Module SOA Et les Services Web

Maroua Idi
marouaidi@gmail.com

Année Universitaire: 2024-2025

iTeam University

Chapitre1: Technologies XML

Plan

- **Présentation**
- **Données structurées**
- **Organisation**
- **Syntaxe XML**
- **Documents XML bien formés**
- **Constructions utiles**
- **Les espaces de noms**
- **Validation des documents XML**
- **Schéma XML**

Présentation

❖ XML (eXtensible Markup Language) : langage à balises Extensible
- permettant de définir de nouvelles balises.
- permettant de mettre en forme des documents grâce à des balises.

❖ Contrairement à HTML (langage **défini** et **figé** avec un nombre de balises limité), XML est un **métalangage** permettant de définir d'autres langages.

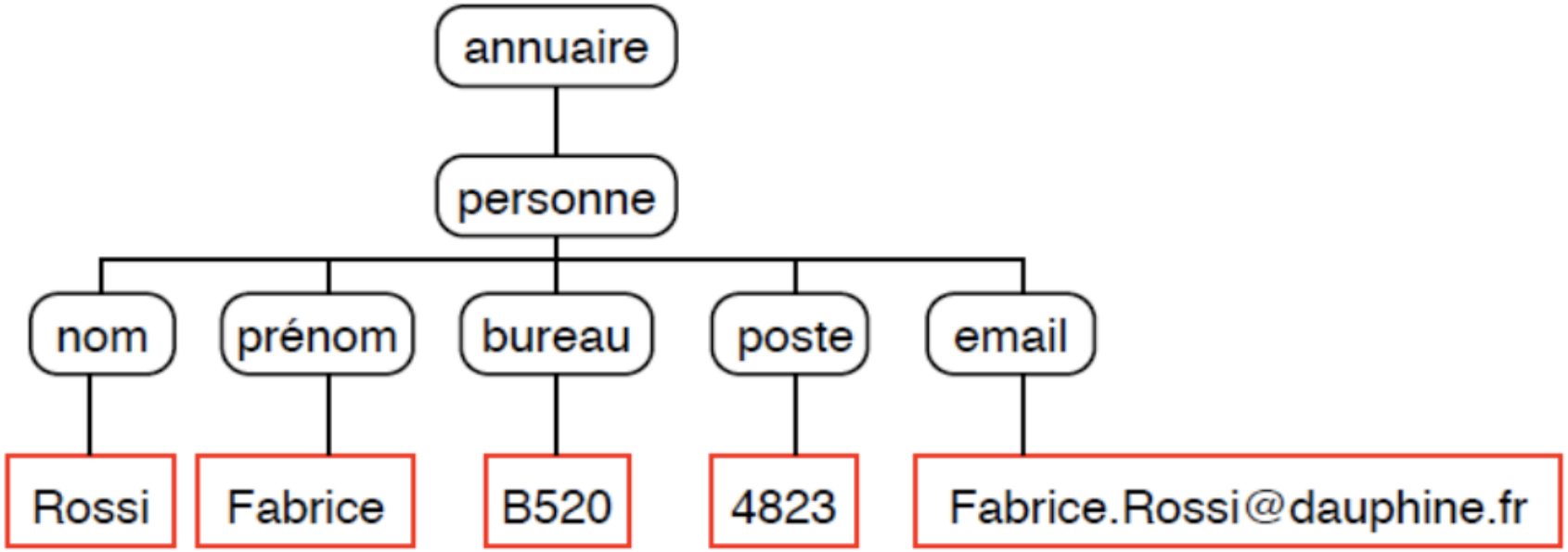
❖ Peut décrire n'importe quel domaine de données grâce à son extensibilité.
❖ Permet de structurer, poser le vocabulaire (grammaire) des données.

Données structurées

- XML permet de représenter des données structurées :
  - **données textuelles organisées** :
    - un **document** constitué d'**éléments.**
    - un **élément** peut être constitué de texte ou contenir d'autres éléments (ou un mélange des deux).
    - un élément peut être associé à des informations complémentaires, les **attributs.**
  - **la structure est celle d'un arbre :**
    - un document XML = un arbre
    - un élément = un nœud de l'arbre
- Le standard XML indique comment traduire l'arbre en un document XML, **pas comment organiser** les données

Exemple : Annuaire (1/4)

- **But** : stocker l'annuaire de Dauphine (nom, prénom, bureau, numéro de poste, email)
- Le texte du document : **les informations** !
- Organisation : s'arranger pour que les informations restent correctement groupées (ne pas mélanger les données !)
- **Solution 1:**



Exemple : Annuaire (2/4)

Traduction en XML de l'arbre :

