Bases de données et web – CM3

L3 Informatique & L3 MIASHS

XML Schema

XML Schema a la même vocation que DTD. Un fichier XML Schema permet de spécifier les règles sémantiques des documents XML qui lui sont associés. Voici une comparaison entre DTD et XML Schema :

- ***DTD***
  - 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 même type d'attributs pour des éléments différents.
  - Format XML.

Entre d'autres termes, XML Schema est plus moderne, possède une plus grande expressivité et est plus utilisé que DTD.

Lier un schéma à un document

La balise ouvrante de l'élément racine du fichier XML contient les informations sur le schéma à utiliser pour le document.

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

Lier un schéma à un document

La balise ouvrante de l'élément racine du fichier XML contient les informations sur le schéma à utiliser pour le document.

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

On indique dans un premier temps que l'on va utiliser la *bibliothèque de fonctions* de XML Schema.

Lier un schéma à un document

La balise ouvrante de l'élément racine du fichier XML contient les informations sur le schéma à utiliser pour le document.

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

On indique dans un premier temps que l'on va utiliser la *bibliothèque de fonctions* de XML Schema.

On utilise ensuite cette espace de nom pour déclarer le schéma associé à notre fichier XML.

Contenu d'un schéma

Le schéma `maisons.xsd` est lui-même un fichier XML et doit donc être associé à la *bibliothèque de fonctions* XML Schema afin de pouvoir en utiliser les fonctionnalités.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3      <xs:element name="maisons">
4          ...
5      </xs:element>
6      ...
7  </xs:schema>
```

Une fois qu'on a déclarer l'utilisation des balises XML Schema dans le fichier `xsd`, on peut utiliser les balises mots-clefs, comme par exemple ici ***xs:element***.

Contenu d'un schéma

Le schéma `maisons.xsd` est lui-même un fichier XML et doit donc être associé à la *bibliothèque de fonctions* XML Schema afin de pouvoir en utiliser les fonctionnalités.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3      <xs:element name="maisons">
4          ...
5      </xs:element>
6      ...
7  </xs:schema>
```

Une fois qu'on a déclarer l'utilisation des balises XML Schema dans le fichier `xsd`, on peut utiliser les balises mots-clefs, comme par exemple ici ***xs:element***.

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 clefs.
- ...

Toutes ces règles sont décrites dans le fichier XML Schema (qui est un document XML) en utilisant des balises mots-clefs XML Schema.

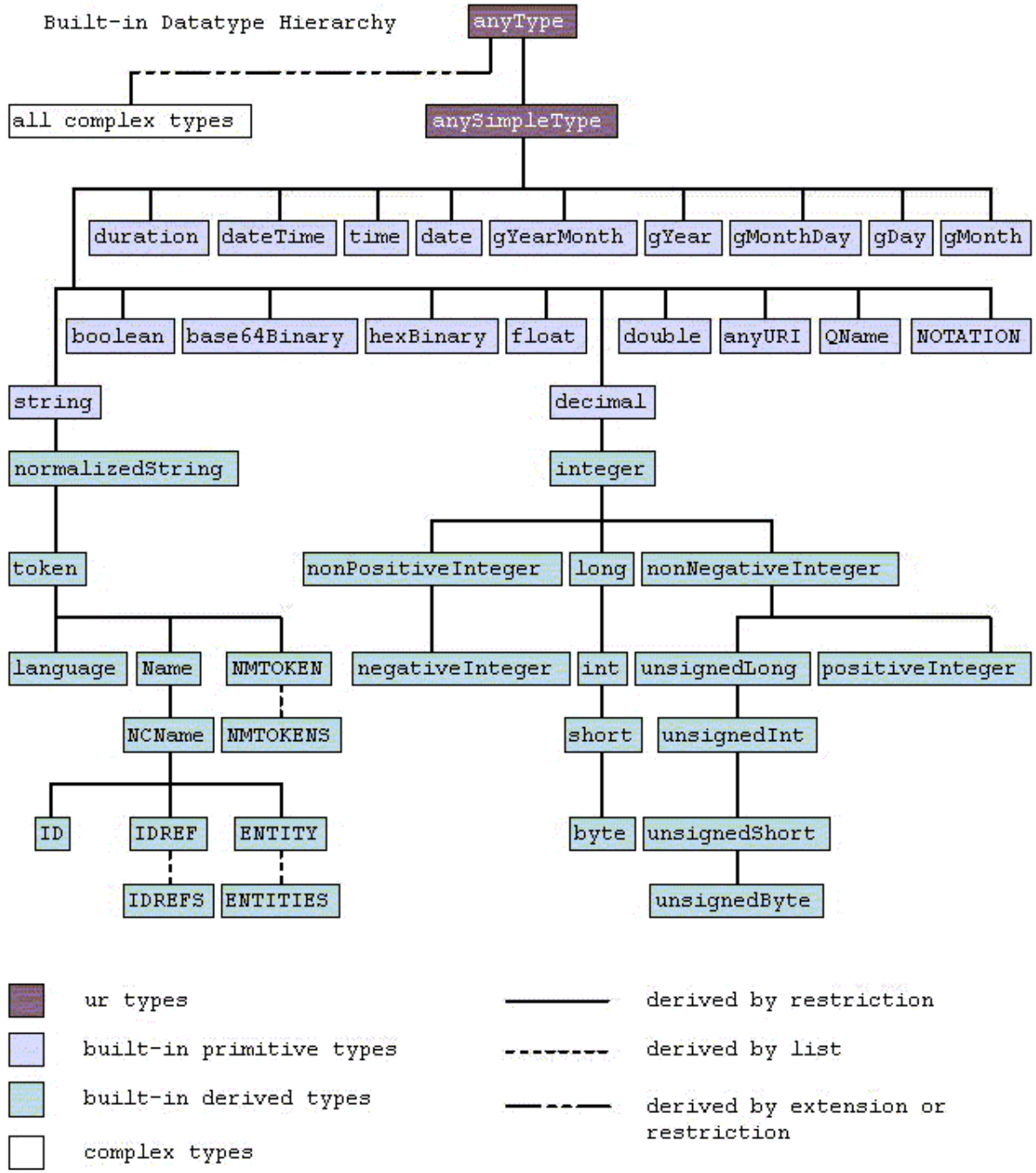
Définitions de types

Il existe des types prédéfinis en XML Schema. Les types sont hiérarchisés sous forme d'arbre dont la racine est ***anyType***.

Il est possible de définir de nouveaux types, par exemple comme on le ferait dans un langage de programmation.

Il y a une distinction entre les ***types simples*** et les ***types complexes*** :

- Les types simples sont utilisés pour les déclaration d'attributs et les déclarations d'éléments dont le contenu se limite à des données atomiques (pas d'élément), sans attribut.
- Les types complexes s'utilisent dans tous les autres cas.


