Les Schémas XML
Richesse des types

Yves Bekkers

Mise à jour : 30 septembre 2011

Plan

- Introduction par un exemple
- Les éléments du langage XMLSchema
- Modèle de contenu d'élément
- Contraintes pour les attributs
- Définitions locales
- Réutilisation : les groupes
- Validation
- Conclusion

Introduction par un exemple

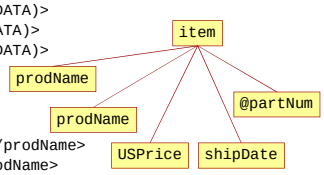
DTD versus XMLSchéma

Un exemple introductif

- Définition d'un élément `<item>` par une DTD

```
<!ELEMENT item (prodName+,USPrice,shipDate?)>
<!ATTLIST item partNum CDATA #implied>
<!ELEMENT prodName (#PCDATA)>
<!ELEMENT USPrice (#PCDATA)>
<!ELEMENT shipDate (#PCDATA)>
```

Séquence



- Exemple d'instance

```
<item partNum="124-PQ">
  <prodName>imprimante</prodName>
  <prodName>printer</prodName>
  <USPrice>150</USPrice>
  <shipDate>10/4/04</shipDate>
</item>
```


Élément racine d'un schéma xmlschema

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


...
...

</xs:schema>
```


XMLSchema un dialecte XML

- Espace de noms du dialecte XMLSchema
  - `http://www.w3.org/2001/XMLSchema`
  - Préfixe recommandé `xs:`

Définition par schémaXML

```
<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="prodName"
          type="xs:string" maxOccurs="5"/>
      <xs:element name="USPrice"
          type="xs:decimal"/>
      <xs:element name="shipDate"
          type="xs:date" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="partNum" type="SKU"/>
  </xs:complexType>
</xs:element>
```

Séquence

Type utilisateur

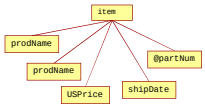
Types prédéfinis

Définition d'un type utilisateur

```
<!– Code Produit -->
<xs:simpleType name="SKU">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
```

- XMLSchema est un dialecte XML
  - C'est plus bavard que les DTDs …

DTD versus XMLSchema



- Plus de contraintes exprimées avec XMLSchema

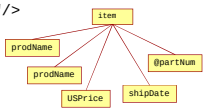
Composant	DTD	XMLSchema
prodName	#PCDATA	xs:string
USPrice	#PCDATA	xs:decimal
ShipDate	#PCDATA	xs:date
@partNum	CDATA	xs:string + *restriction*

- Des types de base primitifs prédéfinis
  - `xs:string, xs:decimal, xs:string, …`
- Des types définis par l'utilisateur

Déclaration d'un type dérivé

- Le type `SKU` (*code produit)* est défini à l'aide d'une déclaration de type dérivée du type `xs:string`

```
<xs:simpleType name="SKU">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
```



- Une contrainte sur la chaîne
  - 3 chiffres + "-" + 2 lettres majuscules
  - Exemple : `344-AB`
- La même restriction pourrait s'appliquer à un contenu d'élément

Contraintes d'occurrence dans les modèles de contenu

- Un modèle de contenu peut contenir des contraintes d'occurrence. Exemple :
  - *N* le nombre d'éléments `<prodName>`
  - Contrainte : $0 \leq N \leq 5$

```
<xs:element name="prodName"
      type="xs:string" minOccurs="0"
          maxOccurs="5"/>
```


Contrainte : $7 \leq N \leq 12$

- DTD

```
<!ELEMENT biscuit(#PCDATA)>
<!ELEMENT commande(biscuit,biscuit,biscuit,biscuit,
    biscuit,biscuit,biscuit,
    biscuit?,biscuit?,biscuit?,biscuit?,biscuit?)>
```

- XMLSchéma

```
<xs:element name="commande">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="biscuit" type="xs:string"
       minOccurs="7" maxOccurs="12" />
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Types énumérés pour les attributs

- DTD

```
<!ELEMENT person (description du contenu de person)>
<!ATTLIST person couleur (red | green ) "red">
```

