XML Schema

Cours « Document et Web Sémantique »

Nicolas Malandain, Nicolas Delestre

Architecture des Systèmes d'Information

Architecture des Systèmes d'Information

Limite des DTD

- Préciser quelle doit être la racine du document ✗
- Préciser le type d'un attribut ou d'un élément ✗
- Spécifier précisément l'arité d'un élément ✗
- Préciser différemment la nature d'un élément en fonction de sa position dans le fichier XML ✗
- Utiliser la grammaire en tant que document XML (utilisable en entrée d'un processeur XSL) ✗
- Conditionner la valeur d'un nœud text ✗

Présentation de XML Schema

- Application XML
- Langage de modélisation (typage des données, génération automatique de formulaire)
- Validation (tend à remplacer les DTD)
- Documentation (cf. *xs :annotation*)
- Amélioration de XPath/XSLT (connaissance du schéma)

Site de référence

`http://www.w3.org/XML/Schema`

Définitions

Le modèle de contenu des éléments

vide	aucun sous élément ou nœud textuel
simple	uniquement des nœuds textuels
complexe	uniquement des sous éléments
mixte	mélange de sous éléments et de nœuds textuels

Le type des éléments

simple	: modèle de contenu simple et aucun attribut
complexe	: toutes les autres combinaisons

Remarque : Les attributs sont toujours de type simple (contenu uniquement textuel, ni sous éléments ni attributs)

Document XML de référence — fic1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bibliotheque xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="file:/home/
    nicolas/Cours/TechnoWeb/CM-XMLSchema
    /exemples/schema1.xsd">

<livre id="l0123456789" disponible="true">
  <isbn>0123456789</isbn>
  <titre lang="fr">La folle histoire d'ASI
    </titre>
  <type>Science Fiction</type>
  <auteur id="SC">
    <nom>Stéphane Canu</nom>
    <datenaissance>1960-09-08</
      datenaissance>
  </auteur>
  <personnage id="AR">
    <nom>Alain Rakotomamonjy</nom>
    <datenaissance>1970-07-09</
      datenaissance>
  </personnage>
  <personnage id="PL">
    <nom>Philippe Leray</nom>
    <datenaissance>1970-10-27</
      datenaissance>
  </personnage>
</livre>

<livre id="l9876543210" disponible="false">
  <isbn>9876543210</isbn>
  <titre lang="fr">Tout sur ma moto</titre
    >
  <type>Biographie</type>
  <auteur id="ND">
    <nom>Nicolas Delestre</nom>
    <datenaissance>1970-06-28</
      datenaissance>
  </auteur>
  <personnage id="EB">
    <nom>Edouard Bracame</nom>
  </personnage>
  <personnage id="PP">
    <nom>Paul Posichon</nom>
  </personnage>
</livre>

</bibliotheque>
```

Définition du XML Schema

Types simples

```
<xs:element name="type" type="xs:string"/>
<xs:element name="nom" type="xs:string"/>
<xs:element name="datenaissance" type="xs:date"/>
<xs:element name="datedeces" type="xs:date"/>
<xs:element name="isbn" type="xs:integer"/>
<xs:attribute name="id" type="xs:ID"/>
<xs:attribute name="disponible" type="xs:boolean"/>
<xs:attribute name="lang" type="xs:language"/>
```

- Les types simples sont les mêmes pour les attributs et les éléments
  `<xs:element name="isbn" type="xs:integer"/>`
  `<xs:attribut name="isbn" type="xs:integer"/>`
- L'ordre d'apparition des définitions n'a aucune importance

Type complexe (sequence)

```
<xs:element name="auteur">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="nom" />
      <xs:element ref="datenaissance" />
      <xs:element minOccurs="0"
                  ref="datedeces" />
    </xs:sequence>
    <xs:attribute ref="id" />
  </xs:complexType>
</xs:element>
```

Type complexe (extension)

```
<xs:element name="titre">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="lang" />
       </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Listing : `schema1.xsd`

Structure plate / Structure en profondeur

Plusieurs structures sont possibles pour un même modèle

- Structure plate
  Toutes les définitions sont situées au niveau global (`schema1.xsd`)
  $\Rightarrow$ plus de modularité
- Structure en profondeur
  Les définitions sont situées localement
  $\Rightarrow$ Définition d'éléments de même nom avec des structures locales différentes