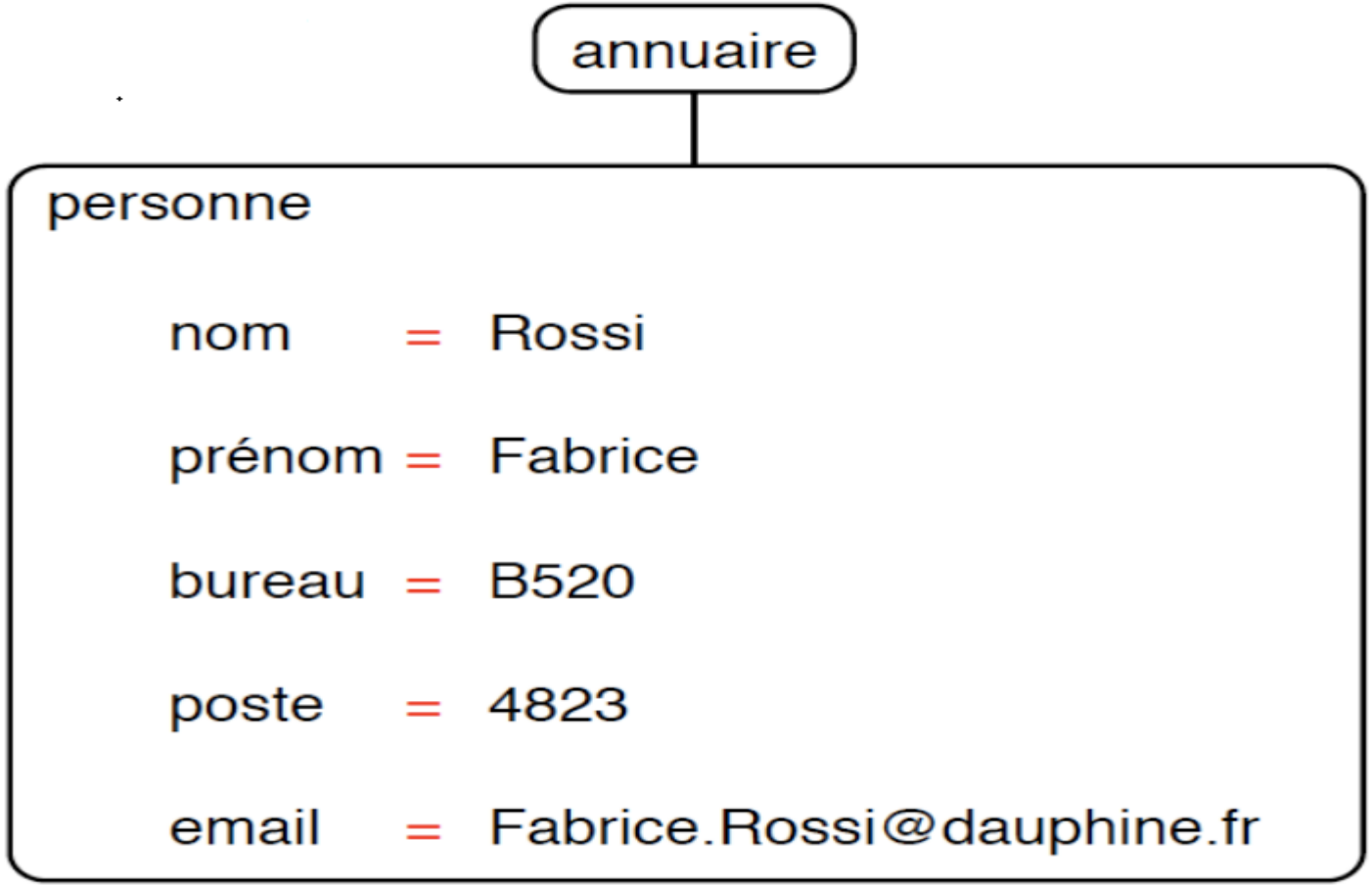
annuaire1.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3      <personne>
4          <nom>Rossi</nom>
5          <prenom>Fabrice</prenom>
6          <bureau>B520</bureau>
7          <poste>4823</poste>
8          <email>Fabrice.Rossi@dauphine.fr</email>
9      </personne>
10     <!-- suite de l'annuaire -->
11 </annuaire>
```

- Inclusion textuelle ⟺ relation mère-fille dans l'arbre.
- Balise ouvrante ou fermante ⟺ nom d'un nœud
- Texte => feuille de l'arbre
- Ne pas confondre les éléments (information) et les balises (syntaxe).

Exemple: Annuaire (3/4)

Solution 2 :



Exemple : Annuaire (4/4)

Traduction en XML de l'arbre :

annuaire2.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <annuaire>
3      <personne    nom="Rossi"
4                   prenom="Fabrice"
5                   bureau="B520"
6                   poste="4823"
7                   email="Fabrice.Rossi@dauphine.fr"/>
8      <!-- suite de l'annuaire -->
9  </annuaire>
10
```

- Attributs ⟺ annotations d'un nœud
- élément vide => feuille

Organisation

- On organise les données en décidant de la structure de l'arbre :
  - ✓ le nom des éléments
  - ✓ l'ordre des éléments
  - ✓ les relations d'inclusion
  - ✓ la position des données (c'est-à-dire du texte)
  - ✓ les contraintes sur les données (texte quelconque, valeur numérique, etc.)
  - ✓ les attributs
- Une organisation particulière forme un **vocabulaire** XML, par exemple:
  - ✓ MathML : pour décrire des équations
  - ✓ SVG : dessin vectoriel
  - ✓ etc.

