

XML Schema

Jamel Eddine Jridi

Introduction

doc bien formé respecte les regles de base de langage

- Un document **bien formé n'est pas suffisant**
- **Validation** d'un document XML :
 - DTD : Document Type Definition
 - **XSD : XML Schema Definition**
 - RNC : Relax NG Compact
- **Objectif de XML Schema**
 - Amélioration du typage type string
 - Amélioration de la structure
 - Dépasser les limites du DTD

DTD vs. XML Schema

- Limites de DTD :

XML description
Typage
contrôle
espaces de noms
réutil/extension

- Description non XML de la structure d'un document
- Typage des données est extrêmement limité.
- Peu ou pas de contrôle des valeurs contenues dans la structure
- Pas de support pour les espaces de noms
- Très limité pour la réutilisation et l'extensibilité

DTD ne base pas sur les balises

on peut pas utiliser un bloc dans un autre schema

- Avantages du XML Schema

- Syntaxe basée sur XML
- Soutenir les espaces de noms
- Plusieurs types de données
- Capable de créer des types de données complexes
- Notion d'héritage: les éléments peuvent hériter du contenu et des attributs d'un autre élément.

Espaces de noms XML

- Lever l'ambiguïté sur le noms des balises
- Un espace de noms est **identifié par une URI**
- La déclaration d'un espace de noms doit se faire **dans un élément**
- Utilisable pour l'élément et ses descendants
- Préfixe et identification : `xmlns:prefix="Identification">`
- `xmlns` sans préfixe = espace de nom par défaut
- Utilisation conjointe de balise `<nom>` dans une structure XML
- Distingués par l'utilisation d'un espace de noms
 - `<liv:nom xmlns:liv="http://www.umontreal.ca/book">`
 - `<aut:nom xmlns:aut="http://www.umontreal.ca/writer">`

Y a 2 facons de declarer un espace de noms en haut est a l'interieur d'un element et en bas on l'a dans la racine qui sera la facon generique pour tt le document

DOC1

```
<personne>
  <prof>
    <nom> jamel </nom>
  </prof>
  <student>
    <nom> felix </nom>
  </student>
</personne>
```

DOC2

```
<personnes>

  <personne>
    <A: nom> jamel </nom>
  </personne>

  <personne>
    <B: nom> felix </nom>
  </personne>
</personnes>
```

ici au contraire du doc1 on peut pas differencir le nom
est appartient a prof ou student donc on utiliser des
espace de nom comme A: prof et B:Etudiant
A et B sont des URI pour identifier des ressources

Déclaration d'un schéma

- Pour faire référence à une DTD dans un document XML:

```
<?xml version="1.0"?>  
<!DOCTYPE root SYSTEM "url">  
<root> ... </root>
```

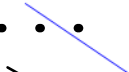
- Pour faire référence à un schéma XML dans un document XML:

```
<?xml version="1.0"?>  
<root
```

espace de nom generique
pour tt le document

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

```
...  
</root>  
  <a>  
    <xsi: nom>....</nom>  
  </a>
```



Structure d'un schéma XSD

- Un document XSD doit commencer par :

```
<?xml version="1.0"?>  
<xs:schema  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

doit être respecté - et tout les descendants doit l'avoir aussi

- `<schema>` est la racine de tout document XSD qui aura les attributs :
 - `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
 - Nécessaire pour spécifier l'endroit où toutes les balises XSD sont définies
 - `elementFormDefault="qualified"`
 - Tous les éléments XML doivent être qualifiés (utiliser un espace de noms)
 - Il est hautement souhaitable de qualifier tous les éléments, ou des problèmes se posera lorsque un autre schéma est ajouté

Structure d'un schéma XSD

- Un document XSD doit commencer par :

```
<?xml version="1.0" encoding="UTF-8"?>  
<xs:schema  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

- `<schema>` est la racine de tout document XSD qui aura les attributs :
 - `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
 - Nécessaire pour spécifier l'endroit où toutes les balises XSD sont définies
- Une structure XML est hiérarchique
- Définition des éléments simples et complexes.

