Chapitre 8
XML schema

Les objectifs visés par la définition de schémas XML, en particulier les similitudes et différences par rapport aux DTDs, ont été largement débattues dans le chapitre 4. Le but de ce chapitre est de présenter les principaux concepts des schémas XML et de décrire de manière rigoureuse la syntaxe utilisée.

8.1 Structure d'un fichier XML schema

Une **définition de schéma XML** s'exprime à l'aide du langage XML. En d'autres termes, un fichier XML schéma est un fichier XML qui respecte la structure décrite à la section 5.2.

A l'instar d'une DTD, un schéma XML définit les types d'éléments pouvant intervenir dans une instance de document XML ainsi que les règles de composition autorisées. Il spécifie également la définition des attributs qui peuvent être associés aux éléments. Il permet aussi l'inclusion de commentaires.

En outre, il permet de spécifier un type pour les éléments terminaux et les attributs - non plus, limités au type "chaînes de caractères" mais pouvant être de type entier, réel, dates, etc. Le nombre d'occurrences autorisées peut être spécifié explicitement. Il offre également des mécanismes, destinés à faciliter la vie des concepteurs, en vue de modulariser la définition de modèles de contenu et, de ce fait, favoriser la réutilisation de définition d'éléments et d'attributs.

8.1.1 Prologue

En tant que fichier XML, un fichier de définition de schéma XML devrait toujours commencer par un **prologue** (cf. 5.2.1) tel que, par exemple:

```
<?xml version="1.0" encoding="UTF-8"?>
```

8.1.2 Définition du schéma

L'élément racine d'un schéma XML est l'élément `xs:schema`.; celui-ci fait référence à l'espace de nom utilisé pour la spécification même du langage de définition des schémas

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

8.1.3 Documentation d'un schéma

Bien que les commentaires usuels (de la forme `<!-- ... -->`) soient bien sûr utilisables dans un schéma, il est recommandé d'utiliser l'élément `xs:annotation` introduit par la norme XML Schema. Cet élément peut être inséré en tant que sous-élément de tout élément de schéma (à l'exception de l'élément `xs:annotation` lui-même et de ses descendants). L'élément `xs:annotation` admet deux types de sous-éléments:

- l'élément `xs:documentation` dont le contenu est destiné à être lu par un humain
- l'élément `xs:appinfo` dont le contenu est destiné à être exploité par une application (un exemple typique est l'utilisation du Dublin Core dont l'utilisation est supportée par bon nombres d'applications)

Ces deux éléments admettent un contenu mixte, à savoir une combinaison de texte et d'éléments (cf section 8.1.4).

8.1.4 Une question de vocabulaire

Conceptuellement, nous distinguons deux modèles de contenu pour les éléments:

- des **éléments structurés**; dont la structure de contenu est exprimée via une règle de composition, faisant potentiellement intervenir trois types de constructeurs: la séquence, le choix et le constructeur all;
- des **éléments terminaux**; dont le contenu est représenté de manière purement textuelle dans une instance de document (i.e. une séquence de caractères ne comportant pas de balises).

La norme XML Schema fait, quant à elle, la distinction suivante quant aux types de contenus des éléments intervenant dans une instance de document XML:

- des *éléments complexes* (nos éléments structurés) dont le contenu est représenté sous la forme d'une arborescence d'éléments, chacun d'eux étant obligatoirement balisé; sont également considérés comme complexes des éléments terminaux auquel on associe un ou plusieurs attributs;
- des *éléments simples* (nos éléments terminaux) dont le contenu est une chaîne de caractères absente de balises;
- des *éléments mixtes* (qui s'apparentent à nos éléments structurés); dont le contenu est représenté sous la forme d'une arborescence d'éléments, certains d'entre eux potentiellement non balisés;
- des *éléments vides* (qui s'apparentent à nos éléments terminaux); dont le contenu est inexistant mais dont la présence permet d'ancrer des attributs (un exemple typique est l'inclusion d'une image dont l'uri est fournie via un attribut).
- des *éléments any* (inclassable); dans la mesure où le modèle de contenu est laissé totalement libre.

8.2 Déclaration d'éléments structurés

8.2.1 Expression des règles de composition

La déclaration d'un élément structuré se fait à l'aide de l'élément `xs:element`.

