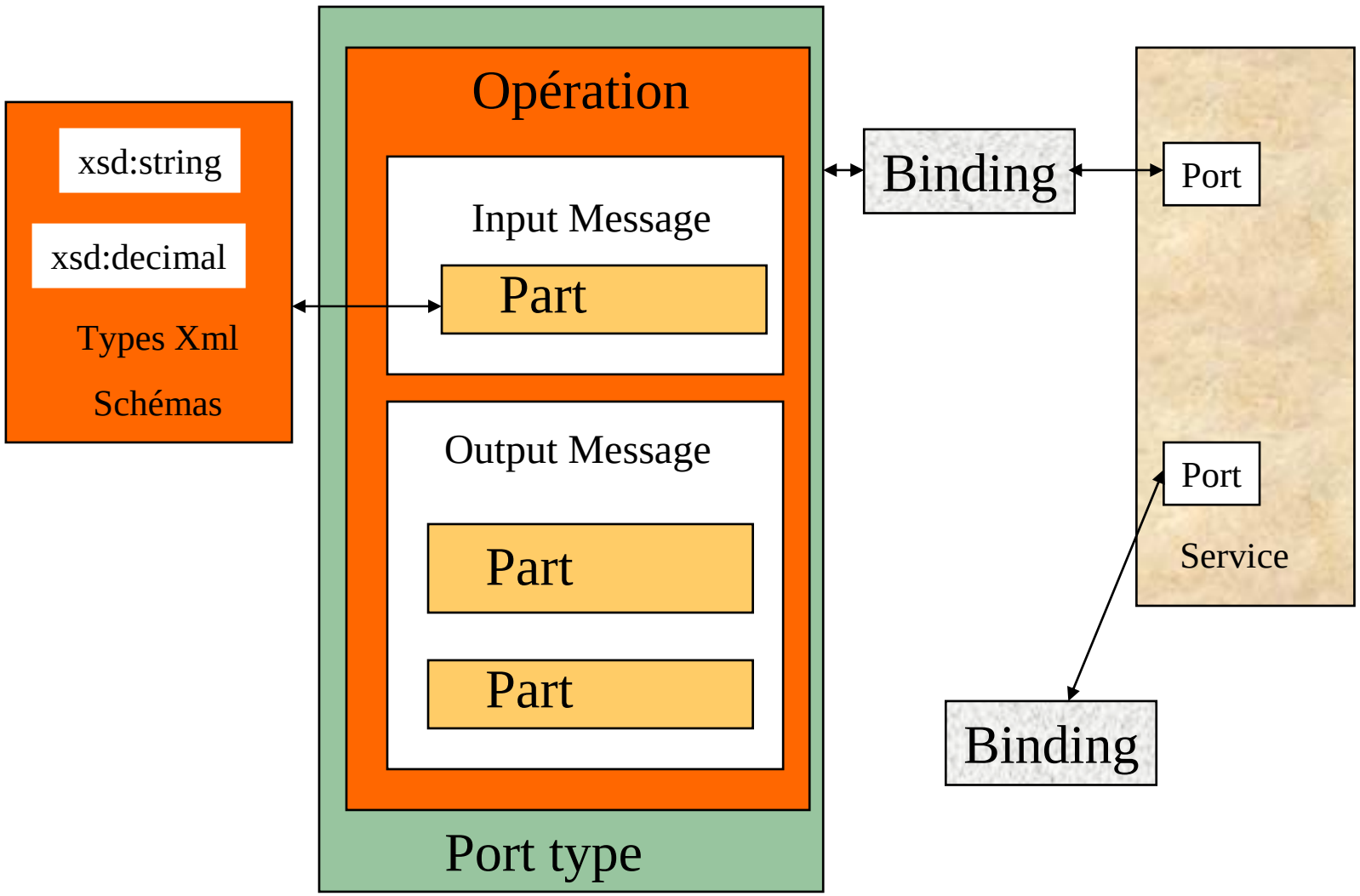
Historique des langages de spécification de services WEB

