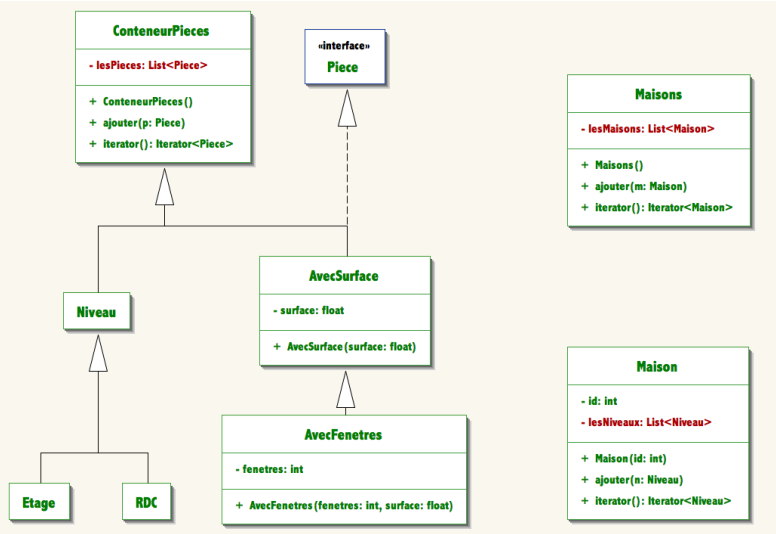
PXML Master MIAGE M1 2011-2012

Cours 4 : Typer les données avec XML-Schema

maisons.xml

```
• <?xml version="1.0" encoding="ISO-8859-1"?>
<maisons>
   <maison id="1">
      <RDC>
         <cuisine surface-m2="12">Evier Inox. Mobilier encastré</cuisine>
         <WC>Lavabo.</WC>
         <séjour surface-m2="38">Cheminée en pierre. Baie vitrée</séjour>
         <bureau surface-m2="14">Bibliothèque</bureau>
         <garage/>
      </RDC>
      <étage>
         <terrasse/>
         <chambre surface-m2="28" fenetre="3">
            <alcove surface-m2="8"/>
         </chambre>
         <chambre surface-m2="18"/>
         <salledeBain surface-m2="15">Douche, baignoire, lavabo</salledeBain>
      </étage>
   </maison>
   <maison id="2">
      <RDC>
         <cuisine surface-m2="12">en ruine</cuisine>
         <garage/>
      </RDC>
      <étage>
         <mirador surface-m2="1">Vue sur la mer</mirador>
         <salledeBain surface-m2="15">Douche</salledeBain>
      </étage>
      </maison>
<maison id="3">
```

Typage



Comparaison entre DTD et XML Schema

- DTD
  - Essentiellement, définition de l'imbrication des éléments, et définition des attributs.
  - Types pauvres
  - Pas de gestion des espaces de nom
  - Pas beaucoup de contraintes sur le contenu d'un document
- XML-Schema
  - Notion de type, indépendamment de la notion d'élément
  - Contraintes d'intégrité d'entité et d'intégrité référentielle, plus précises que les ID/IDREF des DTD.
  - Contraintes de cardinalité
  - Gestion des espaces de noms
  - Réutilisation de mêmes types d'attributs pour des éléments différents
  - Format XML

Lier un schéma à un document

Un peu comme pour une DTD

La balise ouvrante de l'élément racine du fichier XML contient des informations sur le schéma.

```
<exemple xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:noNamespaceSchemaLocation="monSchema.xsd">
```

Lier un schéma à un document

Le schéma `monSchema.xsd` est lui-même un fichier XML et doit donc être associé au schéma qui définit ce qu'on peut utiliser dans un schéma !

