Tutoriel XML Schema

Introduction aux schémas W3C XML Schema.

Eric van der Vlist, Dyomedea (vdv@dyomedea.com).
18 décembre 2000

Cet article a été publié sur XML.com sous le titre "Using W3C XML Schema".

Plan

Retour au plan

2. Notre premier schéma.

Commençons par examiner ce document **XML** assez simple:

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

<book isbn="0836217462">
 <title>
  Being a Dog Is a Full-Time Job
 </title>
 <author>Charles M. Schulz</author>
 <character>
  <name>Snoopy</name>
  <friend-of>Peppermint Patty</friend-of>
  <since>1950-10-04</since>
  <qualification>
    extroverted beagle
  </qualification>
 </character>
 <character>
  <name>Peppermint Patty</name>
  <since>1966-08-22</since>
  <qualification>bold, brash and tomboyish</qualification>
 </character>
</book>
```

La première approche, pour écrire un schéma, est de suivre tout simplement sa structure et de définir chaque élément au moment où nous le rencontrons.

En rencontrant le prologue du document, nous ouvrons un élément "*schema*" :

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
     xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
```

L'élément "*schema*" ouvre un schéma **W3C XML Schema**. Il peut également contenir la définition de l'espace de nom cible et d'autres options dont nous verrons certaines dans les prochaines pages.

Ensuite, correspondant à la balise ouvrante de notre élément "book", nous définissons un élément qui aura ce nom. Cet élément ayant des attributs et des sous éléments, nous devrons le définir comme " "*complexType*" ", l'autre type de données ("*simpleType*") étant réservé aux éléments et attributs sans sous-éléments ou attributs et ne contenant donc que des valeurs.

La liste des sous éléments sera quand à elle décrite à l'intérieur d'un élément "*sequence*" :

```
<xsd:element name="book">
 <xsd:complexType>
  <xsd:sequence>
```

L'élément "*sequence*" définit une liste ordonnée de sous éléments. Nous verrons les deux autres possibilités ("*choice*" et "*all*" dans les pages suivantes.

Ensuite nous décrivons les éléments "title" et "author" comme des types simples puisqu'ils n'ont ni attributs ni sous-éléments. Le type que nous utilisons ("xsd:string") est préfixé par l'espace de noms que nous avons choisi pour le schéma indiquant que nous utilisons un type prédéfini de **W3C XML Schema**:

```
   <xsd:element name="title" type="xsd:string"/>
   <xsd:element name="author" type="xsd:string"/>
```

A présent, nous rencontrons l'élément "character" qui doit être considéré comme un type complexe et que nous pouvons également définir immédiatement. Remarquer comment sa cardinalité est déclarée:

```
   <xsd:element name="character"
        minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
```

Contrairement) d'autres langages de schémas, **W3C XML Schema** permet de définir la cardinalité des éléments (c'est à dire le nombre de leurs occurrences) avec beaucoup de précision et nous pouvons définir à la fois "minOccurs" (le nombre minimum d'occurrences) et "maxOccurs" (le nombre maximum d'occurrences). Ici, maxOccurs est déclaré "unbounded" c'est à dire que l'élément peut être répété autant de fois que l'auteur du document le souhaite. Ces deux attributs ont une valeur égale à 1 par défaut.

Nous spécifions ensuite la liste de ses éléments comme nous l'avons déjà vu:

```
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="friend-of" type="xsd:string"
            minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="since" type="xsd:date"/>
      <xsd:element name="qualification" type="xsd:string"/>
```

Et pouvons terminer la description de l'élément "character" en fermant les éléments "*complexType*" et "*element*" .

```
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
```

La séquence des éléments pour l'élément "books" est également terminée:

```
  </xsd:sequence>
```

Et c'est le moment de définir les attributs de cet élément, les attributs devant toujours être déclarés après les éléments. Il ne semble pas avoir de raison particulière pour cela, mais le Groupe de Travail **W3C XML Schema** a considéré qu'il était plus simple de fixer l'ordre relatif des déclarations des attributs et des éléments et qu'il était plus naturel de déclarer les attributs après les éléments.

```
  <xsd:attribute name="isbn" type="xsd:string"/>
```

Notre schéma est terminé et nous pouvons refermer tous les éléments encore ouverts:

```
 </xsd:complexType>
</xsd:element>

</xsd:schema>
```

C'est tout...

Ce premier style de schémas, parfois appelé "poupées russes" est très proche de la structure d'un document significatif du vocabulaire, chaque élément ou attribut du document étant décrit quand on le rencontre.

Une de ses caractéristiques est de définir chaque élément ou attribut dans son contexte et de permettre de définir des occurrences multiples d'un même nom d'élément ayant des structures différentes en fonction du contexte dans le document.