- Le *nom de l'élément* défini est fourni via l'attribut `name`, qui lui est associé;
- La *règle de composition* des éléments est définie via l'élément fils `xsd:complex-Type`; celui-ci peut comporter trois types de constructeurs: la séquence, le choix et le all;
- Les *indicateurs d'occurrences* sont exprimés via les attributs `minOccurs` et `maxOccurs`. La valeur autorisée pour `minOccurs` est un nombre entier positif; la valeur autorisée pour `maxOccurs` est un nombre entier positif ou la valeur prédéfinie `unbounded`, spécifiant que le nombre autorisé d'occurrences n'est pas borné. Les attributs `minOccurs` et `maxOccurs` peuvent être associés aux éléments `xs:element` ou aux constructeurs *xs:sequence*, `xs:choice` et *xs:all*. La valeur par défaut des attributs `minOccurs` et `maxOccurs` est 1.

L'exemple 8.1 illustre la définition de l'élément recipe exprimant la structure générale d'une recette (telle que proposée par la DTD de l'exemple 5.3.). Le nom de l'élément (recipe) est défini via la valeur donnée à l'attribut `name` associé à `xs:element`. La règle de compostion (exprimée à l'aide de l'élément `xs:complexType`) est une simple séquence (utilisation de l'élément `xs:sequence`); elle comporte les éléments suivants, dont les occurrences sont contrôlées par les attributs `minOccurs` et `maxOccurs`: un élément `title`, 0 à n occurrences de l'élément `comment`, 1 à n occurrences de l'élément `item` et un élément optionnel `picture`. Les éléments `title` et `comment` sont des éléments terminaux; ils sont déclarés de type `xs:string` via l'attribut `type` (l'équivalent du type #PCDATA dans les DTDs). L'élément `item` est un élément structuré dont la structure doit encore être décrite. L'élément `picture` est un élément vide; nous le doterons d'un attribut indiquant l'adresse de l'image (cf section 8.3).

Exemple 8.1

```
<xs:element name="recipe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="comment" type="xs:string"
                        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="item" maxOccurs="unbounded"/>
      <xs:element name="picture" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

L'exemple 8.2 complète l'exemple 8.1 en y intégrant la définition de l'élément `item`, dont la règle de composition est plus complexe. L'élément `item` est constitué d'une séquence comportant un `header` optionnel et le choix entre deux modes de constructions: une suite d'éléments `ingredient` suivi d'une suite d'éléments `steps` ou, une suite combinée d'éléments `ingrédient` et `step`. Cet exemple illustre, notamment, l'utilisation de l'attribut `maxOccurs` en conjonction avec le constructeur `xs:choice`.

Exemple 8.2

```
<xs:element name="recipe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="comment" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="header" type="xs:string" minOccurs="0"/>
          <xs:choice>
            <xs:sequence>
              <xs:element name="ingredient" type="xs:string"
                          maxOccurs="unbounded"/>
               <xs:element name="step" type="xs:string" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:choice maxOccurs="unbounded">
              <xs:element name="ingredient" type="xs:string"/>
              <xs:element name="step" type="xs:string"/>
            </xs:choice>
          </xs:choice>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="picture" minOccurs="0"/>
  </xs:sequence>
  </xs:complexType>
</xs:element>
```

La norme XML Schema a introduit un nouveau constructeur (`xs:all`), une variante limitée du constructeur aggrégat de SGML. Celui-ci permet de définir un élément comme étant constitué d'une collection de sous-éléments. A la différence du constructeur `xs:sequence`, les sous-éléments peuvent apparaître dans n'importe quel ordre dans l'instance du document; leur occurrence est limitée à 0 ou 1.

A titre d'exemple, imaginons que nous complétions notre schéma de recette en ajoutant au niveau principal un descripteur constitué des sous-éléments suivants: pays d'origine, titre dans la langue originale et nom du contributeur. L'exemple 8.3 illustre l'utilisation du constructeur `xs:all` dans ce but.

Exemple 8.3

```
<xs:element name="Description">
  <xs:complexType>
    <xs:all>
      <xs:element name="country" type="xs:string"/>
      <xs:element name="originalTitle" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

8.2.2 Définition des modèles de contenu

La norme XML schema propose trois manières de définir des modèles de contenu d'éléments terminaux ou structurés:

- des *définitions locales*: le modèle de contenu d'un élément structuré est défini par des éléments fils de l'élément concerné; la définition d'un élément terminal est faite en utilisant l'attribut `type` faisant référence à un type prédéfini. L'exemple 8.2 ne fait usage que de définitions locales.
- des *définitions globales*: des éléments peuvent être nommément défini au niveau du shéma et la définition d'autres éléments peuvent y faire référence via l'attribut `href`. Cette façon de faire vise à *factoriser* des définitions de modèles de contenu, la modification d'un modèle de contenu se répercutant naturellement sur les définitions d'éléments y faisant référence. L'exemple 8.4 propose une nouvelle définition de l'élément `recipe` qui fait usage de ce mécanisme. L'élément structuré `item` est défini au niveau du shéma et référencé dans la définition de l'élément `recipe`. Les éléments terminaux `ingredient` et `step` sont également définis de manière globale et référencés dans la définition de l'élément `item`.
- des définitions reposant sur une *déclaration explicite de type*: outre le fait qu'elle permet de factoriser des définitions de modèles de contenu, l'utilisation d'une définition explicite de type permet au concepteur du schéma de la ré-utiliser en l'adaptant lors de la définition d'un nouveau type, via un mécanisme de dérivation similaire au concept d'héritage proposé par les langages de programmation objets (cf section xx). L'exemple 8.5 illustre cette dernière option où la définition de l'élément `item` fait référence, via l'attribut `type,` à une déclaration explicite du type `itemType`.