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="exemple">
        ...
        </xsd:element>
...
</xsd:schema>
```

ici on a choisi `xsd` comme préfixe ; on utilisera aussi `xs`.

Contenu d'un schéma

Un XML-Schema est composé de

- Définitions de types
- Déclaration d'attributs
- Déclaration d'éléments
- Définitions de groupes d'attributs
- Définitions de groupes de modèles
- Définitions de contraintes d'unicité ou de clés
- Déclarations de notations
- ...

Définitions de types

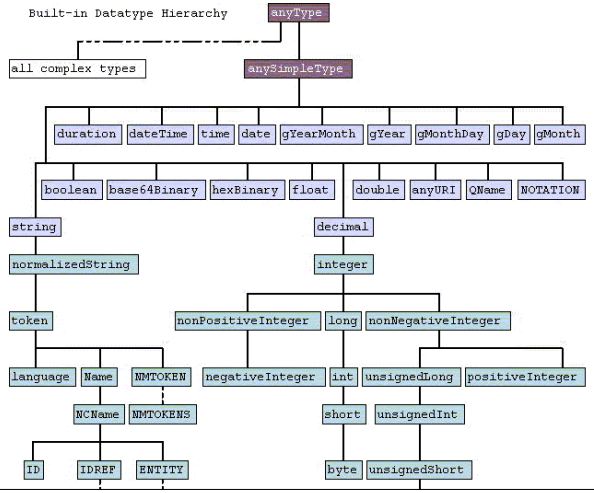
Typage

- Existence de types prédéfinis : hiérarchie de types, dont la racine est le type anyType.
- Possibilité de définir de nouveaux types
- Distinction types simples et types complexes

1. Les types simples sont utilisés pour les déclarations d'attributs, les déclarations d'éléments dont le contenu se limite à des données atomiques, et qui n'ont pas d'attributs.
2. Les types complexes s'utilisent dans tous les autres cas.

Les types simples prédéfinis

- http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html\#built-in-datatypes



Exemple (XHTML) de définition d'un type simple

```
<xs:simpleType name="tabindexNumber">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="32767"/>
  </xs:restriction>
</xs:simpleType>
```

Les types simples

Un type simple est caractérisé par 3 ensembles : son espace de valeurs, son espace lexical, ses facettes.

1. espace de valeurs = les valeurs autorisées pour ce type

2. espace lexical = syntaxe des littéraux. Par exemple `100` et `1.0E2` sont deux littéraux qui représentent la même valeur, de type `float`.

3. facette = propriété disponible sur ce type. Toutes les facettes ne s'appliquent pas à tous les types.

Les types simples

Un type simple peut être un type atomique, type union ou type list

- type atomique : comme en SQL, on retrouve des types caractères, numériques, temporels, . . .

- `union` : union de plusieurs types (i.e. union des espaces de valeurs, des espaces lexicaux).

- `list` : la valeur d'une liste est une séquence de valeurs atomiques. Un littéral liste est une séquence de littéraux du type atomique, séparés par des espaces.

Facettes

Les facettes fondamentales définissent le type de données. Il en existe 5 disponibles sur tous les types :

1. `equal`
2. `ordered` : `false`, `partial` ou `ordered`, indique si l'espace de valeurs est ordonné
3. `bounded` : valeur booléenne, indique si l'espace de valeurs est borné
4. `cardinality` : indique si l'espace de valeur est `finite` ou `countably infinite`
5. `numeric` : valeur booléenne, indique si l'espace de valeurs est numérique

Facettes de contrainte

Les facettes de contraintes sont optionnelles et permettent de restreindre l'espace des valeurs. Elles ne sont pas toutes disponibles sur tous les types.

1. `length` : longueur, qui peut être le nombre de caractères pour un type `string`, le nombre d'éléments pour un type `list`, le nombre d'octets pour un type binaire.
2. `maxLength` : longueur maximale
3. `minLength` : longueur minimale
4. `pattern` : expression régulière qui décrit les littéraux du type (donc les valeurs)
5. `maxExclusive` : valeur maximale au sens strict (<)
6. `maxInclusive` : valeur maximale au sens large (≤)
7. `minExclusive` : valeur minimale au sens strict
8. `minInclusive` : valeur minimale au sens large

Facettes de contrainte

9. `enumeration` : énumération des valeurs possibles
10. `fractionDigits` : nombre maximal de décimales après le point
11. `totalDigits` : nombre maximal de chiffres pour une valeur décimale
12. `whiteSpace` : règle pour la normalisation des espaces dans une chaîne

Facettes de contrainte

- Pour un type `union` : Les seules facettes de contrainte possibles sont `pattern` et `enumeration`.
- Pour un type `list` : Les facettes de contraintes disponibles sont `length`, `maxLength`, `minLength`, `enumeration`, `pattern`, `whiteSpace`.

Les types simples créés par l'utilisateur

On dérive un type simple à partir d'un autre type. Il existe 3 façons de dériver un type simple :

1. par restriction : on crée un type dont l'espace de valeurs est inclus dans l'espace de valeurs d'un type existant. Pour cela, on utilise les facettes pour restreindre l'espace des valeurs,
2. par liste,
3. par union.

Exemples (XHTML) de restrictions d'un type atomique