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

Exemple d'utilisation d'un type simple prédéfini

Voici un extrait d'un document XML décrivant un ensemble de maisons.

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <maisons>
4     <maison id="1">
5        <RDC>
6           <cuisine surface-m2="12">Evier Inox. Mobilier encastré</cuisine>
7           <WC>Lavabo.</WC>
8           <séjour surface-m2="38">Cheminée en pierre. Baie vitrée</séjour>
9           <bureau surface-m2="14">Bibliothèque</bureau>
10          <garage/>
11       </RDC>
```

Dans le fichier XML Schema on décrit le type de l'attribut ***surface-m2*** et ***fenetre*** de la manière suivante :

```
<xs:attribute name="surface-m2" type="xs:decimal"/>
<xs:attribute name="fenetre" type="xs:positiveInteger"/>
```

Nous choisissons ici de spécifier que l'attribut `surface-m2` est un nombre décimal prédéfini par XML Schema en indiquant `xs:decimal`. Pour l'attribut `fenetre` on utilise un entier strictement positif avec `xs:positiveInteger`.

Les types simples

Un type simple est caractérisé par son espace de valeurs, son espace lexical et ses facettes :

- ***Espace de valeurs*** : les valeurs autorisées pour ce type.
- ***Espace lexical*** : syntaxe des littéraux. Par exemple 100 et 1.0E2 sont deux littéraux qui représentent la même valeur, de type float.
- ***Facette*** : des restrictions supplémentaires sur le type.

Chacun de ces trois éléments est potentiellement différent pour chaque type prédéfini de XML Schema. Il est également possible de spécifier ces caractéristiques lors de la création d'un type personnalisé.

Les facettes

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

- ***equal***.
- ***ordered*** : ***false***, ***partial*** ou ***ordered***.
- ***bounded*** : valeur booléenne, indique si l'espace de valeurs est borné.
- ***cardinality*** : indique si l'espace de valeur est ***finite*** ou ***countably infinite***.
- ***numeric*** : valeur booléenne, indique si l'espace de valeurs est numérique.

Ces facettes fondamentales permettent de comprendre finement la définition des types standards de XML Schema. Lors de la création de nouveau type nous nous baserons sur des types préexistant, sans avoir besoin de spécifier chacune de ces facettes fondamentales.

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.

- ***length*** : spécifie le nombre de caractères pour un type `string`, le nombre d'éléments pour un type `list`, le nombre d'octets pour un type binaire.
- ***maxLength*** : longueur maximale.
- ***minLength*** : longueur miniale.
- ***pattern*** : expression régulière qui décrit les littéraux du type (et donc les valeurs possibles).
- ***maxExclusive*** : valeur maximale au sens strict ($<$).
- ***maxInclusive*** : valeur maximale au sens large ($\leq$).
- ***minExclusive*** : valeur minimale au sens strict.
- ***minInclusive*** : valeur minimale au sens large.
- ***enumeration*** : énumération des valeurs possibles.
- ***fractionDigits*** : nombre maximal de décimales après le point.
- ***totalDigits*** : nombre maximal de chiffres pour une valeur décimale.
- ***whiteSpace*** : règle pour la normalisation des espaces dans une chaîne de caractères.

Les types simples créés par l'utilisateur

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

- Par restriction : on créer 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.
- Par liste : on créer un type on indiquant précisément la liste des valeurs possibles.
- Par union : on créer un type en indiquant que le type créé est issu de l'union (ensembliste) d'autres types.

Exemples de restrictions d'un type atomique

```
5  <xs:simpleType name="indiceTableau">
6      <xs:restriction base="xs:nonNegativeInteger">
7          <xs:minInclusive value="0"/>
8          <xs:maxInclusive value="32767"/>
9      </xs:restriction>
10 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de la valeur minimum (valeur min incluse). Ici 0.
- Déclaration de la valeur maximum (valeur max incluse). Ici 32767.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
5  <xs:simpleType name="indiceTableau">
6      <xs:restriction base="xs:nonNegativeInteger">
7          <xs:minInclusive value="0"/>
8          <xs:maxInclusive value="32767"/>
9      </xs:restriction>
10 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de la valeur minimum (valeur min incluse). Ici 0.
- Déclaration de la valeur maximum (valeur max incluse). Ici 32767.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
5  <xs:simpleType name="indiceTableau">
6      <xs:restriction base="xs:nonNegativeInteger">
7          <xs:minInclusive value="0"/>
8          <xs:maxInclusive value="32767"/>
9      </xs:restriction>
10 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de la valeur minimum (valeur min incluse). Ici 0.
- Déclaration de la valeur maximum (valeur max incluse). Ici 32767.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
5  <xs:simpleType name="indiceTableau">
6      <xs:restriction base="xs:nonNegativeInteger">
7          <xs:minInclusive value="0"/>
8          <xs:maxInclusive value="32767"/>
9      </xs:restriction>
10 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de la valeur minimum (valeur min incluse). Ici 0.
- Déclaration de la valeur maximum (valeur max incluse). Ici 32767.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
5  <xs:simpleType name="indiceTableau">
6      <xs:restriction base="xs:nonNegativeInteger">
7          <xs:minInclusive value="0"/>
8          <xs:maxInclusive value="32767"/>
9      </xs:restriction>
10 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de la valeur minimum (valeur min incluse). Ici 0.
- Déclaration de la valeur maximum (valeur max incluse). Ici 32767.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
5  <xs:simpleType name="indiceTableau">
6      <xs:restriction base="xs:nonNegativeInteger">
7          <xs:minInclusive value="0"/>
8          <xs:maxInclusive value="32767"/>
9      </xs:restriction>
10 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de la valeur minimum (valeur min incluse). Ici 0.
- Déclaration de la valeur maximum (valeur max incluse). Ici 32767.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
12 <xs:simpleType name="jour-de-de-la-semaine">
13    <xs:restriction base="xs:token">
14       <xs:enumeration value="lundi"/>
15       <xs:enumeration value="mardi"/>
16       <xs:enumeration value="mercredi"/>
17       <xs:enumeration value="jeudi"/>
18       <xs:enumeration value="vendredi"/>
19       <xs:enumeration value="samedi"/>
20       <xs:enumeration value="dimanche"/>
21    </xs:restriction>
22 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'énumération. Seuls les données présentes dans l'énumération peuvent être utilisées avec ce type.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
12  <xs:simpleType name="jour-de-de-la-semaine">
13      <xs:restriction base="xs:token">
14          <xs:enumeration value="lundi"/>
15          <xs:enumeration value="mardi"/>
16          <xs:enumeration value="mercredi"/>
17          <xs:enumeration value="jeudi"/>
18          <xs:enumeration value="vendredi"/>
19          <xs:enumeration value="samedi"/>
20          <xs:enumeration value="dimanche"/>
21      </xs:restriction>
22  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'énumération. Seuls les données présentes dans l'énumération peuvent être utilisées avec ce type.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
12  <xs:simpleType name="jour-de-de-la-semaine">
13      <xs:restriction base="xs:token">
14          <xs:enumeration value="lundi"/>
15          <xs:enumeration value="mardi"/>
16          <xs:enumeration value="mercredi"/>
17          <xs:enumeration value="jeudi"/>
18          <xs:enumeration value="vendredi"/>
19          <xs:enumeration value="samedi"/>
20          <xs:enumeration value="dimanche"/>
21      </xs:restriction>
22  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'énumération. Seuls les données présentes dans l'énumération peuvent être utilisées avec ce type.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
12  <xs:simpleType name="jour-de-de-la-semaine">
13      <xs:restriction base="xs:token">
14          <xs:enumeration value="lundi"/>
15          <xs:enumeration value="mardi"/>
16          <xs:enumeration value="mercredi"/>
17          <xs:enumeration value="jeudi"/>
18          <xs:enumeration value="vendredi"/>
19          <xs:enumeration value="samedi"/>
20          <xs:enumeration value="dimanche"/>
21      </xs:restriction>
22  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'énumération. Seuls les données présentes dans l'énumération peuvent être utilisées avec ce type.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
12  <xs:simpleType name="jour-de-de-la-semaine">
13      <xs:restriction base="xs:token">
14          <xs:enumeration value="lundi"/>
15          <xs:enumeration value="mardi"/>
16          <xs:enumeration value="mercredi"/>
17          <xs:enumeration value="jeudi"/>
18          <xs:enumeration value="vendredi"/>
19          <xs:enumeration value="samedi"/>
20          <xs:enumeration value="dimanche"/>
21      </xs:restriction>
22  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'énumération. Seuls les données présentes dans l'énumération peuvent être utilisées avec ce type.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
24 <xs:simpleType name="id-special">
25     <xs:restriction base="xs:string">
26         <xs:pattern value="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
27     </xs:restriction>
28 </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'expression régulière qui va définir notre chaîne de caractère. Ici : 3 chiffres, un tiret, 2 chiffres, un tiret, 4 chiffres.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
24  <xs:simpleType name="id-special">
25      <xs:restriction base="xs:string">
26          <xs:pattern value="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
27      </xs:restriction>
28  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'expression régulière qui va définir notre chaîne de caractère. Ici : 3 chiffres, un tiret, 2 chiffres, un tiret, 4 chiffres.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
24  <xs:simpleType name="id-special">
25      <xs:restriction base="xs:string">
26          <xs:pattern value="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
27      </xs:restriction>
28  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'expression régulière qui va définir notre chaîne de caractère. Ici : 3 chiffres, un tiret, 2 chiffres, un tiret, 4 chiffres.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
24  <xs:simpleType name="id-special">
25      <xs:restriction base="xs:string">
26          <xs:pattern value="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
27      </xs:restriction>
28  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'expression régulière qui va définir notre chaîne de caractère. Ici : 3 chiffres, un tiret, 2 chiffres, un tiret, 4 chiffres.
- Fermeture des balises du type nouvellement créé.