Exemple 8.4

```
<xs:element name="ingredient" type="xs:string"/>

<xs:element name="step" type="xs:string"/>

<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="header" type="xs:string" minOccurs="0"/>
      <xs:choice>
        <xs:sequence>
          <xs:element ref="ingredient" maxOccurs="unbounded"/>
          <xs:element ref="step" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:choice maxOccurs="unbounded">
          <xs:element ref="ingredient"/>
          <xs:element ref="step"/>
        </xs:choice>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="recipe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="comment" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
      <xs:element name="picture" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Exemple 8.5

```
<xs:complexType name="itemType">
  <xs:sequence>
    <xs:element name="header" type="xs:string" minOccurs="0"/>
    <xs:choice>
      <xs:sequence>
        <xs:element name="ingredient" type="xs:string" maxOccurs="unbounded"/>
        <xs:element name="step" type="xs:string" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="ingredient" type="xs:string"/>
        <xs:element name="step" type="xs:string"/>
      </xs:choice>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:element name="recipe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="comment" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="item" type="itemType"/>
      <xs:element name="picture" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Remarque importante: quelque soit la manière utilisée, parmi les trois possibilités existgantes pour définir les éléments, celles-ci n'ont aucune incidence sur le document balisé.

8.2.3 Déclaration d'éléments mixtes

La déclaration de types autorisant un contenu mixte se fait en associant l'attribut mixed à l'élément `xs:complexType` et en lui donnant la valeur `true`. Imaginons, à titre d'exemple, que nous autorisions dans la description des étapes de la recette d'inclure des passages importants (phase critique) et des termes à mettre en évidence (entrée de glossaire). L'exemple 8.6 illustre une telle variante de la définition du type `stepType`.

Exemple 8.6

```
<xs:complexType name="stepType" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="important" type="xs:string"/>
    <xs:element name="emphasizedTerm" type="xs:string"/>
  </xs:choice>
</xs:complexType>
```

8.3 Déclaration d'éléments terminaux

Remarque préliminaire: concernant le contenu d'éléments terminaux (ou la valeur d'attributs) la norme XML Schema fait la distinction entre deux espaces: *l'espace lexical*, dans lequel le contenu est considéré comme une suite de caractères, *l'espace de valeur*, dans lequel le contenu peut être interprété par une application en fonction du type qui lui a été associé.

8.3.1 Utilisation de types prédéfinis

La norme propose une série de types de base prédéfinis (résumés à la figure 1). Chacun de ces types possède son propre espace lexical et espace de valeur, ainsi que les règles déterminant quelle forme lexicale correspond à quelle valeur. Pour bon nombre de types prédéfinis, une valeur peut prendre différentes formes lexicales. Ainsi, par exemple, la même valeur d'un contenu de type `float` pourra indifféremment être représenté dans une instance de document comme "2.57" ou "+2.57" ou encore ".257E1". Certains de ces types sont définis via un mécanisme de dérivation (décrit à la section 8.3.2).

Les types prédéfinis peuvent schématiquement être rangés dans les grandes catégories suivantes:

- Les types *chaînes de caractères* qui ne sont pas destinées à faire l'objet de calculs et dont l'espace lexical correspond à l'espace de valeur: il s'agit du type `string` et de ses dérivés ainsi que des types assimilables: `hexBinary`, `base64Binary`, `anyURI`, `QName` et `NOTATION`;
- Les types *numériques*: leur définition reposent sur quatre types de base: `decimal`, `double`, `float` et `boolean`;
- Les types liés à la représentation des *dates* et du *temps*: `duration`, `dateTime`, `time`, `date`, `gYearMonth`, `gYear`, `gMonthDay`, `gMonth` et `gDay`.