Syntaxe XML

- Deux niveaux syntaxiques :
  - ✓ bas niveau : document **bien formé**
  - ✓ haut niveau : document **valide** (haut niveau => bas niveau)
- Du point de vue utilisateur/concepteur :
  - ✓ le bas niveau est **obligatoire** : **mal formé** => pas XML
  - ✓ le bas niveau est **fixé** par la norme
  - ✓ le haut niveau est **facultatif** : **bien formé** => XML
  - ✓ le haut niveau est **entièrement de la responsabilité** du concepteur : il définit les contraintes syntaxiques (noms de éléments, organisation, etc.)
  - ✓ le haut niveau peut se mettre en œuvre de différentes façons (DTD, schémas W3C, Relax NG, etc.)

Documents XML bien formés (1/9)

❖ La balise racine

✓ Tous les documents XML doivent avoir une balise racine, qui est également le premier élément du document XML. Dans tous les documents XML, une seule paire de balise permet de définir l'élément racine, tous les autres éléments doivent être imbriqués dans cet élément.

✓ Tous les éléments peuvent avoir des sous éléments (enfants).
✓ Les sous éléments doivent être correctement imbriqués dans les éléments parents.

Documents XML bien formés (2/9)

❖ Les éléments :

- ❑ <truc> : **balise ouvrante :**
  - ✓ doit toujours correspondre à une **balise fermante** (</truc>)
  - ✓ le texte entre <> est le nom de l'**élément** : constitué de lettres, chiffres, '.', '-', '_' et ' :'

✓ En XML tous les éléments doivent avoir des balises fermantes, comme ceci :

```
<p>Ceci est un paragraphe</p>
<p>Ceci est un autre paragraphe</p>
```

Documents XML bien formés (3/9)

- </quantite>: **balise fermante**:
  - ✓ depuis une balise ouvrante jusqu'à une balise fermante : le contenu d'un élément, un nœud de l'arbre.
- <et_hop/> : **balise mixte**, ouvrante et fermante, pour les éléments vides

❖ **Casse :** A la différence du HTML, les balises XML sont sensibles à la casse.

- ✓ Ainsi en XML, la balise <**l**ettre> est différente de la balise<**L**ettre>
- ✓ Les balises ouvrantes et fermantes doivent respecter la même casse:

<Message>Ceci n'est pas correct</message>

Documents XML bien formés (4/9)

❖ Imbrication

- Tous les éléments XML doivent être correctement imbriqués.
- En HTML, les éléments peuvent être imbriqués de façon incorrecte comme ceci :

** ce texte est gras et italique **

- En XML, tous les éléments doivent être imbriqués correctement les uns dans les autres :

** ce texte est gras et italique **

Documents XML bien formés (5/9)

❖ Attributs d'élément:

- Les attributs sont utilisés pour distinguer les éléments du même nom.
- Un attribut spécifie une propriété **unique** pour l'élément, en utilisant une paire nom / valeur (entre des cotes).
- Un élément peut avoir plusieurs attributs:

  <element atta="x" attb="y" …>....</element>
- Exemple élément non vide avec attributs:

  <adresse **ville="Sfax" codeP="3000"** >Rue 1235 maison 12</adresse>
- Ou élément vide avec attributs:

  <adresse **rue="1235" maison= "12" codeP="3000" ville="Sfax"** />

Documents XML bien formés (6/9)

❖ Les attributs :

Exemple : <font name="times">

name est un **attribut** de l'élément **font**, de valeur **times**.

- ne peut apparaître que dans une balise ouvrante ou mixte.
- doit **toujours** avoir une valeur.
- la valeur est **toujours** délimitée par des cotes " ou des apostrophes ' dans la valeur, < est interdit.
- pour le nom d'un attribut, même contrainte que pour les éléments.
- dans une même balise ouvrante ou mixte, chaque attribut ne peut apparaître qu'une fois.

Documents XML bien formés (7/9)

Grammaire de base :

- un document XML est un arbre d'éléments:
  - ✓ racine est **unique**
  - ✓ le contenu d'un élément est :
    - d'autres éléments
    - du texte (les *character data*) : **< et & sont interdits**

Documents XML bien formés (8/9)

Exemples : Mal formé ou Bien formé?

```
<a><b></a></b>
<p>bla, bla, bla<br>bla, bla, bla</p>
<nom.pas:tres_bien-choisit/>
 <a></a><b></b>
 <a>3<2</a>
 <a>3>2</a>
 <a>bla <br/>bla</a>
```

Documents XML bien formés (9/9)

Exemples (correction):

- <a> : **Mal formé**
- bla, bla, bla
bla, bla, bla : **Mal formé**
- <nom.pas:tres_bien-choisit/> : **Bien formé**
- <a> : **Mal formé**
- <a>3<2 : **Mal formé**
- <a>3>2 : **Bien formé**
- <a>bla
bla : **Bien formé**

Constructions utiles

- **Commentaires**:

La syntaxe pour écrire des commentaires en XML est similaire à celle de HTML :

