

# Formalisation - Structure d'un XMLSchema

## 1 Conventions de nommage en vigueur dans ce cours

En XMLSchema, les noms des éléments et des types d'éléments doivent seulement respecter les conventions de nommage des noms des éléments en XML. Nous ajouterons à ces règles des *conventions de nommage* propres à ce cours, mais communément admises notamment en Java ou C#.

- les noms des éléments commencent par une minuscule et peuvent comporter des majuscules pour exprimer plusieurs mots
- les noms des *types simples* et des *types complexes* commencent par une majuscule suivie d'une minuscule et comportent une majuscule au début de chaque mot.

## 2 Exemple

Dans la suite de ce chapitre, nous allons créer le XMLSchema de l'instance XML suivante:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <étudiant
4   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
5   xmlns='http://www.uga.fr/etudiant'
6   xsi:schemaLocation='http://www.uga.fr/etudiant Etudiant.xsd'>
7
8   <nom>Kuzbidon</nom>
9   <prénom>Alex</prénom>
10  <sexe>H</sexe>
11  <dateNaissance>1991-06-15</dateNaissance>
12  <téléphone>+33 6 00 00 00 00</téléphone>
13  <adresse>
14    <numéro>42</numéro>
15    <rue>impasse de l'Univers</rue>
16    <codepostal>42420</codepostal>
17    <ville>Lunivers</ville>
18  </adresse>
19  <l3Validée>true</l3Validée>
20  <ue nom="FDD" coeff="1">
21    <note>18.5</note>
22    <validée>true</validée>
23  </ue>
24  <ue nom="IHM" coeff="1">
25    <note>15.0</note>
26    <validée>true</validée>
27  </ue>
28  <ue nom="Anglais" coeff="1">
29    <note>5.0</note>
30    <validée>false</validée>
31  </ue>
32 </étudiant>

```

Figure VI.1: Exemple d'instance XML.

### 3 Élément racine

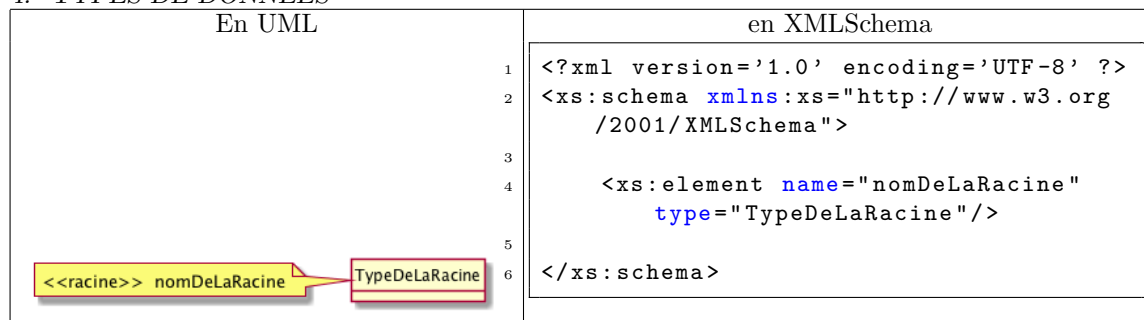
Un XMLSchema est aussi un document XML. Il commence donc par un prologue `<?xml version='1.0' encoding='UTF-8' ?>` et a pour racine l'élément `<schema>`.

Pour désigner l'élément racine du nouveau vocabulaire<sup>1</sup> en XMLSchema, on utilise l'élément `element` avec l'attribut `name` qui désigne le nom de la racine et l'attribut `type` qui désigne son type (nous y reviendrons plus tard).

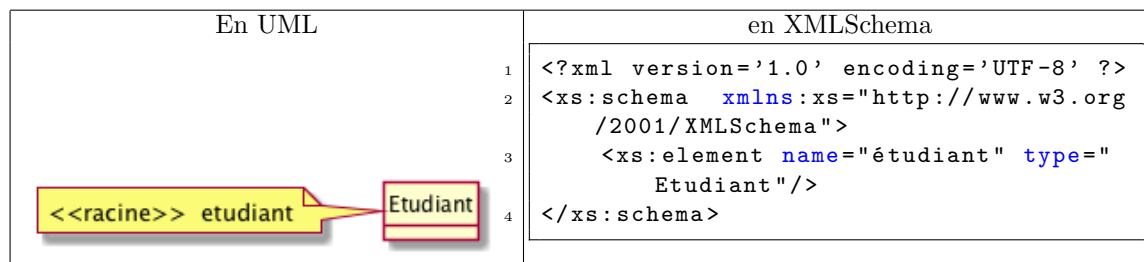
La racine, `<schema>` comporte des attributs, en particulier les attributs `xmlns`. Dans les exemples ci-dessous, la racine est préfixée `xs`, se notant ainsi `<xs:schema>` et l'attribut `xmlns` est suffixé `xs` et a pour valeur `http://www.w3.org/2001/XMLSchema`. Il s'agit de l'espace de nom définissant le vocabulaire XMLSchema ; celui-ci est lié aux mots de son vocabulaires par le préfixe `xs`. Ce point fait l'objet de la section finale sur les espaces de nom. Pour le moment, ces précisions (suffixes, ...) sont données dans les exemples afin que ces derniers soient valides ; vous pouvez cependant les ignorer pour le moment.

La racine s'exprime donc de la manière suivante en UML et XMLSchema:

<sup>1</sup>i.e. des instances XML qui seront valides par rapport à ce schéma



Par exemple, dans l'exemple de la figure VI.1 on obtiendrait l'UML et le XMLSchema suivant:



#### Remarque 1:

La déclaration d'un élément par `<element name="nomElement" type="TypeElement"/>` correspond à signifier qu'il est possible de créer un document XML (une instance XML) ayant pour racine l'élément `nomElement`. C'est une déclaration d'instance... au même titre qu'une déclaration d'instance en Java (ou tout autre langage) qu'on ferait en écrivant `TypeElement nomElement;`. Voir la remarque 3 faite dans la section *Schéma et Instance*.