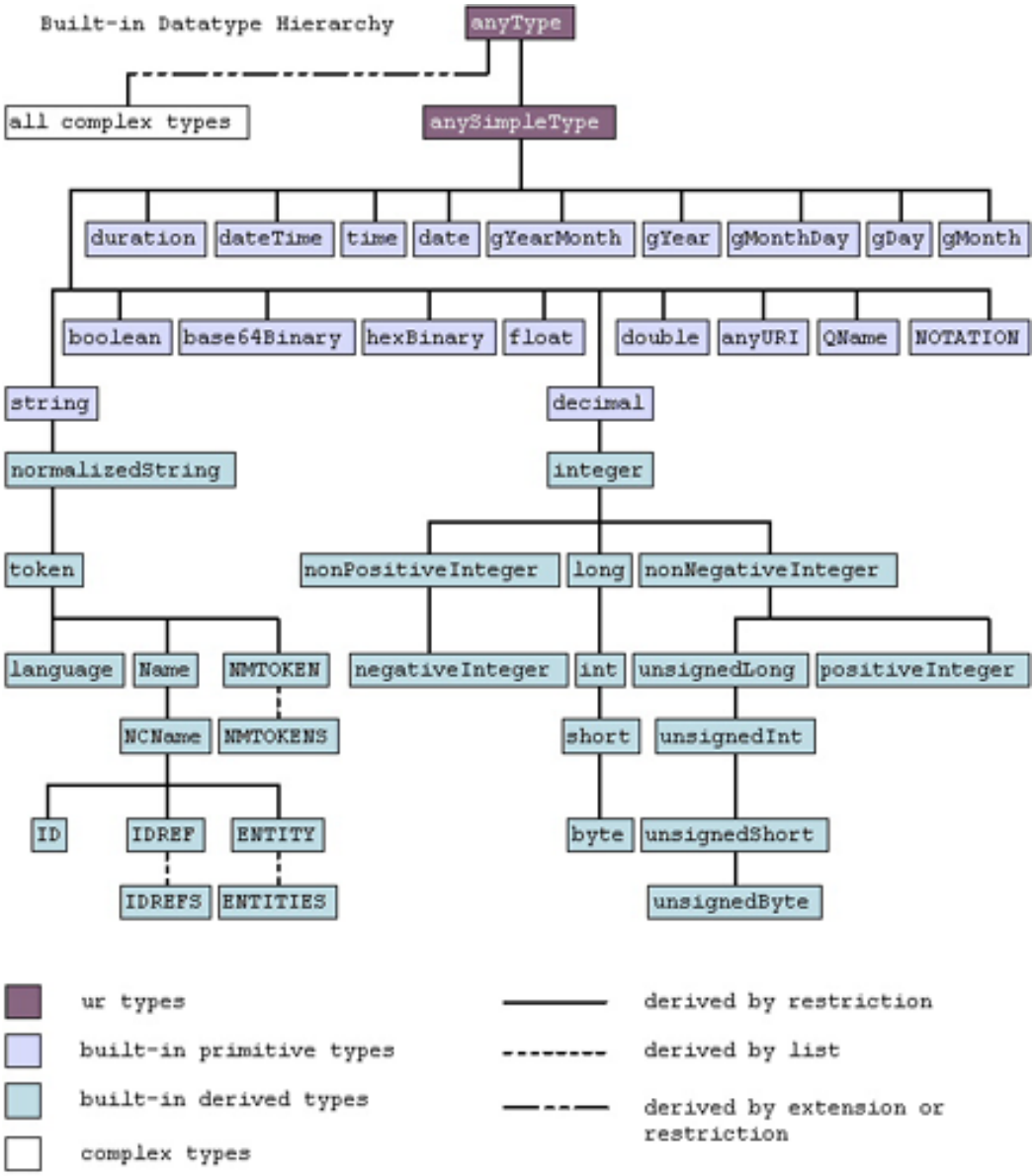

Figure 1: Hiérarchie des types prédéfinis

8.3.2 Définition de types par dérivation

Le concepteur d'un schéma XML a la possibilité de définir ses propres types en utilisant le mécanisme de dérivation proposé par la norme XML Schema; il existe trois manières de dériver des types simples:

- La *dérivation par restriction*: elle consiste à définir un nouveau type en exprimant des contraintes sur un type de base de manière à limiter l'espace des valeurs autorisées; elle ne modifie pas la sémantique associée au type de base;

- La *dérivation par liste*: elle consiste à définir un nouveau type dont la valeur autorisée est une liste d'éléments d'un type spécifié; dans l'instance du document les éléments de la liste doivent être séparés par un ou plusieurs espaces;
- La *dérivation par union*: elle consiste à défnir un nouveau type dont la valeur autorisée peut être d'un type choisi parmi un ensemble spécifié de types candidats.

Dérivation par restriction

La dérivation par restriction se fait par l'usage de ce que la norme qualifie de **facettes**.