- WIDL : ‘ Web Interface Definition Language’
- SDL : ‘Service Definition Language’
- SCL : ‘Soap Contract Language’
- NASSL : 'Network Accessible Service Specification Language'

Définition WSDL

- Un dialecte XML pour décrire des services réseaux.
- Offerts par un ensemble de points d'accès ('endpoint').
- Une approche qui se veut générale => basée message.
- Approche classique à deux niveaux:
  - Une vision abstraite : définition d'interface.
  - Une vision concrète : projection 'binding' sur un service de communication (par exemple http) avec des règles de sérialisation.

Architecture WSDL



Liste des principaux concepts WSDL

- **Types:** La déclaration de types de données utiles.
- **Message:** La description abstraite de données échangées.
- **Opération:** La description abstraite d'une action.
- **Type de port ('Port type'):** Un ensemble d'opérations définies abstraitement.
- **Liaison ('Binding):** Un protocole réel et un format de donnée réel pour implanter un type de port.
- **Port:** Un point d'accès de service défini par une adresse réseau et une liaison.
- **Service:** Un ensemble de ports

Structure d'une description (1)

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri"?>
  <wsdl:types>
      <xsd:schema> …. </xsd:schema>
  </wsdl:types>
  <wsdl:message name="nmtoken">
     <part name="nmtoken" element="qname"? type="qname"?/>
  </wsdl:message>
  <wsdl:portType name="nmtoken">
    <wsdl:operation name="nmtoken">
       <wsdl:input name="nmtoken"? message="qname">
       </wsdl:input>
       <wsdl:output name="nmtoken"? message="qname">?
       </wsdl:output>
       <wsdl:fault name="nmtoken" message="qname">
       </wsdl:fault>
    </wsdl:operation>
  </wsdl:portType>
```

Structure d'une description (2)

```
<wsdl:binding name="nmtoken" type="qname">
  <wsdl:operation name="nmtoken">
    <wsdl:input>
    </wsdl:input>
    <wsdl:output>
    </wsdl:output>
    <wsdl:fault name="nmtoken">
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="nmtoken">
  <wsdl:port name="nmtoken" binding="qname">
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

1 L'élément types

```
<types>
<schema targetNamespace='http://www.entreprise.org/types'
xmlns='http://www.w3.org/2001/XMLSchema'
xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'>
<element name="TradePrice">
    <complexType>
        <all> <element name="price" type="float"/> </all>
    </complexType>
</element>
</schema>
</types>
```

2 L'élément message

```
<message name='AdditionRequest'>
  <part name= 'PremierOperande' type='xsd:decimal'/>
  <part name= 'SecondOperande' type='xsd:decimal'/>
</message>
<message name='AdditionResponse'>
  <part name='Resultat' type='xsd:decimal'/>
</message>
```

3 L’élément type de port

```
<portType name= ’ExempleTypePortSoap'>
<operation name='Add'
   parameterOrder=’PremierOperande SecondOperande'>
      <input message='AdditionRequest' />
      <output message='AdditionResponse' />
  </operation>
  <operation name='Sub'
   parameterOrder=’PremierOperande SecondOperande'>
      <input message=’SoustractionRequest' />
      <output message=’SoustractionResponse' />
</operation>
</portType>
```

4 L'élément opération

Il permet de décrire une opération selon quatre modes:

- ***Request-response****: Le point d'accès reçoit un message et retourne un message.*
- ***Solicit-response****: Le point d'accès émet un message et reçoit un message.*
- ***One-way****: Le point d'accès reçoit un message.*
- ***Notification****: Le point d'accès émet un message .*

Opération One-way

Le sous élément input spécifie le format abstrait d'un unique message entrant.

```
<wsdl:operation name="nmtoken">
    <wsdl:input name="nmtoken"? message="qname"/>
</wsdl:operation>
```

Opération Notification

Le sous élément output spécifie le format abstrait d'un unique message sortant.

```
<wsdl:operation name="nmtoken">
    <wsdl:output name="nmtoken"? message="qname"/>
</wsdl:operation>
```