Exemples de restrictions d'un type atomique

```
24  <xs:simpleType name="id-special">
25      <xs:restriction base="xs:string">
26          <xs:pattern value="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
27      </xs:restriction>
28  </xs:simpleType>
```

- Déclaration du nom de notre type issu d'une restriction d'un type atomique.
- Déclaration du type atomique utilisé comme base.
- Déclaration de l'expression régulière qui va définir notre chaîne de caractère. Ici : 3 chiffres, un tiret, 2 chiffres, un tiret, 4 chiffres.
- Fermeture des balises du type nouvellement créé.

Exemple de type *union*

```
30  <xs:simpleType name="taillePolice">
31     <xs:union>
32        <xs:simpleType>
33           <xs:restriction base="xs:positiveInteger">
34              <xs:minInclusive value="8"/>
35              <xs:maxInclusive value="72"/>
36           </xs:restriction>
37        </xs:simpleType>
38
39        <xs:simpleType>
40           <xs:restriction base="xs:NMTOKEN">
41              <xs:enumeration value="petit"/>
42              <xs:enumeration value="moyen"/>
43              <xs:enumeration value="grand"/>
44           </xs:restriction>
45        </xs:simpleType>
46     </xs:union>
47  </xs:simpleType>
```

- Déclaration du nom de notre type basé sur une union de deux types.
- Déclaration du premier type de l'union.
- Déclaration du deuxième type de l'union.

Exemple de type *union*

```
30  <xs:simpleType name="taillePolice">
31      <xs:union>
32          <xs:simpleType>
33              <xs:restriction base="xs:positiveInteger">
34                  <xs:minInclusive value="8"/>
35                  <xs:maxInclusive value="72"/>
36              </xs:restriction>
37          </xs:simpleType>
38
39          <xs:simpleType>
40              <xs:restriction base="xs:NMTOKEN">
41                  <xs:enumeration value="petit"/>
42                  <xs:enumeration value="moyen"/>
43                  <xs:enumeration value="grand"/>
44              </xs:restriction>
45          </xs:simpleType>
46      </xs:union>
47  </xs:simpleType>
```

- Déclaration du nom de notre type basé sur une union de deux types.
- Déclaration du premier type de l'union.
- Déclaration du deuxième type de l'union.

Exemple de type *union*

```
30  <xs:simpleType name="taillePolice">
31    <xs:union>
32      <xs:simpleType>
33        <xs:restriction base="xs:positiveInteger">
34          <xs:minInclusive value="8"/>
35          <xs:maxInclusive value="72"/>
36        </xs:restriction>
37      </xs:simpleType>
38
39      <xs:simpleType>
40        <xs:restriction base="xs:NMTOKEN">
41          <xs:enumeration value="petit"/>
42          <xs:enumeration value="moyen"/>
43          <xs:enumeration value="grand"/>
44        </xs:restriction>
45      </xs:simpleType>
46    </xs:union>
47  </xs:simpleType>
```

- Déclaration du nom de notre type basé sur une union de deux types.
- Déclaration du premier type de l'union.
- Déclaration du deuxième type de l'union.

Exemple de type *union*

```
30  <xs:simpleType name="taillePolice">
31     <xs:union>
32        <xs:simpleType>
33           <xs:restriction base="xs:positiveInteger">
34              <xs:minInclusive value="8"/>
35              <xs:maxInclusive value="72"/>
36           </xs:restriction>
37        </xs:simpleType>
38
39        <xs:simpleType>
40           <xs:restriction base="xs:NMTOKEN">
41              <xs:enumeration value="petit"/>
42              <xs:enumeration value="moyen"/>
43              <xs:enumeration value="grand"/>
44           </xs:restriction>
45        </xs:simpleType>
46     </xs:union>
47  </xs:simpleType>
```

- Déclaration du nom de notre type basé sur une union de deux types.
- Déclaration du premier type de l'union.
- Déclaration du deuxième type de l'union.