D'un point de vue conceptuel, on peut distinguer trois types de facettes:

- les facettes agissant sur l'espace de valeurs

- la facette (`pattern`) agissant sur l'espace lexical

- les facettes ayant une influence sur la manière dont les espaces sont gérés lors du parsing du document

D'un point de vue syntaxique, une dérivation par restriction se fait en utilisant l'élément `xs:restriction`; la facette utilisée étant représentée par un (ou plusieurs) élément fils de `xs:estriction`. Le type simple auquel s'applique la facette, dénommé le *base datatype*, peut soit être référencé via l'attribut `base` (cf exemple 8.7) ou défini de manière locale dans l'élément restriction (cf exemple 8.8).

Exemple 8.7

```
<xs:simpleType name =”myInteger”>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="-2"/>
    <xs:maxExclusive value="5"/>
  </xs:restriction>
</xs:simpleType>
```

Exemple 8.8

```
<xs:simpleType name=”myInteger”>
  <xs:restriction>
    <xs:simpleType>
      <xs:restriction base=”xs:integer”>
        <xs:maxExclusive value=”5”/>
      </xs:restriction>
    </xs:simpleType>
    <xs:minInclusive value="-2"/>
  </xs:restriction>
</xs:simpleType>
```

Dérivation par liste

L'exemple 8.9 définit un type permettant, dans un instance de document, de fournir un élément constitué d'une liste de valeur entières (les éléments de la liste devant être

séparés par des espaces). Le type d'élément de la liste est spécifié par l'attribut `itemType` associé à `xs:list`.

Exemple 8.9

```
<xs:simpleType name="integerListType">
  <xs:list itemType="xs:integer">
  </xs:list>
</xs:simpleType>
```

Si, dans un schéma, l'élément integerList est défini de type integerListType, l'instance suivante est valide

```
<integerList>1 -25000 1000</integerList>
```

La définition d'un type liste peut également se faire en imbriquant la définition du type des éléments de la liste, comme l'illustre l'exemple 8.10qui permet de représenter une liste de valeurs entières (l'usage de la facette `xs:maxInclusive` borne à 100 la valeur maximale autorisée pour les éléments de la liste).

Exemple 8.10

```
<xs:simpleType name="myIntegerListType">
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:maxInclusive value="100"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:list>
</xs:simpleType>
```

Si, dans un schéma, l'élément myIntegerList est défini de type myIntegerListType, l'instance suivante est valide

```
<myIntegerList>1 -25000 100</myIntegerList>
```

Dérivation par union

La définition d'un type union se fait en utilisant l'élément xs:union; l'ensemble des types autorisés est spécifié via l'attribut memberTypes qui lui est associé (la valeur de l'attribut est une liste de types séparés par des espaces).

L'exemple 8.11 illustre l'utilisation du mécanisme d'union pour définir un type autorisant comme valeur indifféremment des entiers ou des dates.

Exemple 8.11

```
<xs:simpleType name="integerOrDate">
  <xs:union memberTypes="xs:integer xs:date"/>
</xs:simpleType>
```

Il est également possible de définir un type union en imbriquant les définitions des types simples le constituant comme l'illustre l'exemple 8.12, définissant un type permettant de spécifier unhe valeur entière ou une valeur conventionnelle "undefined".