#### Remarque 2:

Dans un fichier XMLSchema, il est tout à fait possible

- de ne déclarer aucune instance d'élément (le fichier XMLSchema ne contenant alors que des définitions de types), auquel cas il ne sera pas possible de créer d'instances XML (de document XML) ; il faut pour cela une racine. Cela peut se faire lorsqu'on écrit des types dans un schéma XML qu'on appelle (par inclusion ou importation) depuis un autre schéma.
- de déclarer plusieurs instances d'éléments (le fichier XMLSchema contenant alors des définitions de types et 1 à plusieurs déclarations d'instances d'éléments), auquel cas il sera possible de créer plusieurs instances de documents XML différentes, ayant chacune une racine différentes.

## 4 Types de Données

L'objectif d'un XMLSchema est de définir les types des données. On distingue 2 types de types de données:

- les types basiques
- les types complexes (`complexType`)

On appelle un type *basique* un type de données non décomposable, c'est-à-dire qu'un élément de type basique n'aura pas de sous-élément (exemple `<nom>Toto</nom>`) ni d'attributs. Un type basique va de plus expliciter les valeurs possibles pour la donnée (par exemple un entier positif).

On appelle type *complexe* un type de données qui contient d'autres données sous la forme d'éléments et/ou d'attributs (exemple `<étudiant><nom>Toto</nom></étudiant>` ou `<étudiant nom="Toto"/>`). Un type complexe décrit l'organisation d'une données, c'est-à-dire les sous-éléments et attributs qui la composent.

## 4.a Types basiques

Un type *basique* peut être soit

- un type prédéfini
- un type simple (`simpleType`)

### Type prédéfini

Il s'agit de types de base directement utilisables dans le langage XMLSchema (il n'y a rien à redéfinir).

Les types prédéfinis d'XMLSchema que nous utiliserons le plus souvent sont les suivant

**string** *chaîne de caractères*

**description** : Succession de caractères. Tous les caractères définis dans l'alphabet (précisé dans l'encodage) sont autorisés.

**note** : C'est le type par défaut c'est-à-dire que si aucun type n'est précisé, c'est que le type est **string**.

**int** *entiers*

**description** : Entiers (positifs et négatifs)

**note** : Attention aux limites : en XML, un entier est stocké sur 64 bits. Les valeurs possibles des entiers sont donc dans l'intervalle [-2147483648...+2147483647].

**double** *réels*

**description** : Nombres à virgule

**note** : Encodage sur 256 bits.

**date** *date*

**description** : Expression de la date au format : yyyy-mm-dd

**note** : exemple 2014-04-21

**boolean** *booléens*

**description** : Un booléen a 2 valeurs possibles : vrai (**true**) ou faux (**false**).

Il existe de nombreux autres types prédéfinis en XMLSchema que vous pouvez retrouver dans la figure VI.2 ou aux URL suivantes du site <https://www.w3schools.com/> :

- **Strings**
- **Dates et temps**
- **Valeurs numériques**
- **Autres**

Les types prédéfinis des éléments s'expriment de la manière suivante en UML et XMLSchema:

En UML	En XMLSchema
- nomElement : typePrédéfini	<element name="nomElement" type="typePrédéfini"/>

Par exemple, dans l'exemple de la figure VI.1 on obtiendrait, sur la partie XML suivante, l'UML et le XMLSchema du tableau:

```

1 <nom>Kuzbidon</nom>
2 <prénom>Alex</prénom>
3 <dateNaissance>1991-06-15</dateNaissance>
4 <l3Validée>true</l3Validée>
5 <note>18.5</note>

```

### 3 Built-in datatypes

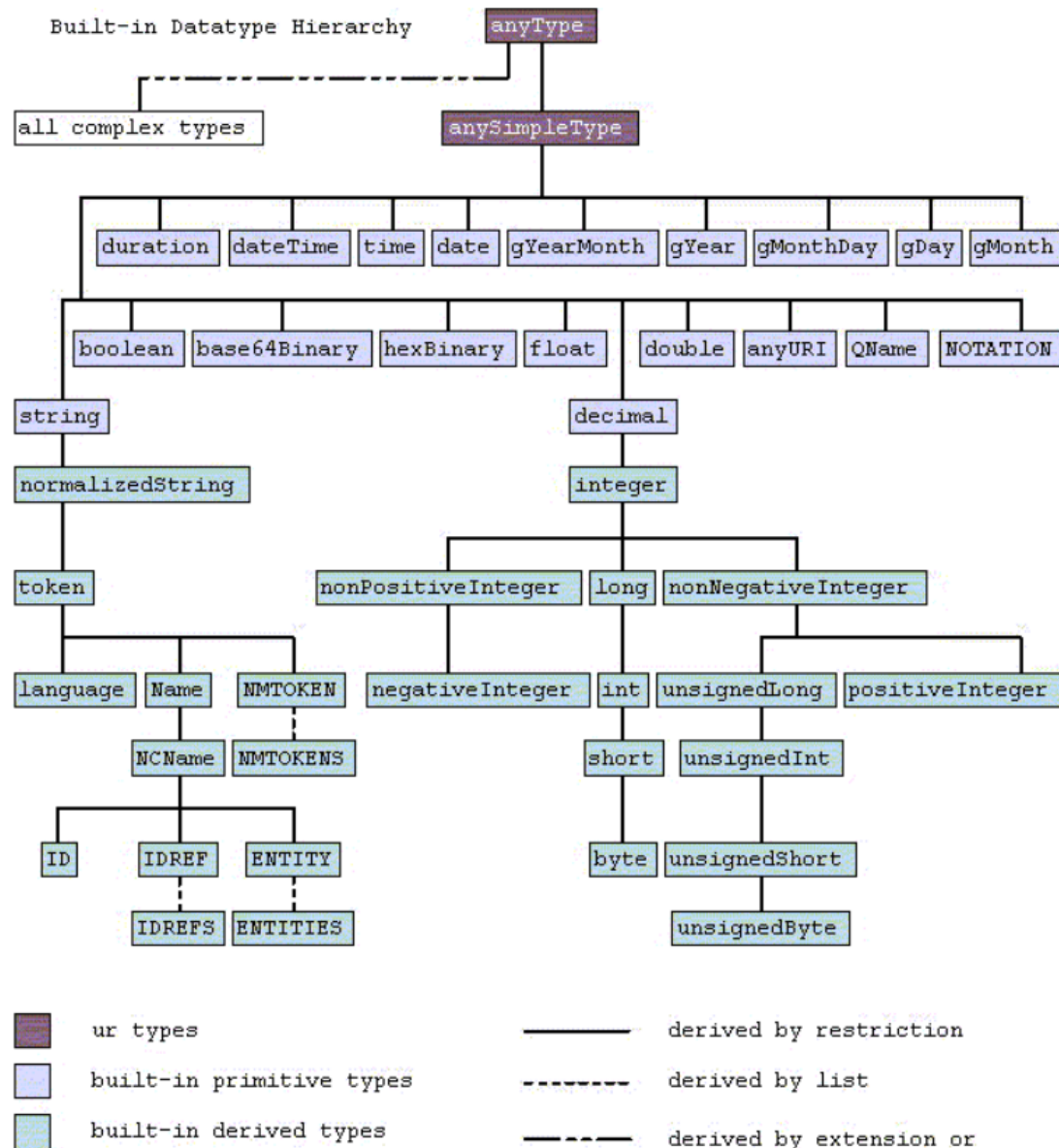


Figure VI.2: Les types prédéfinis en XMLSchema (à titre indicatif).

En UML	En XMLSchema
- nom : string	<element name="nom" type="string"/>
- prénom	<element name="prénom"/>
- dateNaissance : date	<element name="dateNaissance" type="date"/>
- l3Validée : boolean	<element name="l3Validée" type="boolean"/>
- note : double	<element name="note" type="double"/>

*Remarque:* On peut observer dans la deuxième ligne que pour l'élément **prénom**, le type n'a pas été précisé. C'est qu'il s'agit du type par défaut, c'est-à-dire **string**.

#### Types simples

Un type prédéfini permet de définir des données non décomposables en spécifiant des valeurs dans des ensembles prédéfinis (**int**, **double**, **string**, etc.). Ces ensembles sont très généraux et non dédiés à une application ou à un domaine particulier. Par exemple, si l'on choisit le type **double** pour un élément **note**, des notes de -45.7 ou 592 pourront être valides.

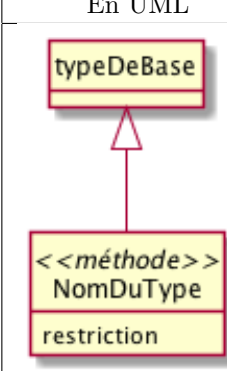
Les *types simples* permettent de réduire le champ des valeurs possibles pour un type prédéfini.

Par définition, un *type simple* (*Simple Type*) est **basé** sur un type prédéfini et le **restreint** à un sous-ensemble de valeurs à l'aide d'une *méthode de restriction*.

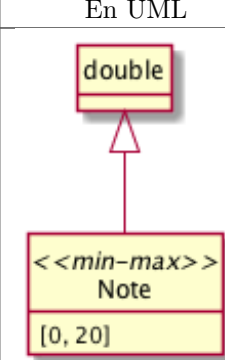
Les méthodes de restriction possibles sont exprimées dans le tableau suivant:

Type prédéfini de base	Méthode de restriction	En XMLSchema
int	«min»	<minInclusive value="..."/> <minExclusive value="..."/>
double	«max»	<maxInclusive value="..."/> <maxExclusive value="..."/>
date	«enumeration»	<enumeration value="..."/>
string	«pattern» «enumeration»	<pattern value="exp. régulière"/> <enumeration value="..."/>

En XMLSchema, un type simple est exprimé à l'aide de l'élément `simpleType` et un sous-élément `restriction` précisant le type de base et la restriction:

En UML	En XMLSchema
	<pre> 1 &lt;simpleType name="NomDuType"&gt; 2   &lt;restriction base="typeDeBase"&gt; 3     &lt;méthode value="..." /&gt; 4     ... 5   &lt;/restriction&gt; 6 &lt;/simpleType&gt; </pre>

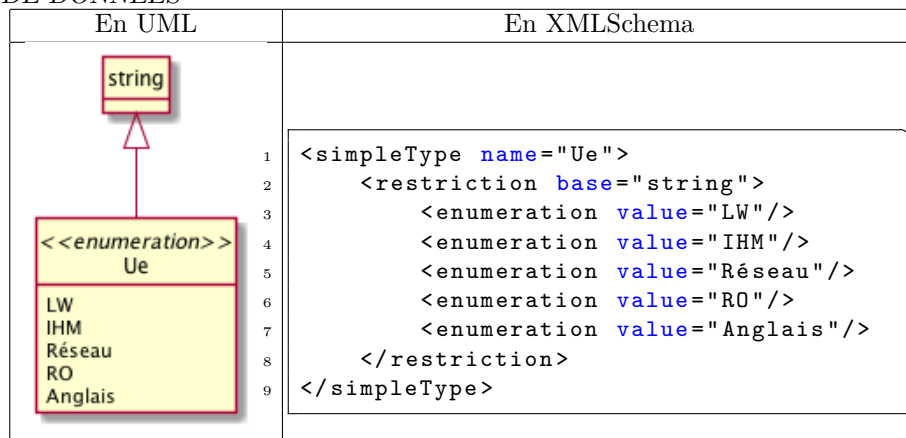
Par exemple, pour exprimer un élément `note` entre 0 et 20, on pourrait utiliser:

En UML	En XMLSchema
	<pre> 1 &lt;simpleType name="Note"&gt; 2   &lt;restriction base="double"&gt; 3     &lt;minInclusive value="0.0" /&gt; 4     &lt;maxInclusive value="20.0" /&gt; 5   &lt;/restriction&gt; 6 &lt;/simpleType&gt; </pre>

Autre exemple, si l'on doit choisir des noms d'UE parmi les UE suivantes:

- LW
- IHM
- Réseau
- RO
- Anglais

On peut alors créer le type simple suivant:



### Expression régulière

Une expression régulière est une expression mathématique qui utilise des symboles prédéfinis pour représenter un ensemble de valeurs. Le tableau suivant résume les principaux symboles que nous serons amenés à utiliser.

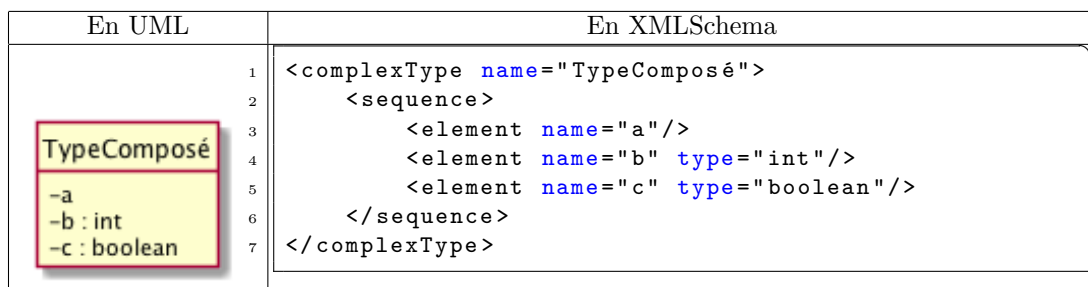
Symbole	Signification
\d	<i>digit</i> = chiffre de 0 à 9
\s	caractère de séparation (espace ou tabulation)