Pour cela, **W3C XML Schema** est un langage contextuel hiérarchique avec champ de définition, scope chaque définition n'étant visible que dans l'élément dans laquelle elle est effectuée ainsi que dans tous ses descendants.

Exemple complet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
     xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:element name="book">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="title" type="xsd:string"/>
   <xsd:element name="author" type="xsd:string"/>
   <xsd:element name="character"
        minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="friend-of" type="xsd:string"
            minOccurs="0" maxOccurs="unbounded"/>
```

```
      <xsd:element name="since" type="xsd:date"/>
      <xsd:element name="qualification" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="isbn" type="xsd:string"/>
 </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Retour au plan

3. Découpons notre premier schéma.

Bien que le style de schéma que nous venons de voir soit très simple à implémenter, il peut rapidement devenir difficile à lire et à maintenir si la structure du document se complexifie. Il présente également l'inconvénient d'être très différent des la structure des **DTD**s, ce qui peut être gênant lors de la conversion manuelle ou automatique de **DTD**s en **W3C XML Schema** ou encore pour développer des méthodologies génériques de génération de schémas.

Le deuxième style de schéma que nous allons aborder utilise les éléments que nous avons déjà vu de manière différente pour construire, à la manière des **DTD**s, un simple catalogue éléments présents dans le document en précisant, pour chacun, la liste de ses attributs et éléments.

Pour ceci, nous allons utiliser des références à des éléments et attributs définis dans le champ de définition du référenceur et créer une structure extrêmement plate:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<!-- definition of simple type elements -->

<xsd:element name="title" type="xsd:string"/>
<xsd:element name="author" type="xsd:string"/>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="friend-of" type="xsd:string"/>
<xsd:element name="since" type="xsd:date"/>
<xsd:element name="qualification" type="xsd:string"/>

<!-- definition of attributes -->

<xsd:attribute name="isbn" type="xsd:string"/>

<!-- definition of complex type elements -->

<xsd:element name="character">
 <xsd:complexType>
  <xsd:sequence>
   <!-- the simple type elements are referenced using
        the "ref" attribute                          -->
   <xsd:element ref="name"/>
   <!-- the definition of the cardinality is done
        when the elements are referenced         -->
   <xsd:element ref="friend-of"
        minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="since"/>
   <xsd:element ref="qualification"/>
  </xsd:sequence>
 </xsd:complexType>
```

```
</xsd:element>

<xsd:element name="book">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="title"/>
   <xsd:element ref="author"/>
   <xsd:element ref="character"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute ref="isbn"/>
 </xsd:complexType>
</xsd:element>

</xsd:schema>
```

L'utilisation d'une référence à un élément ou un attribut est comparable au clonage d'un objet: l'élément ou attribut est défini (tout comme l'objet serait créé) et dupliqué ailleurs dans le schéma par l'utilisation de l'attribut "ref" tout comme l'objet serait dupliqué par clonage. Les deux éléments ou attributs deviennent deux instances d'une même classe que nous avons créé implicitement et nous allons maintenant voir comment nous pouvons définir cette classe de manière explicite.

Retour au plan

4. Définition de types nommés.

Nous avons vu pour le moment comment nous pouvons définir des éléments ou attributs lorsque nous en avons besoin (poupée russe) ou les créer puis les utiliser par référence. **W3C XML Schema** nous donne une troisième manière de procéder qui est en quelque sorte intermédiaire et consiste à définir des types de données pouvant être simples (et utilisable uniquement par des attributs ou des éléments au contenu simple -PCDATA-) ou complexes (utilisables uniquement pour définir des éléments) et d'utiliser ensuite ces types pour définir nos éléments et attributs.

Ceci est réalisé tout simplement en donnant un nom aux éléments "*simpleType*" et "*complexType*" et en les définissant en dehors de toute définition d'élément ou d'attribut.

Nous allons également voir comment dériver un type à partir d'un autre type en donnant une restriction sur les valeurs acceptées par ce type de données.

Ainsi, pour définir un type de données que nous appellerons "nameType" et qui sera une chaîne de caractères acceptant un maximum de 32 caractères, nous écrirons:

```
<xsd:simpleType name="nameType">
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="32"/>
  </xsd:restriction>
 </xsd:simpleType>
```

L'élément "*simpleType*" définit le nom du type de données. L'élément "*restriction*" et son attribut base indique que ce type de données est dérivé par restriction -c'est à dire en donnant une contrainte sur les données acceptées- du type de données "string" appartenant à l'espace de noms **W3C XML Schema**. L'élément "*maxLength*" est appelé une "facette" et définit cette restriction comme étant le fait que la taille maximale est de 32, l'unité étant fonction du type de données est ici le caractère.

Une deuxième facette très puissante est la facette "*pattern*" qui permet de donner une expression régulière qui devra être vérifiée pour que la valeur soit acceptée. Si nous écrivons un noméro ISBN sans signe "-", nous pouvons définir un type ISBN comme étant une chaîne de dix caractères numériques de la manière suivante:

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

Nous verrons d'autres facettes et les deux autres méthodes de dérivation ("*list*" et "*union*") dans la suite de ce tutoriel.

Les types complexes sont également définis en donnant un nom à un "*complexType*" .

Schéma complet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<!-- definition of simple types -->

<xsd:simpleType name="nameType">
 <xsd:restriction base="xsd:string">
  <xsd:maxLength value="32"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="sinceType">
 <xsd:restriction base="xsd:date"/>
</xsd:simpleType>

<xsd:simpleType name="descType">
 <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

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

<!-- definition of complex types -->

 <xsd:complexType name="characterType">
  <xsd:sequence>
   <xsd:element name="name" type="nameType"/>
   <xsd:element name="friend-of" type="nameType"
        minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element name="since" type="sinceType"/>
   <xsd:element name="qualification" type="descType"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="bookType">
  <xsd:sequence>
   <xsd:element name="title" type="nameType"/>
   <xsd:element name="author" type="nameType"/>
   <!-- the definition of the "character" element is
        using the "characterType" complex type    -->
   <xsd:element name="character" type="characterType"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
```

```
  <xsd:attribute name="isbn" type="isbnType" use="required"/>
 </xsd:complexType>

<!-- Reference to "bookType" to define the
     "book" element -->
<xsd:element name="book" type="bookType"/>

</xsd:schema>
```

La définition et l'utilisation de types de données est comparable à la création d'une classe et à son utilisation pour créer un objet. Un type de données est une notion abstraite qui n'a pas d'instanciation et tant que tel et ne peut être utilisé que pour instancier un élément ou un attribut.

Retour au plan

5. Toujours plus loin: groupes, compositeurs et dérivation.

Des groupes d'éléments et d'attribute peuvent également être définis:

```
<!-- définition d'un groupe d'éléments -->

 <xsd:group name="mainBookElements">
  <xsd:sequence>
   <xsd:element name="title" type="nameType"/>
   <xsd:element name="author" type="nameType"/>
  </xsd:sequence>
 </xsd:group>

<!-- definition d'un groupe d'attributs -->

 <xsd:attributeGroup name="bookAttributes">
  <xsd:attribute name="isbn" type="isbnType" use="required"/>
  <xsd:attribute name="available" type="xsd:string"/>
 </xsd:attributeGroup>
```

et utilisés pour définir des types complexes:

```
 <xsd:complexType name="bookType">
  <xsd:sequence>
   <xsd:group ref="mainBookElements"/>
   <xsd:element name="character" type="characterType"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="bookAttributes"/>
 </xsd:complexType>
```

Ces groupes ne sont pas des types de données à proprement parler, mais des conteneurs permettant de manipuler des ensembles d'éléments ou attributs de manière groupée pour la définition de types complexes.

Jusqu'à présent, nous avons vu le compositeur "*sequence*" qui définit un groupe ordonné d'éléments. **W3C XML Schema** supporte deux autres types de compositeurs qui peuvent -avec certaines restrictions- être utilisés comme particules et être combinés.

Les compositeurs eux-mêmes peuvent avoir des attributs minOccurs et maxOccurs pour définir leur cardinalité.

Le compositeur "*choice*" définit un groupe de particules dont une seule devra être

présente. Ainsi, la construction suivante acceptera soit un élément "name" unique, soit un groupe de trois éléments (firstName, middleName et lastName) dont un (middleName) sera optionnel:

```
<xsd:group name="nameTypes">
 <xsd:choice>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:sequence>
   <xsd:element name="firstName" type="xsd:string"/>
   <xsd:element name="middleName" type="xsd:string" minOccurs="0"/>
   <xsd:element name="lastName" type="xsd:string"/>
  </xsd:sequence>
 </xsd:choice>
</xsd:group>
```

Le compositeur "*all*" définit un ensemble non ordonné d'éléments. La définition suivante décrit donc un type complexe dont les éléments peuvent apparaître dans un ordre quelconque:

```
<xsd:complexType name="bookType">
 <xsd:all>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="author" type="xsd:string"/>
  <xsd:element name="character" type="characterType"
       minOccurs="0" maxOccurs="unbounded"/>
 </xsd:all>
 <xsd:attribute name="isbn" type="isbnType" use="required"/>
</xsd:complexType>
```

Pour éviter des combinaisons pouvant devenir complexes et même ambiguës, **W3C XML Schema** impose une série de restrictions draconiennes sur l'utilisation des compositeurs "*all*" : ils ne peuvent apparaître que comme élément unique tout en haut de la définition d'un modèle de contenu et les particules qui les composent ne peuvent être que des éléments et leur cardinalité ne peut être supérieure à un.