- XMLSchéma

```
<xs:simpleType name="color">
  <xs:restriction base="xs:string">
    <xs:enumeration value="red"/>
    <xs:enumeration value="green"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="person" type="person_type">
  <xs:complexType>
    <xs:attribute name="couleur" type="color"/>
  </xs:complexType>
</xs:element>
```


DTD et modèle de contenu d'élément

- DTD
  - Impossible de spécifier un *type énuméré* comme contenu d'élément
- XMLSchéma
  - Comme un attribut, un contenu d'élément peut être de type énuméré
  - Exemple

```
<xs:element name="shoes" type="shoe_color">
```


Valeur par défaut

- Element

```
<xs:element name="shoes"
  type="shoe_color" default="red">
```

- Attribut

```
<xs:attribute
 name="shoes" type="shoe_color"
 default="red">
```


DTD versus XMLSchema résumé

- Plus de sémantique exprimée par le schéma
  - L'élément `<shipDate>` contient une *date*
  - L'élément `<USPrice>` contient un *décimal*
  - L'attribut `@partNum` contient une *chaîne restreinte* à certains modèles
- Traitement orthogonale des attributs et des contenus d'élément (e.g. les types énumérés)
- Contraintes d'occurrences plus souples

Vision W3C d'un schéma XML

- Un schéma XML est construit sur deux ensembles complémentaires de constructions
  - Les *modèles de contenu*
    - Modèle de *contenu vide*
    - Modèle de *contenu simple* (que du texte)
    - Modèle de *contenu complexe* (que des éléments)
    - Modèle de *contenu mixte* (texte + éléments)
  - Les *types de données*
    - *Type simple* (attributs + éléments à contenu simple sans attribut)
    - *Type complexe* (éléments à contenu simple avec attribut + tous les autres)

Déclaration versus définition

- Les schéma XML contiennent
  - Des *déclarations* pour les composants d'instance de document
    - Éléments
    - Attributs
    - Notations
  - Des *définitions* pour les composants internes au schéma
    - Types
    - Groupe *modèles de contenu*
    - Groupe d'attributs

Étude des éléments du langage XMLSchema

Élément <xs:schema>

- Élément racine d'un schéma XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema name="…"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- constructions de niveau 0 -->
</xs:schema>
```

- Constructions de niveau 0 (*composants globaux*)
  - Des déclarations d'élément
  - Des définitions de type complexes.
  - Des définitions de types simples
  - D'autres encore …/…

Composants globaux ou locaux

- Les composants globaux apparaissent au premier niveau au sein de l'élément <xs:schema>
  - Ils sont toujours nommés (attribut name="…")
  - Leur nom doit être unique au sein de leur type de composant
- Les composants locaux
  - Leur nom a une portée locale au type complexe dans lequel ils sont définis
  - Types simples et types complexes définis localement sont anonymes (ils ne peuvent être réutilisés)

Conflit de noms

- Deux types ne peuvent avoir le même nom
- Deux éléments de type différents dans la même portée ne peuvent avoir le même nom
- Un type et un élément peuvent avoir le même nom

Type et élément de même nom

```
<xs:complexType name="figure" abstract="true">
  <xs:attribute name="id" type="xs:string"/>
 </xs:complexType>
 <xs:element name="root">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="figure" type="a:figure"
 maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```


Type simple et type complexe

- Type simple permettent de donner un type aux :
  - Éléments qui ne contiennent que du texte et sans d'attribut
    - Exemple <prénom>Yves</prénom>
  - Attributs (leur contenu n'est que textuel !)
- Type complexe permettent de donner un type aux :
  - Éléments qui ne contiennent que du texte mais avec attributs
    - Exemple <ville codePostal="35000">Rennes</ville>
  - Éléments qui contiennent d'autres éléments, à contenu mixte ou non

Élément <xs:complexeType>

- Déclaration de *type complexe*

```
<xs:complexeType name="…" mixed="…">
  <!-- un modèle de contenu -->
  <!-- des déclarations d'attributs -->
</xs:complexeType>
```

1. Déclaration d'attribut
   `<xs:attribute name="…" type="…" use="…">`
   - L'attribut `type` fait référence à un type simple
   - L'attribut `use` prend une valeur parmi
     - `required, optional, prohibited`
2. Modèle de contenu … *à suivre* …