```
<xs:simpleType name="tabindexNumber">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="32767"/>
  </xs:restriction>
</xs:simpleType>
```

Exemples de restrictions d'un type atomique

```
<xsd:simpleType name="jour-de-la-semaine">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="lundi"/>
    <xsd:enumeration value="mardi"/>
    <xsd:enumeration value="mercredi"/>
    <xsd:enumeration value="jeudi"/>
    <xsd:enumeration value="vendredi"/>
    <xsd:enumeration value="samedi"/>
    <xsd:enumeration value="dimanche"/>
  </xsd:restriction>
</xsd:simpleType>
```

Exemples (XHTML) de restrictions d'un type atomique

```
<!-- single or comma-separated list of media descriptors -->
<xs:simpleType name="MediaDesc">
  <xs:restriction base="xs:string">
    <xs:pattern value="[^,]+(,\s*[^,]+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Exemple de type `union`

```
<!-- permet de faire <font size="34"> ou <font size="medium">-->

<xsd:simpleType name="fontSize">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:positiveInteger">
        <xsd:minInclusive value="8"/>
        <xsd:maxInclusive value="72"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="small"/>
        <xsd:enumeration value="medium"/>
        <xsd:enumeration value="large"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

Exemples de types `list`

```
<xsd:simpleType name="sizes">
  <xsd:list itemType="xsd:decimal"/>
</xsd:simpleType>
```

Permet d'écrire `<dimensions>12 5.6 67</dimensions>`

```
<xsd:simpleType name='listOfString'>
  <xsd:list itemType='xsd:string'/>
</xsd:simpleType>
```

Attention, la liste ci-dessous, de type `listOfString`, a 18 éléments, pas 3 (le séparateur est l'espace)

```
<someElement>
this is not list item 1
this is not list item 2
this is not list item 3
</someElement>
```

Exemple de restriction d'un type `list`

```
<xs:simpleType name='myList'>
  <xs:list itemType='xs:integer'/>
</xs:simpleType>

<xs:simpleType name='myRestrictedList'>
  <xs:restriction base='myList'>
    <xs:pattern value='123 (\d+\s)*456'/>
  </xs:restriction>
</xs:simpleType>

<someElement>123 456</someElement>
<someElement>123 987 456</someElement>
<someElement>123 987 567 456</someElement>
```

Les types complexes

Type complexe

- La définition d'un type complexe est un ensemble de déclarations d'attributs et un type de contenu.
- Extension d'un type : ajout d'un attribut ou d'un sous-élément à un type existant.
- Type complexe à contenu simple : type d'un élément dont le contenu est de type simple mais qui possède un attribut (un type simple n'a pas d'attribut). On le définit par extension d'un type simple par ajout d'un attribut.
- Type complexe à contenu complexe : permet de déclarer des sous-éléments et des attributs. On le construit à partir de rien ou on le définit par restriction d'un type complexe, ou par extension d'un type.

Exemple de type complexe à contenu simple

```
<!-- type length0 dont le contenu a pour type simple
xs:nonNegativeInteger
et qui possde un attribut unit de type xs:NMTOKEN -->
<xs:complexType name="length0">
  <xs:simpleContent>
    <xs:extension base="xs:nonNegativeInteger">
      <xs:attribute name="unit" type="xs:NMTOKEN"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<!-- exemple de déclaration d'élément width de type length0 -->
<xs:element name="width" type="length0"/>

<!-- exemple d'élément width -->
<width unit="cm">25</width>
```

Exemples de types complexes à contenu complexe (1)

Construit à partir de zéro.

```
<xsd:complexType name="type-duree">
   <xsd:sequence>
      <xsd:element name="du" type="xsd:date"/>
      <xsd:element name="au" type="xsd:date"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:element name="duree" type="type-duree"/>

<duree>
    <du>2010-09-13</du>
    <au>2010-09-25</au>
</duree>
```

Exemples de types complexes à contenu complexe (1)

Construit à partir de zéro.