Opération Request-response

Les sous éléments input et output spécifient le format abstrait des messages ‘request’ et ‘response’.

```
<wsdl:operation name="nmtoken" parameterOrder="nmtokens">
    <wsdl:input name="nmtoken"? message="qname"/>
    <wsdl:output name="nmtoken"? message="qname"/>
    <wsdl:fault name="nmtoken" message="qname"/>*
</wsdl:operation>
```

Opération Solicit-response

Les sous éléments input et output spécifient le format abstrait des messages requêtes et réponses.

```
<wsdl:operation
   name="nmtoken" parameterOrder="nmtokens">
      <wsdl:output name="nmtoken"? message="qname"/>
      <wsdl:input name="nmtoken"? message="qname"/>
      <wsdl:fault name="nmtoken" message="qname"/>*
</wsdl:operation>
```

5 La liaison ‘binding’

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
  <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
  <input>
  <soap:body use="literal"
   namespace="http://example.com/stockquote.xsd"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
  <soap:body use="literal"
   namespace="http://example.com/stockquote.xsd"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
  </operation>
</binding>
```

Eléments de la liaison SOAP (1)

<soap:address> *Sous élément de <port>.*

Définit l' URI du point d'accès soap.

<soap:binding> *Sous élément de <binding>.*

Définit le style document ou RPC et le protocole de transport.

<soap:body> *Sous élément de <input> ou <output> dans <binding>.*

Définit l'espace de nommage du corps et son style d'encodage (document ou literal)

<soap:fault> *Sous élément de fault.*

Définit l'élément fault, son nom, son espace de nommage et son style d'encodage.

Eléments de la liaison SOAP (2)

<soap:header> *Sous élément de <input> ou <output> dans <binding>.*

Définit le message qui utilise ce header, son espace de nommage et son style d'encodage.

<soap:headerfault> *Sous élément de <input> <output> dans <binding> .*

Définit le message qui utilise ce headerfault, son espace de nommage et son style d'encodage.

<soap:operation> *Sous élément de operation dans <binding> .*

Définit le code de la méthode à exécuter (la valeur de soapaction) quand on invoque l'opération et le style RPC ou document.

6 L'élément port

```
<wsdl:definitions .... >
   <wsdl:service .... > *
      <wsdl:port name="nmtoken" binding="qname">
         <-- extensibility element (1) -->
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

7 L'élément service

```
<wsdl:definitions .... >
    <wsdl:service name="nmtoken"> *
        <wsdl:port .... />*
        <wsdl:port .... />*
    </wsdl:service>
</wsdl:definitions>
```

Exemple Amazon (1)

- http://soap.amazon.com/schemas2/AmazonWebServices.wsdl
- Début du fichier :

```
<wsdl:definitions xmlns:typens="http://soap.amazon.com"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="http://soap.amazon.com"
   name="AmazonSearch">
```

Exemple Amazon (2)

- **Poursuite du fichier**

```
<wsdl:types>
<xsd:schema xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://soap.amazon.com">
<xsd:complexType name="ProductLineArray">
<xsd:complexContent>
<xsd:restriction base="soapenc:Array">
<xsd:attribute ref="soapenc:arrayType" wsdl:arrayType="typens:ProductLine[]"
    />
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ProductLine">
<xsd:all>
<xsd:element name="Mode" type="xsd:string" minOccurs="0" />
<xsd:element name="ProductInfo" type="typens:ProductInfo" minOccurs="0" />
</xsd:all>
.....
```

Exemple Amazon (3)

```
.....
<message name="KeywordSearchRequest">
<part name="KeywordSearchRequest" type="typens:KeywordRequest"
    />
</message>
<message name="KeywordSearchResponse">
<part name="return" type="typens:ProductInfo" />
</message>
.....
<portType name="AmazonSearchPort">
<!-- Port for Amazon Web APIs  -->
<operation name="KeywordSearchRequest">
<input message="typens:KeywordSearchRequest" />
<output message="typens:KeywordSearchResponse" />
</operation>
.....
```