Modèle de contenus pour éléments

Modèle de contenu

- 4 sortes de modèles de contenu
  - Contenu vide (par défaut)
  - Contenu simple (seulement pour les contenus caractères)
    - Type simple comme les attributs

      `<xs:simpleContent>`
      …
      `</xs:simpleContent>`
  - Contenu composé d'éléments
    - Expression régulière
      - Combinaison `<xs:sequence>, <xs:choice>, <xs:all>`
      - Chacun contenant des références à élément de la forme
      - `<xs:element ref="…" minOccurs="…" maxOccurs="…"/>`
      - L'attribut `ref` fait référence à une définition d'élément
  - Attribut `mixed="true | false"` contenu mixte ou non

Contenu vide avec attribut

- Exemple

```
<prénom val="yves"/>
```

- Définition

```
<xs:element name="prénom">
  <xs:complexType>
     <xs:attribute name="val"
          type="xs:string"/>
  </xs:complexType>
</xs:element>
```


Contenu simple textuel

- Exemple

```
<prénom>Yves</prénom>
```

- Définition 1

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

- Définition 2

```
<xs:element name="prénom">
   <xs:complexType mixed="true"/>
</xs:element>
```

- Définition 3

```
<xs:element name="prénom">
   <xs:complexType>
     <xs:simpleContent>
       <xs:extension base="xs:string"/>
     </xs:simpleContent>
   <xs:complexType>
</xs:element>
```


Contenu simple textuel avec attribut

- Exemple

```
<ville code="35770">vern sur seiche</ville>
```

- Définition

```
<xs:element name="ville">
   <xs:complexType mixed="true">
     <xs:attribute name="code"
         type="xs:string" />
   </xs:complexType>
</xs:element>
```

Autre moyen –l'extension d'un <simpleContent>

- Exemple

```
<ville code="35770">vern sur seiche</ville>
```

- Définition

```
<xs:element name="ville">
   <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="xs:string">
      <xs:attribute name="code" type="xs:string" />
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
 </xs:element>
```


Contenu simple avec attribut et restriction du type de contenu (bis)

- Exemple

```
<price currency="dollard">15.5</price>
```

- Définition

```
<xs:element name="price">
   <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="xs:decimal">
      <xs:attribute name="currency" type="xs:string" />
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
 </xs:element>
```


Contenu complexe Expressions régulières

- Trois opérateurs de composition
  - Élément <xs:sequence> séquence d'éléments
  - Élément <xs:choice> choix d'éléments
  - Élément <xs:all> permutation d'éléments

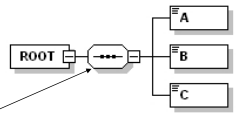
Séquence d'éléments

- DTD

```
<!ELEMENT ROOT (A,B,C) >
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

- XMLSchema

```
<xs:element name="ROOT">
 <xs:complexType mixed="false">
  <xs:sequence>
    <xs:element name="A" type="xs:string"/>
    <xs:element name="B" type="xs:string"/>
    <xs:element name="C" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```



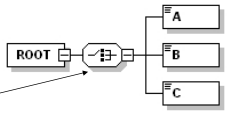
Choix entre éléments

- DTD

```
<!ELEMENT ROOT (A|B|C) >
```

- XMLSchema

```
<xs:element name="ROOT">
 <xs:complexType mixed="false">
  <xs:choice>
    <xs:element name="A" type="xs:string"/>
    <xs:element name="B" type="xs:string"/>
    <xs:element name="C" type="xs:string"/>
  </xs:choice>
 </xs:complexType>
</xs:element>
```



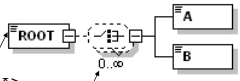
Élément à contenu mixte

- DTD

```
<!ELEMENT ROOT (#PCDATA|A|B)*>
```

- XMLSchema

```
<xs:element name="ROOT">
 <xs:complexType mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="A" type="xs:string"/>
    <xs:element name="B" type="xs:string"/>
  </xs:choice>
 </xs:complexType>
</xs:element>
```



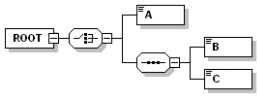
Expression régulière (1)

