XML Schemas (1)

XML, un langage d'arbres

Année 2011-12

Plan du cours

1. les espaces de noms dans XML
2. les schemas XML

Les espaces de noms

Les espaces de noms

Un document XML peut être composé
- de nombreuses balises différentes,
- qui ont des rôles différents
- et qui doivent être traitées par des applications XML différentes

Les noms simples des éléments ne permettent pas de gérer facilement de tels documents :
- traitement des homonymies sur les éléments
- détection rapide des éléments à traiter pour les applications

Les espaces de noms

Les espaces de noms permettent d'étendre le nommage en qualifiant le nom de l'élément par un URI.

```
<eltRacine>
  <monElt xmlns="http://www.chezmoi.net/mesElements">
    <a> ... </a>
  </monElt>
  <monElt xmlns="http://www.chezlui.net/sesElements">
    <b> ... </b>
  </monElt>
</eltRacine>
```

Les deux élément `monElt` sont différents : ils n'appartiennent pas au même espace de noms

Les espaces de noms

- habituellement, déclaration de tous les espaces de noms nécessaires dans l'élément racine du document
- à chaque espace de nom est associé un préfixe
- les éléments sont préfixés pour indiquer à quel espace de noms ils appartiennent.

```
<moi:eltRacine
     xmlns:moi="http://www.chezmoi.net/mesElements"
     xmlns:lui="http://www.chezlui.net/sesElements">
  <moi:monElt>
    <a> ... </a>
  </moi:monElt>
  <lui:monElt>
    <b> ... </b>
  </lui:monElt>
</moi:eltRacine>
```

Une autre manière de typer les documents

- les *DTDs* : utilisation courante
- pensées pour des documents basés sur le texte
- mais
  - limitées dans leur possibilité de typage des attributs
  - pas de typage pour le contenu des éléments
  - pas de différenciation de niveau entre les éléments
  - peu adaptées à des documents qui contiennent des informations destinées à l'échange entre applications, sauvegarde de configuration ...

Une autre manière de typer les documents

- les schémas XML (première édition Mai 2001, seconde édition octobre 2004) :
  - types prédéfinis riches
  - possibilité de définir de nouveaux types
  - possibilité de typer des éléments
  - possibilité de créer des types

Structure d'un schéma XML

Un schéma XML :

- est un document XML (contrairement à une DTD)
- entête xml
- élément racine : `xs:schema`

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

```
...
</xs:schema>
```

- définition d'élément : `<xs:element>`
- définition d'attribut : `<xs:attribute>`

Un premier exemple

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="recette">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="titre" type="xs:string"/>
    <xs:element name="descriptif" minOccurs="0"
                type="xs:string"/>
    <xs:element name="nb-personnes" type="xs:integer" />
    <xs:element name="ingredient" minOccurs="0"
                type="xs:string" maxOccurs=""unbounded/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Classification

- Les éléments sont définis via *des règles* qui spécifient leur *contenu* et leurs attributs ;
- Le contenu est donné en terme de noeuds fils et de noeuds textuels autorisés ;
- On parle alors de modèle de contenu.
- La première information à donner sur un élément est : est-il de *type simple* (`simpleType`) ou de *type complexe* (`complexType`) ?

Classification

- Un modèle de contenu est vide s'il ne contient ni sous-élément ni noeud textuel.
- Un modèle de contenu est simple s'il ne contient que des noeuds textuels.
- Un modèle de contenu est complexe s'il ne contient que des sous-élts.
- Un modèle de contenu est mixte s'il contient des noeuds textuels et des sous-élts.

Tout cela en dehors de la possibilité d'avoir des attributs, des noeuds commentaires, des instructions de traitement...