```
<xs:complexType name="personName">
  <xs:sequence>
    <xs:element name="title" minOccurs="0"/>
    <xs:element name="forename" minOccurs="0"
                                maxOccurs="unbounded"/>
    <xs:element name="surname"/>
  </xs:sequence>
</xs:complexType>
```

Exemple de type complexe à contenu complexe (2)

Obtenu par extension d'un type simple ou complexe

```
<xs:complexType name="extendedName">
  <xs:complexContent>
    <xs:extension base="personName">
      <xs:sequence>
        <xs:element name="generation" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="addressee" type="extendedName"/>

<addressee>
  <forename>Albert</forename>
  <forename>Arnold</forename>
  <surname>Gore</surname>
  <generation>Jr</generation>
</addressee>
```

Exemple de type complexe à contenu complexe (3)

Obtenu par restriction d'un type complexe

```
<xs:complexType name="length2">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:element name="size" type="xs:nonNegativeInteger"/>
        <xs:element name="unit" type="xs:NMTOKEN"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Exemple de type complexe à contenu complexe (4)

Obtenu également par restriction d'un type complexe

```
<xs:complexType name="simpleName">
  <xs:complexContent>
    <xs:restriction base="personName">
      <xs:sequence>
        <xs:element name="forename" minOccurs="1" maxOccurs="1"/>
        <xs:element name="surname"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:element name="who" type="simpleName"/>

<who>
  <forename>Barack</forename>
  <surname>Obama</surname>
</who>
```

Groupe de Modèles

Pour construire un type complexe, on peut utiliser des constructeurs de groupes de modèles. On a déjà rencontré le constructeur `sequence`, il existe 3 constructeurs de groupes de modèles :

- `sequence` : les éléments d'une séquence doivent apparaître dans l'ordre où ils sont déclarés.
- `choice` : un seul élément parmi ceux du `choice` doit apparaître
- `all` : les éléments contenus dans un `all` peuvent apparaître dans n'importe quel ordre.

Exemple d'utilisation de `choice`

```
<xsd:complexType name="SurfaceOuVolume">
  <xsd:choice>
    <xsd:element name="surface" type="length0"/>
    <xsd:element name="volume" type="length0"/>
  </xsd:choice>
</xsd:complexType>

<!-- element occupation du type SurfaceouVolume -->
<occupation>
  <surface unit="cm2">453</surface>
</occupation>
```

Exemple d'utilisation de `all`

```
<xsd:complexType name="Identite">
  <xsd:all>
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="prenom" type="xsd:string"/>
    <xsd:element name="datenaiss" type="xsd:date"/>
  </xsd:all>
</xsd:complexType>
```

Exemple d'utilisation de `all`

```
<!-- elements de type Identite -->
<identite>
  <nom>meurisse</nom>
  <prenom>paul</prenom>
  <datenaiss>1912-12-21</datenaiss>
</identite>

<identite>
  <datenaiss>1948-05-31</datenaiss>
  <nom>bonham</nom>
  <prenom>john</prenom>
</identite>
```

Déclaration d'attribut

- Un attribut est de type simple, par exemple un type prédéfini, une énumeration, une liste ...
- On peut préciser le caractère obligatoire ou facultatif de l'attribut (attribut `use`)
- On peut lui donner une valeur par défaut (attribut `default`) ou une valeur fixe (attribut `fixed`)

Exemples de déclarations d'attributs

```
<xsd:attribute name="size"
                      type="fontSize"
                      use="required"/>

<xs:attribute name="jour" default="lundi"
                    type="jour-de-la-semaine"/>

<xs:attribute name="version"
                    type="xs:number"
                    fixed="1.0"/>
```

Déclaration d'élément

- On définit le contenu de l'élément grâce aux types
- Lorsqu'on n'attribue pas de type à un élément, il est considéré comme de type `xs:anyType` et peut donc contenir n'importe quoi
- Pour un élément de contenu mixte (texte et sous-éléments), on utilise l'attribut `mixed` de `xs:complexType`.
- Pour un élément vide, on définit un type complexe qui n'a pas de sous-élément