< !- - This is a comment - - >

- **CDATA** (texte) :

cdata.xml

```
<![CDATA[ <a>contenu<b> non interprete,
 non analyse, ne fait pas</a> partie
de l'arbre</b> ]]>
```

- **Entités** :

gestion de la structure physique des documents XML

mécanisme de “macro” XML :

- inclusion d’un document dans un autre
- référence externe
- remplacement d’un texte par un autre

Entités

- Entités de bas niveau :
  - syntaxe : &nom;
  - “prédéfinies” pour les caractères spéciaux :

caractère	<	>	&	’	"
entitée	<	>	&	'	"

  - accès aux caractères par leur code UNICODE : &#nombre en base 10; ou &#xnombre en base 16;. Par exemple ' correspond à '.
- Entités de haut niveau :
  - à définir dans la grammaire (DTD, Schema XML)
  - inclusion
  - remplacement

Exemple complet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<carnet>
    <fiche>
        <nom>Rossi</nom><prenom>Fabrice</prenom>
        <adresse>
            <service>UFR MD</service>
            <rue>Place du Marechal de Lattre de Tassigny</rue>
            <code>75016</code><ville>Paris</ville>
        </adresse>
        <telephone>
            <fixe>01 44 05 48 23</fixe>
            <fax>01 44 05 40 36</fax>
            <portable>06 06 06 06 06</portable>
        </telephone>
        <email>
            <nom>Fabrice.Rossi</nom>
            <domaine>dauphine.fr</domaine>
        </email>
    </fiche>
</carnet>
```

Entête

- il est **conseillé** de commencer un document XML par :
  - < ?xml version="1.0" ?>
- l'attribut **encoding** permet d'indiquer la représentation physique des caractères du fichier :
  - < ?xml version="1.0" encoding="UTF-16" ?>
  - < ?xml version="1.0" encoding="UTF-8" ?> par défaut
  - < ?xml version="1.0" encoding="ISO-8859-1" ?> sous linux
- < ?nom ?> : une ***Processing Instruction*** indique aux logiciels comment traiter le document :
  - encodage
  - associer une feuille de style à un document

Les espaces de noms (1/6)

- Un document XML peut contenir des éléments et des attributs qui correspondent à plusieurs domaines distincts (i.e., à plusieurs dialectes).

Problème: comment gérer les collisions ?

Solution => Les espaces de noms (***namespaces***) :

- **permet d'introduire des collections de noms utilisables pour les éléments et les attributs d'un document XML**
- **principes :**
  - ✓ chaque collection est identifiée par un **URI (**Uniform Resource Identifier)
  - ✓ un **préfixe** est associé à un URI

Les espaces de noms (2/6)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html:html xmlns:html="http://www.w3.org/TR/REC-html40">
    <html:head>
        <html:title>Demonstration</html:title>
    </html:head>
    <html:body>Un contenu</html:body>
</html:html>
```

- ✓ L'attribut xmlns:html associe le préfixe html à l'URI "http ://www.w3.org/TR/REC-html40"
- ✓ L'association n'est valable que dans les descendants (au sens large) de l'élément qui contient xmlns:html
- ✓ L'association s'applique aux éléments et aux attributs

Les espaces de noms (3/6)

- Déclaration d'un *namespace* :
  - xmlns:préfixe="URI" : association du préfixe à l'URI
  - xmlns="URI" : définition de l'URI associé à l'espace de noms par défaut (sans préfixe)
- **Nom qualifié** :
  - préfixe : nom local
  - peut être utilisé pour les attributs et les éléments
  - le préfixe **doit** être déclaré par un ascendant
- Remarques :
  - c'est l'URI qui assure l'absence d'ambiguïté, pas le préfixe
  - le dernier qui parle a raison

Les espaces de noms (4/6)

- **Attributs**:
  - les attributs avec un nom qualifié sont traités comme les éléments
  - les attributs sans préfixe sont dans un espace de noms local propre à l'élément dans lequel ils apparaissent (c'est le cas classique sans *namespace*)
- Dans un élément donné, les attributs doivent être distincts, c'est-à-dire :
  - ou bien être des attributs qualifiés qui se distinguent par leur partie locale ou par leur URI
  - ou bien être des attributs sans préfixe distincts

Les espaces de noms (5/6)

Exemples

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/TR/REC-html40">
      <head>
            <title>Demonstration</title>
      </head>
      <body>Un contenu</body>
</html>
```

```
<x xmlns:n1=http://www.w3.org
      xmlns="http://www.w3.org" >
      <good a="1" b="2" />
      <good a="1" n1:a="2" />
</x>
```

Les espaces de noms (6/6)

- Quelques espaces de noms classiques
  - XML: http://www.w3.org/XML/1998/namespace
  - Xinclude: http://www.w3.org/2001/XInclude
  - Xlink: http://www.w3.org/1999/xlink
  - MathML: http://www.w3.org/1998/Math/MathML
  - XHTML: http://www.w3.org/1999/xhtml
  - SVG: http://www.w3.org/2000/svg
  - Schémas: http://www.w3.org/2001/XMLSchema
  - XSLT: http://www.w3.org/1999/XSL/Transform

Exercice

Observez le document XML suivant :