Exemples de types *list*

```
49 <xs:simpleType name="tailles">
50    <xs:list itemType="xs:decimal"/>
51 </xs:simpleType>
```

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

```
53 <xs:simpleType name="listeChaines">
54    <xs:list itemType="xs:string"/>
55 </xs:simpleType>
```

Attention, la liste ci-dessous, de type `listeChaines`, a 12 éléments, et non pas 2 (l'espace est un séparateur).

```
<unElement>
 ceci est une chaîne de caractères
 ceci est une chaîne de caractères
</unElement>
```

Exemple de restriction d'un type *list*

```
57 <xs:simpleType name="maListe">
58    <xs:list itemType="xs:integer"/>
59 </xs:simpleType>
60
61 <xs:simpleType name="maListeRestreinte">
62    <xs:restriction base="maListe">
63       <xs:pattern value="123 (\d+\s)*456"/>
64    </xs:restriction>
65 </xs:simpleType>
```

Ce qui autorise à écrire les éléments suivants :

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

Les types complexes

La définition d'un type complexe est un ensemble de déclarations d'attributs et un type de contenu.

- ***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

```
5  <xs:complexType name="TypePiece">
6     <xs:simpleContent>
7        <xs:extension base="xs:string">
8           <xs:attribute name="surface-m2" type="xs:decimal"/>
9           <xs:attribute name="fenetre" type="xs:positiveInteger"/>
10       </xs:extension>
11    </xs:simpleContent>
12 </xs:complexType>
13
14 <xs:element name="chambre" type="TypePiece"/>
```

- Déclaration du type complexe.
- Déclaration du contenu simple. L'élément est une chaîne de caractère (des données brutes) avec deux attributs potentiels.
- Déclaration de l'élément `chambre` ayant comme type celui que nous venons de créer.

Dans le fichier XML associé, on peut alors, par exemple, déclarer un élément chambre de 17,5m^2.

```
<chambre surface-m2="17.5">Exposition plein sud</chambre>
```

Exemple de type complexe à contenu simple

```
5   <xs:complexType name="TypePiece">
6       <xs:simpleContent>
7           <xs:extension base="xs:string">
8               <xs:attribute name="surface-m2" type="xs:decimal"/>
9               <xs:attribute name="fenetre" type="xs:positiveInteger"/>
10          </xs:extension>
11      </xs:simpleContent>
12  </xs:complexType>
13
14  <xs:element name="chambre" type="TypePiece"/>
```

- Déclaration du type complexe.
- Déclaration du contenu simple. L'élément est une chaîne de caractère (des données brutes) avec deux attributs potentiels.
- Déclaration de l'élément `chambre` ayant comme type celui que nous venons de créer.

Dans le fichier XML associé, on peut alors, par exemple, déclarer un élément chambre de 17,5m^2.

```
<chambre surface-m2="17.5">Exposition plein sud</chambre>
```

Exemple de type complexe à contenu simple

```
5  <xs:complexType name="TypePiece">
6      <xs:simpleContent>
7          <xs:extension base="xs:string">
8              <xs:attribute name="surface-m2" type="xs:decimal"/>
9              <xs:attribute name="fenetre" type="xs:positiveInteger"/>
10         </xs:extension>
11     </xs:simpleContent>
12 </xs:complexType>
13
14 <xs:element name="chambre" type="TypePiece"/>
```

- Déclaration du type complexe.
- Déclaration du contenu simple. L'élément est une chaîne de caractère (des données brutes) avec deux attributs potentiels.
- Déclaration de l'élément `chambre` ayant comme type celui que nous venons de créer.

Dans le fichier XML associé, on peut alors, par exemple, déclarer un élément chambre de 17,5m^2.

```
<chambre surface-m2="17.5">Exposition plein sud</chambre>
```

Exemple de type complexe à contenu simple

```
5  <xs:complexType name="TypePiece">
6     <xs:simpleContent>
7        <xs:extension base="xs:string">
8           <xs:attribute name="surface-m2" type="xs:decimal"/>
9           <xs:attribute name="fenetre" type="xs:positiveInteger"/>
10       </xs:extension>
11    </xs:simpleContent>
12 </xs:complexType>
13
14 <xs:element name="chambre" type="TypePiece"/>
```

- Déclaration du type complexe.
- Déclaration du contenu simple. L'élément est une chaîne de caractère (des données brutes) avec deux attributs potentiels.
- Déclaration de l'élément `chambre` ayant comme type celui que nous venons de créer.

Dans le fichier XML associé, on peut alors, par exemple, déclarer un élément chambre de 17,5m^2.

```
<chambre surface-m2="17.5">Exposition plein sud</chambre>
```

Exemple de type complexe à contenu simple

```
5  <xs:complexType name="TypePiece">
6     <xs:simpleContent>
7        <xs:extension base="xs:string">
8           <xs:attribute name="surface-m2" type="xs:decimal"/>
9           <xs:attribute name="fenetre" type="xs:positiveInteger"/>
10       </xs:extension>
11    </xs:simpleContent>
12 </xs:complexType>
13
14 <xs:element name="chambre" type="TypePiece"/>
```

- Déclaration du type complexe.
- Déclaration du contenu simple. L'élément est une chaîne de caractère (des données brutes) avec deux attributs potentiels.
- Déclaration de l'élément `chambre` ayant comme type celui que nous venons de créer.

Dans le fichier XML associé, on peut alors, par exemple, déclarer un élément chambre de 17,5m^2.

```
<chambre surface-m2="17.5">Exposition plein sud</chambre>
```

Exemple de type complexe à contenu complexe

```
16 <xs:complexType name="type-duree">
17    <xs:sequence>
18       <xs:element name="du" type="xs:date"/>
19       <xs:element name="au" type="xs:date"/>
20    </xs:sequence>
21 </xs:complexType>
22
23 <xs:element name="duree" type="type-duree"/>
```

- Déclaration de la sequence du type complexe. Cela signifie que ce type est composé de deux éléments qui doivent apparaître dans cet ordre.
- Déclaration de l'élément `duree` du type que nous venons de créer.

Dans le fichier XML associé, on peut alors déclarer un élément `duree` en spécifiant les valeurs `du` et `au`, ces valeurs doivent être des dates au format américain car elles sont de type `xs:date`.

```
<duree>
 <du>2020-09-13</du>
 <au>2020-09-25</au>
</duree>
```

Exemple de type complexe à contenu complexe

```
16  <xs:complexType name="type-duree">
17      <xs:sequence>
18          <xs:element name="du" type="xs:date"/>
19          <xs:element name="au" type="xs:date"/>
20      </xs:sequence>
21  </xs:complexType>
22
23  <xs:element name="duree" type="type-duree"/>
```

- Déclaration de la sequence du type complexe. Cela signifie que ce type est composé de deux éléments qui doivent apparaître dans cet ordre.
- Déclaration de l'élément `duree` du type que nous venons de créer.