Les types de données simples sont définis dérivation d'autres types pouvant être prédéfinis par **W3C XML Schema** (identifié par son espace de noms) ou définis ailleurs dans un schéma.

Pour l'instant, nous avons vu des examples de types de données simples dérivés par restriction (en utilisant l'élément "*restriction*"). Les différents types de restrictions qui peuvent être appliquées sur un type de données sont appelés "facettes". **W3C XML Schema** dispose de nombreuses facettes, en plus des deux facettes que nous avons vu (pattern et maxLength), permettant de définir des contraintes sur les longueurs, les valeurs extrêmes, la précision et l'échelle, la période et la durée, la liste des valeurs possibles, ...

Deux autres méthodes de dérivation sont également disponibles qui permettent de définir des listes de valeurs séparés par des espaces et des unions de types de données.

La définition suivante utilise une dérivation par "*union*" afin de permettre au type de données ISBN d'accepter également les valeurs TDB et NA:

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

```
  <xsd:simpleType>
   <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="TBD"/>
    <xsd:enumeration value="NA"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:union>
</xsd:simpleType>
```

L'élément "*union*" est ici appliquée aux deux types de données simples qui sont définis à l'intérieur permettant d'accepter les chaînes de 10 valeurs numériques (premier type) ainsi que les valeurs "TBD" et "NA" (deuxième type).

Ce type de données peut être à son tour utilisé pour constituer un nouveau type (isbnTypes) qui sera une liste séparée par des espaces de valeurs de type "isbnType" et une dérivation par restriction pourra ensuite être appliquée pour restreindre le nombre des valeurs dans la liste qui devra être compris entre 1 et 10:

```
<xsd:simpleType name="isbnTypes">
 <xsd:list itemType="isbnType"/>
</xsd:simpleType>

<xsd:simpleType name="isbnTypes10">
  <xsd:restriction base="isbnTypes">
   <xsd:minLength value="1"/>
   <xsd:maxLength value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Retour au plan

6. Types de contenus

Nous avons vu pour l'instant le type de contenu appliqué par défaut, permettant de définir des documents XML de type "données" dans lesquels les types complexes ont uniquement des noeuds attributs et éléments et les types simples des noeuds texte.

W3C XML Schema supporte également la définition de modèles de contenus vide, simple avec attributs et mixtes.

Les éléments à contenu vide (c'est à dire ne pouvant contenir que des attributs) sont définis simplement en omettant de définir des éléments dans un type complexe. La construction suivante définit ainsi un élément "book" à contenu vide acceptant un attribut "isbn":

```
<xsd:element name="book">
 <xsd:complexType>
  <xsd:attribute name="isbn" type="isbnType"/>
 </xsd:complexType>
</xsd:element>
```

Les éléments à contenu simple, c'est à dire des éléments ne contenant que du texte et des attributs sont dérivés des types de données simples en utilisant l'élément "*simpleContent*" . L'élément book pourrait donc être modifié de la manière suivante pour accepter également un contenu de type chaîne de caractères:

```
<xsd:element name="book">
 <xsd:complexType>
  <xsd:simpleContent>
```

```
   <xsd:extension base="xsd:string">
    <xsd:attribute name="isbn" type="isbnType"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
```

Noter la position de la définition de l'attribut qui indique que l'extension est effectuée par l'ajout de l'attribut.

Cette définition acceptera, par exemple, l'élément XML suivant:

```
<book isbn="0836217462">
        Funny book by Charles M. Schulz.
        Its title (Being a Dog Is a Full-Time Job)
        says it all !
</book>
```

W3C XML Schema supporte les modèles de contenu mixtes au moyen de l'attribut "mixed" qui doit être placé dans l'élément "*complexType*" et la définition suivante:

```
<xsd:element name="book">
 <xsd:complexType mixed="true">
  <xsd:all>
   <xsd:element name="title" type="xsd:string"/>
   <xsd:element name="author" type="xsd:string"/>
  </xsd:all>
  <xsd:attribute name="isbn" type="xsd:string"/>
 </xsd:complexType>
</xsd:element>
```

décrira un élément XML de la forme:

```
<book isbn="0836217462">
        Funny book by <author>Charles M. Schulz</author>.
        Its title (<title>Being a Dog Is a Full-Time
        Job</title>) says it all !
</book>
```

Le fait d'écrire mixed="true" indique à **W3C XML Schema** d'accepter des caractères n'importe où entre les éléments qui le compose et, contrairement à ce qui se passe avec les **DTD**s, n'affecte pas les contrôles que l'on effectue sur ces éléments. On peut donc définir les mêmes restrictions, notamment de cardinalité sur un élément de type mixte que sur un élément de type simple. Bien que cela représente une amélioration par rapport aux **DTD**s, on remarquera qu'il n'est pas possible de définir la position relative des chaînes de caractères par rapports aux éléments.

Retour au plan

7. Contraintes référentielles.

W3C XML Schema nous fournit plusieurs fonctionnalités s'appuyant sur XPath et permettant de décrire des contraintes d'unicité et des contrôles référentiels.

La première d'entre elles est une simple déclaration d'unicité et utilise l'élément "*unique*" . La déclaration suivante, faite à l'intérieur de l'élément "book" indique que le nom d'un personnage doit être unique:

```
<xsd:unique name="charName">
 <xsd:selector xpath="character"/>
 <xsd:field xpath="name"/>
</xsd:unique>
```

La position de la déclaration de cette contrainte (à l'intérieur de la définition de l'élément "book") donne le contexte à partir duquel le contrôle sera effectué: l'élément "book". Le choix de ce contexte détermine également la portée du test qui sera effectué: dans notre cas, le nom du personnage sera unique à l'intérieur d'un élément "book" mais pourra être répété dans un autre élément "book".

Les deux chemins **XPath** spécifiés dans l'élément "*unique*" seront évalués par rapport à ce noeud contexte. Le premier d'entre eux est défini dans l'élément "*selector*" et, comme son nom l'indique, il sélectionne l'élément qui doit être unique et doit donc pointer un noeud de type élément.

Le deuxième chemin **XPath** est évalué par rapport au noeud sélectionné et spécifie le noeud identifiant l'élément qui doit être unique. Il peut pointer un noeud de type élément ou attribut. C'est ce noeud dont l'unicité va être testée.

La seconde déclaration, "*key*" est similaire à "*unique*" et spécifie en outre que cette valeur unique pourra être utilisée comme une clé, ce qui lui donne deux contraintes supplémentaires: elle doit être non nulle et être référençable. Pour spécifier que le nom d'un personnage est une clé, il suffit de remplacer "*unique*" par "*key*" :

```
<xsd:key name="charName">
 <xsd:selector xpath="character"/>
 <xsd:field xpath="name"/>
</xsd:key>
```

La dernière déclaration, "*keyref*" définit une référence à une clé. Nous allons ainsi pouvoir tester que le nom inclus dans l'élément "friend-of" correspond à un personnage du même livre:

```
<character>
 <name>Snoopy</name>
 <friend-of>Peppermint Patty</friend-of>
 <since>1950-10-04</since>
 <qualification>
   extroverted beagle
 </qualification>
</character>
```

Pour ceci, nous allons ajouter un élément "*keyref*" sous la définition de l'élément "book", au même niveau que l'élément "*key*" :

```
<xsd:keyref name="charNameRef" refer="charName">
 <xsd:selector xpath="character"/>
 <xsd:field xpath="friend-of"/>
</xsd:keyref>
```

Ces fonctionnalités utilisant **XPath** apparaissent comme une pièce rajoutée sur l'architecture orientée objet de **W3C XML Schema** et sont pratiquement indépendantes du schéma, le seul point d'ancrage étant le noeud contexte dans lequel elles sont définies.

Retour au plan

8. Schémas réutilisables.

La première précaution à prendre pour écrire des schémas réutilisable est sans doute de les documenter. **W3C XML Schema** a prévu une alternative aux commentaires et Processing Instructions **XML** qui devrait être plus facile à gérer pour les outils les supportant.

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

Les éléments "*annotation*" peuvent être ajoutés "au début de la plupart des constructions" comme le montre cet exemple:

```
<xsd:element name="book">
 <xsd:annotation>
        <xsd:documentation xml:lang="en">
                Top level element.
        </xsd:documentation>
        <xsd:documentation xml:lang="fr">
                Element racine.
        </xsd:documentation>
        <xsd:appInfo source="http://example.com/foo/">
                <bind xmlns="http://example.com/bar/">
                        <class name="Book"/>
                </bind>
        </xsd:appInfo>
 </xsd:annotation>
```

Pour ceux d'entre nous qui souhaitons définir des schémas en utilisant plusieurs documents (que ce soit pour découper un gros schéma ou constituer des bibliothèques de schémas, **W3C XML Schema** a prévu deux mécanismes d'inclusion.

Le premier, "*include*" , est similaire à un copié/collé des définitions contenues dans le schéma qui est inclus. Les définitions ne peuvent donc pas être redéfinies dans le schéma effectuant l'inclusion. Un exemple d'inclusion pourrait être:

```
<xsd:include schemaLocation="character.xsd"/>
```

Le deuxième, "*redefine*" , est semblable à "*include*" , mais permet de redéfinir des déclarations effectuées dans le schéma inclus:

```
<xsd:redefine schemaLocation="character12.xsd">
<xsd:simpleType name="nameType">
 <xsd:restriction base="xsd:string">
  <xsd:maxLength value="40"/>
 </xsd:restriction>
</xsd:simpleType>
</xsd:redefine>
```

Noter que les déclarations qui sont redéfinies doivent l'être à l'intérieur de l'élément "*redefine*" .

Nous avons déjà vu plusieurs fonctionnalités qui peuvent être utilisées avec "*include*" et

"*redefine*" pour créer de véritables bibliothèques de schémas. Nous avons vu notamment comment nous pouvions référencer des éléments et attributs déjà définis, comment nous pouvons définir des types de données par dérivation et les utiliser, comment définir et utiliser des groupes d'éléments et d'attributs. Nous avons également vu la similitude entre ces notions et les techniques de programmations orientées objets et le parallèle entre éléments, attributs et objets et entre types de données et classes. **W3C XML Schema** nous réserve quelques autres fonctionnalités empruntées à la programmation orientée objet que nous allons aborder maintenant.

La première d'entre elle est appelée "groupes de substitutions". Contrairement à un type de données ou un groupe d'éléments, un groupe de substitutions de se déclare pas en utilisant un élément spécifique, mais en référençant (au moyen de l'attribut "substitutionGroup") un élément choisi comme "tête" du groupe de substitution.

L'élément tête n'a d'autre particularité que de devoir être global et d'être suffisamment générique pour que les types des éléments du groupe de substitution puissent tous être dérivés de son type. Si nous déclarons:

```
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="surname" type="xsd:string"
         substitutionGroup="name" />
```

nous formons un groupe de substitutions formé des éléments "surname" et "name" et ces éléments peuvent être utilisés indifféremment partout où "name" a été défini.

Nous aurions également pu vouloir définir que la tête du groupe de substitutions soit un élément générique "name-elt" qui ne puisse pas être utilisé directement dans un document mais uniquement sous une de ses formes dérivées. Pour cela, **W3C XML Schema** permet de déclarer des éléments abstraits, tout comme on peut déclarer une classe abstraite en programmation orientée objet. Ainsi par exemple:

```
<xsd:element name="name-elt" type="xsd:string" abstract="true"/>
<xsd:element name="name" type="xsd:string"
         substitutionGroup="name-elt" />
<xsd:element name="surname" type="xsd:string"
         substitutionGroup="name-elt" />
```

définit "name-elt" comme un élément abstrait qui doit être remplacé par "name" ou "surname" lorsqu'il est employé dans un document.

Nous pourrions également souhaiter, au contraire, interdire ou contrôler les dérivations appliquées sur un type de données et **W3C XML Schema** permet également de définir des éléments et des types complexes comme étant "finaux" en utilisant l'attribut "final". Cet attribut peut prendre les valeurs "restriction", "extension" ou "#all" et bloque respectivement les dérivations par restriction, extension ou toutes les dérivations. Le fragment suivant interdirait donc toute dérivation du type "characterType".

```
 <xsd:complexType name="characterType" final="#all">
```

L'attribut "final" ne s'applique qu'aux éléments et aux types complexes et un mécanisme encore plus fin permet de controler la dérivation de types simples facette par facette. L'attribut correspondant ("fixed") s'applique aux différentes facettes et lorsque se valeur est "true", la facette ne peut plus être restreinte. L'exemple suivant montre comment on peut interdire toute restriction ultérieure sur la taille maximum du type "nameType":

```
<xsd:simpleType name="nameType">
 <xsd:restriction base="xsd:string">
```

```
  <xsd:maxLength value="32" fixed="true"/>
 </xsd:restriction>
</xsd:simpleType>
```

Retour au plan

9. Espaces de noms.

Le support des espaces de noms est une des principales motivations qui ont conduit au développement de **W3C XML Schema** et ce support est particulièrement souple et simple. Il permet non seulement de gérer les espaces de noms en faisant abstraction des préfixes qui sont utilisées, mais également de définir des schémas ouverts qui acceptent l'ajout d'éléments et d'attributs provenant d'espaces de noms connus ou non.

Chaque schéma est lié à un espace de nom particulier (ou à l'absence d'espace de nom) et il faudra donc définir au moins un schéma par espace de nom que l'on veut spécifier. Les éléments et attributs sans espace de noms peuvent eux être définis dans n'importe quel tschéma.

Le déclaration de l'espace de nom défini par un schéma est effectuée en utilisant l'attribut "targetNamespace" de l'élément "*schema*" . Dans les exemples vus jusqu'à présent, cet attribut avait été omis, ce qui signifiait que nous travaillions sans espace de nom. Si nous modifions notre document "book" pour qu'il appartienne à un espace de noms:

```
<book isbn="0836217462" xmlns="http://example.org/ns/books/">
```

la manière la plus simple d'adapter notre schéma est de rajouter les attributs suivants à l'élément "*schema*" :

```
<xsd:schema
     xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
          xmlns="http://example.org/ns/books/"
          targetNamespace="http://example.org/ns/books/"
          elementFormDefault="qualified"
          attributeFormDefault="unqualified" >
```

Examinons plus en détail ces déclarations... La première (xmlns:xsd="http://www.w3.org/2000/10/XMLSchema") est une déclaration classique conforme à la spécification des espaces de noms. **W3C XML Schema** l'utilise également pour mémoriser que le préfixe "xsd" est attaché à l'espace de noms identifiant XML Schema et nous permettre de préfixer les types de données XML Schema par "xsd". Il est important de remarquer que nous aurions pu utiliser n'importe quel préfixe au lieu de "xsd" et que nous aurions même pu déclarer l'espace de noms XML Schema comme étant l'espace de noms par défaut. Dans ce cas, les types prédéfinis n'auraient pas du être préfixés.

Ensuite, comme notre document utilise l'espace de nom "http://example.org/ns/books/", nous le définissons comme étant l'espace par défaut, ce qui nous évitera d'avoir à préfixer nos propres types de données, éléments et attributs dans ce schéma. Ici encore, nous aurions pu choisir n'importe quel préfixe.

"targetNamespace" définit, indépendamment des déclarations de préfixes, l'espace de nom qui est décrit dans ce schéma. Si vous devez référencer des objets appartenant à cet espace de noms (ce qui est les cas sauf si vous utilisez une structure de type "poupée

russe" pure, il faut également définir un préfixe pour cet espace de noms.

Les deux derniers attributs ("elementFormDefault" et "attributeFormDefault") sont liés à la possibilité offerte par **W3C XML Schema** de définir dans un même schéma des attributs et éléments qualifiés (c'est à dire appartenant à un espace de noms) et non qualifiés (sans espace de noms).

La différentiation entre éléments et attributs qualifiés ou non qualifié est faite objet par objet au moyen de l'attribut "form" pouvant prendre la valeur "qualified" ou "unqualified" et les attributs "elementFormDefault" et "attributeFormDefault" que nous voyons ici définissent la valeur par défaut, dans ce schéma, des attributs "form" pour les éléments et les attributs.

L'exemple ci-dessus spécifie donc que, par défaut (c'est à dire lorsque l'attribut "form" est absent des déclarations), les éléments seront qualifiés (et appartiendont à l'espace de noms spécifié par l'attribut "targetNamespace") et les attributs seront non qualifiés (et n'appartiendont à aucun espace de noms).

Il faut toutefois noter une restriction importante: les éléments et attributs globaux (c'est à dire définis immédiatement sous l'élément "*schema*" ne peuvent pas être considérés comme étant non qualifiés.

Nous avons vu que **W3C XML Schema**, à la manière de XSLT/XPATH, utilise les préfixes des espaces de noms dans la valeur des attributs "ref" ou "type" pour identifier l'espace de nom dans lequel l'objet à été défini.

Nous avons utilisé cette fonctionnalité tout au long de nos exemples pour identifier les types de données prédéfinis en les préfixant par xsd (préfixe que nous avons associé à "http://www.w3.org/2000/10/XMLSchema") et pour identifier nos propres objets sans les préfixer et nous allons maintenant voir comment nous pouvons le faire pour identifier des objets appartenant à d'autres espaces de noms.

Ceci est réalisé au moyen d'un processus en trois étapes qui est nécessaire y compris pour utiliser des attributs préfixés par "xml" et appartenant à l'espace de nom **XML** 1.0 et nous allons voir comment procéder en utilisant l'espace de noms **XML** 1.0, cette question faisant partie des foires aux questions **W3C XML Schema**... La première étape est de définir un préfixe pour cet espace de noms de manière standard, par exemple:

```
<xsd:schema
     xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
          targetNamespace="http://example.org/ns/books/"
          xmlns:xml="http://www.w3.org/XML/1998/namespace"
          elementFormDefault="qualified" >
```

La deuxième étape consiste à indiquer à **W3C XML Schema** où il pourra trouver le schéma correspondant à cet espace de nom en utilisant l'élément "*import*" :