Exemple Amazon (4)

```
<operation name="ModifyShoppingCartItemsRequest">
<soap:operation soapAction="http://soap.amazon.com" />
  <input> <soap:body use="encoded"
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://soap.amazon.com" /> </input>
   <output><soap:body use="encoded"
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="http://soap.amazon.com" /> </output>
</operation>
</binding>
<service name="AmazonSearchService"> <!--Endpoint Amazon WebAPI -->
<port name="AmazonSearchPort"
  binding="typens:AmazonSearchBinding">
  <soap:address location="http://soap.amazon.com/onca/soap2" />
</port>
</service>
</wsdl:definitions>
```

Conclusion WSDL

- **Avantages**
  - Description de services assez simple**.**
  - Facile à interroger sur l'Internet**.**
  - Autorise une liaison statique, dynamique, retardée.
  - Extensible.
- **Inconvénients**
  - **Peu de fonctionnalités** offertes**.**
    - Peu d'aspects de déploiement.
    - Pas de spécification sémantique.

Bibliographie WSDL

- **Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001**
  http://www.w3.org/TR/2001/NOTE-wsdl-20010315
- Another small SOAP tutorial Benoît Marchal/Gamelan java journal
- Beginners introduction to SOAP
  http://www.4guysfromrolla.com/webtech/070300-2.shtml
- Web Services Description Language (WSDL) 1.1:
  http://www.w3.org/TR/wsdl

Chapitre IV

UDDI 'Universal Description Discovery and Integration'

Introduction UDDI

- Un annuaire pour exposer des services offerts par des entreprises.
- Utilise SOAP pour l'accès aux informations de l’annuaire.
- Créateur Ariba + IBM, Microsoft, HP, Oracle, SAP ...

Grandes lignes de UDDI

- Recherche d'une entreprise selon la nature de l'activité.
  - Exemple: NAICS ('North American Industry Classification System') = 3341 Fabricants d'ordinateurs.
- Recherche d'un service offert par l'entreprise: pour obtenir la description de son type (notion de 'tmodel').

UDDI: enregistrement et découverte de services

Pages Blanches : Fournir des informations sur une entreprise

Nom de l'entreprise, Identificateurs commerciaux.

Adresses diverses.

Description générale de l'activité.

Nom des personnes à contacter.

Pages Jaunes : Classer les entreprises et leurs services selon une taxinomie standard

Entreprises : Classifications normalisées par secteurs d 'activités

Localisation Géographique : taxinomie couples nom/valeur.

Pages Vertes : Fournir des informations techniques sur les services

Description des services,

Description des processus d'utilisation.