En pratique il existe un juste équilibre entre les deux

Structure plate — schema1.xsd

```
<xs:element name="bibliotheque">
    <xs:complexType>
        <xs:sequence>
            <xs:element maxOccurs="unbounded" ref="livre"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="livre">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="isbn"/>
            <xs:element ref="titre"/>
            <xs:element ref="type"/>
            <xs:element minOccurs="0" ref="auteur"/>
            <xs:element maxOccurs="unbounded" minOccurs="0"
                        ref="personnage"/>
        </xs:sequence>
        <xs:attribute ref="id"/>
        <xs:attribute ref="disponible"/>
    </xs:complexType>
</xs:element>
```

Structure en profondeur — schema2.xsd

```
<xs:element name="bibliotheque">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="livre">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="isbn"/>
            <xs:element ref="titre"/>
            <xs:element ref="type"/>
            <xs:element minOccurs="0" ref="auteur"/>
            <xs:element maxOccurs="unbounded" minOccurs="0"
                        ref="personnage"/>
          </xs:sequence>
          <xs:attribute ref="id"/>
          <xs:attribute ref="disponible"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Type simple / Type complexe

Definition (Type simple)

correspond à la spécification de l'espace des valeurs d'un élément terminal ou d'un attribut

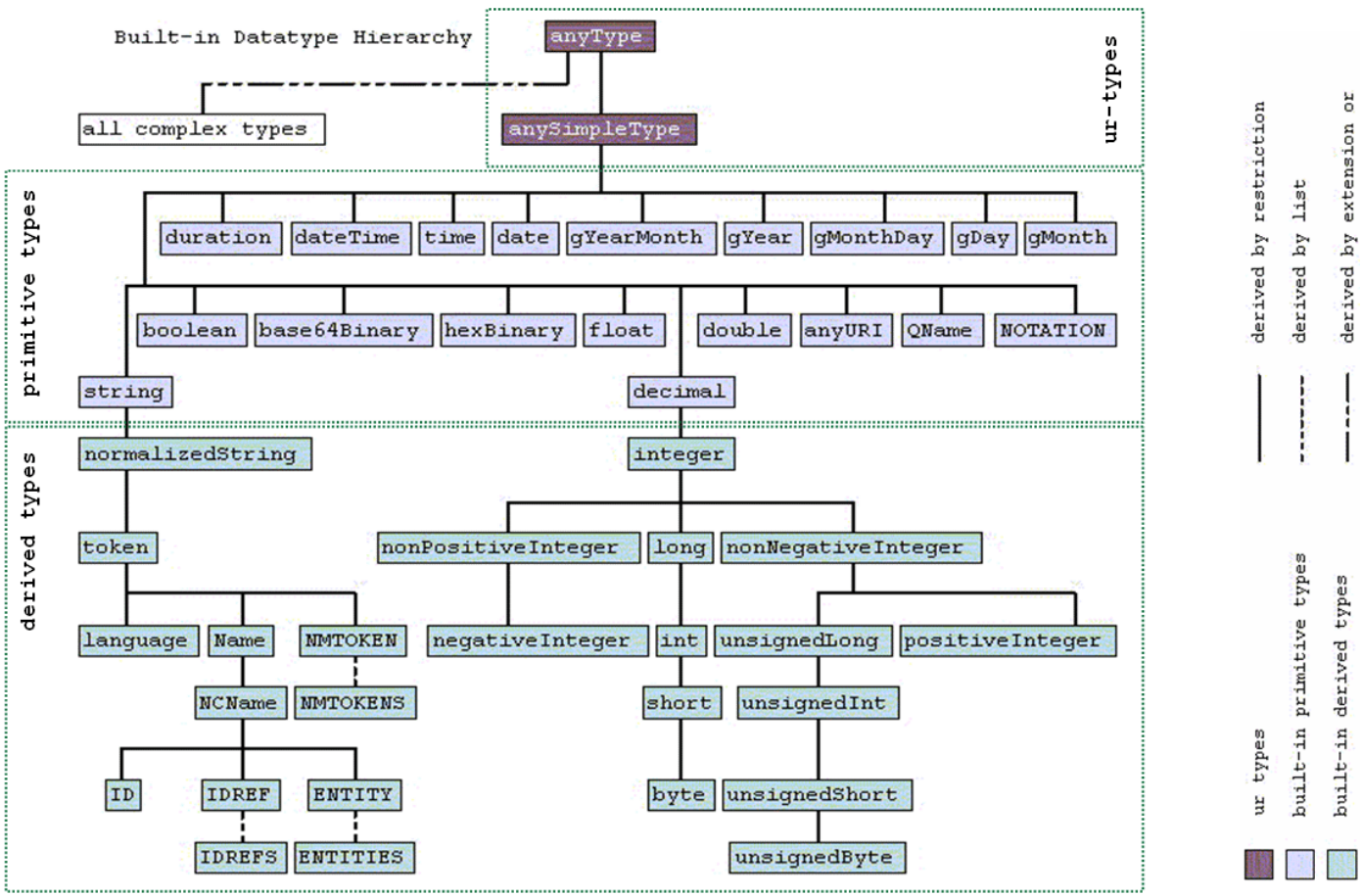
Definition (Type complexe)

correspond à la spécification du modèle de contenu (structure et attributs)

Types prédéfinis

La recommandation XML Schema du W3C fournit des types prédéfinis

- types de base (primitifs)
- types dérivés à partir des types de base



http://dret.net/lectures/xml-fall11/img/xsd-type-hierarchy.gif

Espace lexical et Espace des valeurs

Passage d'une valeur par quatres espaces :

Espace physique	texte brut encodé dans le fichier
Espace normalisé	conversion en Unicode, normalisation des fins de lignes (nœuds textuels) et des séparateurs (attributs) ⇒*tabulation, retour chariot et nouvelle ligne remplacés par un espace*
