Définir le format d'un document XML : DTD et Schémas

par G. Chagnon (Autres articles sur Developpez.com)

Date de publication : 16 mai 2002

Dernière mise à jour : 20 janvier 2009

Un fichier XML doit non seulement respecter des règles d'écriture vues précédemment ; il peut aussi, si on le désire, suivre des règles strictes d'enchâssements des éléments. Il existe deux grands langages de description pour ce faire.

Définition de Type de Document : Une DTD permet de décrire les éléments et leurs attributs autorisés dans un document XML.

Schémas XML : Les schémas XML permettent de décrire plus finement que les DTD l'ensemble des éléments et attributs autorisés dans un fichier XML.

I - Définition de Type de Document

I-A - Introduction

Il peut être parfois nécessaire de préciser les balises et attributs auxquels on a droit lors de la rédaction d'un document **XML**, par exemple si l'on veut pouvoir partager le même type de document avec une communauté d'autres rédacteurs. Deux solutions sont possibles : les « **Schémas XML** » et les « *Document Type Definition* ». Ces dernières sont les plus simples à manipuler et sont apparues en premier, alors que les Schémas n'étaient pas encore définis. Ce sont les raisons pour lesquelles nous allons nous limiter à elles pour le moment. Il faut néanmoins garder à l'esprit qu'il existe une autre solution, plus complexe certes, mais aussi plus puissante. Elle permet notamment d'informer plus efficacement l'utilisateur sur les balises auxquelles il a droit, ou bien de spécifier de manière plus détaillée le format autorisé pour le contenu de l'élément ou de l'attribut.

I-B - Types de DTD

I-B-1 - Introduction

Une **DTD** peut être stockée dans deux endroits différents. Elle peut être incorporée au document **XML** (elle est alors dite interne), ou bien être un fichier à part (on parle alors de **DTD** *externe*). Cette dernière possibilité permet de la partager entre plusieurs documents **XML**. Il est possible de mêler **DTD** interne et externe.

Il existe de surcroît deux types de **DTD** externes : privé ou public. Les **DTD** privées sont accessibles uniquement en local (sur la machine de développement), tandis que les publiques sont disponibles pour tout le monde, étant accessibles grâce à un URI (*Uniform Resource Identifier*).

Une déclaration de type de document est de la forme :

```
<!DOCTYPE elt.racine ... "..." "...">
```

Nous verrons progressivement par quoi remplacer les points de suspension. Cette déclaration se place juste après le prologue du document. L'élément racine du document **XML** rattaché à cette **DTD** est alors obligatoirement **elt.racine**. Par exemple...

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE commande ... "..." "boncommande.dtd">
 <commande>
 <item>(...)</item>
 <item>(...)</item>
 <item>(...)</item>
 </commande>
```

I-B-2 - Syntaxe

Le contenu ne change pas suivant le type de **DTD**, mais les déclarations d'une **DTD** interne sont écrites à l'intérieur du prologue du document **XML** alors que celles d'une **DTD** externe sont stockées dans un fichier... externe.

Exemple de déclarations pour une **DTD** interne :

```
<!DOCTYPE biblio[
 <!ELEMENT biblio (livre)*>
 <!ELEMENT livre (titre, auteur, nb_pages)>
   <!ATTLIST livre
```

```
    type (roman | nouvelles | poemes | théâtre) #IMPLIED
    lang CDATA "fr"
  >
 <!ELEMENT titre (#PCDATA)>
 <!ELEMENT auteur (#PCDATA)>
 <!ELEMENT nb_pages (#PCDATA)>
```

I-B-3 - DTD externe

Les deux types de **DTD** externes sont les **DTD** de type public et les **DTD** de type system. Le mot-clef **SYSTEM** indique que le fichier spécifié se trouve sur l'ordinateur local et qu'il est disponible uniquement à titre privé. Le mot-clé **PUBLIC** indique une ressource disponible pour tous sur un serveur distant.

Exemple de déclaration de **DTD** externe de type **SYSTEM** :

```
<!DOCTYPE biblio SYSTEM "bibliographie.dtd">
```

Le fichier associé est le suivant :

```
<!ELEMENT biblio (livre*)>
 <!ELEMENT livre (titre, auteur, nb_pages)>
   <!ATTLIST livre
     type (roman | nouvelles | poemes | théâtre) #IMPLIED
     lang CDATA "fr"
   >
 <!ELEMENT titre (#PCDATA)>
 <!ELEMENT auteur (#PCDATA)>
 <!ELEMENT nb_pages (#PCDATA)>
```

Exemple de déclaration de **DTD** externe de type **PUBLIC** :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
```

Dans l'exemple précédent, la chaîne de caractères après le mot PUBLIC fait référence tout d'abord à l'identifiant de la DTD (ici -, qui signifie que la DTD n'a pas de numéro d'enregistrement officiel), au propriétaire de la DTD (ici le W3C), puis son nom, enfin sa langue.

I-C - Déclarations d'éléments

I-C-1 - Généralités

Une déclaration d'élément est de la forme :

```
<!ELEMENT nom type_element>
```

nom est le nom de l'élément et **type_element** est le type auquel il est associé. Un élément peut être de type texte, vide, séquence ou choix d'élément. Dans ces deux derniers cas, on indique la liste des éléments-enfants. Passons ces types en revue.

I-C-2 - Élément texte

Cet élément est le plus répandu, puisque c'est celui qui contient... du texte. Il se déclare ainsi :

```
<!ELEMENT elt (#PCDATA)>
```

I-C-3 - Élément vide

Un élément vide est, comme son nom l'indique, un élément qui n'a aucun contenu -que ce soit de type texte, ou bien un autre élément. Le mot-clef utilisé pour la déclaration de ce type d'élément est **EMPTY** :

```
<!ELEMENT elt EMPTY>
```

Exemple d'utilisation :

```
<elt />
```

Un élément vide peut cependant fort bien posséder un ou plusieurs attributs. Par exemple

```
<img width="100" height="30" src="paysage.jpg" alt="Paysage de neige" />
```

I-C-4 - Séquence d'éléments

Une séquence d'éléments est une liste ordonnée des éléments qui doivent apparaître en tant qu'éléments-enfants de l'élément que l'on est en train de définir. Ce dernier ne pourra contenir *aucun* autre élément que ceux figurant dans la séquence. Cette liste est composée d'éléments séparés par des virgules et est placée entre parenthèses.

Chaque élément-enfant doit de plus être déclaré par ailleurs dans la **DTD** (avant ou après la définition de la liste, peu importe). Dans le fichier **XML**, ils doivent apparaître dans l'*ordre* de la séquence.

```
<!ELEMENT elt0 (elt1, elt2, elt3)>
```

Exemple d'utilisation valide :

```
<elt0>
   <elt1>(...)</elt1>
   <elt2>(...)</elt2>
   <elt3>(...)</elt3>
 </elt0>
```

Exemples d'utilisations non valides :

```
<elt0>
   <elt1>(...)</elt1>
   <elt3>(...)</elt3>
 </elt0>
```

... car l'élément **elt2** est manquant.

```
<elt0>
   <elt1>(...)</elt1>
   <elt3>(...)</elt3>
   <elt2>(...)</elt2>
 </elt0>
```

... car l'ordre des éléments n'est pas respecté.