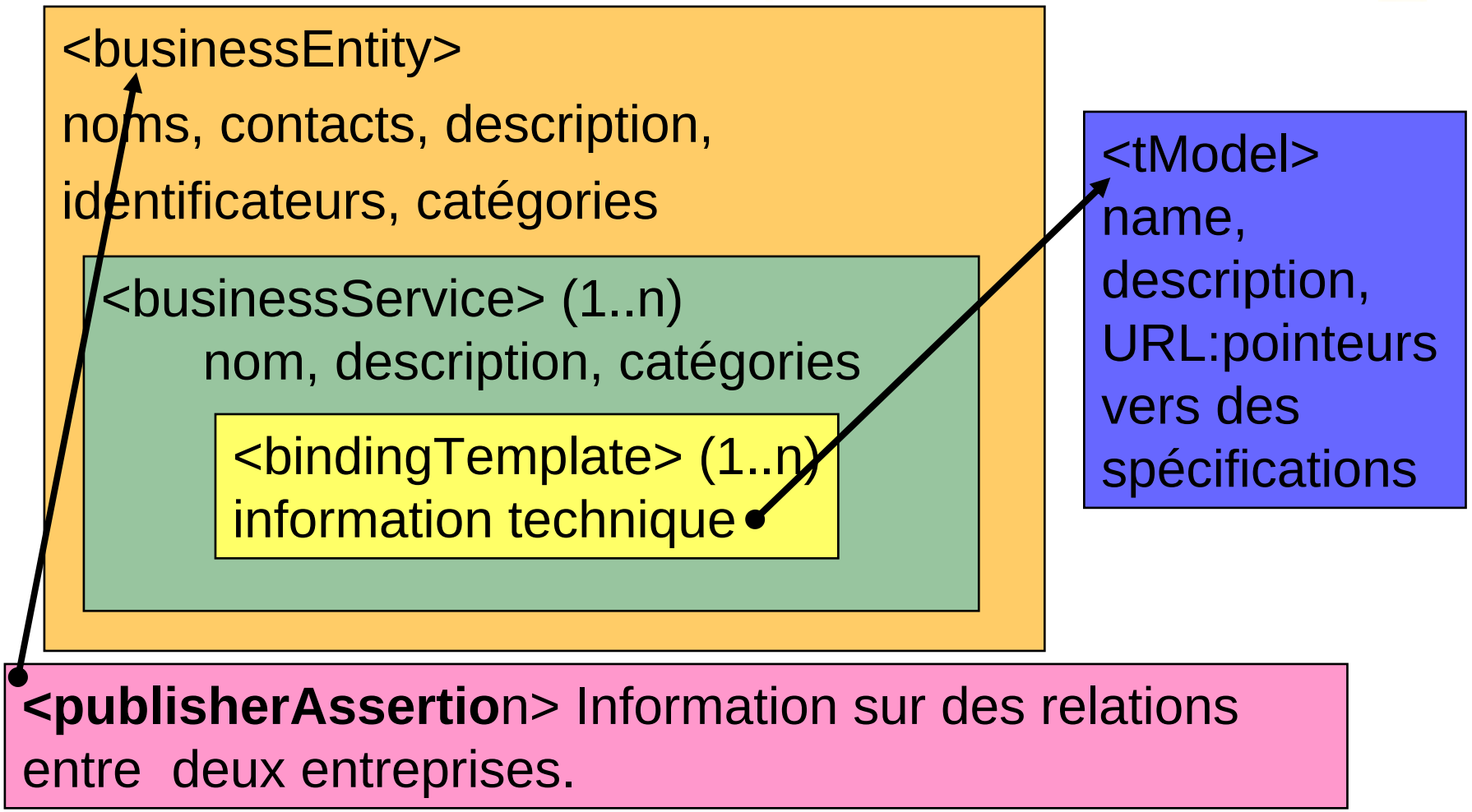
Informations de liaison (protocoles utilisés).

Les structures de données de l'annuaire UDDI

Quelles informations

- Qui : Le nom de l'entreprise, les contacts ...
- Quoi : Les classes, les noms des services
- Ou : Les adresses d'accès aux services
  - URI, adresses mail
- Comment : Les informations concernant les interfaces, les propriétés (tmodels)...

Structuration des informations



Un exemple de service

```
<businessService>
...
   <bindingTemplates>
        <bindingTemplate>
...
            <accessPoint urlType="http">http://www.foo.com/
             </accessPoint>
            <tModelInstanceDetails>
                 <tModelInstanceInfo tModelKey="xxxxx"> ...
                  </tModelInstanceInfo>
            </tModelInstanceDetails>
...
        </bindingTemplate>
...
   <bindingTemplates>
<businessService>
```

Elément businessEntity

```
<element name = "businessEntity"> <complexType>
   <sequence>
      <element ref = "discoveryURLs" minOccurs = "0"/>
      <element ref = "name" maxOccurs = "unbounded"/>
      <element ref = "description" minOccurs = "0"
   maxOccurs = "unbounded"/>
      <element ref = "contacts" minOccurs = "0"/>
      <element ref = "businessServices" minOccurs =
   "0"/>
      <element ref = "identifierBag" minOccurs = "0"/>
      <element ref = "categoryBag" minOccurs = "0"/>
   </sequence>
<attribute ref = "businessKey" use = "required"/>
<attribute ref = "operator"/>
<attribute ref = "authorizedName">
</complexType> </element>
```