Modèles de contenu simple

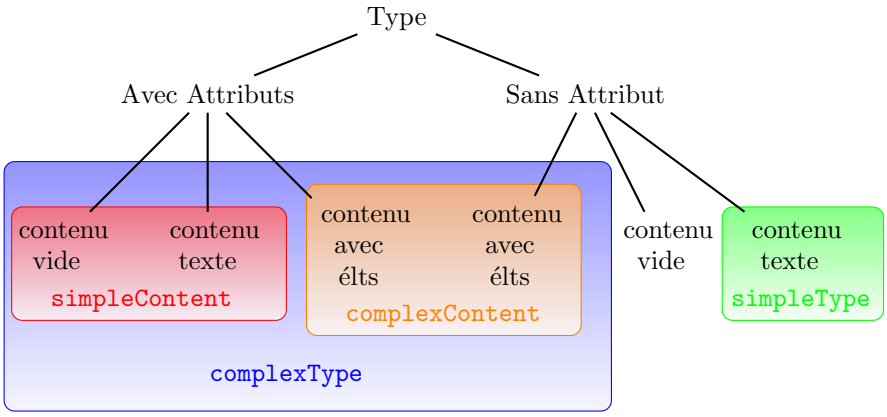
Pour les éléments qui ont un modèle de contenu simple, on fait ensuite deux distinctions sur le type :

- les éléments qui sont simples et n'ont pas d'attributs sont de *type simple*

– tous les autres éléments (avec un autre modèle de contenu ou possédant des attributs) : ils sont de *type complexe*.

Remarque : les attributs sont toujours de type simple.

Types et modèles de contenus



Exemple

```
<?xml version="1.0" encoding="iso-8859-1"?>

<personne
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:nonamespaceSchemaLocation="exemple.xsd">
  <civilite type="Melle" />
  <nom>Dupont</nom>
  <prenom>Jeannette</prenom>
  <age>23</age>
</personne>
```

Exemple

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="civilite">
  <xs:complexType>
    <xs:attribute name="type" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="personne">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="civilite" minOccurs="0"/>
      <xs:element name="nom" type="xs:string"/>
```

```
      <xs:element name="prenom" type="xs:string"/>
      <xs:element name="age"  type="xs:integer"
          minOccurs="0"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Remarques

- l'ordre des règles n'a pas d'importance dans les déclarations (langage déclaratif)
- *les éléments qui sont déclarés juste sous l'élément* `schema` :
  - ont une portée globale
  - peuvent être réutilisés ailleurs dans le schéma par référence
  - ils peuvent être élément racine d'un document instance

Remarques

- *les éléments qui sont déclarés à l'intérieur d'autres éléments* :
  - ont une portée locale ;
  - ne peuvent pas être réutilisés ailleurs ;
- intéressant, car :
  - il est possible de déclarer deux éléments de même nom différemment suivant le contexte
  - les éléments déclarés en interne ne peuvent pas apparaître à la racine d'un document.

Exemple

Le document suivant *est une instance valide* de notre schéma exemple :

```
<?xml version="1.0" encoding="iso-8859-1"?>

<civilite
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   type="Madame"/>
```

Le document suivant n'est pas une instance valide de notre schéma exemple :

```
<?xml version="1.0" encoding="iso-8859-1"?>

<prenom
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  Suzette est aussi un très joli prénom.
</prenom>
```

Remarques

Pour les attributs : même chose.

Le choix entre *une déclaration globale* et *une déclaration locale* est
- parfois une obligation,
- parfois une question de choix de type de programmation.

Une autre possibilité est de laisser les éléments locaux définis localement... mais de déclarer globalement leur type.

Exemple

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="TCivilite">
  <xs:attribute name="type" type="xs:string"/>
</xs:complexType>
<xs:element name="personne">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="civilite"
          type="TCivilite" minOccurs="0"/>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="prenom" type="xs:string"/>
      <xs:element name="age"  type="xs:integer"
          minOccurs="0"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Mauvais Exemple

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="recette">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="titre" type="xs:string"/>
    <xs:element name="descriptif" minOccurs="0"
                type="xs:string"/>
    <xs:element name="nb-personnes" type="xs:integer" />
    <xs:element name="ingredients" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="ingredient" minOccurs="0"
                   maxOccurs="unbounded">
```

Mauvais Exemple

```
        <xs:complexType>
         <xs:sequence>
          <xs:element name="nom" type="xs:string" />
          <xs:element name="quantite" type="xs:integer" />
         </xs:sequence>
        </xs:complexType>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Bon Exemple