{m}	L'élément précédent est répété m fois. Par exemple \d{m} équivaut à \d\d\d\d\d
{x..y}	L'élément précédent est répété de x à y fois.
[A..E]	Toutes les lettres entre A et E
[1..8]	Tous les chiffres entre 1 et 8

Si l'on souhaite par exemple modéliser un numéro de téléphone sous la forme +33 4 56 52 00 12, on aurait une expression régulière du type `\+\d\d\s\d(\s\d\d){4}`.

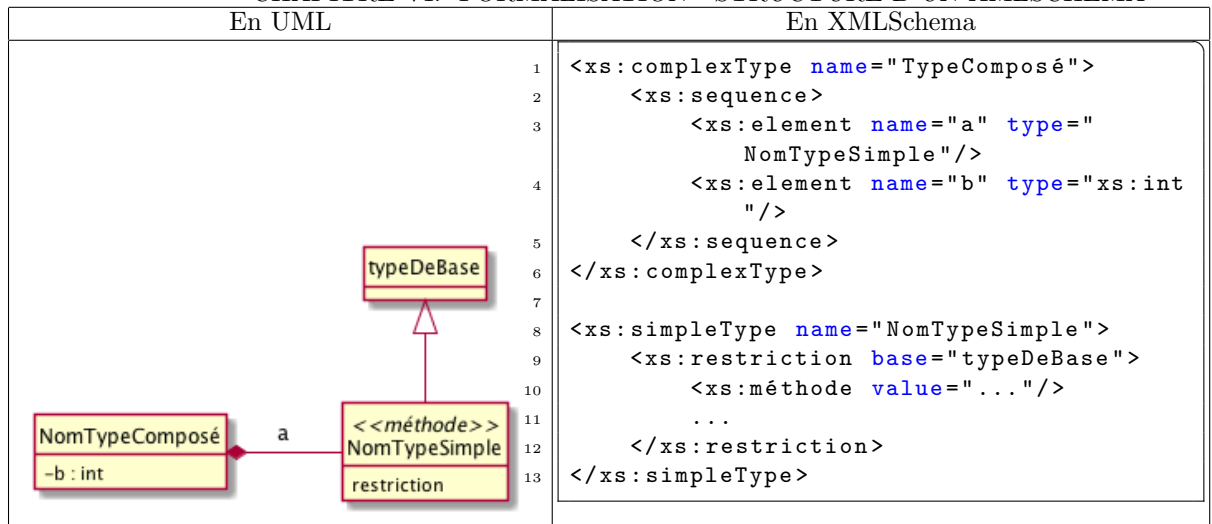
### 4.b Types complexes

La composition permet d'utiliser des types basiques ou complexes dans d'autres types complexes.

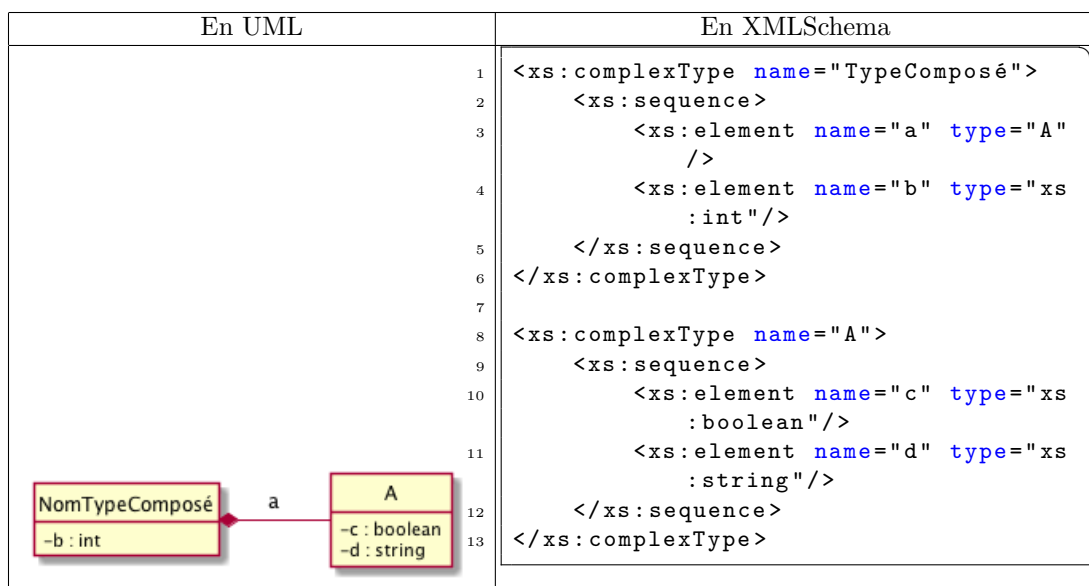
- exemple avec des types prédéfinis:



- exemple avec un type simple



- exemple avec un autre type complexe



Dans les exemples précédents, l'élément `sequence` du langage XMLSchema indique que les sous-éléments devront apparaître dans le même ordre dans les instances XML valides par rapport au schéma. On aurait également pu utiliser l'élément `choice` qui permettrait d'en utiliser 1 parmi les éléments cités ou `group` qui permet d'utiliser tous les éléments cités dans un ordre indifférent.

Si l'on reprend la partie l'exemple de la figure VI.1 ci-dessous:

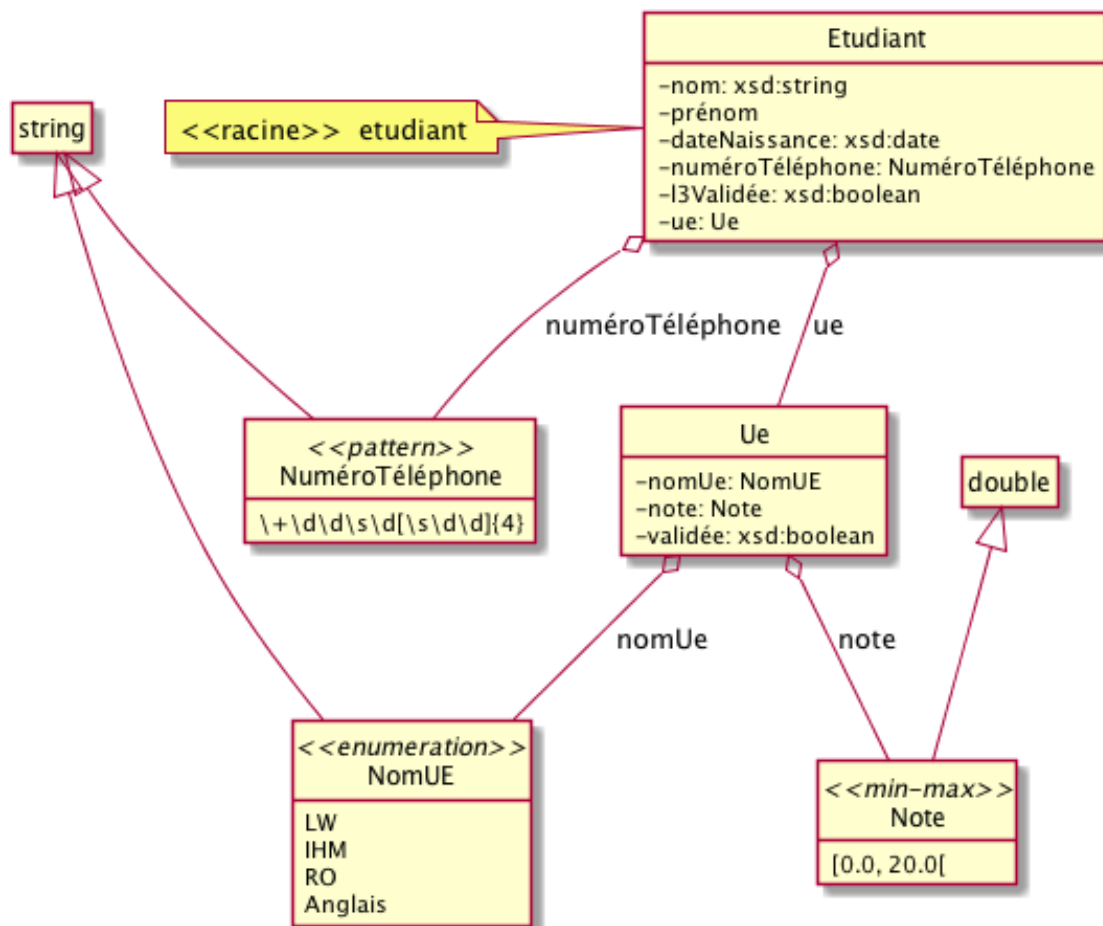
```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <etudiant>
3   <nom>Kuzbidon</nom>
4   <prénom>Alex</prénom>
5   <dateNaissance>1991-06-15</dateNaissance>
6   <numéroTéléphone>+33 6 00 00 00 00</numéroTéléphone
7   >
8   <l3Validée>true</l3Validée>
9   <ue nom="LW">
10     <note>18.5</note>
11     <validée>true</validée>
12   </ue>
13 </etudiant>

```

On obtient alors le diagramme UML et le XMLSchema suivants:





```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4     <xs:element name="etudiant" type="Etudiant"/>
5
6     <xs:complexType name="Etudiant">
7         <xs:sequence>
8             <xs:element name="nom" type="xs:string"/>
9             <xs:element name="prénom"/>
10            <xs:element name="dateNaissance" type="xs:date"/>
11            <xs:element name="numéroTéléphone" type="NuméroTéléphone"/>
12            <xs:element name="l3Validée" type="xs:boolean"/>
13            <xs:element name="ue" type="UE"/>
14        </xs:sequence>
15    </xs:complexType>
16
17    <xs:simpleType name="NuméroTéléphone">
18        <xs:restriction base="xs:string">
19            <xs:pattern value="[+]\d{2}\s\d(\s\d\d){4}"/>
20        </xs:restriction>
21    </xs:simpleType>
22
23    <xs:complexType name="UE">
24        <xs:sequence>
25            <xs:element name="note" type="Note"/>
26            <xs:element name="validée" type="xs:boolean"/>
27        </xs:sequence>
28        <xs:attribute name="nomUE" type="NomUE"/>

```

```

29 </xs:complexType>
30
31 <xs:simpleType name="NomUE">
32   <xs:restriction base="xs:string">
33     <xs:enumeration value="LW"/>
34     <xs:enumeration value="IHM"/>
35     <xs:enumeration value="R0"/>
36     <xs:enumeration value="Anglais"/>
37   </xs:restriction>
38 </xs:simpleType>
39
40 <xs:simpleType name="Note">
41   <xs:restriction base="xs:double">
42     <xs:minInclusive value="0"/>
43     <xs:maxExclusive value="20"/>
44   </xs:restriction>
45 </xs:simpleType>
46
47 </xs:schema>

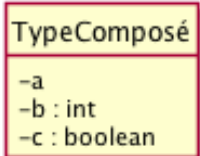
```

## 5 Attributs

Nous avons vu dans le chapitre 2 page 30 qu'il est possible de décrire des notions sous forme de sous-éléments ou bien d'attributs. Dans l'exemple précédent, le choix a été fait de n'utiliser que des sous-éléments.

Si l'on souhaite ajouter un attribut à un élément, alors cet élément devient automatiquement de type complexe. Attention toutefois à garder à l'esprit que le type de l'attribut lui même est forcément *basique*, c'est-à-dire non décomposable.

En UML, la notation des attributs est la même que pour les sous éléments.

En UML	En XMLSchema
 <pre> 1  TypeComposé 2  -a 3  -b : int 4  -c : boolean </pre>	<pre> 1 &lt;complexType name="TypeComposé"&gt; 2   &lt;attribute name="a"/&gt; 3   &lt;attribute name="b" type="int" default="42"/&gt; 4   &lt;attribute name="c" type="boolean" use="required"/&gt; 5 &lt;/complexType&gt; </pre>

*Note* : Si le type composé contient des sous-éléments *et* des attributs, les attributs sont déclarés dans le `complexType` après l'élément `sequence` (ou `choice` ou `group`).

On peut spécifier qu'un attribut est obligatoire en utilisant l'attribut `use` à la valeur `required`. De même, on peut donner une valeur par défaut si l'attribut n'est pas spécifié.

Par contre, on ne peut pas imposer un ordre pour les attributs. Ils pourront apparaître dans le document XML dans n'importe quel ordre.

## 6 Occurrences et Cardinalités

Dans tous les exemples précédents, lorsqu'un type complexe comportait un sous élément `<element name="..." type="..." />`, il en comportait par défaut 1 et 1 seul.

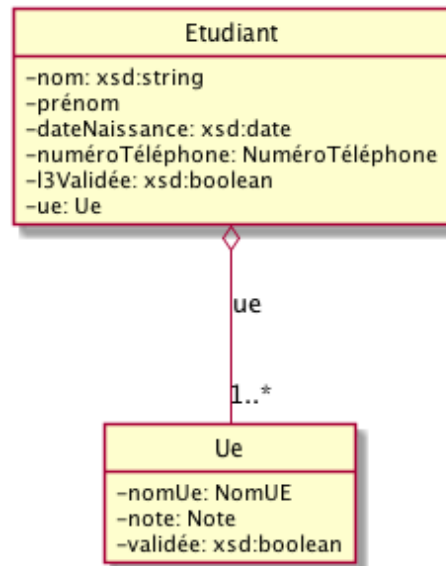
XMLSchema va permettre de préciser le nombre d'occurrences minimum et/ou maximum de chaque élément.

En UML, on va utiliser les nombres et le symbole `*` pour préciser *autant de fois que l'on souhaite*. Ainsi, on aura:

- `0..1` signifie que l'élément peut apparaître au minimum 0 fois et au maximum 1 fois. Cela signifie en fait que cet élément est optionnel.

- 0..\* signifie que l'élément peut apparaître au minimum 0 fois et autant de fois que nécessaire.
- 1..\* signifie que l'élément doit apparaître au moins une fois.
- 1..1 signifie que l'élément doit apparaître une et une seule fois (c'est l'option par défaut).

Par exemple, pour que dans l'exemple VI.1 un étudiant puisse choisir plusieurs options, on utilisera:



En XMLSchema, ce sont les attributs `minOccurs` et `maxOccurs` de l'élément `element` de XMLSchema qui vont permettre de préciser le nombre d'occurrences d'un élément.

