Ontologies et Web Sémantique
Master 2

Chapitre I:XML

I.5 **xml schema**

Mr Bougherara.S

Plan du cours

Introduction

Un schéma W3C (schéma) est une grammaire définie dans un formalisme XML.

Sauf pour la gestion des entités, on peut considérer les schémas comme remplaçant des DTD.

Limites DTD

- Syntaxe non XML
- Pas de support des espaces de noms
- Modularité limitée
- Modèle limité :
- Contraintes de structures trop simplistes(nombre, modèle mixte, etc.).
- Pas de contraintes sur les données

xml schema :Caractéristiques

- gestion des espaces de noms ;
- types de base riches et extensibles ;
- réutilisation par importation et héritage ;
- davantage de souplesse dans les cardinalités ;

Structure d’un doc XMLSchema

- C'est un document XML bien formé(arbre) avec :
  - Une racine schema,
  - Une suite de définitions de types,d'éléments et d'attributs

- Un schéma XML se compose essentiellement de déclarations d'éléments et d'attributs et de définitions de types.Chaque élément est déclaré avec un type qui peut être, soit un des types prédéfinis, soit un nouveau type défini dans le schéma

- L'espace de noms des schémas XML est identifié par l'URI http://www.w3.org/2001/XMLSchema. Il est généralement associé, comme dans l'exemple suivant au préfixe xsd ou à xs. Tout leschéma est inclus dans l'élément xsd:schema

Structure d’un doc XMLSchema

La structure globale d'un schéma est donc la suivante :

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

Types élément et attribut

- ***Types des formes (élément)***
  - Vide : pas de contenu (EMPTY en DTD) .
  - Simple : du texte (#PCDATA en DTD)
  - Complexe : un ou plusieurs éléments.
  - Mixte : un mélange d'éléments et de texte

- ***Types des formes (attribut)***
  - Simple : du texte (#PCDATA en DTD)

Type simple

Types prédéfinis(de base)

•Les schémas possèdent de nombreux types prédéfinis(xsd:int,xsd:float, xsd:ID)

•Les types de base sont toujours liés à l'espace de noms des schémas pour éviter toute collision avec votre propre type.

• le type est déclaré par l'attribut type à l'intérieur de l'élément

•Lorsqu'on nomme un type de base, on le préfixe par xs: pour signifier la présence d'un espace de noms (et non pour obliger à utiliser le préfixe xs).

Type simple

Types prédéfinis(de base)