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="TIngredients">
  <xs:sequence>
   <xs:element name="ingredient" minOccurs="0"
               maxOccurs="unbounded" type="TIngredient">
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="TIngredient">
  <xs:sequence>
    <xs:element name="nom" type="xs:string" />
    <xs:element name="quantite" type="xs:integer" />
  </xs:sequence>
 </xs:complexType>
```

Bon Exemple (suite)

```
 <xs:element name="recette">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="titre" type="xs:string"/>
    <xs:element name="descriptif" minOccurs="0"
                type="xs:string"/>
    <xs:element name="nb-personnes" type="xs:integer" />
    <xs:element name="ingredients" minOccurs="0"
                type="TIngredients"/>
   </xs:sequence>
```

```
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Différents types de contenus

- Un élément de contenu simple est déclaré par `simpleContent`
  - ne contient que des noeuds textuels
  - pas d'attributs
- tous les autres éléments sont de type complexe !
  - éléments à contenu simple mais avec des attributs
  - éléments à contenu complexe, i.e. qui ont pour fils des éléments (avec ou sans attributs)

Différents types de contenus

- Un élément de contenu complexe est déclaré par `complexType`.
- Il peut devenir un élément de contenu mixte si :
  - on lui donne un contenu.
  - on lui ajoute l'attribut `mixed="true"`. Cet attribut indique que des noeuds textuels peuvent être insérés n'importe où entre les sous-éléments de l'élément ainsi défini.

Exemples

```
<xs:element name="signature">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element ref="nom"/>
      <xs:element ref="prenom"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Exemples

Et un document instance :

```
<signature>
  ceci est un document instance du schéma précédent.
  <nom>Durand</nom>
  avec des zones de texte
  <prenom>Ginette</prenom>
  entre deux éléments
</signature>
```

Différents types de contenus

Un élément de contenu vide est
- un élément sans contenu
- avec ou sans attributs

Exemple : deux éléments vides

```
<xs:element name="difficulte">
  <xs:complexType>
    <xs:attribute name="niveau" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="nomDuChat" />
```

Différents types de contenus

Un élément de contenu simple avec attributs est un type complexe :

```
<xs:complexType name="qte">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="unite" type="token" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Structuration des contenus

Dans un élément complexe :
- la *séquence* impose l'ordre des éléments
- la présence des éléments peut être assouplie par les attributs :
  - `minOccurs` : une occurrence minimale à zéro remplace le `?` des DTDs ;
  - `maxOccurs` : une occurrence maximale à `unbounded` remplace le `+` ou le `*` des DTDs ;
  - `maxOccurs` et `minOccurs` sont par défaut à `1`.

Exemples

```
<xs:element name="vol">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ville" type="xs:string"
                  minOccurs="2" maxOccurs="2" />
    </xs:sequence>
  </xs:complexType>
```

```
</xs:element>

<vol>
  <ville>Paris</ville>
  <ville>Barcelone</ville>
</vol>
```

Exemples

```
<xs:element name="vol">
  <xs:complexType>
    <xs:sequence minOccurs="2" maxOccurs="2" >
      <xs:element name="ville" type="xs:string"/>
      <xs:element name="horaire" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<vol>
  <ville>Paris</ville>
  <horaire>18h15</horaire>
  <ville>Barcelone</ville>
  <horaire>20h05</horaire>
</vol>
```

Structuration des contenus

- le choix avec la balise `<xs:choice>` remplace le `|` des DTDs ;
- une présence non ordonnée avec `<xs:all>`

Attention :

- un schéma XML doit être *complètement déterministe* : le validateur doit pouvoir toujours déterminer par rapport à quelle 'branche' du type il doit valider l'instance ;
- pour `<xs:all>`, on ne peut faire varier les minima et maxima d'occurrences que de 0 à 1.