Espace lexical	compactage des caractères d'espacement (non systématique) ⇒*suppression des espaces de tête, de queue et remplacement de plusieurs espaces consécutifs par un seul espace*
Espace des valeurs	valeur après la prise en compte du type

Exemple :

une valeur / plusieurs représentations lexicales

1.5 , 01.5, 1.50000, . . .

Ces formes lexicales sont égales si de type `xs:float`, mais différentes si de type `xs:string`

Quelques types simples 1 / 3

- Chaînes de caractères

`xs:string`	non normalisé
`xs:normalizedString`	normalisé mais non compacté
tous les autres	normalisés et compactés

  - unité lexicale : `xs:token` (compacté de `xs:normalizedString`), `xs:NMTOKEN` (sans blanc), `xs:ID`, `XS:IDREF`, ...
  - format binaire : `xs:hexBinary`, `xs:base64Binary`
  - URI : `xs:anyURI`
  - ...

Quelques types simples 2 / 3

- Numériques
  `xs:decimal`, `xs:integer`, `xs:nonPositiveInteger`, `xs:negativeInteger`, `xs:long`, `xs:unsignedByte`, ...
- Date / Heure
  - `xs:dateTime` point dans le temps *2002-01-19T13 :50 :20+00 :00*
    toutes les données sont obligatoires
  - `xs:date` date *2003-10-30+02 :00*, *2004-02-26*
  - `xs:time` heure *19 :55 :00+02 :00*, *2004-02-26*
  - `xs:gYearMonth` mois *2003-10+01 :00*, *-1050-08*
  - `xs:gYear` année *2001+02 :00*, *-1000*
- Types liste : `xs:NMTOKENS`, `xs:IDREFS`, `xs:ENTITIES`

Quelques types simples 3 / 3

Modification des types simples schema3.xsd

```
<xs:element name="type" type="xs:token"/>
<xs:element name="nom" type="xs:token"/>
<!-- chaînes de caractères compactées -->
<xs:element name="datenaissance" type="xs:date"/>
<xs:element name="datedeces" type="xs:date"/>
<xs:element name="isbn" type="xs:NMTOKEN"/>
<!-- chaine de caractères sans blanc (9 chiffres + chiffre ou 'X') -->
<xs:attribute name="id" type="xs:ID"/>
<xs:attribute name="disponible" type="xs:boolean"/>
<xs:attribute name="lang" type="xs:language"/>
```

Création de types simples

Trois méthodes de dérivation :

- **restriction** ajout de contraintes sans changement de sens
- **liste** création d'une liste de valeurs d'un même type
- **union** mise en commun de plusieurs types, conservation de la sémantique commune

Syntaxe générale