```
<element name="nom" type="Type" minOccurs="..." maxOccurs="..." />
```

On utilisera le terme `unbounded` pour préciser *autant de fois que l'on souhaite*.

Par exemple,

- `<element name="ue" type="Ue" minOccurs="0" maxOccurs="1" />` signifie que le champ `ue` est optionnel (présent une fois ou absent).
- `<element name="ue" type="Ue" minOccurs="0" maxOccurs="unbounded" />` signifie que le champ `ue` est optionnel et peut apparaître autant de fois que nécessaire
- `<element name="ue" type="Ue" minOccurs="1" maxOccurs="unbounded" />` signifie que le champ `ue` est obligatoire (doit apparaître au moins une fois) et peut apparaître plusieurs fois.

### 6.a Types complexes - Extensions

Dans les langages de programmation orientés objet, on utilise l'héritage pour définir des objets de plus en plus complexes et la transmission de caractéristiques des objets, les objets fils héritant des caractéristiques (attributs et méthodes) des objets parents. XMLSchema permet de réaliser une forme d'héritage : les types enfants héritant des éléments et attributs des types parents.

En XMLSchema, nous avons vu les restrictions. Nous allons maintenant voir les extensions.

Le type complexe que l'on souhaite étendre et que l'on notera *type complexe étendu* comporte une extension. Une extension a une base, c'est à dire un type qui peut être un type simple (ou prédéfini) ou un type complexe. De la même manière qu'une restriction contenait les contraintes à apporter à la base, l'extension contiendra ce qui étendra le type, par exemple une séquence d'éléments et/ou des attributs.

Lorsque le contenu du type complexe étendu a une base correspondant à un type simple et que l'extension consiste qu'à rajouter des attributs, on dira que ce type complexe étendu a un contenu simple noté `simpleContent`. Dans ce cas, le type complexe étendu prendra la forme suivante :

```

1 <xs:complexType name="TypeComplexeEtendu">
2   <xs:simpleContent>
3     <xs:extension base = "NomTypeSimple">
4       <xs:attribute name="attr1" type="TypeAttribut1"/>
5       <xs:attribute name="attr2" type="TypeAttribut2"/>
6       ...
7     </xs:extension>
8   </xs:simpleContent>
9 </xs:complexType>

```

Lorsque le contenu du type complexe étendu a une base formée d'un type complexe contenant des éléments et/ou qu'on ajoute une séquence d'éléments, alors on dira que ce type complexe étendu a un contenu complexe noté **complexContent**. Dans ce cas, le type complexe étendu prendra la forme suivante :

```

1 <xs:complexType name="TypeComplexeEtendu">
2   <xs:complexContent>
3     <xs:extension base = "NomTypeSimpleOUComplexe">
4       <!-- sequence obligatoire si base = type simple -->
5       <xs:sequence>
6         <xs:element name="nomElement1" type="TypeElement1"/>
7         <xs:element name="nomElement2" type="TypeElement2"/>
8         ...
9       </xs:sequence>
10      <!-- attributs facultatifs -->
11      <xs:attribute name="attr1" type="TypeAttribut1"/>
12      <xs:attribute name="attr2" type="TypeAttribut2"/>
13      ...
14    </xs:extension>
15  </xs:complexContent>
16</xs:complexType>

```

Voyons maintenant les deux cas de figure, à commencer par les types complexes à contenu simple. Parfois, on souhaite stocker du texte directement dans un élément doté d'attributs, sans que cet élément ne contienne de sous-éléments. Par exemple, considérons `<note coeff="2">12.5</note>`. Comme un tel élément dispose d'attributs (un seul dans l'exemple), il s'agit d'un type complexe. Pour autant, il ne contient pas de séquence. Pour permettre à cet élément de contenir du texte, nous allons indiquer que cet élément est un *contenu simple*, autrement dit un **simpleContent**. Ce contenu simple dont la base sera un type simple, sera en revanche étendu (extension) d'un ou plusieurs attributs. Là où une restriction ajoute des contraintes à une base (un type prédéfini), une extension permet l'ajout de caractéristiques à cette base.