Exemples

```
<xs:element name="identite">
  <xs:complexType>
    <xs:choice>
      <xs:sequence>
        <xs:element name="nom" type="xs:string"/>
        <xs:element name="prenom" type="xs:string"/>
      </xs:sequence>
      <xs:element name="pseudo" type="xs:string"/>
```

```
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Exemples

Deux exemples de documents instances :

```
<identite>
  <nom>dupond</nom>
  <prenom>jean</prenom>
</identite>

<identite>
  <pseudo>toto</pseudo>
</identite>
```

Exemples

```
<xs:element name="identite2">
  <xs:complexType>
    <xs:all>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="prenom" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<identite2>
  <prenom>georgette</prenom>
  <nom>dupont</nom>
</identite2>
```

Exemple

Pour un élément comme

```
<!ELEMENT elt (elt1|elt2|elt3)* >
```

on écrira :

Exemple

```
<xs:element name="elt">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="elt1"/>
      <xs:element name="elt2"/>
      <xs:element name="elt3"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Création de modèles de type complexe

Pour un type complexe à contenu complexe :

- balise `<xs:complexType>`
- à l'intérieur :
  - liste des sous-éléments ;
  - puis la liste des attributs.

```
<xs:complexType name="Tidentite">
  <xs:sequence>
    <xs:element name="nom" type="xs:string" />
    <xs:element name="prenom" type="xs:string" />
  </xs:sequence>
  <xs:attribute name="civilite" type="xs:string" />
</xs:complexType>
```

Création de modèles de type complexe

Les structures de contrôle sont des connecteurs

- `xs:sequence`, `xs:choice`
- `xs:all` définit une liste non ordonnée. Son caractère non-déterministe déclenche une série de contraintes sur son utilisation :
  - il doit être le groupe de plus haut niveau dans son modèle de contenu (n'apparaît pas dans `xs:group`, ni dans `xs:choice` ou `xs:sequence`)
  - son nombre d'occurrences est 0 ou 1, idem pour ses sous-éléments.

Création de modèles

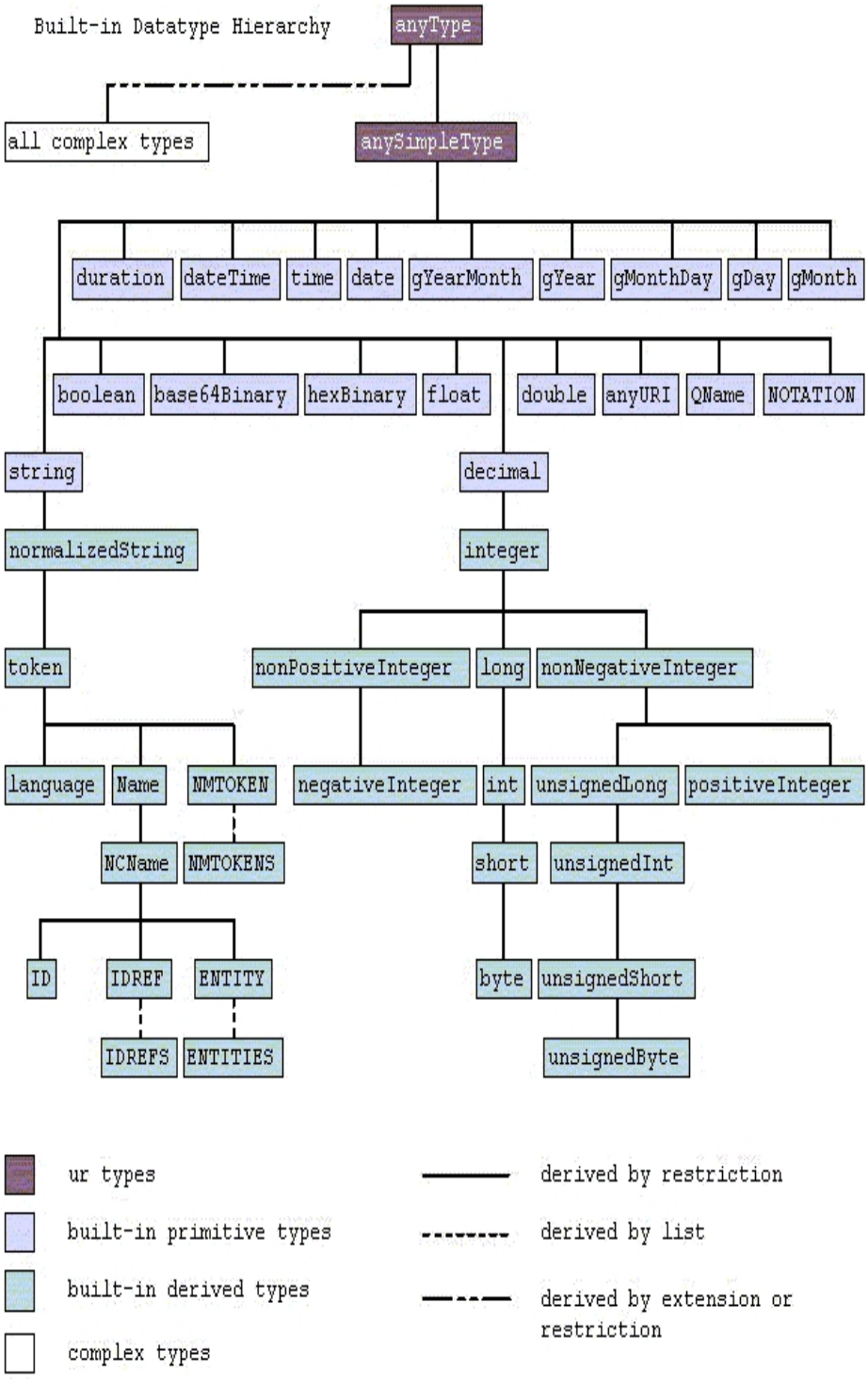
- `xs:attribute` permet de définir un attribut.
  - toujours d'un *type simple*
  - valeur par défaut : `default`
  - `use = (optional | prohibited | required) : optional`
  - déclaration locale ou globale
- `xs:group` permet de définir un groupe d'éléments
- `xs:attributeGroup` permet de définir un groupe d'attributs