```
<xs:simpleType id=ID name=NCName ... >
  (annotation? , (restriction | list | union))
</xs:simpleType>
```

Lors d'une définition globale (nommée) l'attribut `name` est obligatoire, il est inutile lors d'une définition locale (anonyme)

Dérivation par restriction

Ajout de nouvelles contraintes au type de base :

Syntaxe

```
<xs:simpleType name="nouveautype">
  <xs:restriction base="typedebase">
     (simpleType?, facettes*)
  </xs:restriction>
</xs:simpleType>
```

- Si non utilisation de `base` alors obligation de `simpleType`
- Possibilité d'ajouter de nouvelles restrictions à un type déjà restreint

Interdiction d'étendre le type de base lors d'une dérivation (par exemple : passer de 8 chiffres à 12)

Types et facettes 1 / 2

chaînes de caractères	xs :enumeration, xs :length, xs :maxLength, xs :minLength, xs :pattern, . . .
les nombres	`xs:enumeration`, `xs:maxExclusive`, `xs:minExclusive`, `xs:maxInclusive`, `xs:minInclusive`, `xs:pattern`, `xs:totalDigits` (entiers et dérivés), `xs:fractionDigits` (décimaux)
date/heure	`xs:enumeration`, `xs:maxExclusive`, `xs:minExclusive`, `xs:maxInclusive`, `xs:minInclusive`, `xs:pattern`

- La facette `xs:length` ne peut être changée si un de ses parents la possède déjà
- Facette fixée : attribut `fixed="true"`, cette facette ne pourra être réutilisée

Types et facettes 2 / 2

Exemple 1

```
<xs:simpleType name="entierlimite1">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1" />
    <xs:maxExclusive value="10" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="entierlimite2">
  <xs:restriction>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:maxExclusive value="10" />
      </xs:restriction>
    </xs:simpleType>
    <xs:minInclusive value="1" />
  </xs:restriction>
</xs:simpleType>
```

Exemple 2

```
<xs:simpleType name="floatpattern">
  <xs:restriction base="xs:float">
    <xs:pattern value="0\.[0-9]+[eE]\-?[0-9]+"
        />
  <xs:restriction>
</xs:simpleType>
```

Exemple 3

```
<xs:simpleType name="floatpatternenum">
  <xs:restriction base="floatpattern">
    <xs:enumeration value="0.5e0" />
    <xs:enumeration value="0.3e-4" />
    <xs:enumeration value="0.314159E1" />
  </xs:restriction>
</xs:simpleType>
```

Dérivation par liste 1 / 3

Obtention d'un type liste à partir d'un type simple :

Syntaxe

```
<xs:simpleType name="nouveautype">
  <xs:list itemType="typedebase">
    (xs:simpleType?)
  </xs:list>
</xs:simpleType>
```

Dérivation par restriction d'une liste, les facettes disponibles sont : `xs:length`, `xs:maxLength`, `xs:minLength`, `xs:enumeration`, `xs:whiteSpace`

Interdiction de faire des listes de listes

Dérivation par liste 2 / 3

Exemple 1

```
<xs:simpleType name="listefloatpattern">
  <xs:list itemType="floatpattern" />
</xs:simpleType>
```

Exemple 2

```
<xs:simpleType name="listeentiers">
    <xs:list>
        <xs:simpleType>
            <xs:restriction base="xs:integer">
                <xs:totalDigits value="3" />
            </xs:restriction>
        </xs:simpleType>
    </xs:list>
</xs:simpleType>
```

Dérivation par liste 3 / 3

Exemple 3

```
<xs:simpleType name="listedixentiers">
    <xs:restriction base="listeentiers">
        <xs:maxLength value="10"></xs:maxLength>
    </xs:restriction>
</xs:simpleType>
```

Dérivation par l'union 1 / 2

Création d'un type par fusion d'espaces lexicaux d'autres types :

Syntaxe