```

1 <xs:simpleType name="ValeurCoeff">
2   <xs:restriction base="xs:double">
3     <xs:minInclusive value="0.0"/>
4   </xs:restriction>
5 </xs:simpleType>
6
7 <xs:complexType name="Note">
8   <xs:simpleContent>
9     <xs:extension base = "xs:double">
10      <xs:attribute name="coeff" type="ValeurCoeff"/>
11    </xs:extension>
12  </xs:simpleContent>
13</xs:complexType>

```

Remarquons que le type simple qui sert de base à l'extension dans un **simpleContent** n'est pas nécessairement un type prédéfini de XMLSchema. On peut en effet tout à fait utiliser une type simple (comportant des restrictions) comme base pour une telle extension. Par exemple :

```

1 <xs:simpleType name="ValeurCoeff">
2   <xs:restriction base="xs:double">
3     <xs:minInclusive value="0.0"/>

```

```

4      </xs:restriction>
5 </xs:simpleType>
6
7 <xs:simpleType name="ValeurNote">
8     <xs:restriction base="xs:double">
9         <xs:minInclusive value="0.0"/>
10        <xs:maxInclusive value="20.0"/>
11    </xs:restriction>
12 </xs:simpleType>
13
14 <xs:complexType name="Note">
15     <xs:simpleContent>
16         <xs:extension base = "ValeurNote">
17             <xs:attribute name="coeff" type="ValeurCoeff"/>
18         </xs:extension>
19     </xs:simpleContent>
20 </xs:complexType>

```

Finissons par un exemple très complet pour illustrer les types complexes étendus à contenu complexe. Considérons l'extension d'une personne pour en faire un étudiant de L3 Miage :

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <xs:schema version="1.0"
3     xmlns:xs="http://www.w3.org/2001/XMLSchema"
4     xmlns:etu="http://www.uga.fr/etudiant"
5     targetNamespace="http://www.uga.fr/etudiant"
6     elementFormDefault="qualified">
7
8     <xs:element name="étudiant" type="etu:Etudiant"/>
9
10    <xs:complexType name="Etudiant">
11        <xs:complexContent>
12            <xs:extension base="etu:Contact">
13                <xs:sequence>
14                    <xs:element name="l3Validée" type="xs:boolean"/>
15                    <xs:element name="ue" type="etu:UE" maxOccurs="unbounded"
16                        />
17                </xs:sequence>
18            </xs:extension>
19        </xs:complexContent>
20    </xs:complexType>
21
22    <xs:complexType name="Contact">
23        <xs:complexContent>
24            <xs:extension base="etu:Personne">
25                <xs:sequence>
26                    <xs:element name="téléphone" type="etu:NuméroTéléphone"/>
27                    <xs:element name="adresse" type="etu:Adresse"/>
28                </xs:sequence>
29            </xs:extension>
30        </xs:complexContent>
31    </xs:complexType>
32
33    <xs:complexType name="Adresse">
34        <xs:sequence>
35            <xs:element name="numéro" type="xs:int"/>
36            <xs:element name="rue"/>
37            <xs:element name="codepostal" type="xs:int"/>
38            <xs:element name="ville"/>
39        </xs:sequence>
40    </xs:complexType>
41
42    <xs:simpleType name="NuméroTéléphone">
43        <xs:restriction base="xs:string">

```

```

43      <xs:pattern value="[+]\d{2}\s\d(\s\d\d){4}"/>
44    </xs:restriction>
45  </xs:simpleType>
46
47  <xs:complexType name="Personne">
48    <xs:sequence>
49      <xs:element name="nom"/>
50      <xs:element name="prénom"/>
51      <xs:element name="sexe" type="etu:Sexe"/>
52      <xs:element name="dateNaissance" type="xs:date"/>
53    </xs:sequence>
54  </xs:complexType>
55
56  <xs:simpleType name="Sexe">
57    <xs:restriction base="xs:string">
58      <xs:enumeration value="H"/>
59      <xs:enumeration value="F"/>
60    </xs:restriction>
61  </xs:simpleType>
62
63  <xs:simpleType name="ValeurCoeff">
64    <xs:restriction base="xs:double">
65      <xs:minInclusive value="0.0"/>
66    </xs:restriction>
67  </xs:simpleType>
68
69  <xs:simpleType name="ValeurNote">
70    <xs:restriction base="xs:double">
71      <xs:minInclusive value="0.0"/>
72      <xs:maxInclusive value="20.0"/>
73    </xs:restriction>
74  </xs:simpleType>
75
76  <xs:complexType name="UE">
77    <xs:sequence>
78      <xs:element name="note" type="etu:ValeurNote"/>
79      <xs:element name="validée" type="xs:boolean"/>
80    </xs:sequence>
81
82    <xs:attribute name="nom" type="etu:NomUE"/>
83    <xs:attribute name="coeff" type="etu:ValeurCoeff"/>
84  </xs:complexType>
85
86  <xs:simpleType name="NomUE">
87    <xs:restriction base="xs:string">
88      <xs:enumeration value="FDD"/>
89      <xs:enumeration value="IHM"/>
90      <xs:enumeration value="RO"/>
91      <xs:enumeration value="Anglais"/>
92    </xs:restriction>
93  </xs:simpleType>
94
95 </xs:schema>

```