```
<?xml version="1.0"?>
<!-- this is a note -->
<note date=3 janvier>
 <to>Bob</To>
 <from>Alice</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
<note date="5 janvier" <!-- this is another note --> >
 <to>Alice</to>
 <from>Bob
 <body>No problem & see you soon</body>
</note>
<note /
```

1. Ce document est-il bien formé (i.e. respecte-t-il la syntaxe XML) ?
2. S'il ne l'est pas, corrigez les erreurs

Exercice (correction)

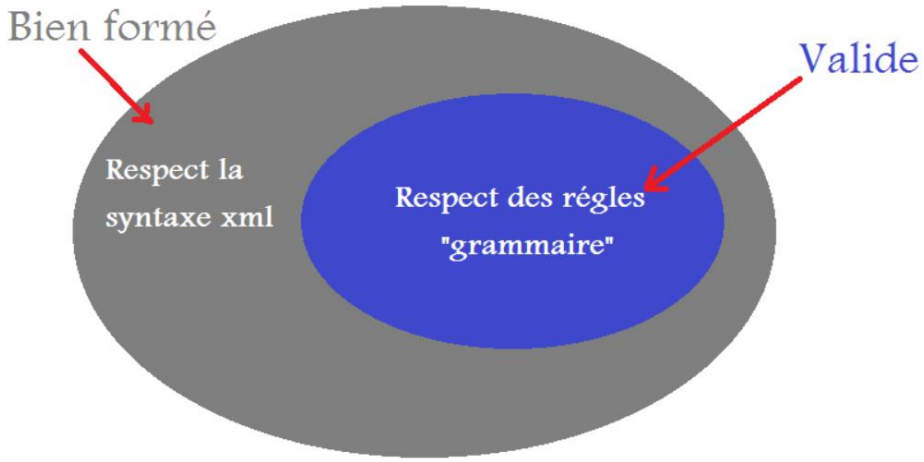
Le document corrigé :

```
<?xml version="1.0"?>
<!-- this is a note -->
<document>
<note date="3 janvier">
<to>Bob</to>
<from>Alice</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
<note date="5 janvier">
<!-- this is another note -->
<to>Alice</to>
<from>Bob</from>
<body>No problem, see you soon</body>
<note/>
</document>
```

Validation des documents XML

Les DTD et les Schémas XML(1/2)

- Un document XML est dit valide
  - s'il est bien formé (Il respecte la syntaxe XML).
  - Fait référence à une DTD (Document Type Definition) ou schéma XML (XSD) .



Les DTD et les Schémas XML(2/2)

- Soit par une **DTD (Document Type Definition):** Qui est une grammaire qui permet de définir une structure type de document XML.
  - ✓ Limitées: structure simple.
  - ✓ Syntaxe: issue de SGML (Pas XML).
- Ou par un **schéma XML** qui est un langage de description de format de document XML permettant de définir la structure et le type de contenu d'un document XML.
  - ✓ syntaxe XML.
  - ✓ Supporte tout type des attributs.

Note: Les DTD ne sont pas abordés dans ce cours, nous focalisons sur les schémas, en particulier : Schémas du W3C

Objectifs de Schéma XML

Un document XML doit pouvoir être validé relativement à son schéma XSD (XML Schema Definition).

- **Structures**
  - Définir la structure et les contenus des documents.
  - Définir des relations d'héritage.
- **Typage des données**
  - Fournir un ensemble de types primitifs.
  - Définir un système de typage suffisamment riche.
  - Distinguer les aspects reliés à la représentation lexicale des données de ceux gouvernant les données.
  - Permettre de créer des types de données usagers dérivés de types existants en contraignant certaines propriétés (domaine, précision, longueur, format).

Principes de schéma XML (1/2)

Un schéma XML est un document XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Déclaration de deux types d ’éléments -->
<xsd:element name="nom" type="xsd:string" />
<xsd:element name= "prenom" type="xsd:string" />
</xsd:schema>
```

Principes des s

```
<?xml version="1.0"?>
<Addresse_postale_france  xmlns:xsd=http://www.w3.org/2001/XMLSchema
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://www.w3.org/2001/XMLSchema
Addresse_postale_france.xsd      pays="FR">
<nom>Mr Jean Dupont</nom>
<rue>rue Camille Desmoulins</rue>
<ville>Paris</ville>
<departement>Seine</departement>
<code_postal>75600</code_postal>
</Addresse_postale_france >
```

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

Document XML (Adresse.xml)

Schéma (Adresse.xsd)

Schéma XML

Les composants primaires

Un schéma XML est construit par assemblage de différents composants (13 sortes de composants rassemblés en différentes catégories).

❖ Composants de déclaration

- Déclaration d'éléments.
- Déclaration d'attributs.

❖ Composants de définition de types

- Définition de types simples (*Simple type*).
- Définition de types complexes (*Complex type*).

Schéma XML

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

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

Schéma XML

Déclaration des attributs (1/3)

- Un attribut est déclaré par la balise ***attribute***
- Un attribut est une valeur, nommée et typée, associée à un élément.
- Le type d'un attribut défini en XML schéma est obligatoirement **simple**.