```
<xs:simpleType name="nouveautype">
  <xs:union memberTypes="liste de types">
    (xs:simpleType*)
  </xs:union>
</xs:simpleType>
```

Dérivation par l'union 2 / 2

Exemple

```
<xs:simpleType name="unionfloatinfini">
  <xs:union memberTypes="xs:float">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="+infini" />
        <xs:enumeration value="-infini" />
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

Attention à l'ordre des dérivations

ordreDesDerivations.xsd

```
<xs:simpleType name="ListeDUnions">
  <xs:list>
    <xs:simpleType>
      <xs:union memberTypes="xs:positiveInteger xs:negativeInteger" />
    </xs:simpleType>
  </xs:list>
</xs:simpleType>
<xs:simpleType name="UnionDeListes">
  <xs:union>
    <xs:simpleType>
      <xs:list itemType="xs:positiveInteger" />
    </xs:simpleType>
    <xs:simpleType>
      <xs:list itemType="xs:negativeInteger" />
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

ordreDesDerivations.xml

```
<tns:eltListeDUnions>
   1 2 3 -4
</tns:eltListeDUnions>
```

```
<tns:eltUnionDeListes>
   1 2 3 4
</tns:eltUnionDeListes>
```

```
<tns:eltUnionDeListes>
   -1 -2 3 -4
</tns:eltUnionDeListes>
<!-- ne valide pas -->
```

Modification du schéma de base — schema4.xsd

```
<xs:simpleType name="string255">
  <xs:restriction base="xs:token">
    <xs:maxLength value="255"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="string32">
  <xs:restriction base="xs:token">
    <xs:maxLength value="32"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="typelivre">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Biographie"/>
    <xs:enumeration value="Science_Fiction
        "/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="langues">
  <xs:restriction base="xs:language">
    <xs:enumeration value="en"/>
    <xs:enumeration value="fr"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="isbn">
    <xs:restriction base="xs:NMTOKEN">
        <xs:length value="10"/>
        <xs:pattern value="[0-9]\{9\}[0-9X]
            "/>
    </xs:restriction>
</xs:simpleType>
```

```
<xs:element name="type" type="typelivre"/>
<xs:element name="nom" type="string32"/>
<xs:element name="isbn" type="isbn"/>
<xs:attribute name="lang" type="langues"/>
```

Présentation

- Description de la structure du balisage
- Utilisation des types simples pour les éléments terminaux et les attributs

Rappel :

Modèle de contenu	mixte	complexe	simple	vide
un ou plusieurs éléments enfants	oui	oui	non	non
un ou plusieurs nœuds textuels enfants	oui	non	oui	non

Certains éléments de définition de types complexes sont les mêmes que pour les types simples, mais leur signification est différente

Création de types complexes 1 / 2

Contrairement aux types simples qui sont uniquement créés par dérivation, les types complexes peuvent être créés de zéro.

Syntaxe générale

```
<xs:complexType   abstract=boolean: false
                  block=(#all | List of (extension | restriction))
                  final=(#all | List of (extension | restriction))
                  id=ID
                  mixed=boolean: false
                  name=NCName>
  (annotation?,
    (simpleContent | complexContent |
      ( (group | all | choice | sequence)?,
        ((attribute | attributeGroup)*, anyAttribute?))))
</xs:complexType>
```

Lors d'une définition globale (nommée) l'attribut `name` est obligatoire, il est inutile lors d'une définition locale (anonyme)

Création de types complexes 2 / 2

abstract le type peut il être instancié ?

block empêche d'utiliser, à la place de ce type complexe, un type complexe ayant le type de dérivation spécifié :

- **extension** empêche des types complexes dérivés par extension d'être utilisés à la place de ce type complexe.
- **restriction** empêche des types complexes dérivés par restriction d'être utilisés à la place de ce type complexe.
- **#all** les deux

final empêche la dérivation du type (suivant le type de dérivation)

mixed le type est il mixte ?

Création de types complexes à contenu simple

Élément avec attribut et contenu simple (uniquement du texte)

Syntaxe

Extension par ajout d'attributs à un type simple

```
<xs:complexType>
  <xs:simpleContent>
    <xs:extension base="typedebase">
      (annotation?, ((attribute | attributeGroup)*, anyAttribute?))
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Dérivation par extension