- DTD

```
<!ELEMENT ROOT (A|(B,C))>
```

- XMLSchema

```
<xs:element name="ROOT">
 <xs:complexType mixed="false">
  <xs:choice>
   <xs:element name="A" type="xs:string"/>
    <xs:sequence>
     <xs:element name="B" type="xs:string"/>
     <xs:element name="C" type="xs:string"/>
    </xs:sequence>
  </xs:choice>
 </xs:complexType>
</xs:element>
```



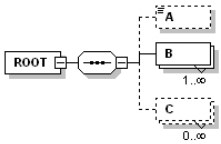
Expression régulière (2)

- DTD

```
<!ELEMENT ROOT (A?,B+,C*) >
```

- XMLSchema

```
<xs:element name="ROOT">
 <xs:complexType mixed="false">
  <xs:sequence>
   <xs:element name="A" minOccurs="0"/>
   <xs:element name="B" maxOccurs="unbounded"/>
   <xs:element name="C" minOccurs="0"
                        maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```



Valeurs par défaut : 1

Réutilisation d'un type

- DTD
  - On utilise les entités paramètres

  Déclaration `<!ENTITY % seqABC "A,B,C">`

  Utilisation `<!ELEMENT root (&seqABC;)>`

Modèle importé dans sa totalité

- XMLSchema :
  - Un type déclaré au niveau 1 de l'élément `<xs:schema>`
  - Il est global, il porte un nom, il peut être réutilisé

```
<xs:complexType mixed="false" name="root">
  <xs:sequence>
    <xs:element name="A" type="xs:string"/>
    <xs:element name="B" type="xs:string"/>
    <xs:element name="C" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```


Types différents pour un même élément

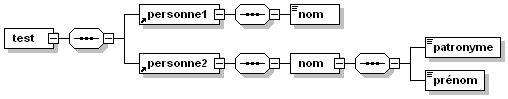
Deux types différents pour le même nom d'élément - 1

```
<xs:element name="personne1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nom" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```


Deux types différents pour le même nom d'élément - 2

```
<xs:element name="personne2">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="nom">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="patronyme" type="xs:string"/>
      <xs:element name="prénom" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Deux types différents pour le même nom d'élément - 3




```
<xs:element name="test">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="personne1"/>
      <xs:element ref="personne2"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```


Contraintes pour les attributs

Attribut chaîne obligatoire

- DTD

```
<!ELEMENT ROOT EMPTY>
<!ATTLIST ROOT a CDATA #REQUIRED>
```

- XMLSchema

```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:attribute name="a" type="xs:string"
      use="required"/>
  </xs:complexType>
</xs:element>
```

obligatoire

Attribut chaîne optionnel

- DTD

```
<!ELEMENT ROOT EMPTY>
<!ATTLIST ROOT a CDATA #IMPLIED>
```

- XMLSchema

```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:attribute name="a" type="xs:string"
      use="optional"/>
  </xs:complexType>
</xs:element>
```

optionnel

Type énuméré pour un attribut

- DTD

```
<!ELEMENT ROOT EMPTY>
<!ATTLIST ROOT a (x|y|z) #REQUIRED>
```

- XMLSchema

```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:attribute name="a" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="x"/>
          <xs:enumeration value="y"/>
          <xs:enumeration value="z"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

Définition d'un type énuméré
Par restriction du type `xs:string`

Attribut à valeur fixe

- DTD

```
<!ELEMENT ROOT EMPTY>
<!ATTLIST ROOT a CDATA #FIXED "x">
```

- XMLSchema

```
<xs:element name="ROOT">
 <xs:complexType mixed="false">
  <xs:attribute name="a" type="xs:string"
      fixed="x"/>
 </xs:complexType>
</xs:element>
```

Valeur fixe

Types simples

- Les types simples peuvent être
  - Primitifs (ne dérivant pas d'un autre)
  - Dérivés d'un autres type simple
    - Par une *liste* : séquence de types séparés par des blancs
    - Par une *union* : union d'autres types simples
    - Par une *restriction* :
      - `length, minLength, maxLength` (longueur de listes)
      - `enumeration` (liste de valeurs)
      - `pattern` (expression régulière à la Perl)
      - `whitespace` (préserver, remplacer, réduire les espaces)
      - `minInclusive, maxInclusive` (intervales bornés de valeurs)