Dans le fichier XML associé, on peut alors déclarer un élément `duree` en spécifiant les valeurs `du` et `au`, ces valeurs doivent être des dates au format américain car elles sont de type `xs:date`.

```
<duree>
 <du>2020-09-13</du>
 <au>2020-09-25</au>
</duree>
```

Exemple de type complexe à contenu complexe

```
16  <xs:complexType name="type-duree">
17      <xs:sequence>
18          <xs:element name="du" type="xs:date"/>
19          <xs:element name="au" type="xs:date"/>
20      </xs:sequence>
21  </xs:complexType>
22
23  <xs:element name="duree" type="type-duree"/>
```

- Déclaration de la sequence du type complexe. Cela signifie que ce type est composé de deux éléments qui doivent apparaître dans cet ordre.
- Déclaration de l'élément `duree` du type que nous venons de créer.

Dans le fichier XML associé, on peut alors déclarer un élément `duree` en spécifiant les valeurs `du` et `au`, ces valeurs doivent être des dates au format américain car elles sont de type `xs:date`.

```
<duree>
 <du>2020-09-13</du>
 <au>2020-09-25</au>
</duree>
```

Exemple de type complexe à contenu complexe

```
16  <xs:complexType name="type-duree">
17      <xs:sequence>
18          <xs:element name="du" type="xs:date"/>
19          <xs:element name="au" type="xs:date"/>
20      </xs:sequence>
21  </xs:complexType>
22
23  <xs:element name="duree" type="type-duree"/>
```

- Déclaration de la sequence du type complexe. Cela signifie que ce type est composé de deux éléments qui doivent apparaître dans cet ordre.
- Déclaration de l'élément `duree` du type que nous venons de créer.

Dans le fichier XML associé, on peut alors déclarer un élément `duree` en spécifiant les valeurs `du` et `au`, ces valeurs doivent être des dates au format américain car elles sont de type `xs:date`.

```
<duree>
 <du>2020-09-13</du>
 <au>2020-09-25</au>
</duree>
```

Exemple de type complexe à contenu complexe

```
25  <xs:complexType name="nomPersonne">
26      <xs:sequence>
27          <xs:element name="titre" minOccurs="0"/>
28          <xs:element name="prenom" minOccurs="0" maxOccurs="unbounded"/>
29          <xs:element name="nom-famille"/>
30      </xs:sequence>
31  </xs:complexType>
```

- Déclaration des sous-éléments de `nomPersonne`. Ici on indique que `titre` est un élément optionnel (entre 0 et 1) et `prenom` possède une cardinalité indéterminée (entre 0 et n).
- Déclaration du sous-élément `nom-famille`. Par défaut, la cardinalité est `minOccurs="1"` et `maxOccurs="1"`.

Exemple de type complexe à contenu complexe

```
25  <xs:complexType name="nomPersonne">
26      <xs:sequence>
27          <xs:element name="titre" minOccurs="0"/>
28          <xs:element name="prenom" minOccurs="0" maxOccurs="unbounded"/>
29          <xs:element name="nom-famille"/>
30      </xs:sequence>
31  </xs:complexType>
```

- Déclaration des sous-éléments de `nomPersonne`. Ici on indique que `titre` est un élément optionnel (entre 0 et 1) et `prenom` possède une cardinalité indéterminée (entre 0 et n).
- Déclaration du sous-élément `nom-famille`. Par défaut, la cardinalité est `minOccurs="1"` et `maxOccurs="1"`.

Exemple de type complexe à contenu complexe

```
25  <xs:complexType name="nomPersonne">
26      <xs:sequence>
27          <xs:element name="titre" minOccurs="0"/>
28          <xs:element name="prenom" minOccurs="0" maxOccurs="unbounded"/>
29          <xs:element name="nom-famille"/>
30      </xs:sequence>
31  </xs:complexType>
```

- Déclaration des sous-éléments de `nomPersonne`. Ici on indique que `titre` est un élément optionnel (entre 0 et 1) et `prenom` possède une cardinalité indéterminée (entre 0 et n).
- Déclaration du sous-élément `nom-famille`. Par défaut, la cardinalité est `minOccurs="1"` et `maxOccurs="1"`.

Exemple de type complexe à contenu complexe

```
33 <xs:complexType name="nomPersonneEtendu">
34    <xs:complexContent>
35       <xs:extension base="nomPersonne">
36          <xs:sequence>
37             <xs:element name="rang" minOccurs="0"/>
38          </xs:sequence>
39       </xs:extension>
40    </xs:complexContent>
41 </xs:complexType>
42
43 <xs:element name="personneEtendue" type="nomPersonneEtendu"/>
```

- Le type `nomPersonneEtendu` est une extension du type `nomPersonne`. Cette extension ajoute le rang dans la fratrie, qui est optionnel.

Cette spécification autorise la création de balise `personneEtendue` comme présenté ci-dessous. Remarque : le fait qu'il n'y est pas de `titre` est autorisé par le schéma.

```
<personneEtendue>
 <prenom>Jean</prenom>
 <prenom>Michel</prenom>
 <nom-famille>Dupont</nom-famille>
 <rang>1</rang>
</personneEtendue>
```

Exemple de type complexe à contenu complexe

```
33  <xs:complexType name="nomPersonneEtendu">
34    <xs:complexContent>
35      <xs:extension base="nomPersonne">
36        <xs:sequence>
37          <xs:element name="rang" minOccurs="0"/>
38        </xs:sequence>
39      </xs:extension>
40    </xs:complexContent>
41  </xs:complexType>
42
43  <xs:element name="personneEtendue" type="nomPersonneEtendu"/>
```

- Le type `nomPersonneEtendu` est une extension du type `nomPersonne`. Cette extension ajoute le rang dans la fratrie, qui est optionnel.

Cette spécification autorise la création de balise `personneEtendue` comme présenté ci-dessous. Remarque : le fait qu'il n'y est pas de `titre` est autorisé par le schéma.

```
<personneEtendue>
 <prenom>Jean</prenom>
 <prenom>Michel</prenom>
 <nom-famille>Dupont</nom-famille>
 <rang>1</rang>
</personneEtendue>
```

Exemple de type complexe à contenu complexe

```
33  <xs:complexType name="nomPersonneEtendu">
34    <xs:complexContent>
35      <xs:extension base="nomPersonne">
36        <xs:sequence>
37          <xs:element name="rang" minOccurs="0"/>
38        </xs:sequence>
39      </xs:extension>
40    </xs:complexContent>
41  </xs:complexType>
42
43  <xs:element name="personneEtendue" type="nomPersonneEtendu"/>
```

- Le type `nomPersonneEtendu` est une extension du type `nomPersonne`. Cette extension ajoute le rang dans la fratrie, qui est optionnel.