I-C-5 - Choix d'éléments

Un choix d'élément donne... le choix dans une liste de plusieurs éléments possibles. L'utilisation précise dépend des indicateurs d'occurence. De même que pour la séquence, les éléments-enfants doivent être déclarés dans la **DTD**. Cette liste est composée d'éléments séparés par le caractère | (combinaison de touches **AltGr+6** sur un clavier AZERTY).

```
<!ELEMENT elt0 (elt1 | elt2 | elt3)>
```

Exemple d'utilisation valide :

```
<elt0><elt2>(...)</elt2></elt0>
```

Exemple d'utilisation non valide :

```
<elt0>
   <elt2>(...)</elt2>
   <elt3>(...)</elt3>
 </elt0>
```

I-C-6 - Indicateurs d'occurence

I-C-6-a - Syntaxe

Lors de la déclaration de séquence ou de choix d'éléments, à chaque élément enfant peut être attribuée une indication d'occurence (**?**, **+** ou *****).

Exemples d'indicateur d'occurences :

```
<!ELEMENT elt0 (elt1, elt2?, elt3+, elt*)>
```

- **elt1** ne comprend aucune indication d'occurence. Il doit donc apparaître une *seule et unique fois* dans l'élément **elt0** ;

- **elt2** a pour indication d'occurence **?**. Cela signifie que l'élément doit apparaître *au maximum* une fois (il peut ne pas apparaître du tout) ;
- **elt3** a pour indication d'occurence **+**. Cela signifie que l'élément doit apparaître *au moins* une fois ;
- **elt4** a pour indication d'occurence *****. Cela signifie que l'élément peut apparaître autant de fois que l'auteur le désire, voire pas du tout.

I-C-6-b - Exemples

Les indicateurs d'occurence peuvent être utilisés en conjonction avec les séquences ou les choix d'éléments. Ainsi...

```
<!ELEMENT elt0 (elt1+, elt2*, elt3?)>
```

... permet d'indiquer une séquence composée d'au moins un élément **elt1**, puis d'un nombre indéterminé d'éléments **elt2** (éventuellement nul), enfin au plus un élément **elt3**.

Exemple d'utilisation d'un choix d'éléments avec indicateurs d'occurence par élément :

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

Exemple d'utilisation valide :

```
<elt0>
   <elt2>(...)</elt2>
   <elt2>(...)</elt2>
 </elt0>
```

Exemples d'utilisation non valide :

```
<elt0>
   <elt3>(...)</elt3>
   <elt2>(...)</elt2>
 </elt0>
```

```
<elt0>
   <elt2>(...)</elt2>
   <elt3>(...)</elt3>
 </elt0>
```

Exemple d'utilisation d'un choix d'éléments avec indicateur d'occurence global :

```
<!ELEMENT elt0 (elt1 | elt2 | elt3)*>
```

Exemple d'utilisation valide :

```
<elt0>
   <elt2>(...)</elt2>
   <elt3>(...)</elt3>
```

```
   <elt1>(...)</elt1>
 </elt0>
```

Dans ce dernier cas, il n'y a pas de contrainte visible sur l'ordre d'apparition des éléments. C'est la déclaration la plus souple possible.

I-C-7 - Élément quelconque

L'élément quelconque est l'élément-« fourre-tout » dans une **DTD**. Il peut contenir tout autre élément défini dans la **DTD**, aussi bien qu'être vide ou contenir du texte. Les éléments-enfants éventuels peuvent apparaître dans n'importe quel ordre, et en quantité non définie. Il est préférable de ne pas utiliser trop souvent ce type de déclaration, car on perd les avantages qu'offre la rédaction d'une **DTD**, qui sont de fixer des contraintes précises sur la structure du document **XML** qui lui est lié. Le mot-clef utilisé pour la déclaration de ce type d'élément est **ANY**.

```
<!ELEMENT elt ANY>
```

I-C-8 - Élément à contenu mixte

Un élément à contenu mixte peut contenir aussi bien du texte que des éléments-enfants. Il se présente comme une liste de choix, avec des indicateurs d'occurence bien choisis. Le texte contenu peut se trouver à n'importe quel endroit dans l'élément, et peut être une section **CDATA**.

Exemple de déclaration :

```
<!ELEMENT citation (#PCDATA | auteur)*>
```

Exemple d'utilisation :

```
<citation>
   <auteur>Shakespeare</auteur>Être ou ne pas être
 </citation>
```

I-C-9 - Exercice : Écriture d'une DTD avec éléments

Rédiger une DTD pour une bibliographie. Cette bibliographie :

- contient des livres et des articles ;
- les informations nécessaires pour un livre sont :
  - son titre général ;
  - les noms des auteurs ;
  - ses tomes et pour chaque tome, leur nombre de pages ;
  - des informations générales sur son édition comme par exemple le nom de l'éditeur, le lieu d'édition, le lieu d'impression, son numéro ISBN ;
- les informations nécessaires pour un article sont :
  - son titre ;
  - les noms des auteurs ;

- ses références de publication : nom du journal, numéro des pages, année de publication et numéro du journal
- on réservera aussi un champ optionnel pour un avis personnel.

Tester cette DTD avec un fichier XML que l'on écrira ex-nihilo et validera.

Correction

I-D - Déclarations d'attributs

I-D-1 - Introduction

Comme on peut trouver dans un document **XML** des éléments possédant des attributs, il est normal que la **DTD** permette de définir des contraintes sur ces derniers. On peut déclarer et attacher à un élément donné chaque attribut séparément, mais il est souvent préférable, afin d'améliorer la lisibilité du code, de les réunir sous la forme d'une liste. Chaque attribut défini dans la liste possède un nom et un type. On peut lui donner une valeur par défaut, ou bien le spécifier obligatoire. Le mot-clef de cette déclaration est **ATTLIST**.

I-D-2 - Type chaîne de caractères

Il s'agit là du type d'attribut le plus courant. Une chaîne de caractères peut être composée de caractères ainsi que d'**entités analysables**. Le mot-clef utilisé pour la déclaration de chaîne de caractère est **CDATA**.

Exemple de déclaration de **CDATA** (nous reviendrons dans un instant sur la signification du mot-clef **#IMPLIED**) :

```
<!ELEMENT elt (...)>
   <!ATTLIST elt attr CDATA #IMPLIED>
```

Exemples d'utilisations :

```
<elt attr="Chaîne de caractères">(...)</elt>
```

```
<!ENTITY car "caractères">
 <elt attr="Chaîne de &car;">(...)</elt>
```

I-D-3 - Valeurs par défaut

Chaque attribut peut être requis, optionnel ou fixe et avoir une valeur par défaut. Les exemples suivants montrent la déclaration d'un attribut appelé **attr** attaché à un élément nommé **elt**.