Types simples prédéfinis

- Ils peuvent être primitifs ou dérivés
- Il y en a une cinquantaine
- Exemples :

`string`	`negative-integer`
`normalized-string`	`date`
`name`	`time`
`NCName`	`datetime`
`langage`	`duration`
`float`	`ID`
`double`	`IDREF`
`long`	`IDREFS`
`int`	`boolean`
`short`	`hexBinary`
`positive-integer`	`anyURI`

Restrictions d'un type

Exemple : restriction de la longueur d'une chaîne

```
<xs:simpleType name="chaine32">
  <xs:restriction base="xs:string">
    <xs:maxLength value="32"/>
  </xs:restriction>
</xs:simpleType>
```


Longueur minimum d'un identificateur

```
<xs:simpleType name="longName">
  <xs:restriction base="xs:NCName">
    <xs:minLength value="6"/>
  </xs:restriction>
</xs:simpleType>
```


Exemple : restriction du type `date`

```
<xs:element name="ROOT">
  <xs:simpleType name="maDate">
    <xs:restriction base="xs:date">
      <xs:pattern value="\d{4}-05-\d{2}"/>
    </xs:restriction>
  </xs:simpleType>
 </xs:element>
```

Entier borné

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


Type énuméré - 1

```
<xs:element name="car">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```


Type énuméré - 2

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

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```


Restriction d'un type simple

- Contenu
  - xs:minExclusive, xs:minInclusive, xs:maxExclusive, xs:maxInclusive, xs:totalDigits, xs:fractionDigits, xs:length, xs:maxLength, xs:minLength: xs:enumeration: xs:whiteSpace, xs:pattern
  - xs:attribute, xs:attributeGroup, xs:anyAttribute (contenu simple d'un type complexe)

Union de types simples

```
<qtes>
 <qte valeur="12"/>
 <qte valeur="*"/>
</qtes>
```


Union de types simples (bis)

```
<xs:attribute name="valeur" use="required">
  <xs:simpleType>
    <xs:union memberTypes="xs:decimal star"/>
  </xs:simpleType>
</xs:attribute>
<xs:simpleType name="star">
  <xs:restriction base="xs:string">
    <xs:enumeration value="*"/>
  </xs:restriction>
</xs:simpleType>
```

Restriction d'un type complexe

- Permet de réduire le nombre d'instances valides (la description de contenu doit être valide pour le contenu de base, les attributs inchangés peuvent être omis)
- Contenu
  - xs:group, xs:all, xs:choice, xs:sequence
  - xs:attribute, xs:attributeGroup, xs:anyAttribute

Réutilisation d'un type simple

- DTD
  - On utilise les entités paramètres

  Déclaration `<!ENTITY % ContentType "CDATA">`

  Utilisation `<!ATTLIST elem at %ContentType; "x">`
- XMLSchema
  - Un type déclaré au niveau 0 de l'élément `<xs:schema>` est global
  - Il porte un nom, il peut être réutilisé

```
<xs:simpleType name='ContentType'>
  <xs:restriction base='xs:string'/>
</xs:simpleType>
```


Définitions locales/globales

Premier style d'écriture : "*à plat*"

- Utiliser une liste de définitions et déclarations *globales*
  - Toutes les définitions et déclarations sont directement au niveau 0 de l'élément `<xs:schema>`

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="ROOT" type="root"/>
<xs:element name="A" type="xs:string"/>
<xs:element name="B" type="xs:string"/>
<xs:complexType mixed="false" name="root">
  <xs:choice>
    <xs:element ref="A"/>
    <xs:element ref="B"/>
  </xs:choice>
 </xs:complexType>
</xs:schema>
```

Définition globale de type
- peut être dérivé

Définitions globales d'éléments
- peuvent être réutilisées partout

Second style d'écriture : *"en poupées russes"*

- Définitions et déclarations sont mises en ligne

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:choice>
      <xs:element name="A" type="xs:string"/>
      <xs:element name="B" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Définition locale de type
- ne peut pas être dérivée

déclaration locale d'élément
- ne peut pas être réutilisé ailleurs
- y compris comme racine de document