Exemple 8.12

```
<xs:simpleType name="integerOrUndefined">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:integer"/>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="undefined"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

Synthèse

Le tableau fourni en annexe résume l'ensemble des facettes disponibles et indique à quels types d'éléments elles peuvent d'appliquer.

Les catégories de types sont regroupés de la manière suivante dans le tableau (à lire en conjonction avec la figure 1):

- les chaînes de caractères compressibles (il s'agit de types chaînes de caractères qui font l'objet d'une phase de compression d'espaces lors du parsing du document): `xs:Entity`, `xs:ID`, `xs:IDREF`, `xs:language`, `xs:Name`, `xs:NCName`, `xs:NMTOKEN`, `xs:token`, `xs:anyURI`, `xs:base64Binary`, `xs:NOTATION` et `xs:QName`
- les chaînes de caractères proprement dites (qui ne sont pas concernées par la phase de compression): `xs:string` et `xs:normalizedString`
- les types: `xs:double` et `xs:float`
- le type: `xs:decimal`
- les types entiers et dérivés: `xs:byte`, `xs:int`, `xs:integer`, `xs:long`, `xs:negativeIneger`, `xs:nonNegativeInteger`, `xs:nonPositiveInteger`, `xs:positiveInteger`, `xs:short`, `xs:unsignedByte`, `xs:unsignedInt`, `xs:unsignedLong` et xs:`unsignedShort`
- le type booléen: `xs:boolean`
- les types se rapportant à l'expression des dates et du temps: `xs:date`, `xs:dateTime`, `xs:duration`, `xs:gDay`, `xs:gMonth`, `xs:gMonthDay`, `xs:gYear`, `xs:gYearMonth` et `xs:time`
- les types définis par le mécanisme de liste

- les types definis par le mécanisme d'union

8.4 Déclaration d'attributs

Un attribut est une information qui peut être associée à un élément en vue de le qualifier; par exemple:

- le niveau de difficulté d'une recette: associé à l'élément `recipe`, l'attribut `difficulty` peut prendre les valeurs `easy`, `medium` ou `difficult`;
- le temps requis pour effectuer une étape de la recette: associé à l'élément `step`, l'attribut `duration` permet de spécifier une durée;
- la localisation d'une image: associé à l'élément `picture`, l'attribut `source` est utilisé pour indiquer l'adresse où la photo est disponible.

8.4.1 Types des attributs

Les attributs définis dans un schema XML doivent être de *type simple*; i.e. d'un type prédéfini par la norme (cf figure 1) ou d'un type défini par l'utilisateur via un des mécanismes de restriction de type simple décrit à la section 8.3.

L'exemple 8.13 illustre la définition de l'attribut `source` associé à l'élément optionnel `picture`; le type de l'attribut `source` est `xs:anyURI`, un type prédéfini par la norme XML schema.

Exemple 8.13

```
<xs:element name="picture" minOccurs="0">
  <xs:complexType>
    <xs:attribute name="source" type="xs:anyURI"/>
  </xs:complexType>
</xs:element>
```

L'exemple 8.14 illustre la définition de l'attribut `difficulty` associé à l'élément `recipe`; le type de l'attribut `difficulty` est un type dérivé défini par le concepteur du schéma qui fait usage de la facette `xs:enumeration` afin de restreindre le type prédéfini `xs:string`.

Exemple 8.14

```
<xs:element name="recipe">
  <xs:complexType>
       ...
      <xs:attribute name="difficulty">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="easy"/>
            <xs:enumeration value="medium"/>
            <xs:enumeration value="difficult"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
 </xs:element>
```

8.4.2 Contraintes d'occurrences et valeurs par défaut

Le concepteur d'un schéma XML peut spécifier le caractère *obligatoire* ou *optionnel* d'un attribut associé à un élément de son schéma via l'attribut `use` associé à la définition d'un attribut.

Ainsi, si dans notre exemple de recette, nous permettons de spécifier un niveau de difficulté (sans toutefois l'exiger) nous définirons l'attribut `difficulty` en donnant la valeur `optional` à l'attribut `use` (cf exemple 8.15)

Exemple 8.15

```
<xs:element name="recipe">
  <xs:complexType>
       ...
      <xs:attribute name="difficulty" use=”optional”>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="easy"/>
            <xs:enumeration value="medium"/>
            <xs:enumeration value="difficult"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
 </xs:element>
```

Si, de plus, nous souhaitons donner une *valeur par défaut* à l'attribut optionnel `difficulty` (imaginons que les contributeurs aux instances de documents proviennent d'une école hotelière et que, par conséquent, ils fournissent essentiellement des recettes difficiles à exécuter!), nous utiliserons l'attribut `default` et lui donnerons la valeur `difficult`. (cf exemple 8.16)

Exemple 8.16

```
<xs:element name="recipe">
  <xs:complexType>
       ...
      <xs:attribute name="difficulty" use=”optional” default=”difficult”>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="easy"/>
            <xs:enumeration value="medium"/>
            <xs:enumeration value="difficult"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
 </xs:element>
```

Il est également possible, pour des attributs optionnels ou obligatoires, d'imposer une valeur d'attribut fixée dans le schéma; cela se fait en utilisant l'attribut `fixed`. Un exemple typique d'une telle utilisation serait, par exemple, d'autoriser un commentaire en anglais à nos recettes, auquel cas nous pourrions associer à un élément `englishComment` un attribut `language` qui aurait systématiquement la valeur fixée `english`.

NB: Il n'est pas possible de spécifier une valeur par défaut pour un attribut obligatoire.

Enfin, mentionnons encore que l'attribut `use`, en plus des valeurs `required` et `optional`, peut prendre une troisième valeur (`prohibited`); celle-ci n'a d'intérêt que pour exclure la présence d'attributs lors de la conception objets de schémas (lors de la dérivation de types complexes par restriction).

8.4.3 Définition globales vs définition locales

Il existe deux manières de définir et associer des attributs à un élément: de manière *locale* (c'est la méthode qui a été illustrée dans la section 8.4.1) ou de manière *globale*.

La définition d'attributs globaux se fait en définissant les attributs au niveau du schéma; c'est-à-dire en les définissant en tant qu'éléments fils de l'élément `xs:schema`. L'avantage de cette pratique est de factoriser des définitions d'attributs en vue de les associer potentiellement à plusieurs éléments du schéma.

Imaginons que nous faisons évoluer notre schéma de définition de recette et que nous décidons d'associer des illustrations photographiques à d'autres éléments que `recipe` (par exemple, à `step`, `ingredient`, etc.). L'attribut `format` associé à `picture`, actuellement défini comme une énumération de valeurs admissibles limité à `jpeg` et `png` pourrait être redéfini de manière à mentionner le format `gif`. Il serait judicieux dans un tel cas de définir l'attribut au niveau gobal et de le référencer dans la définition de l'élément `picture` comme l'illustre la figure 8.17.

Exemple 8.17