1. Déclaration d'un attribut avec une valeur par défaut :

```
<!ELEMENT elt (...)>
   <!ATTLIST elt attr CDATA "valeur">
```

Un tel attribut n'est pas obligatoire. S'il est omis dans le fichier **XML** lors de l'utilisation de l'élément **elt**, il est considéré comme valant valeur. Dans cet exemple, si on écrit **<elt>(...)</elt>**, cela est équivalent à écrire **<elt attr="valeur">(...)>/elt>**.

2. Déclaration d'un attribut requis :

```
<!ELEMENT elt (...)>
   <!ATTLIST elt attr CDATA #REQUIRED>
```

Un tel attribut est obligatoire. Son absence déclenche une erreur du vérificateur syntaxique sur le fichier **XML**.

3. Déclaration d'un attribut optionnel :

```
<!ELEMENT elt (...)>
   <!ATTLIST elt attr CDATA #IMPLIED>
```

4. Déclaration d'un attribut avec une valeur fixe :

```
<!ELEMENT elt (...)>
   <!ATTLIST elt attr CDATA #FIXED "valeur">
```

L'utilité d'un tel attribut peut sembler bizarre à première vue, puisqu'il ne peut prendre qu'une seule valeur. Cette fonctionnalité est cependant utile lors d'une mise à jour d'une **DTD**, pour anticiper la compatibilité avec des versions ultérieures.

I-D-4 - Type ID

Ce type sert à indiquer que l'attribut en question peut servir d'*identifiant* dans le fichier **XML**. Deux éléments ne pourront pas posséder le même atribut possédant la même valeur.

Exemple de déclaration de type **ID** optionnel :

```
<!ELEMENT elt (...)>
   <!ATTLIST elt attr ID #IMPLIED>
 <!ELEMENT elt1 (...)>
   <!ATTLIST elt1 attr ID #IMPLIED>
 <!ELEMENT elt2 (...)>
   <!ATTLIST elt2 attr ID #IMPLIED>
```

La déclaration précédente interdit par exemple...

```
<elt1 attr="machin"></elt1>
 <elt2 attr="truc"></elt2>
 <elt1 attr="machin"></elt1>
```

... ainsi que

```
<elt1 attr="machin"></elt1>
 <elt2 attr="machin"></elt2>
 <elt1 attr="truc"></elt1>
```

I-D-5 - Type énuméré

On peut parfois désirer limiter la liste de valeurs possibles pour un attribut. On le définit alors comme étant de type énuméré. Donner une autre valeur dans le fichier **XML** provoque une erreur.

Exemple de déclaration d'une liste de choix d'attributs :

```
<!ELEMENT img EMPTY>
   <!ATTLIST img format (GIF | JPEG | PNG) "PNG">
```

Cet exemple déclare un attribut **format** d'un élément **img**. La valeur de cet attribut peut être **PNG**, **GIF** ou **JPEG**. Si aucune valeur n'est affectée à cet attribut, c'est la valeur par défaut qui le sera, ici **PNG**. On notera l'absence de guillemets dans la liste des valeurs possibles. En ajouter est une erreur courante dans la rédaction d'une **DTD**.

I-D-6 - Utilisation de liste pour les attributs

On utilise le fait qu'il est possible de « factoriser » le nom de l'élément. Par exemple...

```
<!ELEMENT elt (...)>
   <!ATTLIST elt
     attr1 CDATA #IMPLIED
     attr2 CDATA #REQUIRED
   >
```

I-D-7 - Exercice : Écriture d'une DTD avec attributs

Modifier la DTD précédente...

- ... en ajoutant un attribut optionnel soustitre à l'élément titre ;
- ... en faisant de l'élément tome un élément vide et en lui ajoutant un attribut requis nb_pages et un attribut optionnel soustitre ;
- ... en faisant de l'élément nom_journal un attribut de l'élément journal et en lui donnant comme valeur par défaut Feuille de Chou ;
- ... en faisant de l'élément annee un attribut de type énuméré, prenant comme valeurs possibles 2000, 2001, 2002, "avant_2000" et "inconnue" et proposant comme valeur par défaut inconnue.

Utiliser cette DTD pour créer un fichier XML valide.

Correction

I-E - Déclarations d'entités

I-E-1 - Introduction

Les déclarations d'entités permettent de disposer de l'équivalent de raccourcis clavier et de caractères a *priori* non accessibles dans le jeu de caractères sélectionné.

I-E-2 - Les entités paramétriques

Elles servent à définir des symboles qui seront utilisés ailleurs dans la **DTD**. Ce sont en quelque sorte des raccourcis d'écriture : partout où une entité est mentionnée, elle peut être remplacée par la chaîne de caractères qui lui est associée. Ce mécanisme s'apparente à un mécanisme de « macro ».

Les entités paramétriques ne peuvent pas être utilisées en-dehors d'une **DTD**.

Exemple tiré de la **spécification du langage HTML** :

```
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
```

L'exemple précédent a pour effet d'indiquer au système que toute occurence de **%heading;** doit être remplacée par **H1|H2|H3|H4|H5|H6** dans la **DTD**.

Ce mécanisme peut également servir à utiliser un nom relativement compréhensible à la place d'une séquence de caractères peu évocatrice. La définition d'une entité peut également faire référence à d'autres entités ; la substitution est alors effectuée de proche en proche.

I-E-3 - Les entités de caractères

Elle servent à donner un nom facilement lisible à des caractères qui ne sont pas représentables dans l'alphabet utilisé, ou qui ne sont pas disponibles au clavier.

Exemples tirés de la **DTD** du langage **HTML 4.01** :

```
<!ENTITY nbsp " ">
 <!ENTITY eacute "&#233;">
```

Les entités de caractères définies dans une **DTD** peuvent être utilisées dans un document **XML** référençant cette **DTD** à l'aide de la notation **&NomEntité;** (par exemple **&ecaute;** pour reprendre une des entités précédentes).

I-E-4 - Les entités internes

Ce sont des symboles pouvant être définis dans une **DTD** et utilisés dans un document **XML** comme raccourcis d'écriture. La définition complète du symbole est entièrement incluse dans la **DTD**. Exemple :

```
<!ENTITY ADN "Acide désoxyribonucléique">
```

Dans le fichier **XML**, l'appel à **&ADN;** sera aumatiquement remplacé, lors de l'affichage ou du traitement, par la chaîne de caractères "*Acide désoxyribonucléique*".

I-E-5 - Les entités externes

Il s'agit...