Exemple 1 : éléments de contenu simple

```
<xsd:element name="long" type="length0"/>

<xsd:element name="nom" type="xsd:string"/>

<xsd:element name="font">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="size" type="fontSize"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Exemple 1 : éléments de contenu simple

Permet d'écrire dans le document XML

```
<long unit="cm">45</long>

<nom>durand</nom>

<font size="medium">machin</font>
```

Exemple 2 : élément de contenu mixte

```
<xsd:element name="toto">
  <xsd:complexType mixed="true">
    <xsd:sequence>
     <xsd:element ref="font" minOccurs="1" maxOccurs="2"/>
     <xsd:element ref="long"/>
     <xsd:element name="long" type="length1"/>
     <xsd:element name="media" type="MediaDesc"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Exemple 2 : instance du schéma précédent

```
<toto>
  <font size="45">blabla</font>
  contenu mixte donc on peut avoir du texte
  <font size="medium">machin</font>
  <!-- les 2 elements long qui suivent -->
  <!-- ne sont pas du meme type -->
  <!-- c'est impossible pour une DTD -->
  <long unit="cm">45</long>
  <long><size>34</size><unit>cm</unit></long>
  et patati et patata
  <media>screen</media>
</toto>
```

Exemple 3 : élément vide

L'élément `br` en XHTML contient uniquement des attributs. Les types utilisés sont définis dans le schéma de XHTML.

```
<xs:element name="br">
  <xs:complexType>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="class" type="xs:NMTOKENS"/>
    <xs:attribute name="style" type="StyleSheet"/>
    <xs:attribute name="title" type="Text"/>
  </xs:complexType>
</xs:element>
```

```
<xsd:element name="vide">
  <xsd:complexType/>
</xsd:element>
```

Eléments, Attributs et Types

- On doit associer les types et les noms d'éléments ou d'attributs.
- Un type peut contenir des déclarations d'éléments ou d'attributs
- Un élément ou attribut peut contenir une déclaration de type
- On distingue : déclaration locale (dans une autre déclaration) ou globale (fils de la racine `xs:schema`)
- On peut faire référence à un type, élément, attribut déjà défini grâce à l'attribut `ref`

Eléments, Attributs et Types

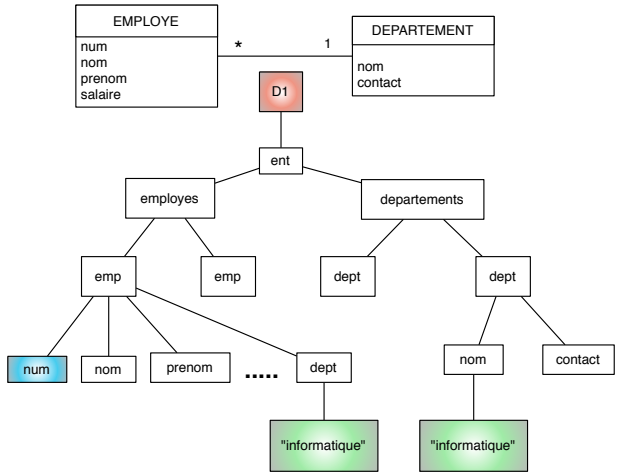
```
<xsd:element name="trimestre">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="type-duree">
        <xsd:attribute name="num" type="type-trimestre"
                       use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="trimestres">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="trimestre" minOccurs="4" maxOccurs="4"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Contraintes d'intégrité

- Modèle relationnel :
  - Contrainte d'unicité `UNIQUE`, de clef primaire (unicité et existence) `PRIMARY KEY`, de clef étrangère `FOREIGN KEY` ... `REFERENCES` ....
  - Une clef primaire est définie pour une relation, et elle composée d'un ou plusieurs attributs.
  - Une clef étrangère fait référence à des attributs d'une relation précise.
- Avec une DTD, on peut définir des identifiants (attribut de type `ID`), et des références d'identifiant (de type `IDREF`). L'existence ou non est définie en choisissant `IMPLIED` ou `REQUIRED`. Mais
  - les références ne sont pas typées (on ne sait pas à quel type de nœud on fait référence)
  - un identifiant est global au document
- XML-Schema : on se rapproche du modèle relationnel

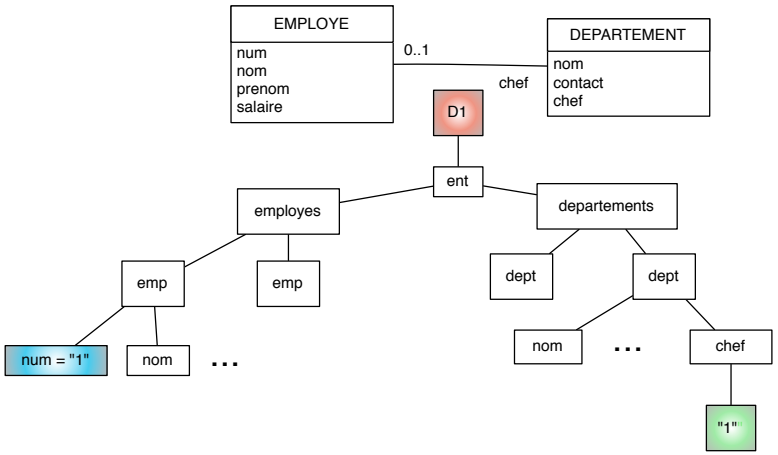
Exemple

Tout employé appartient à un département :



Exemple (suite)

Un département a au plus un chef



Exemple (suite)

On déclare un type pour un département, et pour une séquence de départements