L'extension de types complexes à modèle de contenu simple consiste simplement à ajouter des attributs

Exemple

```
<xs:element name="titre">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="chaineetlang">
        <xs:attribute name="note" type="xs:token"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Dérivation par restriction

Les restrictions portent sur le modèle de contenu et les attributs

Exemple type départ

```
<xs:complexType name="typeorigine">
  <xs:simpleContent>
    <xs:extension base="xs:token">
      <xs:attribute ref="lang"/>
      <xs:attribute name="note"
                    type="xs:integer"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Exemple type restreint

```
<xs:complexType name="typerestreint">
  <xs:simpleContent>
    <xs:restriction base="typeorigine">
      <xs:maxLength value="255"/>
      <xs:attribute name="lang">
        <xs:simpleType>
          <xs:restriction base="langues">
            <xs:enumeration value="fr"/>
            <xs:enumeration value="en"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="note"
                    use="prohibited"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

Syntaxe identique à la dérivation par restriction des types simples

Création de types complexes à contenu complexe

Élément à contenu complexe avec ou sans attribut

Nécessité de définir la liste des sous-éléments ainsi que leur ordre

Syntaxe

```
<xs:complexType>
  (sequence | choice | all)
</xs:complexType>
```

Définition

`sequence`, `choice`, `all` sont des connecteurs constitués de particules pouvant être `element`, `sequence`, `choice`, `any`, `group`

Les connecteurs et particules 1 / 2

Connecteurs

- `sequence` définit une liste ordonnée de particules
- `choice` définit le choix d'une particule dans un ensemble déclaré
- `all` définit une liste non ordonnée de particules

Particules

Définition des occurences de particules :

- `minOccurs` définit le nombre d'occurences minimum de la particule
- `maxOccurs` définit le nombre d'occurences maximum de la particule (valeur possible : `unbounded`)

Par défaut la valeur de ces attributs est de 1.

Les connecteurs et particules 2 / 2

schema5.xsd

```
<xs:element name="auteur">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element ref="nom" />
        <xs:sequence>
          <xs:element ref="prenom" />
          <xs:element ref="nom" />
          <xs:element ref="surnom" maxOccurs="2" />
        </xs:sequence>
      </xs:choice>
      <xs:element ref="datenaissance" />
      <xs:element minOccurs="0" ref="datedeces" />
    </xs:sequence>
    <xs:attribute ref="id" />
  </xs:complexType>
</xs:element>
```

Groupes d'éléments et d'attributs 1 / 2

Groupe

Les groupes sont des structures étiquetées d'éléments et d'attributs utilisables par simple référence dans des définitions.

Syntaxe

La définition des groupes doit être globale (directement sous xs :schema).

- groupe d'éléments

```
<xs:group name=NCName id=ID maxOccurs="valeurmax" minOccurs="
    valeurmin" ref=QName>
  (annotation?, (all | choice | sequence)?)
</xs:group>
```

- groupe d'attributs

```
<xs:attributeGroup id=ID name=NCName ref=QName>
    (annotation?, ((attribute | attributeGroup)*,
        anyAttribute?))
</xs:attributeGroup>
```

Groupes d'éléments et d'attributs 2 / 2

schema6.xsd

```
<xs:group name="nom">
  <xs:choice>
    <xs:element ref="nom" />
    <xs:sequence>
      <xs:element ref="prenom" />
      <xs:element ref="nom" />
      <xs:element ref="surnom"
                   maxOccurs="2" />
    </xs:sequence>
  </xs:choice>
</xs:group>


<xs:element name="auteur">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="nom" />
      <xs:element ref="datenaissance" /
           >
      <xs:element minOccurs="0"
                   ref="datedeces" />
    </xs:sequence>
    <xs:attribute ref="id" />
  </xs:complexType>
</xs:element>
```

```
<xs:attributeGroup name="attributsDeLivre">
  <xs:attribute ref="id" />
  <xs:attribute ref="disponible" />
  <xs:attribute name="agemini" use="optional"
       >
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minExclusive value="0" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:attributeGroup>