Exemple de schema

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

    <xs:element name="titre" type="xs:string"/>
  <xs:element name="numchapitre" type="xs:int"/>
</xs:schema>
```

Exemple de document xml associé

<titre>Un titre **</titre>**

<numchapitre>10 **</numchapitre>**

Type simple

Types prédéfinis(de base)

Chaine de caractères

`xs:string`	le plus simple sans normalisation ni compactage
`xs:normalizedString`	la forme normalisée et non compactée
`xs:token`	la forme normalisée et compactée
`xs:language`	les codes de langue
`xs:NMTOKEN`	pas de blanc
`xs:Name`	forme de NMToken commençant par une lettre
`xs:ID`	chaîne unique dans le document
`xs:IDREF`	chaîne ayant pour valeur un ID
`xs:anyURI`	la valeur de l'URI utilisera des caractères ASCII, conversion automatique

Type simple
Types prédéfinis(de base)
Les types numériques

`xsd:boolean`	Valeur booléenne avec `true` ou `1` pour *vrai* et `false` ou `0` pour *faux*
`xsd:byte`	Nombre entier signé sur 8 bits
`xsd:unsignedByte`	Nombre entier non signé sur 8 bits
`xsd:short`	Nombre entier signé sur 16 bits
`xsd:unsignedShort`	Nombre entier non signé sur 16 bits
`xsd:int`	Nombre entier signé sur 32 bits
`xsd:unsignedInt`	Nombre entier non signé sur 32 bits
`xsd:long`	Nombre entier signé sur 64 bits
`xsd:unsignedLong`	Nombre entier non signé sur 64 bits
`xsd:integer`	Nombre entier sans limite de précision
`xsd:positiveInteger`	Nombre entier strictement positif sans limite de précision
`xsd:negativeInteger`	Nombre entier strictement négatif sans limite de précision
`xsd:nonPositiveInteger`	Nombre entier négatif ou nul sans limite de précision
`xsd:nonNegativeInteger`	Nombre entier positif ou nul sans limite de précision
`xsd:float`	Nombre flottant sur 32 bits conforme à la norme IEEE 754
`xsd:double`	Nombre flottant sur 64 bits conforme à la norme IEEE 754
`xsd:decimal`	Nombre décimal sans limite de précision

Type simple
Types prédéfinis(de base)
dates et heures

`xsd:time`	Heure au format `hh:mm:ss[.sss].` par exemple `14:07:23`. La partie fractionnaire `.sss` des secondes est optionnelle.
`xsd:date`	Date au format `YYYY-MM-DD`, par exemple `2008-01-16`. Tous les champs sont obligatoires
`xsd:dateTime`	Date et heure au format `YYYY-MM-DDThh:mm:ss` comme `2008-01-16T14:07:23`. Tous les champs sont obligatoires
`xsd:gYear`	Année du calendrier grégorien au format `YYYY` comme `2011`
`xsd:gYearMonth`	Année et mois du calendrier grégorien au format `YYYY-MM` comme `1966-06` pour juin 1966
`xsd:gMonth`	Mois du calendrier grégorien au format `MM` comme `01` pour janvier
`xsd:gMonthDay`	Jour et mois du calendrier grégorien au format `MM-DD` comme `11-01 (1er nov)`
`xsd:gDay`	Jour (dans le mois) du calendrier grégorien au format `DD` comme `01` pour le premier jourde chaque mois

Type simple

Créer d’un nouveau type simple

•Pour créer un nouveau types simple en utilise l’element xsd :simpleType

• Un type simple est souvent obtenu par restriction par l'opérateur union.

•

• La déclaration d'un type simple a la forme suivante.

<xsd:simpleType name=[*nom_nouveau_type*]>
[*liste restrictions*]
</xsd:simpleType>

Type simple

Créer d'un nouveau type simple

La restriction

•Permet d'obtenir un type dérivé à partir d'un type de base.

•L'idée générale de la restriction est de définir un nouveau type dont les contenus au sens large sont des contenus du type de base:

•03 types

- •Restriction par intervalle
- •Restriction **par énumération**
- •**Restriction par motif**

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par intervalle**

•Permet de restreindre les contenus en donnant une valeur minimale et/ou une valeur maximale avec les éléments:
• xsd:minInclusive, xsd:minExclusive, xsd:maxInclusive et xsd:maxExclusive.

•Ces contraintes s'appliquent aux types numériques pour lesquels elles ont un sens.

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par intervalle**

- exemple

<xsd:element name="year"> <xsd:simpleType> <xsd:restriction base="xsd:integer"> **<xsd:minInclusive value="1970"/>** **<xsd:maxInclusive value="2050"/>** </xsd:restriction> </xsd:simpleType> </xsd:element>	le type donné à l'élément year est un entier entre 1970 et 2050 inclus
<xsd:attribute name="date"> <xsd:simpleType> <xsd:restriction base="xsd:date"> **<xsd:minExclusive value="2001-12-31"/>** </xsd:restriction> </xsd:simpleType> </xsd:attribute>	Date après le 31 décembre 2001 exclus

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par énumération**

•de donner explicitement une liste des valeurs possibles d'un type prédéfini ou déjà défini avec l'élément xsd:enumeration

•Exemple

```
<xsd:element name="language" type="Language">
        <xsd:simpleType name="Language">
                <xsd:restriction base="xsd:language">
                        <xsd:enumeration value="de"/>
                        <xsd:enumeration value="en"/>
                        <xsd:enumeration value="fr"/>
                </xsd:restriction>
        </xsd:simpleType>
 </xsd:element>
```

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par motif**

•de restreindre les valeurs en donnant, avec l'élément xsd:pattern, une expression rationnelle qui décrit les valeurs possibles d'un type prédéfini ou déjà défini.

•Le contenu est valide s'il est conforme à l'expression rationnelle

•Exemple :pour le type monEntier, ou bien l'entier 10 ou bien l'entier 30.

```
<xs:simpleType nom="monEntier">
<xs:restriction base="xs:int">
        <xs:pattern value="10"/>
        <xs:pattern value="30"/>
</xs:restriction>
</xs:simpleType
```

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par motif**

•de restreindre les valeurs en donnant, avec l'élément xsd:pattern, une expression rationnelle qui décrit les valeurs possibles d'un type prédéfini ou déjà défini.

•Le contenu est valide s'il est conforme à l'expression rationnelle

•Exemple :pour le type monEntier, ou bien l'entier 10 ou bien l'entier 30.