Exemple

```
<xsd:attribute name="année" type="xsd:integer" />
```

Schéma XML

Déclaration des attributs (2/3)

- 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="année" type="xsd:integer" use="required"  value="2024"/>
```

Schéma XML

Déclaration des attributs (3/3)

❖ **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="année" type="xsd:integer" use="required" />
```

Schéma XML

Les Types

- Les schémas sont basés sur la notion de type
- Chaque élément et chaque attribut possède un type
- Approche objet: les **types de base** et **types défini par dérivation**
- Deux grandes catégories de types :

1. Types simples (simpleType) :

- ✓ chaînes de caractères, valeurs numériques, etc.
- ✓ Un élément d'un type simple ne peut ni contenir d'autres éléments ni avoir des attributs.
- ✓ Les attributs ont des types simples.

2. Types complexes (complexType) :

- ✓ tout le reste, en particulier les éléments contenant d'autres éléments et/ou des attributs.

Schéma XML

Les Types: types simples (1/2)

❖ Types simples **prédéfinis** au sens de la norme XML Schémas '***datatypes***':

```
xsd:string      % Une chaîne de caractères (texte)
xsd:decimal     % Des chiffres avant et après la virgule
xsd:integer     % Un nombre entier (positif ou négatif).
xsd:boolean     %  Une valeur booléenne (Vrai ou Faux)
xsd:date        %  Une date au format AAAA-MM-JJ
xsd:time        %  Une heure au format HH:MM:SS
```

Exemple:

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

Valeur par défaut, modifiable si nécessaire.

- **Déclaration d'une valeur par défaut**: <xsd: element name= "code_postal" type= "xsd:integer" default "7500"/>
- **Déclaration d'une valeur figée:** <xsd: element name= "code_postal" type= "xsd:integer" fixed "7500"/>

Valeur obligatoire et inchangeable.

Schéma XML

Les Types: types simples (2/2)

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

- **Exemple : dérivation par restriction**

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

Nouveau type simple est dérivé de xsd: decimal

le nombre doit avoir exactement 2 chiffres après la virgule.

Schéma XML

Types: types complexes

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

Exemple

```
<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**.

Schéma XML

Types: types complexes

- **xsd:sequence** : ordonne les sous-éléments : ils doivent apparaître dans un ordre précis.
- **xsd:choice** : indique qu'un seul des sous-éléments peut apparaître,
- **xsd:all** : les sous-éléments apparaissent dans n'importe quel ordre,

- **maxOccurs** : nombre maximum (par défaut 1). Pour un nombre illimité, utiliser : **maxOccurs**="unbounded",
- **minOccurs** : nombre minimum.

Schéma XML

Types: types complexes (les séquences)

- 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:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="AdresseType">
<xsd:sequence>
<xsd:element name="rue" type="xsd:string" minOccurs="1" maxOccurs="1"/>  <!-- Obligatoire -->
<xsd:element name="ville" type="xsd:string" minOccurs="1" maxOccurs="1"/>  <!-- Obligatoire -->
 <xsd:element name="codePostal" type="xsd:string" minOccurs="1" maxOccurs="1"/>  <!-- Obligatoire -->
<xsd:element name="telephone" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>  <!-- Optionnel -->
</xsd:sequence>
<!-- L'attribut "pays" est défini ici, en dehors de la séquence -->
 <xsd:attribute name="pays" type="xsd:string" use="required"/>
</xsd:complexType>
 <xsd:element name="adresse" type="AdresseType"/>
</xsd:schema>
```

Schéma XML

Types: types complexes (les séquences)

❖ Exemple d'utilisation dans un document XML:

- **Exemple 1** : Sans numéros de téléphone

```
<adresse pays="France">
<rue>123 Rue de Paris</rue>
 <ville> Paris</ville>
<codePostal>75001</codePostal>
</adresse>
```

- **Exemple 2**: Avec plusieurs numéros de téléphone

```
<adresse pays="France">
<rue>123 Rue de Paris</rue>
<ville>Paris</ville>
<codePostal>75001</codePostal>
<telephone>01-23-45-67-89</telephone>
<telephone>02-34-56-78-90</telephone>
</adresse>
```

Schéma XML

Types: types complexes (Choice)

- Un **seul** des éléments listés doit être présent.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="ContactInfoType">
 <xsd:choice>
 <xsd:element name="telephone" type="xs:string"/>
<xsd:element name="email" type="xsd:string"/>
</xsd:choice>
 </xsd:complexType>
<xsd:element name="contactInfo" type="ContactInfoType"/>
</xsd:schema>
```

Schéma XML

Types: types complexes (Choice)

Exemple de document XML avec <xsd: choice>

Exemple 1 : Avec un numéro de téléphone

```
<contactInfo>
<telephone>01-23-45-67-89</telephone>
</contactInfo>
```

Exemple 2 : Avec une adresse email

```
<contactInfo>
<email>example@domaine.com</email>
</contactInfo>
```

Exemple non valide (avec les deux éléments)

```
<contactInfo>
<telephone>01-23-45-67-89</telephone>
<email>example@domaine.com</email>
 </contactInfo>
```

Schéma XML

Types: types complexes (all)

- Les éléments listés doivent être **tous présents** au plus une fois.
- Il peuvent apparaître dans n'importe quel ordre.