Les types prédéfinis

Petit tour d'horizon des types prédéfinis proposés par la recommandation des XML schémas :

- tous les types nécessaires à une bonne traduction à partir de ou vers les bases de données.
- définition des types selon *un graphe* :
- les nouveaux types sont construits par extension/restriction d'autres types : en descendant le graphe, on a des types définis par restriction des types pères (parfois dérivation à partir d'une liste).

Les types prédéfinis


Built-in Datatype Hierarchy
anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
complex types
derived by restriction
derived by list
derived by extension or restriction

Les types prédéfinis

- `anySimpleType` est le type à la racine de la hiérarchie des types. Toute donnée est de type `anySimpleType`
- `boolean`
- les types numériques : `float, double` et `decimal` sont directement en-dessous de `anySimpleType`
- une hiérarchie existe à partir de `decimal` : `integer`, puis en-dessous, les types plus spécifiques `nonPositiveInteger, NonNegativeInteger` ou `long`, etc etc ...

Les types prédéfinis

- `string` : le type le plus haut dans la catégorie des chaînes de caractères. Seul type pour lequel aucune normalisation n'est effectuée, i.e. aucune transformation ou compactage sur les caractères d'espace qui peuvent être contenus dans une donnée de type `xs:string`. En dessous de ce type, on trouvera :
  - `normalizedString` : tous les séparateurs sont remplacés par des blancs, mais il n'y a pas de compactage effectué. Tous les autres types présentés ici sont normalisés et compactés.
  - `token` version compactée de `normalizedString`, `NMTOKEN`, `ID`, `IDREF`, `ENTITY`,...
  - et les versions dérivées par liste : `IDREFS`, `NMTOKENS`, `ENTITIES`

Les types prédéfinis

- `duration` : similaire à `interval` en postgreSQL. Intervalle, ou durée, de temps. S'exprime en années, mois, jours, heures, etc...
- `dateTime` : similaire à `timestamp` en postgreSQL. Combinaison d'une date absolue et d'une valeur absolue.
- `time`, `date`, `gYearMonth`, `gYear`... : correspondent respectivement aux parties horaire, date, mois et année, année d'un `dateTime` (les derniers sont dans le calendrier grégorien)
- plus quelques autres...

Pour terminer, comment énumérer des valeurs...

```
<xs:simpleType name="TUnite">
  <xs:restriction base="xs:token">
    <xs:enumeration value="gr" />
    <xs:enumeration value="kg" />
    <xs:enumeration value="pièce" />
  </xs:restriction>
</xs:simpleType>
```