```
<xs:attribute name="format">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="jpeg"/>
      <xs:enumeration value="png"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<xs:element name="picture" minOccurs="0">
  <xs:complexType>
    <xs:attribute name="format"/>
  </xs:complexType>
</xs:element>
```

8.4.4 Groupes d'attributs

La norme XML schema permet de regrouper une séquence d'attributs et de l'identifier par un nom; une tel *groupe d'attributs* pouvant dès lors être associé à n'importe quel élément du schéma. Les définitions de groupe d'attributs doivent se faire au niveau global et être référencés explicitement dans la définition d'éléments.

Reprenons l'exemple développé à la section 8.4.3, visant à factoriser les attributs associés à un élément de type image. Si nous voulons associer à un tel type d'élément deux attributs: l'un spécifiant son adresse (une URI) et l'autre son format (jpeg ou png ou gif), nous pouvons définir un groupe d'attributs, que nous appelerons `pictureFeatures`, regroupant les deux propriétés que nous souhaitons associer de manière systématique à une image.

L'exemple de la figure 8.18 illustre la définition du groupe d'attributs `pictureFeatures`. Syntaxiquement, ce groupe est défini par l'utilisation de l'élément `xs:attributeGroup`, le nom du groupe d'attributs étant spécifié par l'usage de l'attribut `name`. L'association de ce groupe d'atributs à l'élément `picture` se fait via l'usage de l'attribut `name` associé à l'élément `picture`.

Exemple 8.18

```
<xs:attributeGroup name="pictureFeatures">
  <xs:attribute name="source" type="xs:anyURI" use="required"/>
  <xs:attribute name="format" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="jpeg"/>
        <xs:enumeration value="png"/>
        <xs:enumeration value="gif"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:attributeGroup>