```
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="PersonneType">
 <xsd:all>
 <xsd:element name="nom" type="xs:string"/>
 <xsd:element name="prenom" type="xs:string"/>
 <xsd:element name="dateNaissance" type="xs:date"/>
</xsd:all>
</xsd:complexType>
<xsd:element name="personne" type="PersonneType"/>
</xsd:schema>
```

Schéma XML

Types: types complexes (all)

Exemple 1 : Ordre des éléments tel qu'il apparaît dans le schéma

```
<personne> <nom>Dupont</nom>
<prenom>Jean</prenom>
<dateNaissance>1980-05-12</dateNaissance>
</personne>
```

Exemple 2 : Les éléments dans un ordre différent

```
<personne> <prenom>Jean</prenom>
<dateNaissance>1980-05-12</dateNaissance>
<nom>Dupont</nom> </personne>
```

Exemple non valide (élément manquant)

```
<personne> <nom>Dupont</nom>
<prenom>Jean</prenom> </personne>
```

Les types de données schéma XML

Objectifs de la définition des types

- **Fournir des types primitifs** analogues à ceux qui existent en SQL ou en Java.
- **Définir un système de typage suffisamment riche** pour importer/exporter des données d'une base de données.
- **Distinguer les aspects reliés à la représentation lexicale des données** de ceux gouvernant les ensembles de données sous-jacents.
- **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).

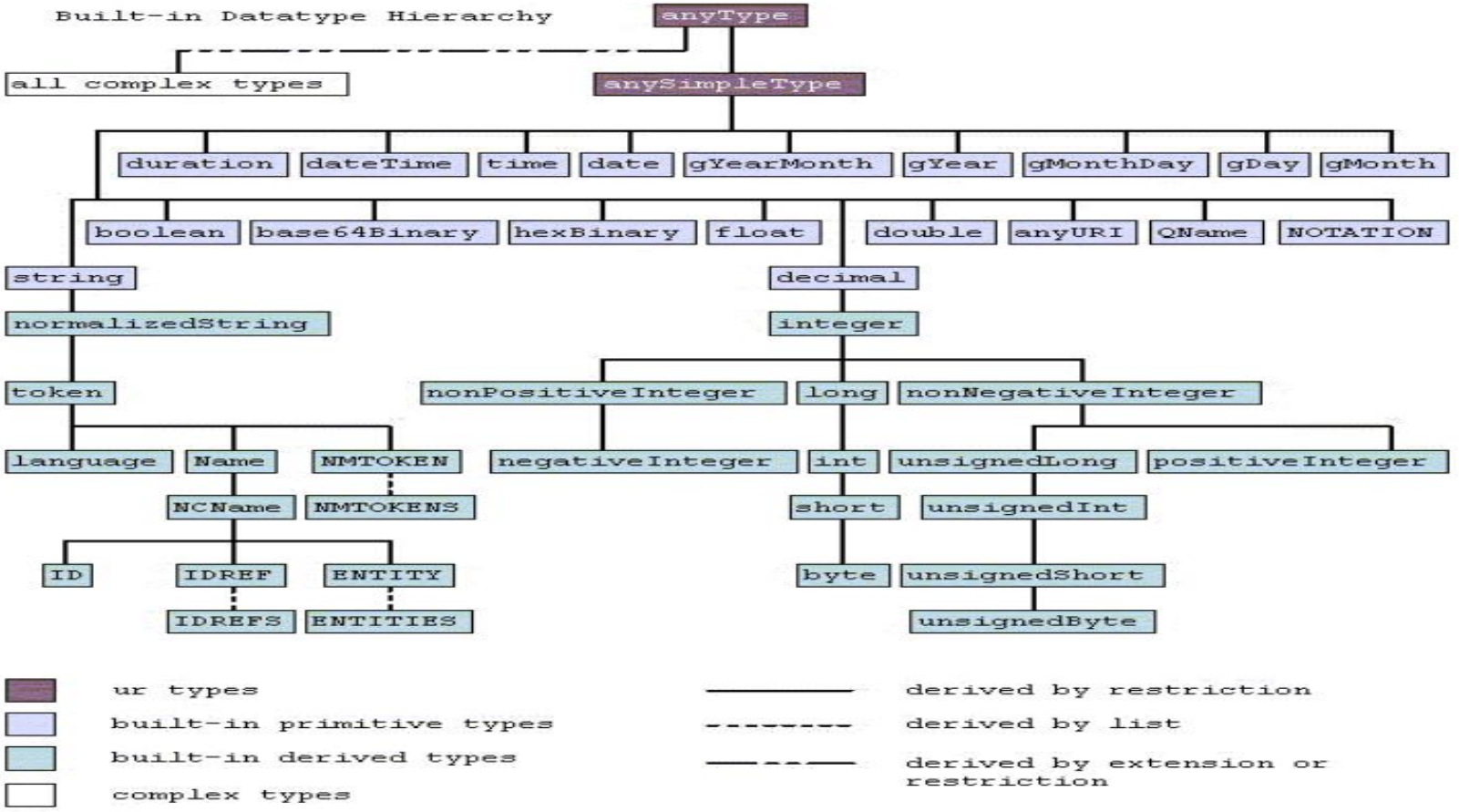
Ex: Le type float est défini par la norme IEEE 754-1985 (c'est un flottant simple, précision sur 32-bit).

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.

Définitions relatives aux types



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**
….	

Les types de données schéma XML

Dérivation de types simples

- **Dérivation par restriction**
- **Dérivation par extension**
- **Dérivation par union**
- **Dérivation par liste**

Les types de données schéma XML

Dérivation par restriction (1/4)

- La dérivation par restriction restreint l'ensemble des valeurs d'un type pré-existant.
- La restriction est définie par des **contraintes de facettes** que vous pouvez utiliser pour dériver un type simple.

Les types de données schéma XML

Dérivation par restriction (2/4)

❖ **Les contraintes de facettes**

- ***lenght*** : la longueur d'une donnée.
- ***minLenght***: la longueur minimum.
- ***maxLenght***: la longueur maximum.
- ***pattern***: défini par une expression régulière.
- ***enumeration***: un ensemble discret de valeurs.
- ***whitespace***: contraintes de normalisation des chaînes relativement aux espaces (preserve, replace, collapse).
- ***maxInclusive***: une valeur max comprise.
- ***maxExclusive***: une valeur max exclue.
- ***minInclusive***: une valeur min comprise.
- ***minExclusive***: une valeur min exclue.
- ***totalDigits***: le nombre total de chiffres.
- ***fractionDigits***: le nombre de chiffres dans la partie fractionnaire.

Les types de données schéma XML

Dérivation par restriction (3/4)

Exemple