... soit de symboles pouvant être définis dans un autre fichier, mais pouvant être utilisés dans un document XML ou la DTD elle-même. Par exemple :

```
<!ENTITY Inclusion SYSTEM "toto.xml">
 <!ENTITY % Inclusion SYSTEM "toto.inc">
```

Dans le fichier **XML**, le contenu du fichier **toto.xml** sera inséré à l'appel de l'entité **&Inclusion;**, et dans la **DTD**, le contenu du fichier **toto.inc** sera inséré à l'appel de l'entité **&Inclusion;**

... soit de symboles pouvant être définis dans une autre **DTD** et utilisés dans la **DTD** courante :

```
<!ENTITY % HTMLSpecial PUBLIC "-//W3C//ENTITIES Special for XHTML//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml-special.ent">
```

Le contenu de cette **DTD** (qui peut être de type **SPECIAL** ou **PUBLIC**) est importé dans la **DTD** courante par l'appel de **%HTMLSpecial;**.

I-E-6 - Exercice : Déclarations d'entités

On visualisera les documents XML avec Internet Explorer ou Mozilla.

- Rédiger un document XML de quelques mots, comportant une DTD externe déclarant deux éléments :
  - elt.racine peut contenir autant de fois que l'on veut du texte ou elt.enfant ;
  - elt.enfant peut contenir du texte.
- Déclarer une entité paramétrique entite1 permettant d'insérer dans la DTD la chaîne de caractères "#PCDATA|elt.enfant". Utiliser cette entité dans la déclaration de l'élément elt.racine.
- Déclarer une entité entite2 permettant d'insérer "<elt.enfant>entité</elt.enfant>" et l'appeler dans un corps de texte d'elt.racine.
- Déclarer l'entité de caractère Eacute comme étant le caractère É (qui correspond à É). L'appeler dans un corps de texte.

Correction : **Fichier XML** - **DTD**

II - Initiation aux Schémas XML

II-A - Introduction

II-A-1 - Limitations des DTD

Lors de son lancement, **XML** a été perçu comme une réelle chance pour les développeurs d'avoir à disposition un langage simple d'utilisation, portable sans difficulté d'une machine -et d'une application- à une autre, et libre de droits. Dans les premiers temps, un fichier **XML**, si on voulait le standardiser en utilisant un vrai langage général de description, devait dépendre d'une **DTD**. Mais ce format de description, hérité de **SGML**, souffre de nombreuses déficiences.

1. Premièrement, les **DTD** ne sont pas au format **XML**. Cela signifie qu'il est nécessaire d'utiliser un outil spécial pour manipuler un tel fichier, différent de celui utilisé pour l'édition du fichier **XML**.
2. Deuxièmement, les **DTD** ne supportent pas les « espaces de nom » (nous reviendrons sur cette notion). En pratique, cela implique qu'il n'est pas possible d'importer des définitions de balises définies par ailleurs dans un fichier **XML** défini par une **DTD**.
3. Troisièmement, le « typage » des données (c'est-à-dire la possibilité de spécifier par exemple qu'un attribut ne doit être qu'un nombre entier) est extrêmement limité.

II-A-2 - Apports des schémas

Conçu pour pallier les déficiences pré-citées des **DTD**, **XML Schema** propose des nouveautés en plus des fonctionnalités fournies par les **DTD** :

- Le typage des données est introduit, ce qui permet la gestion de booléens, d'entiers, d'intervalles de temps... Il est même possible de créer de nouveaux types à partir de types existants.
- La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément..
- Le support des espaces de nom.
- Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif (rappel : dans une **DTD**, on est limité à 0, 1 ou un nombre infini d'occurences pour un élément).
- Les schémas sont très facilement concevables par modules.

II-B - Les premiers pas

II-B-1 - Introduction

Le but d'un schéma est de définir une classe de documents **XML**. Il permet de décrire les autorisations d'imbrication et l'ordre d'apparition des éléments et de leurs attributs, tout comme une **DTD**. Mais il permet aussi d'aller au-delà.

Un premier point intéressant est qu'un fichier **Schema XML** est un document **XML**. Cela permet à un tel document d'être manipulé de la même manière que n'importe quel autre fichier **XML**, et en particulier par une feuille de style **XSL**. Par exemple, il est notamment possible d'automatiser la création d'une documentation à partir d'un schéma, fondée sur les commentaires et explications qui s'y trouvent. C'est d'ailleurs chose facile avec l'éditeur oXygen, via le menu Modules d'extension>Schema Documentation, à partir d'un **exemple de schéma**, de produire la **documentation correspondante**.

Le vocabulaire de **XML Schema** est composé d'environ 30 éléments et attributs. Ce vocabulaire est, a priori de manière bizarrement récursive et « auto-référente », défini dans un Schéma. Mais il existe également une DTD.

II-B-2 - Structure de base

Comme tout document **XML**, un **Schema XML** commence par un prologue, et a un élément racine.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <!-- déclarations d'éléments, d'attributs et de types ici -->
 </xsd:schema>
```

L'élément racine est l'élément **xsd:schema**. Pour le moment, oubliez l'attribut **xmlns:xsd** (dont le rôle est le même que celui que nous avons déjà rencontré lors du cours sur les feuilles de style), et qui fait référence à l'**espace de noms** utilisé pour l'écriture du fichier. Il faut simplement retenir que tout élément d'un schéma doit commencer par le préfixe **xsd**.

Nous allons voir, par la suite, comment déclarer éléments et attributs à l'aide d'un schéma.

II-C - Déclarations d'éléments et d'attributs

II-C-1 - Déclarations d'éléments

Un élément, dans un schéma, se déclare avec la balise **<xsd:element>**. Par exemple,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="contacts" type="typeContacts" />
   <xsd:element name="remarque" type="xsd:string" />
 <!-- déclarations de types ici -->
 </xsd:schema>
```

Le schéma précédent déclare deux éléments : un élément **contacts** et un élément **remarque**. Chaque élément est « typé » -c'est-à-dire qu'il doit respecter un certain format de données. L'élément **contacts** est ainsi du type **typeContacts**, qui est un **type complexe** défini par l'utilisateur. L'élément remarque quant à lui est du type **xsd:string** qui est un **type simple** prédéfini de **XML Schema**.

Chaque élément déclaré est associé à un type de données via l'attribut **type**. Les éléments pouvant contenir des élément-enfants ou posséder des attributs sont dits de type *complexe*, tandis que les éléments n'en contenant pas sont dits de type *simple*. Nous reviendrons plus loin sur cette notion de **type de données**.

II-C-2 - Déclarations d'attributs

II-C-2-a - Déclaration simple

À la différence des éléments, un attribut ne peut être que de type simple. Cela signifie que les attributs, comme avec les **DTD**, ne peuvent contenir d'autres éléments ou attributs. De plus, les déclarations d'attributs doivent être *placées après* les définitions des types complexes, autrement dit, après les éléments **xsd:sequence, xsd:choice** et **xsd:all** (voir plus loin). Pour mémoire, rappelons que dans une **DTD**, l'ordre des déclarations n'a pas d'importance.