```
<xs:simpleType nom="monEntier">
<xs:restriction base="xs:int">
        <xs:pattern value="10"/>
        <xs:pattern value="30"/>
</xs:restriction>
</xs:simpleType
```

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par motif**

Liste des facettes

`xsd:enumeration`	Cette facette permet d'énumérer explicitement les valeurs autorisées. Elle s'applique à tous les types simples y compris les types construits avec `xsd:union` et `xsd:list`
`xsd:pattern`	Cette facette permet de donner une expression rationnelle pour contraindre les valeurs. Elle ne s'applique pas uniquement aux types dérivés de `xsd:string` mais à tous les types simples y compris les types numériques et les types contruits avec `xsd:union` et `xsd:list`. L'utilisation avec `xsd:decimal` permet de restreindre, par exemple, aux nombres ayant 4 chiffres pour la partie entière et 2 pour la partie fractionnaire
`xsd:length`, `xsd:minLength` `xsd:maxLength`	Ces trois facettes donnent respectivement une longueur fixe ou des longueurs minimale et maximale.Elles s'appliquent principalement aux types dérivés de `xsd:string`

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par motif**

Liste des facettes

`xsd:minInclusive`, `xsd:minExclusive`, `xsd:maxInclusive` et `xsd:maxExclusive`	Ces quatre facettes donnent des valeurs minimale et maximale en incluant ou non la borne donnée. Ces facettes s'appliquent à tous les types numériques ainsi qu'à tous les types de date et d'heure.
`xsd:fractionDigits` `xsd:totalDigits`	Ces deux facettes fixent respectivement le nombre maximal de chiffres de la partie fractionnaire (à droite de la virgule) et le nombre maximal de chiffres en tout.
`xsd:fractionDigits` `xsd:totalDigits`	Ces deux facettes fixent respectivement le nombre maximal de chiffres de la partie fractionnaire (à droite de la virgule) et le nombre maximal de chiffres en tout.

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **Restriction par motif**

Liste des facettes

`xsd:whiteSpace`	Cette facette peut prendre les trois valeurs `preserve`, `replace` et `collapse` qui correspondent à trois modes de fonctionnement de l'analyseur lexical. • `Preserve` Dans ce mode, les caractères d'espacement sont laissés inchangés par l'analyseur lexical. • `Replace` Dans ce mode, chaque caractère d'espacement est remplacé par un espace `U+20`. Le résultat est donc du type prédéfini `xsd:normalizedString` • `Collapse` Dans ce mode, le traitement du mode précédent `replace` est d'abord appliqué puis les espaces en début et en fin sont supprimés et les suites d'espaces consécutifs sont remplacées par un seul espace. Le résultat est donc du type prédéfini `xsd:token`

- **Type simple**
  - Créer d'un nouveau type simple
    - **La restriction**
      - **L'union**

•sert à autoriser plusieurs types, ce qui permet d'offrir des alternatives, comme un attribut ayant pour valeur soit un entier soit une constante.
•L'union sert à combiner plusieurs types simples ; il suffira qu'une valeur corresponde à l'un des types déclarés par l'attribut memberTypes pour que la valeur soit correcte.

Exemple :

```
<xs:simpleType>
        <xs:union  memberTypes="xs:int xs:boolean"/>
</xs:simpleType>
```

Dans cet exemple, on autorisera une valeur entière ou un booléen (true ou false

Type complexe

•un type complexe concerne le contenu d'un élément.

•Il caractérise une composition d'éléments ou d'attributs (d'où sa nature complexe). Dans un schéma, le type complexe nécessite toujours une balise complexType

Type complexe

- Type complexe d'éléments seulement (composition)
  - **composition ordonnée (séquence)**

La séquence caractérise des éléments fils présents dans un ordre donné

Exemple :

```
<xs:element name="plan">
   <xs:complexType>
         <xs:sequence>
               <xs:element name="auteurs" type="xs:string"/>
               <xs:element name="chapitres" type="xs:string"/>
         </xs:sequence>
   </xs:complexType>
</xs:element>
```

Type complexe

- Type complexe d'éléments seulement (composition)

•composition non ordonnée

Le connecteur all caractérise un ensemble d'éléments de présence obligatoire mais sans contrainte sur l'ordre (toutes les permutations sont donc valides).

Exemple :

```
<xs:element name="plan">
  <xs:complexType>
        <xs:all>
              <xs:element name="auteur" type="xs:string"/>
              <xs:element name="chapitres" type="xs:string"/>
        </xs:all>
  </xs:complexType>
</xs:element>
```

Dans cet exemple l'élément plan contiendra les éléments auteur et chapitres dans n'importe quel ordre.

Type complexe

- Type complexe d'éléments seulement (composition)
  - **le choix**

Le choix permet de proposer des alternatives d'éléments
Exemple :