```
<xsd:complexType name="TypeDept">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="contact" type="xsd:string"/>
    <xsd:element name="chef" type="xsd:int" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SeqDept">
  <xsd:sequence>
    <xsd:element name="dept" type="TypeDept"
maxOccurs="unbounded"
        minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Exemple (suite)

Même chose pour les employés.

```
<xsd:complexType name="TypeEmploye">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string"/>
    <xsd:element name="prenom" type="xsd:string"/>
    <xsd:element name="salaire" type="xsd:decimal"/>
  <xsd:element name="dept" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="num" type="xsd:int"/>
</xsd:complexType>
```

Exemple (suite)

```
<xsd:complexType name="SeqEmploye">
  <xsd:sequence>
    <xsd:element name="emp"
                 type="TypeEmploye"
                 maxOccurs="unbounded"
                 minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Exemple (suite)

Déclaration des éléments `employes` et `departements` avec les clefs primaires pour les éléments `emp` et `dept`.

```
<xsd:element name="employes" type="SeqEmploye">
  <xsd:key name="clefEmp">
    <xsd:selector xpath="emp"/>
    <xsd:field xpath="@num"/>
  </xsd:key>
</xsd:element>

<xsd:element name="departements" type="SeqDept">
  <xsd:key name="clefDept">
    <xsd:selector xpath="dept"/>
    <xsd:field xpath="nom"/>
  </xsd:key>
</xsd:element>
```

Exemple (suite)

L'élément racine du document avec les clefs étrangères.

```
<xsd:element name="ent">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="employes"/>
      <xsd:element ref="departements"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:keyref name="refDept" refer="clefDept">
    <xsd:selector xpath="employes/emp"/>
    <xsd:field xpath="dept"/>
  </xsd:keyref>
  <xsd:keyref name="refEmp" refer="clefEmp">
    <xsd:selector xpath="departements/dept"/>
    <xsd:field xpath="chef"/>
  </xsd:keyref>
</xsd:element>
```

Explications

- L'endroit où l'on déclare une contrainte détermine la zone où elle doit être vérifiée. En effet, la contrainte s'applique à l'intérieur de l'élément "contexte" de cette contrainte.
- Le `selector` détermine pour quel élément c'est une clef. C'est un chemin XPath qui désigne un ensemble de noeuds (appelés noeuds cibles) contenus dans l'élément contexte de la contrainte.
- Les `field` qui suivent donnent les composants (attributs ou éléments) de la clef. Chaque composant désigne un noeud unique (élément ou attribut), par rapport à 1 noeud cible désigné par le `selector`. De plus, chaque composant doit être de type simple.

Explications

- La syntaxe des chemins XPath est réduite, voir la norme pour plus de précision.
- Dans une clef étrangère, l'attribut `refer` indique à quelle clef primaire elle fait référence.
- Pour définir une contrainte d'unicité, remplacer `xsd:key` par `xsd:unique`.