L'exemple suivant montre la déclaration d'un attribut **maj** de type **xsd:date** (un autre type simple) qui indique la date de dernière mise à jour de la liste des contacts.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="contacts" type="typeContacts" />
   <xsd:element name="remarque" type="xsd:string">     <!-- déclarations de types ici -->
     <xsd:complexType>
     <!-- déclarations du modèle de contenu ici -->
       <xsd:attribute name="maj" type="xsd:date" />
     </xsd:complexType>
 </xsd:element>
 </xsd:schema>
```

II-C-2-b - Contraintes d'occurences

Tout comme dans une **DTD**, un attribut peut avoir un indicateur d'occurences.

L'élément **attribute** d'un **Schema XML** peut avoir trois attributs optionnels : **use, default** et **fixed**. Des combinaisons de ces trois attributs permettent de paramétrer ce qui est acceptable ou non dans le fichier **XML** final (attribut obligatoire, optionnel, possédant une valeur par défaut...). Par exemple, la ligne suivante permet de rendre l'attribut **maj** optionnel, avec une valeur par défaut au 11 octobre 2003 s'il n'apparaît pas (le format de date est standardisé : cette date s'écrit donc à l'anglo-saxonne année-mois-jour ; cela permet en outre de plus facilement classer les dates).

```
<xsd:attribute name="maj" type="xsd:date" use="optional" default="2003-10-11" />
```

Quand l'attribut **fixed** est renseigné, la seule valeur que peut prendre l'attribut déclaré est celle de l'attribut **fixed**. Cet attribut permet de "réserver" des noms d'attributs pour une utilisation future, dans le cadre d'une mise à jour du schéma.

Le tableau suivant présente une comparaison entre le format **DTD** et le **XML Schema**.

DTD	Attribut **use**	Attribut **default**	Commentaire
#REQUIRED	required	-	
"blabla" #REQUIRED	required	blabla	
#IMPLIED	optional	-	
"blabla" #IMPLIED	optional	blabla	
-	prohibited	-	Cet attribut ne doit pas apparaître

*Table 1. Contraintes d'occurences fixables par les attributs **use** et **default**.*

Il est à noter que la valeur de l'attribut **default** doit être conforme au type déclaré. Par exemple...

```
<xsd:attribute name="maj" type="xsd:date" use="optional" default="-43" />
```

... produirait une erreur à la validation du schéma.

Un autre type de déclaration d'attributs dans les DTD, la liste de choix, est possible grâce à une **restriction de type** ; nous y reviendrons.

II-C-2-c - Regroupements d'attributs

XML Schema propose une fonctionnalité supplémentaire, permettant de déclarer des groupes d'attributs (groupes auxquels il est possible de faire appel lors d'une déclaration d'éléments). Cela permet d'éviter de répéter des informations de déclarations.

II-C-2-d - Déclaration d'élément ne contenant que du texte avec un (ou plusieurs) attribut(s)

Un tel élément est de type complexe, car il contient au moins un attribut. Afin de spécifier qu'il peut contenir également du texte, on utilise l'attribut **mixed** de l'élément **xsd;complexType**. Par défaut, **mixed="false"**; il faut dans ce cas forcer **mixed="true"**. Par exemple,

```
<xsd:element name="elt">
   <xsd:complexType mixed="true">
     <xsd:attribute name="attr" type="xsd:string" use="optional" />
   </xsd:complexType>
 </xsd:element>
```

II-C-3 - Déclaration et référencement

La procédure précédente de déclaration d'éléments peut amener à une structure de type « poupée russe » des déclarations. Pour des raisons de clarté, il est beaucoup plus avantageux d'ordonner ces déclarations, ainsi qu'on peut le voir sur cet **exemple** (on ne fera pas attention, pour le moment, aux « définitions de type »).

Il est recommandé de commencer par déclarer les éléments et attributs de type simple, puis ceux de type complexe. On peut en effet faire référence, dans une déclaration de type complexe, à un élément de type simple préalablement défini. Par exemple...

```
<xsd:element name="livre">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="auteur" type="xsd:string" />
       <xsd:element name="pages" type="xsd:positiveInteger" />
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
```

... est plus difficile à maintenir que...

```
<xsd:element name="pages" type="xsd:positiveInteger" />
 <xsd:element name="auteur" type="xsd:string" />
 <xsd:element name="livre">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="auteur" />
       <xsd:element ref="pages" />
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
```

II-D - Les types de données

II-D-1 - Introduction

Ainsi que nous l'avons déjà brièvement signalé, **XML Schema** permet de spécifier des types de données bien plus finement que le langage **DTD**. Il distingue notamment **types simples** et **types complexes**.

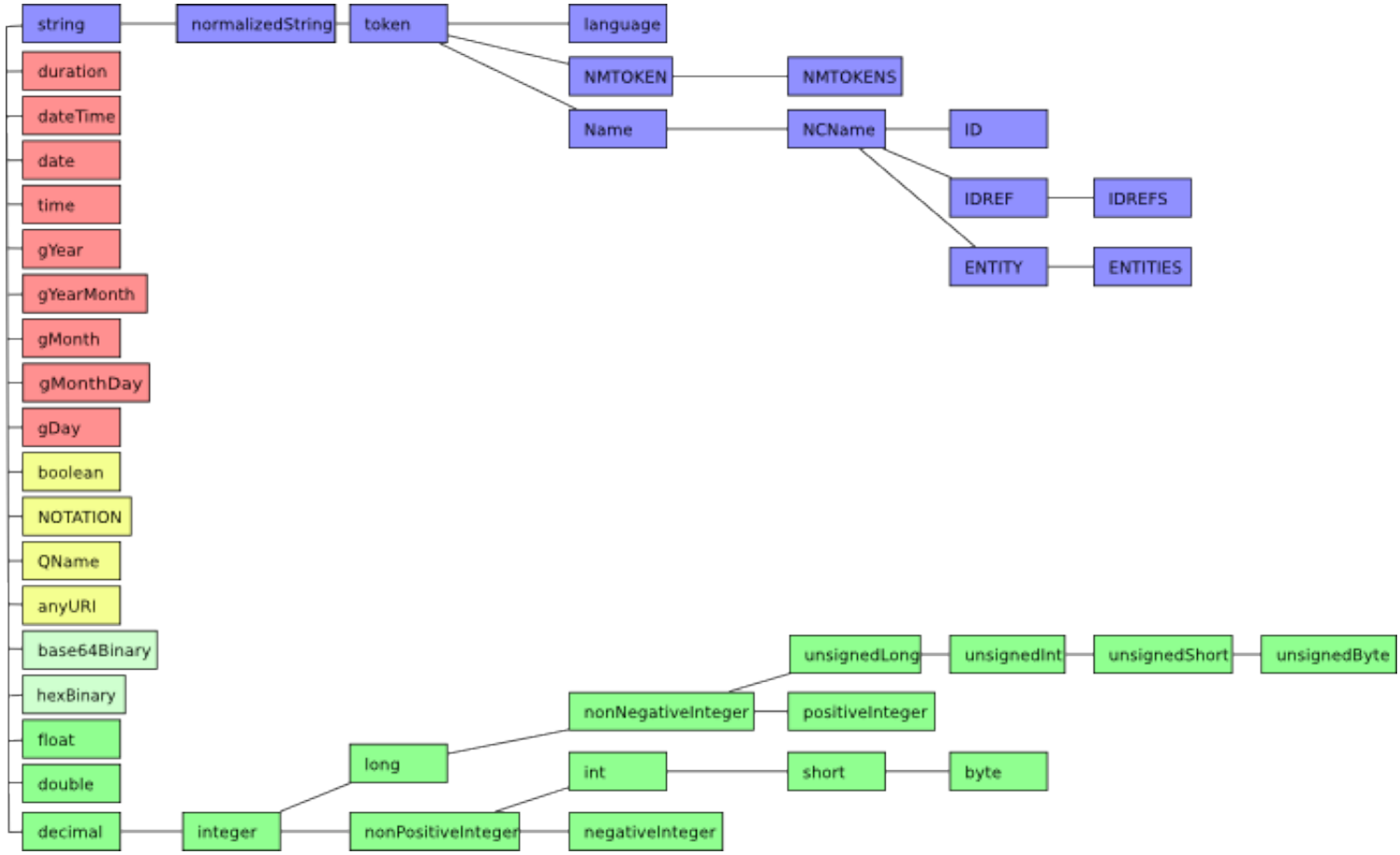
II-D-2 - Types simples

II-D-2-a - Généralités

Les types de données simples ne peuvent comporter ni attributs, ni éléments enfants. Il en existe de nombreux, prédéfinis, mais il est également possible d'en "dériver" de nouveaux (nous reviendrons **plus loin** sur cette fonctionnalité). Enfin, il est possible de déclarer des "**listes**" de types.

II-D-2-b - Bibliothèque de types intégrés

Nombreux sont les types prédéfinis dans la bibliothèque de types intégrés de **XML Schema**. la figure suivante en donne la hiérarchie, et leur **liste détaillée** figure sur le site du W3C.



Les types de données les plus simples (les chaînes de caractères) que permettaient les DTDs sont conservés, mais d'autres ont fait leur apparition. On pourra envisager, par exemple, dans un schéma décrivant un bon de commande, la déclaration d'un attribut **quantite** :