```
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <!-- Définition d'un nouveau type "positiveInteger" dérivé du type "integer" -->
 <xsd:simpleType name="positiveIntegerRestricted">
  <xsd:restriction base="xs:integer">
 <!-- Valeur minimale acceptée : 1 -->
  <xsd:minInclusive value="1"/>
 <!-- Valeur maximale acceptée : 100 -->
  <xsd:maxInclusive value="100"/>
 </xsd:restriction>
</xs:simpleType>
<!-- Utilisation du nouveau type dans un élément -->
 <xsd:element name="age" type="positiveIntegerRestricted"/>
</xsd:schema>
```

Les types de données schéma XML
Dérivation par restriction (4/4)

- **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>
```

Les types de données schéma XML

Dérivation par extension (1/2)

- 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.**

Les types de données schéma XML

Dérivation par extension (2/2)

- **Exemple**

```
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <!-- Définition du type de base "Personne" -->
<xsd:complexType name="Personne">
<xsd:sequence>
 <xsd:element name="nom" type="xs:string"/>
<xsd:element name="prenom" type="xs:string"/>
</xsd:sequence>
</xsd:complexType>
<!-- Dérivation par extension : "Employé" ajoute des éléments à "Personne" -->
<xsd:complexType name="Employe">
<xsd:complexContent>
<xsd:extension base="Personne">
<xsd:sequence>
<xsd:element name="idEmploye" type="xs:string"/>
<xsd:element name="poste" type="xs:string"/>
 </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:schema>
```

Les types de données schéma XML

Dérivation par union

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

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <!-- Définition d'un nouveau type basé sur l'union de "integer" et "string" -->
<xsd:simpleType name="Identifiant">
<xsd:union memberTypes="xsd:integer xsd:string"/>
</xsd:simpleType>
<!-- Utilisation du type "Identifiant" dans un élément -->
 <xsd:element name="userId" type="Identifiant"/>
</xsd:schema>
```

Les types de données schéma XML

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).
- **Exemple**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <!-- Définition d'un type simple basé sur une liste de chaînes de caractères -->
<xsd:simpleType name="ListeDeCouleurs">
<xsd:list itemType="xsd:string"/>
 </xsd:simpleType>
<!-- Utilisation du type "ListeDeCouleurs" dans un élément -->
<xsd:element name="couleurs" type="ListeDeCouleurs"/>
</xsd:schema>
```

XSL, XPath, XQuery

XSL (eXtensible Stylesheet Language)

- Transformer et styliser les documents XML.
- Il comprend XSLT, un langage qui permet de convertir un document XML en un autre format (par exemple HTML, texte, etc.).

Utilité : Stylisation et transformation d'XML.

XPath (XML Path Language)

- Sélectionner des nœuds dans un document XML, permettant de naviguer à travers la structure XML.

- **Utilité** : Accéder à des éléments ou attributs spécifiques dans un document XML.

XQuery (XML Query Language)

- Interroger des documents XML. Il permet d'extraire, de filtrer, et de manipuler des données dans des documents XML.

- **Utilité** : Requête et manipulation de données dans des documents XML, similaire à SQL mais pour XML.