Cette spécification autorise la création de balise `personneEtendue` comme présenté ci-dessous. Remarque : le fait qu'il n'y est pas de `titre` est autorisé par le schéma.

```
<personneEtendue>
 <prenom>Jean</prenom>
 <prenom>Michel</prenom>
 <nom-famille>Dupont</nom-famille>
 <rang>1</rang>
</personneEtendue>
```

Exemple de type complexe à contenu complexe

```
45  <xs:complexType name="nomSimple">
46    <xs:complexContent>
47      <xs:restriction base="nomPersonne">
48        <xs:sequence>
49          <xs:element name="prenom" minOccurs="1" maxOccurs="1"/>
50          <xs:element name="nom-famille"/>
51        </xs:sequence>
52      </xs:restriction>
53    </xs:complexContent>
54  </xs:complexType>
```

- Il est possible de respécifier des propriétés déjà établies. Ici on indique notamment que `prenom` doit apparaitre et n'est plus optionnel.

Dans le document XML associé, on peut alors créer une `personne` comme indiqué ci-dessous.

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

Exemple de type complexe à contenu complexe

```
45  <xs:complexType name="nomSimple">
46    <xs:complexContent>
47      <xs:restriction base="nomPersonne">
48        <xs:sequence>
49          <xs:element name="prenom" minOccurs="1" maxOccurs="1"/>
50          <xs:element name="nom-famille"/>
51        </xs:sequence>
52      </xs:restriction>
53    </xs:complexContent>
54  </xs:complexType>
```

- Il est possible de respécifier des propriétés déjà établies. Ici on indique notamment que `prenom` doit apparaitre et n'est plus optionnel.

Dans le document XML associé, on peut alors créer une `personne` comme indiqué ci-dessous.

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

Exemple de type complexe à contenu complexe

```
45  <xs:complexType name="nomSimple">
46    <xs:complexContent>
47      <xs:restriction base="nomPersonne">
48        <xs:sequence>
49          <xs:element name="prenom" minOccurs="1" maxOccurs="1"/>
50          <xs:element name="nom-famille"/>
51        </xs:sequence>
52      </xs:restriction>
53    </xs:complexContent>
54  </xs:complexType>
```

- Il est possible de respécifier des propriétés déjà établies. Ici on indique notamment que `prenom` doit apparaitre et n'est plus optionnel.

Dans le document XML associé, on peut alors créer une `personne` comme indiqué ci-dessous.

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

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 trois 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.

Remarque : il est *théoriquement* possible de faire un ***all*** en DTD. Supposons que nous ayons un élément `racine` possédant trois enfants, `A`, `B`, et `C`. Quel est la règle en DTD qui permet d'autoriser n'importe quel ordre d'apparation des trois enfants?

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 trois 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.

Remarque : il est *théoriquement* possible de faire un ***all*** en DTD. Supposons que nous ayons un élément `racine` possédant trois enfants, `A`, `B`, et `C`. Quel est la règle en DTD qui permet d'autoriser n'importe quel ordre d'apparation des trois enfants?

```
<!ELEMENT racine ( (A, B, C) | (A, C, B) | (B, A, C) |
(B, C, A) | (C, A, B) | (C, B, A) )>
```

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 trois 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.

Remarque : il est *théoriquement* possible de faire un ***all*** en DTD. Supposons que nous ayons un élément `racine` possédant trois enfants, `A`, `B`, et `C`. Quel est la règle en DTD qui permet d'autoriser n'importe quel ordre d'apparation des trois enfants?

```
<!ELEMENT racine ( (A, B, C) | (A, C, B) | (B, A, C) |
(B, C, A) | (C, A, B) | (C, B, A) )>
```

Si l'élément `racine` possède *n* enfants. Quel est le nombre de sous-élément à écrire pour effectuer un ***all*** en DTD?

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 trois 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.

Remarque : il est *théoriquement* possible de faire un ***all*** en DTD. Supposons que nous ayons un élément `racine` possédant trois enfants, `A`, `B`, et `C`. Quel est la règle en DTD qui permet d'autoriser n'importe quel ordre d'apparation des trois enfants?

```
<!ELEMENT racine ( (A, B, C) | (A, C, B) | (B, A, C) |
(B, C, A) | (C, A, B) | (C, B, A) )>
```

Si l'élément `racine` possède n enfants. Quel est le nombre de sous-élément à écrire pour effectuer un ***all*** en DTD?

$$n!$$

Par exemple pour 6 enfants, cela représente 720 combinaisons.

Exemple d'utilisation de *choice*

```
5  <xs:complexType name="surfaceOuVolume">
6     <xs:choice>
7        <xs:element name="surface" type="xs:float"/>
8        <xs:element name="volume" type="xs:float"/>
9     </xs:choice>
10 </xs:complexType>
```

Ce qui permet d'écrire dans le document XML :

```
<occupation>
 <surface>453</surface>
</occupation>
```

ou :

```
<occupation>
 <volume>453</volume>
</occupation>
```

mais pas :

```
<occupation>
 <surface>453</surface>
 <volume>453</volume>
</occupation>
```

Exemple d'utilisation de *all*

```
12  <xs:complexType name="identite">
13     <xs:all>
14        <xs:element name="nom" type="xs:string"/>
15        <xs:element name="prenom" type="xs:string"/>
16        <xs:element name="datenaissance" type="xs:date"/>
17     </xs:all>
18  </xs:complexType>
```

Ce qui permet d'écrire dans le document XML :

```
<identite>
 <nom>Sotala</nom>
 <prenom>Joona</prenom>
 <datenaiss>1998-03-22</datenaiss>
</identite>
```

ou :

```
<identite>
 <datenaiss>2002-04-08</datenaiss>
 <nom>Desplanches</nom>
 <prenom>Clément</prenom>
</identite>
```

et toute permutation de `nom`, `prenom`, `datenaiss`.

Déclaration d'attribut

Un attribut est de type simple, par exemple un type prédéfini, une énumération, une liste ...

On peut préciser le caractère obligatoire ou facultatif de l'attribut (attribut `use` lors de la déclaration de l'attribut).

On peut lui donner une valeur par défaut (attribut `default`) ou une valeur fixe (attribut `fixed`).

Un attribut se déclare là où on déclare l'élément qui lui est associé.