```
<xsd:attribute name="quantite" type="xsd:positiveInteger" use="default" value="1" />
```

... qui force la valeur de l'attribut à un être un entier positif. Un bon de commande **XML** suivant ce schéma, ayant une commande spécifiant **quantite="-3"** sera alors automatiquement refusé par le système. Un document respectant un tel schéma peut dès lors être enregistré directement dans une base de données, sans contrôle supplémentaire sur ce point.

II-D-2-c - Listes

Les types listes sont des suites de types simples (ou *atomiques*). **XML Schema** possède trois types de listes intégrés : **NMTOKENS, ENTITIES** et **IDREFS**. Il est également possible de créer une liste personnalisée, par « dérivation » de types existants. Par exemple,

```
<xsd:simpleType name="numéroDeTéléphone">
   <xsd:list itemType="xsd:unsignedByte" />
 </xsd:simpleType>
```

Un élément conforme à cette déclaration serait **<téléphone>01 44 27 60 11</téléphone>**.

Il est également possible d'indiquer des contraintes plus fortes sur les types simples ; ces contraintes s'appellent des « facettes ». Elles permettent par exemple de limiter la longueur de notre numéro de téléphone à 10 nombres. Nous reviendrons sur les **facettes**.

II-D-2-d - Unions

Les listes et les types simples intégrés ne permettent pas de choisir le type de contenu d'un élément. On peut désirer, par exemple, qu'un type autorise soit un nombre, soit une chaîne de caractères particuliers. Il est possible de le faire à l'aide d'une déclaration d'union. Par exemple, sous réserve que le type simple **numéroDeTéléphone** ait été préalablement défini (voir précédemment), on peut déclarer...

```
<xsd:simpleType name="numéroDeTéléphoneMnémoTechnique">
   <xsd:union memberTypes="xsd:string numéroDeTéléphone" />
 </xsd:simpleType>
```

Les éléments suivants sont alors des "instances" valides de cette déclaration :

```
<téléphone>18</téléphone>
 <téléphone>Pompiers</téléphone>
```

II-D-3 - Les types complexes

II-D-3-a - Introduction

Un élément de type simple ne peut contenir pas de sous-élément. Il est nécessaire pour cela de le déclarer de type complexe. On peut alors déclarer des **séquences** d'éléments, des types de **choix** ou des **contraintes d'occurences**

II-D-3-b - Séquences d'éléments

Nous savons déjà comment, dans une **DTD**, nous pouvons déclarer un élément comme pouvant contenir une suite de sous-éléments dans un ordre déterminé. Il est bien sûr possible de faire de même avec un schéma.

On utilise pour ce faire l'élément **xsd:sequence**, qui reproduit l'opérateur , du langage **DTD**. Ainsi...

```
<xsd:complexType>
   <xsd:sequence>
     <xsd:element name="nom" type="xsd:string" />
     <xsd:element name="prénom" type="xsd:string" />
     <xsd:element name="dateDeNaissance" type="xsd:date" />
     <xsd:element name="adresse" type="xsd:string" />
     <xsd:element name="adresseElectronique" type="xsd:string" />
     <xsd:element name="téléphone" type="numéroDeTéléphone" />
   </xsd:sequence>
 </xsd:complexType>
```

... est équivalent à une déclaration d'élément, dans une **DTD**, où apparaîtrait (**nom, prénom, dateDeNaissance, adresse, adresseElectronique, téléphone**).

II-D-3-c - Choix d'élément

On peut vouloir modifier la déclaration de type précédente en stipulant qu'on doive indiquer soit l'adresse d'une personne, soit son adresse électronique. Pour cela, il suffit d'utiliser un élément **xsd:choice** :

```
<xsd:complexType name="typePersonne">
   <sequence>
     <xsd:element name="nom" type="xsd:string" />
     <xsd:element name="prénom" type="xsd:string" />
     <xsd:element name="dateDeNaissance" type="xsd:date" />
     <xsd:choice>
       <xsd:element name="adresse" type="xsd:string" />
       <xsd:element name="adresseElectronique" type="xsd:string" />
     </xsd:choice>
   </sequence>
   <xsd:element name="téléphone" type="numéroDeTéléphone" />
 </xsd:complexType>
```

Ce connecteur a donc les mêmes effets que l'opérateur | dans une **DTD**.

II-D-3-d - L'élément all

Cet élément est une nouveauté par rapport aux **DTD**. Il indique que les éléments enfants doivent apparaître une fois (ou pas du tout), mais dans n'importe quel ordre. Cet élément **xsd:all** doit être un enfant direct de l'élément **xsd:complexType**. Par exemple...