Réutilisation : les groupes

Réutilisation – groupe d'éléments

- DTD
  - On utilise les entités paramètres

  Déclaration `<!ENTITY % seqAB "(A,B)">`

  Utilisation `<!ELEMENT root (&seqAB;|C)>`
- XMLSchema : *groupe d'éléments*
  - Déclaration globale

```
<xs:group name="seqAB">
   <xs:sequence>
     <xs:element name="A" type="xs:string"/>
     <xs:element name="B" type="xs:string"/>
   </xs:sequence>
 </xs:group>
```


Réutilisation d'un groupe d'élément

- DTD

  Déclaration `<!ENTITY % seqAB "(A,B)">`

  Utilisation `<!ELEMENT root (&seqAB;|C)>`
- XMLSchema : *groupe d'éléments*
  - Utilisation du groupe `seqAB`

```
<xs:element name="ROOT">
 <xs:complexType mixed="false">
   <xs:choice>
     <xs:group ref="seqAB"/>
     <xs:element name="C" type="xs:string"/>
   </xs:choice>
 </xs:complexType>
</xs:element>
```

Référence

Réutilisation – groupe d'attributs

- DTD - On utilise les entités paramètres
- XMLSchema : *groupe d'attributs*
  - Déclaré au niveau 0 de l'élément `<xs:schema>`

```
<xs:attributeGroup name="g1">
  <xs:attribute name="x" type="xs:string" default="x"/>
</xs:attributeGroup>
```

  - Utilisé dans un type complexe

```
<xs:element name="ROOT">
  <xs:complexType>
     <xs:choice>
           <xs:group ref="seqAB"/>
           <xs:element name="C" type="xs:string"/>
     </xs:choice>
     <xs:attributeGroup ref="g1"/>
  </xs:complexType>
</xs:element>
```

Attention
la référence au groupe d'attributs doit apparaître après le modèle de contenu

Contraintes d'intégrité

Contrainte d'unicité

Contrainte de référencement

Exemple 1

```
<root>
  <AAA>
    <a id="x"/>
    <a id="y"/>
    <a id="z"/>
  </AAA>
</root>
```

- Définition d'un ensemble de clés nommé `myid`

```
<xs:element name="root"  type="root-type">
  <xs:key name="myid">
    <xs:selector xpath="./AAA/a"/>
    <xs:field xpath="@id"/>
  </xs:key>
</xs:element>
```


Exemple 1 - bis

```
<root>
  <AAA>
    <a id="x"/>
    <a id="y"/>
    <a id="z"/>
  </AAA>
  <BBB>
    <b refid="x"/>
    <b refid="y"/>
    <b refid="z"/>
  </BBB>
</root>
```

référence

Exemple 1 - ter

```
<xs:element name="root"  type="root-type">
  <xs:key name="myid">
    <xs:selector xpath="./AAA/a"/>
    <xs:field xpath="@id"/>
  </xs:key>
  <xs:keyref name="dummy" refer="myid">
    <xs:selector xpath="./BBB/b"/>
    <xs:field xpath="@idref"/>
  </xs:keyref>
</xs:element>
```


Contraintes d'intrégrité - bis

- Référence

```
<para>
   ...<bibref code="bek1999"/>...
</para>
```

- Définition

```
<bibitem code="bek1999">
   ...<author>Y Bekkers</author>...
</bibitem>
```


Contraintes d'intrégrité - suite

```
<xs:element name="livre" type="…">
  ←Définition des clés→
  <xs:key name="bib">
    <xs:selector xpath='bibitem'/>
    <xs:field xpath='@code'/>
  </xs:key>
  ←Définition des références→
  <xs:keyref name="dummy" refer="bib">
    <xs:selector xpath=".//bibref"/>
    <xs:field xpath='@code'/>
  </xs:keyref>
</xs:element>
```


Contrainte d'unicité sur deux champs

```
<xsd:element name="catalog"
  type="CatalogType">
  <xsd:unique name="dateAndProdNumKey">
    <xsd:selector
             xpath="department/product"/>
    <xsd:field xpath="number"/>
    <xsd:field xpath="@effDate"/>
  </xsd:unique>
</xsd:element>
```