<xs:element name="picture" minOccurs="0">
  <xs:complexType>
    <xs:attributeGroup ref="pictureFeatures"/>
  </xs:complexType>
</xs:element>
```

8.5 Liens et contraintes d'unicité

Outre les relations hiérarchiques induites des règles de composition d'éléments structurés, il est utile de pouvoir spécifier des références entre éléments de données XML.

La possibilité de spécifier des références dans un document (telles que références à une figure, à une section, etc.) a été introduite dans les DTDs; elle se base sur l'utilisation des attributs de type ID, IDREF et IDREFS. Pour des raisons de compatiblité, la norme XML Schema simule ce mécanisme via les types xs:ID, xs:IDREF et xs:IDREFS. A la différence des DTDs, ces types peuvent aussi bien être utilisés pour spécifier des valeurs d'éléments terminaux que d'attributs.

La norme XML Schema introduit toutefois une autre manière de spécifier des liens; celle-ci est inspirée de la notion de clé d'accès existant dans les bases de données. Le mécanisme proposé repose sur trois constructions:

- `xs:key`; qui permet de spécifier un élément ou un attribut comme clé d'accès;
- `xs:unique`;qui permet de spécifier l'unicité d'une clé dans une sous-arborescence;
- `xs:keyref`; qui permet de spécifier une référence à un élément ou un attribut muni d'une propriété de type `xs:key` ou `xs:unique` .

La spécification d'une clé ou d'une contrainte d'unicité se fait en mentionnant, par rapport à l'élément concerné, le chemin d'accès (via l'élément `xs:selector`) et l'identification du sous-élément ou de l'attribut (via l'élément `xs:field`).

8.6 Sujets non développés

Dans ce chapitre nous avons présenté d'un point de vue technique et de manière synthétique, les principes des schémas XML. Nous avons essayé d'appliquer une certaine rigueur mais n'avons de loin pas traité tous les aspects de ce langage. Le lecteur intéressé se référera au livre: "XML Schema, Eric van der Vlist, éditions O'Reilly, 2002 (isbn: 0-596-00252-1)".

Parmi les thèmes non développés, mentionnons:

- Une discussion approfondie des **types de base** - Ceux-ci sont très proches des types proposés par la majorité des langages de programmation.
- Une description détaillée des **facettes** - Nous avons privilégié l'explication des concepts de base et mis l'accent sur une vue synthétique des possibilités offertes.
- Les mécanismes de **conception objets** des schémas - Afin de faciliter la vie des concepteurs de schémas, la norme XML Schema offre des mécanismes permettant de définir de nouveaux types sur la base de types complexes existants. Les constructions disponibles s'inspirent du paradigme de conception utilisé par les langages de programmation orientés objets. Dans ce sens, elles permettent la définition de nouveaux types complexes en étendant ou restreignant les caractéristiques de types préalablement définis.

8.7 Conclusion

La norme XML Schema propose une alternative aux DTDs (on peut pratiquement dire "supplante les DTDs") pour définir des modèles de données XML.

Si les schémas permettent de définir des classes de documents, au sens introduit par SGML, ils permettent en outre de définir des modèles de données qui se rapprochent de ceux utilisés par les bases de données. Cette évolution laisse présager un usage accru de données au format XML dans les systèmes d'information à l'avenir.

Dans les sections suivantes nous résumons les points essentiels développés dans ce chapitre et portons un regard critique sur la norme XML Schema.

8.7.1 Résumé

Voici de manière résumée les points importants développés dans ce chapitre:

- Le langage XML Schema, exprimé dans la syntaxe XML, étend les possibilités offertes par les DTDs pour la définition de structures génériques.
- Les règles de composition d'éléments structurés s'appuient sur trois constructeurs (la séquence, le choix et le constructeur all); elles permettent de spécifier explicitement le nombre d'occurrences des éléments.

- La définition de modèles de contenus peut se faire de trois manières: via des définitions locales (anonymes), via des définitions globales (dans le but de les factoriser) ou via des déclarations explicites de types (en vue de les réutiliser en les adaptant)
- Contrairement aux DTDS, les schémas XML permettent de spécifier un type pour le contenu des éléments terminaux et attributs, un ensemble de types prédéfinis, très similaire à ceux existants dans la majorité des langages de programmation, est défini par la norme.
- Le concepteur d'un schéma XML peut définir ses propres types sur la base de trois mécanismes de dérivation: par restriction, par liste ou par union.
- Outre l'association classique d'attributs à des éléments, la norme XML Schema permet de définir des groupes d'attributs; les définitions d'attributs peuvent être globales.
- Il existe deux mécanismes pour spécifier des liens entre éléments: l'un hérité de XML (via SGML) destiné à exprimer des références, l'autre inspiré de la notion de clé d'accès des bases de données; il est également possible de spécifier une contrainte d'unicité sur un sous-arbre.

8.7.2 Point de vue critique

Si les schémas XML marquent un pas dans deux directions: la possibilité de produire des documents exploitables par des applications diverses et une approche pour la modélisation de données , on peut néanmoins dresser un certain nombre de critiques à leur égard:

- En termes de modélisation, les schémas XML mélangent plusieurs paradigmes: les documents structurés, les hypertextes, les bases de données et préconisent une conception objets. Toutefois, dans leur formulation actuelle, l'amalgame de ces concepts est loin d'être maîtrisé.
- La distinction entre éléments et attributs, introduite par le langage SGML, soulève des difficultés d'ordre conceptuel; la notion de groupe (avec ses restrictions) introduite par les schémas ne fait qu'ajouter à la confusion.
- Un certain nombre de concepts anciens (par exemple, l'inclusion et l'exclusion) ont été supprimés lors du passage de SGML à XML. Néanmoins, un certain nombre d'entre eux subsistent dans les schémas XML (par exemple, le mécanisme de liens basé sur ID, IDREF et IDREFS)
- Le problème du traitement des espaces et séparateurs (déjà mentionné à propos des DTDs) reste complexe et constitue un frein aux développements d'applications.
- L'interprétation de certaines valeurs d'éléments et d'attributs (hors syntaxe balises) pose un réel problème en termes de standardisation.

- Les mécanismes introduits en vue de faciliter la conception modulaire de schémas sont complexes et font intervenir un grand nombre de possibilités dont l’articulation est loin d’être claire (espaces de noms, conception objets des types complexes, importation, groupes de subsitutions, mécanismes de contrôle des dérivations, etc.)
- Si la forme “verbeuse” de XML est intéressante (pour des questions de portabilité), elle peut poser des problèmes dans certains contextes applicatifs. Des initiatives sont en cours en vue de proposer des méthodes de compression appropriées.