```
<xsd:complexType>
   <xsd:all>
     <xsd:element name="nom" type="xsd:string" />
     <xsd:element name="prénom" type="xsd:string" />
     <xsd:element name="dateDeNaissance" type="xsd:date" />
     <xsd:element name="adresse" type="xsd:string" />
     <xsd:element name="adresseElectronique" type="xsd:string" />
     <xsd:element name="téléphone" type="numéroDeTéléphone" />
   </xsd:all>
 </xsd:complexType>
```

... indique que chacun de ces éléments peut apparaître une fois ou pas du tout (équivalent de l'opérateur **?** dans une **DTD**), et que l'ordre des éléments n'a pas d'importance (cela n'a pas d'équivalent dans une **DTD**).

II-D-3-e - Indicateurs d'occurences

Dans une **DTD**, un indicateur d'occurence ne peut prendre que les valeurs 0, 1 ou l'infini. On peut forcer un élément **sselt** à être présent 378 fois, mais il faut pour cela écrire (**sselt, sselt..., sselt, sselt**) 378 fois. **XML Schema** permet de déclarer directement une telle occurence, car tout nombre entier non négatif peut être utilisé. Pour déclarer qu'un élément peut être présent un nombre illimité de fois, on utilise la valeur **unbounded**. Les attributs utiles sont **minOccurs** et **maxOccurs**, qui indiquent respectivement les nombres minimal et maximal de fois où un élément peut apparaître. Le tableau suivant récapitule les possibilités :

Dans une DTD	Valeur de minOccurs	Valeur de maxOccurs
*	0	unbounded
+	1 (pas nécessaire, valeur par défaut)	unbounded
?	0	1 (pas nécessaire, valeur par défaut)
rien	1 (pas nécessaire, valeur par défaut)	1 (pas nécessaire, valeur par défaut)
impossible	nombre entier n quelconque	nombre entier m quelconque supérieur ou égal à n

Table 2. Liste des indicateurs d'occurence.

II-D-3-f - Création de type complexe à partir de types simples

Il est possible également de créer un type complexe à partir d'un type simple.

On peut avoir besoin de définir un élément contenant une valeur simple, et possédant un attribut, comme **<poids unite="kg">67</poids>**, par exemple. Un tel élément ne peut pas être déclaré de type simple, car il contient un attribut. Il faut *dériver* un type complexe à partir du type simple **positiveInteger** :

```
<xsd:complexType name="typePoids">
   <xsd:simpleContent>
     <xsd:extension base="xsd:positiveInteger">
       <xsd:attribute name="unite" type="xsd:string" />
     </xsd:extension>
   </xsd:simpleContent>
 </xsd:complexType>
```

L'élément **xsd:simpleContent** indique que le nouvel élément ne contient pas de sous-élément.

II-D-3-g - Exercices

Exercice 1. Déclaration d'éléments

Nous allons reprendre le texte d'un **exercice sur les DTD**, mais cette fois-ci avec un schéma. On ne déclarera, pour le moment, que des types de chaînes de caractères.

Rédiger un Schema XML pour une bibliographie. Cette bibliographie :

- contient des livres et des articles ;
- les informations nécessaires pour un livre (élément livre) sont :
  - son titre général (élément titre) ;
  - les noms des auteurs (éléments auteur) ;

    - ses tomes (élément tomes) et pour chaque tome (éléments tome), leur nombre de pages (élément pages) ;
    - des informations générales sur son édition (élément infosEdition)comme par exemple le nom de l'éditeur (élément editeur), le lieu d'édition (élément lieuEdition), le lieu d'impression (élément lieuImpression), son numéro ISBN (élément ISBN) ;
- les informations nécessaires pour un article (élément article) sont :
    - son titre (élément titre) ;
    - les noms des auteurs (éléments auteur) ;
    - ses références de publication (élément infosPublication) : nom du journal (élément nomJournal), numéro des pages (élément pages), année de publication (élément anneePublication) et numéro du journal (élément numéroJournal)
- on réservera aussi un champ optionnel, pour chaque livre et chaque article, pour un avis (élément avis) personnel.

Tester ce Schema XML avec un fichier XML que l'on écrira ex-nihilo et validera.

Correction

Exercice 2. Déclaration d'attributs

Modifier le Schéma précédent... On ne déclarera, pour le moment, que des types de chaînes de caractères.

- ... en ajoutant un attribut optionnel soustitre à l'élément titre ;
- ... en faisant de l'élément tome un élément vide et en lui ajoutant un attribut requis nbPages et un attribut optionnel sousTitre ;
- ... en faisant de l'élément nomJournal un attribut de l'élément infosPublication et en lui donnant comme valeur par défaut Feuille de Chou ;

Utiliser ce Schéma pour créer un fichier XML valide.

Correction

Exercice 3. Déclaration de types

Nous allons modifier le Schema précédent pour tirer parti des fonctionnalités additionnelles de ce format, relatives ayx types de données.

A. Utilisation des types intégrés

1 L'élément pages doit être un entier positif, ainsi que l'élément numéroJournal.
2 L'élément anneePublication doit être déclaré comme... une année.

B. Déclarations de types

Nous allons maintenant définir nos propres types de données, en les dérivant par rapport aux types intégrés.

1 Définir un type simple nommé typeISBN, basé sur une restriction du ype xsd:string, se limitant à une chaîne de caractères composée de 10 chiffres. L'utiliser dans la déclaration de l'élément ISBN.
2 En s'inspirant de l'exemple donné en cours du formatage d'une adresse électronique, déclarer un type typePages, basé sur une restriction du type xsd:string, se limitant à un nombre, puis la chaîne de caractères " à ", puis un autre nombre.

On vérifiera le fonctionnement à l'aide d'un fichier XML.

Correction

II-E - Espaces de nom

II-E-1 - Introduction

La notion d'espace de nom est complexe ; elle permet à un document **XML** quelconque d'utiliser les balises définies dans un schéma donné... quelconque. Nous avons déjà utilisé cette notion :

- dans les feuilles de style **XSL**, où nous utilisions des éléments préfixés par **xsl**, après avoir écrit **<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">**
- dans un schéma, où nous utilisions des éléments préfixés par **xsd**, après avoir écrit **<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">**.

Cela signifie que l'espace de nom auquel ces balises font référence, là où elles sont définies , est un schéma particulier que l'on peut consulter.

II-E-2 - Comment lier un fichier XML à un schéma ?

Nous n'allons pas ici entrer dans les détails de la notion d'espace de nom, mais simplement apprendre à valider un document **XML** d'après un **Schema XML**. On utilise pour ce faire le préfixe **xmlns**.

- Nous avons déjà vu le cas, équivalent à une **DTD** de type **PUBLIC**, où le schéma est... public. Un schéma est en effet un document **XML**, et on trouve dans son élément racine l'attribut **xmlns:xsd="http://www.w3.org/2001/XMLSchema**". Cela signifie que dans le document, tous les éléments commençant par **xsd** sont référencés à cette URL. Donc si on a déposé un schéma à l'adresse http://www.monsite.org/collection_schemas/biblio, on peut l'appeler par **<xsd:biblio xmlns="http://www.monsite.org/collection_schemas/biblio">**.
- Dans le cas d'une référence locale, correspondant à une **DTD** de type **SYSTEM**, on fait référence au schéma dans le document **XML** en utilisant l'attribut **noNamespaceSchemaLocation**, par **<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="lien_relatif_vers_le_schema">**. Par exemple, **<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="biblio10.xsd">**.

II-F - Les dérivations

II-F-1 - Introduction

Les types simples et complexes permettent déjà de faire plus de choses que les déclarations dans le langage **DTD**. Il est possible de raffiner leur déclaration de telle manière qu'ils soient une "restriction" ou une extension d'un type déjà existant, en vue de préciser un peu plus leur forme. Nous allons nous limiter dans ce cours d'initiation à la restriction des types simples.

II-F-2 - Restriction de type

II-F-2-a - Généralités

On peut appliquer une dérivation aussi bien à un type simple, qu'à un type complexe. La dérivation par restriction permet de créer de nouveaux types simples à partir des types simples prédéfinis par le format **XML Schema**. On utilise pour ce faire des *facettes*, qui sont des contraintes supplémentaires appliquées à un type simple particulier.

Une facette permet de placer une contrainte sur l'ensemble des valeurs que peut prendre un type de base. Par exemple, on peut souhaiter créer un type simple, appelé **MonEntier**, limité aux valeurs comprises entre 0 et 99 inclus. On dérive ce type à partir du type simple prédéfini **nonNegativeInteger**, en utilisant la facette **maxExclusive**.