Exemples de déclarations d'attributs

```
<xs:attribute name="taille" type="taillePolice" 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 grace au 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.

La déclaration d'élément peut se faire de manière *récursive*. On peut déclarer un élément, puis dans cet élément, déclarer un élément, puis dans cet élément, déclarer un élément, etc ...

Nous allons structurer nos XML Schema de manière à éviter au maximum des déclarations en cascade.

Exemples d'éléments à contenu simple

```
5  <xs:element name="longueur" type="xs:integer"/>
6  <xs:element name="nom" type="xs:string"/>
7
8  <xs:element name="police">
9     <xs:complexType>
10       <xs:simpleContent>
11          <xs:extension base="xs:string">
12             <xs:attribute name="taille" type="taillePolice"/>
13          </xs:extension>
14       </xs:simpleContent>
15    </xs:complexType>
16 </xs:element>
```

- Déclaration d'éléments d'un certain type.
- Déclaration d'un élément police de type complexe. Cet élément est une extension d'un type de base (ici `xs:string`) auquel on rajoute un attribut `taille` de type `taillePolice`.

Exemples d'éléments à contenu simple

```
5  <xs:element name="longueur" type="xs:integer"/>
6  <xs:element name="nom" type="xs:string"/>
7
8  <xs:element name="police">
9     <xs:complexType>
10       <xs:simpleContent>
11          <xs:extension base="xs:string">
12             <xs:attribute name="taille" type="taillePolice"/>
13          </xs:extension>
14       </xs:simpleContent>
15    </xs:complexType>
16 </xs:element>
```

- Déclaration d'éléments d'un certain type.
- Déclaration d'un élément police de type complexe. Cet élément est une extension d'un type de base (ici `xs:string`) auquel on rajoute un attribut `taille` de type `taillePolice`.

Exemples d'éléments à contenu simple

```
5   <xs:element name="longueur" type="xs:integer"/>
6   <xs:element name="nom" type="xs:string"/>
7
8   <xs:element name="police">
9     <xs:complexType>
10      <xs:simpleContent>
11        <xs:extension base="xs:string">
12          <xs:attribute name="taille" type="taillePolice"/>
13        </xs:extension>
14      </xs:simpleContent>
15    </xs:complexType>
16  </xs:element>
```

- Déclaration d'éléments d'un certain type.
- Déclaration d'un élément police de type complexe. Cet élément est une extension d'un type de base (ici `xs:string`) auquel on rajoute un attribut `taille` de type `taillePolice`.

Exemples d'éléments à contenu simple

L'exemple précédent permet alors d'écrire en XML :

```
<longueur>45</longueur>
<nom>Durand</nom>
<police taille="moyen">texte</police>
```

Remarque : une chaîne de caractère peut simplement être un nombre, comme par exemple ici 45, qui est en fait la chaîne de caractères 45.

Exemples d'éléments à contenu mixte

```
18  <xs:element name="toto">
19     <xs:complexType mixed="true">
20        <xs:sequence>
21           <xs:element ref="police" minOccurs="1" maxOccurs="2"/>
22           <xs:element ref="longueur"/>
23           <xs:element name="longueur2" type="xs:float"/>
24           <xs:element name="media" type="nom"/>
25        </xs:sequence>
26     </xs:complexType>
27  </xs:element>
```

- Déclaration d'un élément à contenu mixte.
- Ici l'utilisation de **ref** indique qu'on *copie-colle* la déclaration précédente de `police` et `longueur`.
- On ajoute également deux éléments de type `xs:float` et `nom`.

Remarque : on peut mélanger `ref` et des attributs de cardinalité comme `minOccurs` et `maxOccurs`.

Exemples d'éléments à contenu mixte

```
18  <xs:element name="toto">
19     <xs:complexType mixed="true">
20        <xs:sequence>
21           <xs:element ref="police" minOccurs="1" maxOccurs="2"/>
22           <xs:element ref="longueur"/>
23           <xs:element name="longueur2" type="xs:float"/>
24           <xs:element name="media" type="nom"/>
25        </xs:sequence>
26     </xs:complexType>
27  </xs:element>
```

- Déclaration d'un élément à contenu mixte.
- Ici l'utilisation de **ref** indique qu'on *copie-colle* la déclaration précédente de `police` et `longueur`.
- On ajoute également deux éléments de type `xs:float` et `nom`.

Remarque : on peut mélanger `ref` et des attributs de cardinalité comme `minOccurs` et `maxOccurs`.

Exemples d'éléments à contenu mixte

```
18  <xs:element name="toto">
19     <xs:complexType mixed="true">
20        <xs:sequence>
21           <xs:element ref="police" minOccurs="1" maxOccurs="2"/>
22           <xs:element ref="longueur"/>
23           <xs:element name="longueur2" type="xs:float"/>
24           <xs:element name="media" type="nom"/>
25        </xs:sequence>
26     </xs:complexType>
27  </xs:element>
```

- Déclaration d'un élément à contenu mixte.
- Ici l'utilisation de **ref** indique qu'on *copie-colle* la déclaration précédente de `police` et `longueur`.
- On ajoute également deux éléments de type `xs:float` et `nom`.

Remarque : on peut mélanger `ref` et des attributs de cardinalité comme `minOccurs` et `maxOccurs`.

Exemples d'éléments à contenu mixte

```
18  <xs:element name="toto">
19     <xs:complexType mixed="true">
20        <xs:sequence>
21           <xs:element ref="police" minOccurs="1" maxOccurs="2"/>
22           <xs:element ref="longueur"/>
23           <xs:element name="longueur2" type="xs:float"/>
24           <xs:element name="media" type="nom"/>
25        </xs:sequence>
26     </xs:complexType>
27  </xs:element>
```

- Déclaration d'un élément à contenu mixte.
- Ici l'utilisation de **ref** indique qu'on *copie-colle* la déclaration précédente de `police` et `longueur`.
- On ajoute également deux éléments de type `xs:float` et `nom`.

Remarque : on peut mélanger `ref` et des attributs de cardinalité comme `minOccurs` et `maxOccurs`.

Exemples d'éléments vide

La balise **br** en XHTML contient uniquement des attributs. Les types utilisés sont définis dans le schéma de XHTML.