Elément businessService

```
<element name = "businessService">
   <complexType>
      <sequence>
         <element ref = "name" maxOccurs = "unbounded"/>
         <element ref = "description" minOccurs = "0"
  maxOccurs = "unbounded"/>
         <element ref = "bindingTemplates"/>
         <element ref = "categoryBag" minOccurs = "0"/>
      </sequence>
     <attribute ref = "serviceKey" use = "required"/>
     <attribute ref = "businessKey"/>
   </complexType>
</element>
```

Elément bindingTemplate

```
<element name = "bindingTemplate">
   <complexType>
   <sequence>
     <element ref = "description" minOccurs = "0"
  maxOccurs = "unbounded"/> <choice>
     <element ref = "accessPoint" minOccurs = "0"/>
     <element ref = "hostingRedirector" minOccurs =
  "0"/>
     </choice>
     <element ref = "tModelInstanceDetails"/>
   </sequence>
   <attribute ref = "bindingKey" use = "required"/>
   <attribute ref = "serviceKey"/>
   </complexType> </element>
```

Elément tModel

```
<element name = "tModel"> <complexType> <sequence>
    <element ref = "name"/>
    <element ref = "description" minOccurs = "0"
  maxOccurs = "unbounded"/>
    <element ref = "overviewDoc" minOccurs = "0"/>
    <element ref = "identifierBag" minOccurs = "0"/>
    <element ref = "categoryBag" minOccurs = "0"/>
</sequence>
<attribute ref = "tModelKey" use = "required"/>
<attribute ref = "operator"/>
<attribute ref = "authorizedName"/>
</complexType>
</element>
```

Exemple de tmodel avec WSDL

```
<tModel authorizedName=”aaa” operator=”bbb”
   tModelKey=”xxxx”>
    <name>ServiceCommande</name>
    <description xml:lang=”fr”>
        Description WSDL du service de prise de commandes
    </description>
    <overviewDoc>
      <description xml:lang=”fr”>Source WSDL
      </description>

   <overviewURL>http://exemple.com/ServiceCommande.wsdl
   </overviewURL>
  </overviewDoc>
</tModel>
```

Elément publisherAssertion

```
<element name = "publisherAssertion">
   <complexType>
      <sequence>
         <element ref = "fromKey"/>
         <element ref = "toKey"/>
         <element ref = "keyedReference"/>
      </sequence>
  </complexType>
</element>
```

Les primitives d’accès à l’annuaire

L'API UDDI de recherche

- Trouver un enregistrement

find_business

find_service

find_binding

find_tModel

- Obtenir des détails complémentaires

get_businessDetail

get_serviceDetail

get_bindingDetail

get_tModelDetail

L'API UDDI de création/destruction

- Enregistrer des informations

save_business

save_service

save_binding

save_tModel

- Détruire des informations

delete_business

delete_service

delete_binding

delete_tModel

Conclusion Uddi

- Un service d'annuaires de plus.
- Orienté métier du commerce électronique.
- Orienté SOAP.

- Comment cette norme va t'elle se développer?

Conclusion Web Services

Avantages

- Un ensemble de propositions assez simples à la base.
- Très adaptées aux problèmes des communications entre applications WEB.

Limitations

- **Beaucoup de travail reste à faire : limitations actuelles des outils annexes.**
- **Limitations des spécifications de service** : **SOAP, WSDL et Uddi ne règlent pas tous les** problèmes de vocabulaire, de sémantique pour faire dialoguer des applications => besoin d'extension des définitions de spécifications ('Extended Service Definition')
  - Gestion des flots d'appels (workflow)
  - Gestion de la dynamique des exécutions (liaison)
  - Gestion des protocoles non standards (connecteurs)
  - Gestion du déploiement des services
- **Problèmes de performances**