Élément simple vs. complexe

- Un élément simple contient du texte et rien d'autre
- Un élément simple ne peut pas avoir des attributs
- Un élément simple ne peut pas contenir d'autres éléments
- Un élément simple ne peut pas être vide
- Si un élément **n'est pas simple**, il est "**complexe**"
- Un élément complexe peut avoir des attributs
- Un élément complexe peut être vide, ou bien il peut contenir du texte, d'autres éléments, ou les deux textes et autres éléments.

Définition d'un élément simple en XML Schema

- Un élément simple est défini par :

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

Avec :

- **name** est le nom de l'élément
- Les valeurs les plus courantes pour l'attribut **type** sont :

xs:boolean	xs:integer
xs:date	xs:string
xs:decimal	xs:time
- Autres attributs d'un élément simple peut être:
 - default="**default value**" : donner une valeur par défaut.
 - fixed="**value**" : aucune autre valeur peut être spécifiée

Définition d'un attribut

- Les attributs eux-mêmes sont toujours déclarés comme des types simples.
- Un attribut est déclaré comme suit :

```
<xs:attribute    name="name"    type="type"  />
```

Avec :

- **name** et **type** sont les mêmes comme les `xs:element`
- Autres attributs d'un élément simple peut être:
 - `default="default value"` : donner une valeur par défaut.
 - `fixed="value"` : aucune autre valeur peut être spécifiée
 - `use="optional"` : l'attribut n'est pas obligatoire (par défaut)
 - `use="required"`

Définition des restrictions

- La forme générale pour mettre une restriction à une valeur de texte est la suivante :

```
<xs:element name="name">                                (ou xs:attribute)
  <xs:restriction base="type">
    ... restrictions ...
  </xs:restriction>
</xs:element>
```

- Exemple

```
<xs:element name="age">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0">
    <xs:maxInclusive value="40">
  </xs:restriction>
</xs:element>
```

Restrictions sur les valeurs

- `minInclusive` -- le nombre doit être \geq à une valeur donnée.
- `minExclusive` -- le nombre doit être $>$ à une valeur donnée.
- `maxInclusive` -- le nombre doit être \leq à une valeur donnée.
- `maxExclusive` -- le nombre doit être $<$ à une valeur donnée.
- `totalDigits` -- le nombre doit avoir un nombre donné de digits
- `fractionDigits` – le nombre ne doit pas avoir plus de chiffres après la virgule

Restrictions sur les string

essaie

- `length` -- la chaîne doit contenir un nombre donné des caractères
- `minLength` -- la chaîne peut avoir au minimum n caractères
- `maxLength` -- la chaîne peut avoir au maximum n caractères
- `pattern` -- la chaîne doit correspondre à une expression régulière
- `whiteSpace` – traitement des espaces
 - `value="preserve"` : garder tous les espaces
 - `value="replace"` : changer tous les caractères blancs aux espaces
 - `value="collapse"` : retirer les espaces avant et après, et remplacer toutes les séquences d'espaces avec un seul espace

Les énumérations

```
<xs:element name="season">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Spring"/>
      <xs:enumeration value="Summer"/>
      <xs:enumeration value="Autumn"/>
      <xs:enumeration value="Fall"/>
      <xs:enumeration value="Winter"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Définition d'un élément complexe

- Un élément complexe est défini par :