Exemple

```
<catalog>
  <department number="021">
    <product effDate="2000-02-27">
      <number>557</number>
      <name>Short-Sleeved Linen Blouse</name>
      <price currency="USD">29.99</price>
    </product>
    <product effDate="2001-04-02">
      <number>557</number>
      <name>Short-Sleeved Linen Blouse</name>
      <price currency="USD">39.99</price>
    </product>
```


Exemple suite

```
    <product effDate="2001-04-02">
      <number>563</number>
      <name>Ten-Gallon Hat</name>
      <price currency="USD">69.99</price>
    </product>
    <product>
      <number>443</number>
      <name>Deluxe Golf Umbrella</name>
      <price currency="USD">49.99</price>
    </product>
  </department>
</catalog>
```

Validation

Rappel : validation par une DTD

- Nécessité de mettre une déclaration `<!DOCTYPE …>`
- Exemple

```
<?xml version="1.0">
<!DOCTYPE racine SYSTEM "maDTD.xml">
<racine>
   …
</racine>
```


XMLSchema deux dialectes XML

- Deux espaces de noms
  - Dialecte XMLSchema
    - `http://www.w3.org/2001/XMLSchema`
    - Préfixe recommandé `xs:`
  - Dialecte pour référencer une instance de XMLSchema
    - `http://www.w3.org/2001/XMLSchema-instance`
    - Préfixe recommandé `xsi:`

Spécifier l'emplacement des schémas **W3C XML Schema**

- `xsi:schemaLocation`
- `xsi:noNamespaceSchemaLocation`
- Deux attributs de l'espace de noms XSI "`http://www.w3.org/2001/XMLSchema-instance`" qui permettent de spécifier l'emplacement des schémas **W3C XML Schema** que l'on souhaite utiliser pour la validation d'un document.

`xsi:schemaLocation`

- Réservé à une suites de références vers des schémas **W3C XML Schema** décrivant des espaces de noms
- Son contenu est une suite de paires de valeurs alternant des URIs d'espaces de noms et les chemins d'accès aux schémas correspondant.

`xsi:schemaLocation`

```
<racine
 xmlns="http://mondomaine.com/monNom"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:SchemaLocation="http://mondomaine.com/monNom
   monSchema.xsd">
 ...
</racine>
```

1. `http://mondomaine.com/monNom` : mon espace de noms
2. `monSchema.xsd` : fichier contenant mon schéma

xsi:noNamespaceSchemaLocation

- réservé aux références à des schémas décrivant des vocabulaires sans espaces de noms et sa valeur est constituée du chemin permettant d'accéder au schéma correspondant :

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


Mon schéma décrit-il un espace de noms ?

- Pour voir si un schéma décrit un espace de noms, il suffit de regarder l'attribut "**targetNamespace**" de l'élément `xs:schema`
- Pas d'attribut targetNamespace :

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

- Pas d'espace de nom

Mon schéma décrit-il un espace de noms ? (bis)

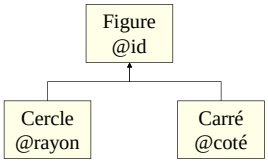
- Présence d'un attribut targetNamspace

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://mondomaine.com/monNom"
  targetNamespace="http://mondomaine.com/monNom"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
```

- Déclaration d'un espace de noms

Type abstrait / héritage et validation

- **Un exemple**



Validation d'un Type abstrait – «figType»

```
<xs:schema targetNamespace= "figureAbstraite"
   xmlns:a="figureAbstraite"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name= "figType" abstract="true">
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="figure" type="a:figType"
    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

définition

utilisation

Cercle – *une extension de type*

```
<xs:complexType name="circle">
  <xs:complexContent>
    <xs:extension base="a:figType">
      <xs:sequence>
        <xs:element name="rayon" type="xs:double"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Carré – *une extension de type*

```
<xs:complexType name="carré">
  <xs:complexContent>
    <xs:extension base="a:figType">
      <xs:sequence>
        <xs:element name="coté" type="xs:double"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```


Exemple de contenu

```
<root
  xmlns:xsi="http://www.w3.org/2001/XMLSc
  hema-instance">
 <figure xsi:type="cercle">
    <rayon>25</rayon>
  </figure>
  <figure xsi:type="carre">
    <coté>25</coté>
  </figure>
</root>
```