```
<import namespace="http://www.w3.org/XML/1998/namespace"
  schemaLocation="myxml.xsd"/>
```

W3C XML Schema sait maintenant où trouver le schéma correspondant à un objet préfixé par "xml" et nous pouvons donc utiliser ces objets:

```
   <xsd:element name="title">
    <xsd:complexType>
          <xsd:simpleContent>
           <xsd:extension base="xsd:string">
```

```
        <xsd:attribute ref="xml:lang"/>
          </xsd:extension>
         </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
```

On peut se demander pourquoi avoir fait référence à l'élément "xml:lang" plutôt que d'avoir créé un élément ayant un type "xml:lang".

Nous l'avons fait en raison d'une différence importante entre utilisation de types et de références lorsque l'on travaille sur des espaces de noms différents:

- Référencer un élément ou un attribut importe sa définition complète, y compris son nom complet avec son espace de noms.
- Référencer un type de données importe sa définition mais vous laisse définir son nom qui ne peut qu'être non qualifié ou appartenir à l'espace de nom du schéma importateur.

Pour définir un attribut "xml:lang", nous n'avons donc pas le choix et devont le définir par référence.

Pour clore ce chapitre à propos des espaces de noms, nous devons encore voir comment, comme promis en introduction, nous pouvons écrire des schémas ouverts et extensibles en autorisant des éléments non déterminés appartenant à d'autres espaces de noms. Ceci est possible grâce à un jeu d'éléments "*any*" et "*anyAttribute*" permettant, respectivement, d'autoriser des éléments ou attributs.

Si nous voulions étendre la définition de notre type "descType" pour qu'il accepte n'importe quel élément XHTML, nous pourrions donc écrire:

```
<xsd:complexType name="descType" mixed="true">
 <xsd:sequence>
  <xsd:any namespace="http://www.w3.org/1999/xhtml"
   minOccurs="0" maxOccurs="unbounded"
   processContents="skip"/>
 </xsd:sequence>
</xsd:complexType>
```

"descType" deviendrait ainsi un type à contenu mixte acceptant un nombre quelconque d'éléments de l'espace de noms "http://www.w3.org/1999/xhtml namespace". L'attribut "processContents" est positionné à "skip" pour indiquer à **W3C XML Schema** que nous ne souhaitons pas valider ces éléments par rapport à un schéma pour **XHTML**. Mes autres valeurs possibles sont "strict" imposant une validation et "lax" demandant de valider si le schéma correspondant est disponible. L'attribut "namespace" accepte une liste d'URIs séparées par des espaces et les valeurs spéciales "##any" (n'importe quel espace de noms), "##local" (éléments non qualifiés), "##targetNamespace" (l'espace de noms cible) et "##other" (tout espace de noms autre que la cible).

Il n'est pas possible de spécifier que l'espace de noms doit être autre que ceux contenus dans une liste.

"*anyAttribute*" permet la même fonctionalités que "*any*" mais est réservé aux attributs.

Retour au plan

10. W3C XML Schema dans les documents XML

Maintenant que nous avons vu la plupart des fonctions que nous pouvons utiliser dans un **W3C XML Schema**, il nous reste à voir quelques extensions pouvant être utilisées dans les documents eux-mêmes. Ces extensions sont identifiées par un espace de noms spécifique "http://www.w3.org/2000/10/XMLSchema-instance" souvent associé au préfixe "xsi".

Les deux premiers attributs permettent de lier un document à des schémas **W3C XML Schema**. Ce lien n'est pas une contrainte absolue et l'association doit également pouvoir être faite au moment de la validation en utilisant des mécanismes dépendant de l'outil utilisé (notamment au moyen de paramètres de la ligne de commande). Ce lien est donc une indication qui aidera les outils **W3C XML Schema** à localiser un schéma en dehors de toute autre information.

Pour associer le document avec un schéma ne définissant pas d'espace de noms, on utilise l'attribut "noNamespaceSchemaLocation":

```
<book isbn="0836217462"
   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file:library.xsd">
```

alors que pour l'associer avec un schéma définissant un espace de nom on utilise "schemaLocation" (attention à la syntaxe associant une URI d'espace de nom et l'URI du schéma séparés par un espace dans un même attribut):

```
<book isbn="0836217462" xmlns="http://example.org/ns/books/"
   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
   xsi:schemaLocation="http://example.org/ns/books/ file:library.xsd">
```

Les deux autres attributs xsi sont "xsi:type" qui définit le type d'un élément et "xsi:null" qui indique que la valeur d'un élément est nulle (l'élément doit avoir été défini comme pouvant être nul au moyen de nullable="true" dans le schéma).

Ces deux attributs peuvent être utilisés dans les documents sans avoir été définis dans le schéma.