```
<xs:element name="name">  
  <xs:complexType>  
    ...
```

```
  </xs:complexType>  
</xs:element>
```

- Exemple

```
<xs:element name="person">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="firstName" type="xs:string"/>  
      <xs:element name="lastName" type="xs:string" />  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```

```
<person>  
  <firstName>  
  </..>  
  <lastName>  
  </...>  
</person>
```

ordre doit etre respecte

Les éléments doivent
apparaître dans cet
ordre

Portée globale et locale

ce sont comme variable globale et var locale

- Les élément déclarés dans `xs:schema` sont disponibles pour une utilisation tout au long du schéma (portée globale)
- Les éléments déclarés dans un `xs:complexType` sont locales a ce dernier.
- Exemple

```
<xs:element name="person"> globale
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstName" type="xs:string"/>
      <xs:element name="lastName" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Déclaration et utilisation

- Jusqu'à présent, nous avons parlé de la façon de déclarer des types, pas comment les utiliser
- Pour utiliser un type que nous avons déclaré, l'utiliser comme la valeur de l'attribut `type = "..."`
- Exemple:

```
<xs:element name ="étudiant" type="person" />
```

```
<xs:element name ="professeur" type="person" />
```
- Vous ne pouvez pas utiliser un type si est local à un autre type

xs:sequence VS. xs:all

- xs:sequence = les éléments doivent apparaître dans un ordre particulier au contraire que le xs:all

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstName" type="xs:string"/>
      <xs:element name="lastName" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

équivalent à "all" en DTD (A|B|C)*,

- Les membres d'un xs:all peuvent apparaître 0 ou une fois.
- Vous pouvez utiliser minOccurs="0" pour indiquer qu'un élément est optionnel (valeur par défaut est 1)

Référence

- Une fois que vous avez défini un élément ou un attribut (avec `name`), vous pouvez vous référer à elle avec `ref`

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="lastName" type="xs:string" />
    </xs:all>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="student" ref="person">
```

Ou juste: `<xs:element ref="person">`

ref va affecter sur element
"element" est ses decendants
ref => "element" sera enfant de
l'ature "element"

Élément textuel avec attribut

- Si un élément textuel a des attributs, il n'est plus un type simple

```
<xs:element name="shoesize">
```

```
  <xs:complexType>
```

```
    <xs:simpleContent>
```

```
      <xs:extension base="xs:integer">
```

```
        <xs:attribute name="country" type="xs:integer">
```

```
      </xs:extension>
```

```
    </xs:simpleContent>
```

```
  </xs:complexType>
```

```
</xs:element>
```

"shoesize" est un "simpleContent" qui étend la base "integer" donc "35", est étendu aussi l'attribut "country" de type "string" donc "France"

string

```
<shoesize country="france">35</shoesize>
```

- `xs:simpleContent` contient des extensions ou des restrictions sur un type complexe textuel seulement ou sur un type simple comme contenu, et **ne contient pas d'éléments**.

Élément vide et mixte

- Tout élément vide est complexe

```
<xs:complexType name="counter">  
  <xs:complexContent> <A alt="32" />  
    <xs:extension base="xs:anyType" />  
    <xs:attribute name="count" type="xs:integer" />  
  </xs:complexContent>  
</xs:complexType>
```

- Élément mixte

```
<xs:complexType name="paragraph" mixed="true">  
  <xs:sequence>  
    <xs:element name="someName" type="xs:anyType" />  
  </xs:sequence>  
</xs:complexType>
```

```
<paragraph>  
  <someName>
```

Les extensions

```
<xs:element name="employee" type="fullpersoninfo"/>
```

```
<xs:complexType name="personinfo"> un type  
  <xs:sequence>  
    <xs:element name="firstname" type="xs:string"/>  
    <xs:element name="lastname" type="xs:string"/>  
  </xs:sequence>  
</xs:complexType>
```

```
<xs:complexType name="fullpersoninfo"> un autre type  
  <xs:complexContent> complex car ce type entend le type en haut et donc on peut avoir  
    <xs:extension base="personinfo"> un element complexe  
      <xs:sequence>  
        <xs:element name="address" type="xs:string"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>
```

`<xs:element name="student" type="fullpersonifo"/>` --> XML compatible

```
<student>
  <firstName> ... </...>
  <lastName>.....</..>
  <adress> ... </...>
</student>
```

le type peut contenir un "space de nom" , regarde l'exemple sur le site