```
<xs:element name="plan">
<xs:complexType>
        <xs:choice>
                <xs:element name="sections" type="xs:string"/>
                <xs:element name="chapitres" type="xs:string"/>
        </xs:choice>
</xs:complexType>
</xs:element>
```

Ce qui autorisera un élément sections ou bien un élément chapitres dans l'élément plan

Type complexe

- Type complexe d'éléments seulement (composition)

  - **Les cardinalités**

Tout comme dans les DTD, les cardinalités sont possibles sur les éléments de connecteur ou sur les connecteurs eux-mêmes

Ces cardinalités sont positionnées par les attributs minOccurs et maxOccurs.

Pour faire le parallèle avec les DTD, nous retrouvons l'équivalent des opérateurs ?, + et * avec :

? : minOccurs="0" maxOccurs="1" ;
+ : minOccurs="1" maxOccurs="unbounded" ;
*: minOccurs="0" maxOccurs="unbounded".
Par defaut , minOccurs et maxOccurs ont la valeur 1.

Type complexe

- Type complexe d'éléments seulement (composition)
  - **Les cardinalités**

Exemple

```
<xs:element name="plan">
 <xs:complexType>
        <xs:sequence>
              <xs:element name="auteur" type="xs:string"
                    maxOccurs="unbounded"/>
              <xs:element name="chapitre" type="xs:string"
                    minOccurs="2"
axOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>
</xs:element>
```

Dans cet exemple, l'élément plan contient un élément auteur suivi d'au moins 2 éléments chapitre

Type complexe

- Type complexe d'éléments seulement (composition)
  - **Les cardinalités**

Exemple2

```
<xs:element name="plan">
<xs:complexType>
        <xs:sequence minOccurs="2" maxOccurs="3">
                <xs:element name="auteur" type="xs:string"/>
                <xs:element name="chapitre" type="xs:string"/>
        </xs:sequence>
</xs:complexType>
</xs:element>
```

Avec cet exemple, nous contrôlons que l'élément plan contient entre 2 et 3 suites d'éléments auteur et chapitre

Type complexe

- Type complexe d'elements avec attribut

•Définition d'un attribut :

L'attribut implique la présence d'un type complexe. Il est toujours placé en dernière position par l'element xs:**attribute** selon la syntaxe **:**

<xs:**attribute name=[***nom attribut***] type=[***type attribut***]/>**

L'attribut en lui-même, ne contenant que du texte, est un type simple.

Type complexe

- Type complexe d'elements avec attribut

•Définition d'un attribut :

Exemple :

```
<xs:element name="personne">
        <xs:complexType>
                …
                <xs:attribute name="nom" type="xs:string"/>
        </xs:complexType>
</xs:element>
```

Dans cet exemple, nous associons à l'élément personne l'attribut nom. L'attribut nom est local à l'élément personne ; il ne peut pas être employé dans un autre élément.

Type complexe

- Type complexe d'elements avec attribut

•limitations sur un attribut

La présence d'un attribut peut être définie par l'attribut use, qui peut prendre les valeurs suivantes :

• prohibited : interdire l'usage d'un attribut par dérivation d'un type complexe.

• optional : l'attribut n'est pas obligatoirement renseigné (employé par défaut).

• required : l'attribut est obligatoire.

Deux autres attributs, default et fixed, servent à définir respectivement une valeur par défaut, si l'attribut est optionnel, et une valeur obligatoire

Type complexe

- Type complexe d'elements avec attribut
  - **limitations sur un attribut**

Example:

```
<xs:element name="a">
   <xs:complexType>
         <xs:attribute name="t1" use="required"
type="xs:int"/>
         <xs:attribute name="t2" use="optional"
type="xs:string"    default="valeur"/>
         <xs:attribute name="t3" use="required"
type="xs:token"    fixed="autre"/>
   </xs:complexType>
 </xs:element>
```

Elément vide

Element quasiment vide

L’élément vide n’a pas de contenu ; sa définition est donc sans typage. On l’écrit, par exemple :
<xs:element name="br"/>

Elément vide

élément vide avec attributs

L’attribut rend la structure de l’élément plus complexe. Son typage est donc également sous cette forme.

exemple :

```
<xs:element name="img">
        <xs:complexType>
                <xs:attribute name="src" type="xs:string"/>
        </xs:complexType>
</xs:element>
```

élément avec contenu complexe et attributs

```
<xs:element name="auteur">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nom" type="xs:string"/>
            <xs:element name="prenom" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:token"/>
            <xs:complexType>
</xs:element>
```

Dans ce cas, l'élément auteur contient un élément nom suivi d'un élément prenom et possède un attribut id.