```
<xsd:simpleType name="monEntier">
   <xsd:restriction base="nonNegativeInteger">
     <xsd:maxExclusive value="100" />
   </xsd:restriction>
 </xsd:simpleType>
```

Il existe un nombre important de facettes qui permettent de :

- fixer, restreindre ou augmenter la longueur minimale ou maximale d'un type simple
- énumérer toutes les valeurs possibles d'un type
- prendre en compte des expressions régulières
- fixer la valeur minimale ou maximale d'un type (voir l'exemple ci-dessus)
- fixer la précision du type...

II-F-2-b - Exemples

On peut utiliser cette fonctionnalité pour reproduire ce qui, dans les **DTD**, permettait de limiter les valeurs de certains attributs. Ainsi...

```
<xsd:attribute name="jour" type="typeJourSemaine" use="required" />
 <xsd:simpleType name="typeJourSemaine">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="lundi" />
     <xsd:enumeration value="mardi" />
     <xsd:enumeration value="mercredi" />
     <xsd:enumeration value="jeudi" />
     <xsd:enumeration value="vendredi" />
     <xsd:enumeration value="samedi" />
     <xsd:enumeration value="dimanche" />
   </xsd:restriction>
 </xsd:simpleType>
```

Pour limiter la longueur d'une chaîne :

```
<xsd:simpleType name="typeMotLangueFrancaise">
   <xsd:restriction base="xsd:string">
     <xsd:length value="21" />
   </xsd:restriction>
 </xsd:simpleType>
```

Plus complexe, on peut utiliser des *expressions régulières*, qui permettent de spécifier quels sont les caractères autorisés, à l'aide de l'élément **xsd:pattern**. Par exemple...

```
<xsd:simpleType name="typeAdresseElectronique">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="(.)+@(.)+" />
   </xsd:restriction>
 </xsd:simpleType>
```

Dans cet exemple, **(.)+** signifie que l'on peut mettre n'importe quel caractère au moins une fois, et qu'entre les deux chaînes doit impérativement apparaître le caractère **@**.

Un numéro ISBN est un référent international pour une publication. Il s'agit d'un numéro à 10 ou 13 chiffres. On peut le déclarer ainsi :

```
<xsd:simpleType name="typeISBN">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="[0-9]{10}" />
   </xsd:restriction>
 </xsd:simpleType>
```

Bien sûr, toutes les facettes ne sont pas acceptées par tous les types. Il serait fastidieux ici d'en donner la liste ; elle est accessible sur le site du W3C à l'adresse **http://www.w3.org/TR/xmlschema-0/#SimpleTypeFacets**.

II-G - Diverses autres fonctionnalités

II-G-1 - Inclusion de schémas

Un **Schema XML** a rapidement tendance à devenir verbeux. Autrement dit, il devient rapidement suffisamment long pour que sa complexité apparente rebute l'□il humain. Il est alors plus raisonnable de le scinder en plusieurs morceaux, chacun chargé de définir précisément tel ou tel sous-domaine du schéma. Il est en effet possible d'*inclure* plusieurs schémas dans un seul, à l'aide de l'élément **xsd:include**. Cela offre de plus l'énorme avantage de modulariser un schéma, et donc de pouvoir sans difficulté importer certaines parties à partir de schémas déjà existants. Nul besoin de réinventer la roue, ou de procéder à de massifs « copier/coller », en ayant besoin de tout reprendre à zéro à chaque fois que l'on fait une mise à jour.

Supposons par exemple que nous ayons défini le schéma d'une bibliographie dans le cadre plus général de l'écriture d'un mémoire. Ce **Schema XML** est stocké à l'URL http://www.monsite.org/schemas/biblio.xsd. Nous avons besoin d'une bibliographie pour une autre aplication -par exemple, dresser la liste d'un ensemble de ressources pour un site Web de e-formation. Nous pouvons inclure le schéma précédemment écrit à l'aide de la commande...

```
<xsd:include schemaLocation="http://www.monsite.org/schemas/biblio.xsd" />
```

Il s'agit d'une sorte de « copier-coller » du contenu de la bibliographie dans le schéma en cours d'écriture. La seule condition est que le **targetNameSpace** soit le même dans le **Schema XML** inclus et dans le **Schema XML** importateur.

II-G-2 - Documentation

XML Schema permet, outre l'utilisation des commentaires comme tout format **XML**, l'adjonction de documentation aux éléments.

La documentation à l'intention des lecteurs humains peut être définie dans des éléments **xsd:documentation**, tandis que les informations à l'intention de programmes doivent être incluses dans des éléments **xsd:appinfo**. Ces deux éléments doivent être placés dans un élément **xsd:annotation**. Ils disposent d'attributs optionnels : **xml:lang** et **source**, qui est une référence à une URI pouvant être utilisée pour identifier l'objectif du commentaire ou de l'information.

Les éléments **xsd:annotation** peuvent être ajoutés au début de la plupart des constructions. Voir par exemple le schéma **biblio10.xsd** déjà donné.

II-G-3 - Attribut null

Il est toujours préférable de pouvoir indiquer explicitement qu'un élément est non renseigné plutôt que d'omettre cet élément. La valeur null des bases de données relationnelles est utilisée dans ce but. **XML Schema** intègre un mécanisme similaire permettant d'indiquer qu'un élément peut être non renseigné.

Nous déclarons maintenant l'élément **courriel** comme pouvant être **null** à l'aide de l'attribut nullable du vocabulaire de **XML Schema** :

```
<personne>
   <nom>Jean Dupont</nom>
   <courriel xsi:null></courriel>
 </personne>
```

Cet attribut doit toujours être préfixé par **xsi**. Quant à l'élément portant cet attribut, il peut contenir d'autres attributs, mais pas de sous-élément.