Validité de document type abstrait

- Document non valide

```
<root>
  <figure/>
  <figure xsi:type="a:carré"><coté>22.5</coté></figure>
</root>
```

- Document valide

```
<root>
  <figure xsi:type="a:cercle"><rayon>22.5</rayon></figure>
  <figure xsi:type="a:carré"><coté>22.5</coté></figure>
</root>
```


Eléments abstraits

- Nécessité d'utiliser les groupes de substitution

```
<element name="comment" type="string"
   abstract="true"/>
<element name="shipComment"
   type="string"
   substitutionGroup="ipo:comment"/>
<element name="customerComment"
   type="string"
   substitutionGroup="ipo:comment"/>
```


Groupe de substitution

- Attribut `substitutionGroup`
- Des éléments globaux peuvent être regroupés sous un autre élément global (appelé la **tête** du groupe).
- Tous les éléments regroupés sous une tête peuvent être substitués à la tête.
- Tous les éléments sous une tête doivent avoir le même type ou un type dérivé de celui de la tête
- La tête peut être déclarée **abstraite**. Dans ce cas seuls les éléments sous la tête peuvent apparaître dans une instance de document.

Validation en Java par un schéma-xml Compilation DOM

```
final String source = "carnetDAdresse.xml";
final String schema = "carnetDAdresse.xsd";
SchemaFactory schemaFactory =
    SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
Schema schemaXSD = schemaFactory.newSchema(new File(schema));
// créer un compilateur
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
// associer le schéma xml au compilateur
factory.setSchema(schemaXSD);
factory.setNamespaceAware(true); // important ne pas oublier
DocumentBuilder parser = factory.newDocumentBuilder();
Document document = parser.parse(new File(source));
```

Les xlmSchemas ont leur DTD

```
<!DOCTYPE xs:schema PUBLIC
   "-//W3C//DTD XMLSCHEMA 200102//EN"
   "http://www.w3c.org/2001/XMLSchema.dtd">
<xs:schema>
   …
</xs:schema>
```


Modularité

- `<xs:include>`
  - Inclure des définitions d'un même espace de nommage
- `<xs:import>`
  - Inclure des définitions d'un autre espace de nommage
- `<xs:redefine>`
  - Redéfinir dans le schéma actuel des types simples et complexes, des groupes et des groupes d'attributs obtenus à partir de fichiers de schéma externes.

Ce que vous venez de voir

- Introduction par un exemple
- Les éléments du langage XMLSchema
- Modèle de contenu d'élément
- Contraintes pour les attributs
- Définitions locales
- Réutilisation : les groupes
- Validation
- Conclusion

Ce qu'il reste à voir

- La subsomption
- Valeur par défaut pour les contenus d'élément et les attributs
- Contraintes d'unicité `<xs:unique>` et `<xs:key>`
  - Xpath est utilisé pour définir des contraintes d'unicité

Conclusion

XMLSchema plus puissant que les DTDs, mais …

- Les schémas XML permettent d'exprimer beaucoup plus de contraintes que les DTDs
- Mais pour vérifier qu'un document est fonctionnel
  - Il restera toujours des contraintes qui ne seront pas exprimables dans un schéma
  - Elles seront vérifiées par les applications
- C'est gros et complexe (200 pages de définitions)
  - Il y a la place pour d'autres schéma de langage plus simples …
- Malgré cela les schémas XML sont en train d'entrer dans les mœurs

D'autres langages de schéma

- RELAX NG
  `http://www.oasis-open.org/committes/relax-ng/ [lien]`
- Schematron
  `http://www.ascc.net/xml/ressource/schematron/schematron.html [lien]`
- DSD *Document Structure Description*
  `http://www.brics.dk./DSD/ [lien]`
- Bien d'autres
  …

Références

- *XML Schema*, Eric van der Vlist, O'Reilly
- Un court tutorial en ligne par le même auteur
  - `http://www.xml.com/pub/a/2000/11/29/schemas/part1.html [lien]`
- *The XML revolution*, Anders Moller & Micheal I. Schwartzbach
  - `http://www.brics.dk/~amoeller/XML [lien]`