```
29 <xs:element name="br">
30    <xs:complexType>
31       <xs:attribute name="id" type="xs:ID"/>
32       <xs:attribute name="class" type="xs:NMTOKENS"/>
33       <xs:attribute name="style" type="StyleSheet"/>
34       <xs:attribute name="title" type="Text"/>
35    </xs:complexType>
36 </xs:element>
```

Un exemple de création d'élément vide.

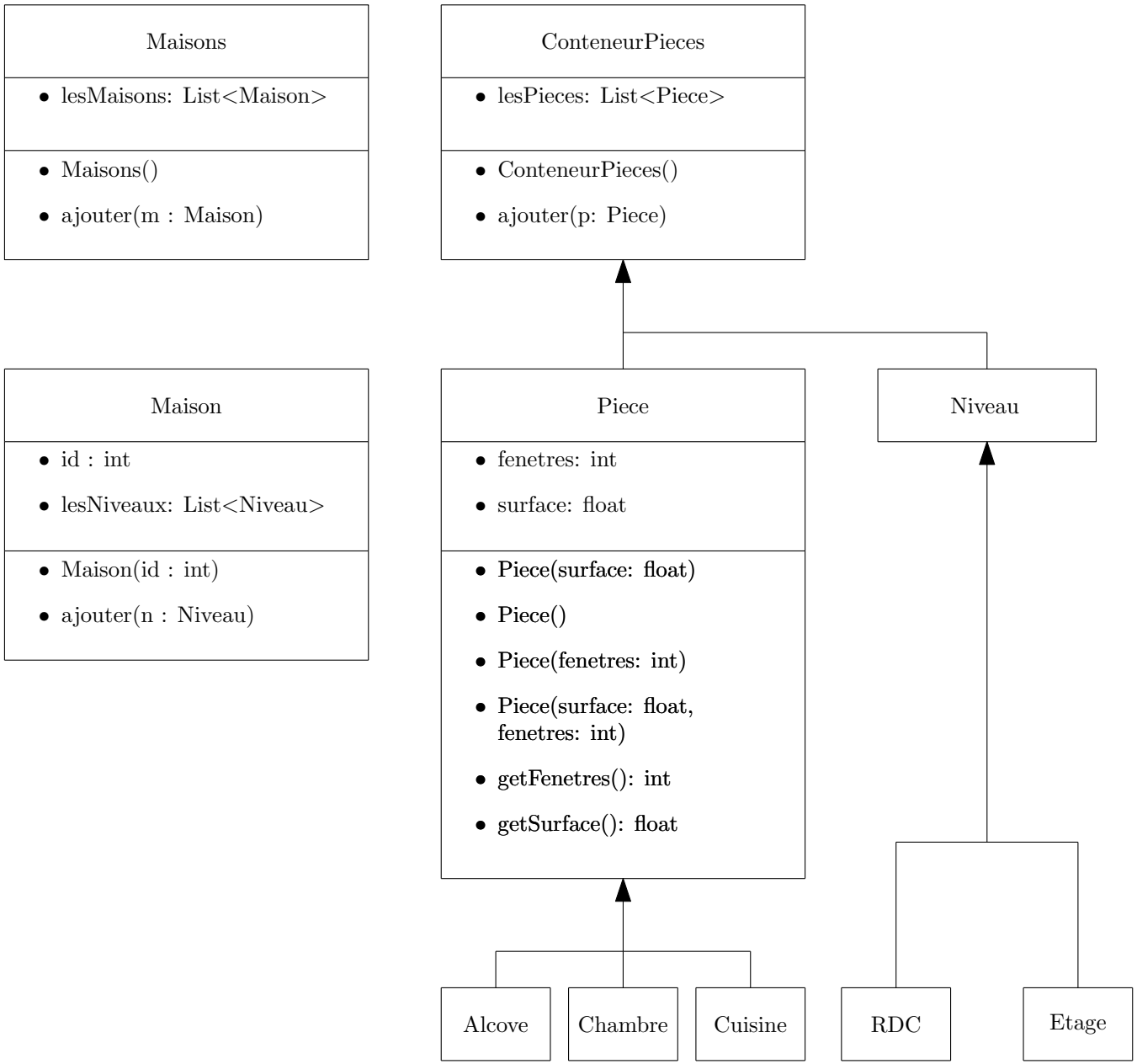
```
38 <xs:element name="vide">
39    <xs:complexType/>
40 </xs:element>
```

Éléments, attributs et types

Quelques rappels sur les interactions possibles entre éléments, 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 ou attribut déjà défini grâce à l'attribut `ref`.

Quizz


Maisons
lesMaisons: List<Maison>
Maisons()
ajouter(m : Maison)
ConteneurPieces
lesPieces: List<Piece>
ConteneurPieces()
ajouter(p: Piece)
Maison
id : int
lesNiveaux: List<Niveau>
Maison(id : int)
ajouter(n : Niveau)
Piece
fenetres: int
surface: float
Piece(surface: float)
Piece()
Piece(fenetres: int)
Piece(surface: float, fenetres: int)
getFenetres(): int
getSurface(): float
Niveau
Alcove
Chambre
Cuisine
RDC
Etage

Quizz

```
5  <xsd:element name="maison">
6     <xsd:complexType>
7        <xsd:sequence>
8           <xsd:choice minOccurs="0" maxOccurs="unbounded">
9              <xsd:element name="etage" type="TypeNiveau"/>
10             <xsd:element name="RDC" type="TypeNiveau"/>
11          </xsd:choice>
12       </xsd:sequence>
13       <xsd:attribute name="id" type="xsd:positiveInteger"
14          use="required"/>
15    </xsd:complexType>
16 </xsd:element>
17
18 <xsd:element name="maisons">
19    <xsd:complexType>
20       <xsd:sequence minOccurs="0" maxOccurs="unbounded">
21          <xsd:element ref="maison"/>
22       </xsd:sequence>
23    </xsd:complexType>
24 </xsd:element>
```

Quizz

```
26 <xsd:complexType name="TypeNiveau">
27    <xsd:sequence>
28       <xsd:choice minOccurs="0" maxOccurs="unbounded">
29          <xsd:element name="alcove" type="TypePiece"/>
30          <xsd:element name="cuisine" type="TypePiece"/>
31          <xsd:element name="séjour" type="TypePiece"/>
32          <xsd:element name="bureau" type="TypePiece"/>
33          <xsd:element name="garage" type="TypePiece"/>
34          <xsd:element name="terrasse" type="TypePiece"/>
35          <xsd:element name="chambre" type="TypePiece"/>
36          <xsd:element name="salledeBain" type="TypePiece"/>
37          <xsd:element name="mirador" type="TypePiece"/>
38          <xsd:element name="WC" type="TypePiece"/>
39       </xsd:choice>
40    </xsd:sequence>
41 </xsd:complexType>
```

Quizz

```
43  <xsd:complexType name="TypePiece" mixed="true">
44     <xsd:sequence>
45        <xsd:choice minOccurs="0" maxOccurs="unbounded">
46           <xsd:element name="alcove" type="TypePiece"/>
47           <xsd:element name="cuisine" type="TypePiece"/>
48           <xsd:element name="séjour" type="TypePiece"/>
49           <xsd:element name="bureau" type="TypePiece"/>
50           <xsd:element name="garage" type="TypePiece"/>
51           <xsd:element name="terrasse" type="TypePiece"/>
52           <xsd:element name="chambre" type="TypePiece"/>
53           <xsd:element name="salledeBain" type="TypePiece"/>
54           <xsd:element name="mirador" type="TypePiece"/>
55           <xsd:element name="WC" type="TypePiece"/>
56        </xsd:choice>
57     </xsd:sequence>
58     <xsd:attribute name="surface-m2" type="xsd:decimal"/>
59     <xsd:attribute name="fenetre" type="xsd:positiveInteger"/>
60  </xsd:complexType>
```