<xs:element name="livre">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="isbn" />
      <xs:element ref="titre" />
      <xs:element ref="type" />
      <xs:element minOccurs="0"
                   ref="auteur" />
      <xs:element maxOccurs="unbounded"
                   minOccurs="0"
                   ref="personnage" />
    </xs:sequence>
    <xs:attributeGroup ref="attributsDeLivre"
          />
  </xs:complexType>
</xs:element>
```

Règle d'attribution de la particule unique

Définition

Un parseur doit pouvoir déterminer si un document XML est valide, simplement si en le parcourant pour chaque nœud il peut vérifier sa validité sans aller voir en avant.

Exemple qui pose problème

```
<xs:group name="nom">
  <xs:choice>
    <xs:element ref="nom" />
    <xs:sequence>
      <xs:element ref="nom" />
      <xs:element ref="prenom" />
      <xs:element ref="surnom"
                  maxOccurs="2" />
    </xs:sequence>
  </xs:choice>
</xs:group>
```

Solution

```
<xs:group name="nom">
  <xs:sequence>
    <xs:element ref="nom" />
    <xs:sequence minOccurs="0">
      <xs:element ref="prenom" />
      <xs:element ref="surnom"
                  maxOccurs="2" />
    </xs:sequence>
  </xs:sequence>
</xs:group>
```

Mais aussi la solution donnée dans l'exemple précédent

Règle de déclaration cohérente

Définition

Cette règle interdit les choix entre des éléments de même nom ayant des types différents.

Exemple qui pose problème

```
<xs:element name="eltTestDeclarationCoherente">
  <xs:complexType>
    <xs:choice>
      <xs:element name="nom" type="xs:string" />
      <xs:element name="nom">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="nomdefamille" type="xs:string" />
            <xs:element name="prenom" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Le connecteur xs:all

Utilisation

xs:all permet la définition d'un modèle de contenu non ordonné. Seules les particules xs:element sont autorisées avec la contrainte d'occurence 0 ou 1

auteur devient ...

```
<xs:element name="auteur">
  <xs:complexType>
    <xs:all>
      <xs:element ref="nom" />
      <xs:element ref="datenaissance" />
      <xs:element minOccurs="0" ref="datedeces" />
    </xs:all>
    <xs:attribute ref="id" />
  </xs:complexType>
</xs:element>
```

xs:sequence qui ne peut pas être remplacé par xs:all

```
<xs:element name="livre">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="isbn" />
      <xs:element ref="titre" />
      <xs:element ref="type" />
      <xs:element minOccurs="0" ref="auteur" />
      <xs:element ref="personnage" minOccurs="0"
                                   maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attributeGroup ref="attributsDeLivre" />
  </xs:complexType>
</xs:element>
```

Solution : changer la structure — schema6toxsall.xsd

```
<xs:element name="livre">
  <xs:complexType>
    <xs:all>
      <xs:element ref="isbn" />
      <xs:element ref="titre" />
      <xs:element ref="type" />
      <xs:element minOccurs="0" ref="auteur" />
      <xs:element name="personnages">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="personnage" minOccurs="0"
                                         maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:all>
   <xs:attributeGroup ref="attributsDeLivre" />
  </xs:complexType>
</xs:element>
```

Dérivation par extension

Principe

Ajouter des attributs et/ou des éléments

Syntaxe

```
<xs:complexContent>
  <xs:extension base="typedebase">
    (annotation?, ((group | all | choice | sequence)?,
                   ((attribute | attributeGroup)*, anyAttribute?)))
  </xs:extension>
</xs:complexContent>
```

Attention

Le principe d'extension est très restreint, on ne peut qu'ajouter des éléments à la suite des éléments du type de base.

schema7.xsd

```
<xs:complexType name="caracteristiquesPersonne">
  <xs:sequence>
    <xs:group ref="nom" />
    <xs:element ref="datenaissance" />
  </xs:sequence>
  <xs:attribute ref="id" />
</xs:complexType>

<xs:element name="auteur">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="caracteristiquesPersonne">
        <xs:sequence>
          <xs:element minOccurs="0" ref="datedeces" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

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

Dérivation par restriction

Principe

La dérivation par restriction consiste à diminuer le nombre d'instances conformes au modèle de base.

Attention

La dérivation doit définir entièrement le nouveau type et être compatible avec le type dérivé

Syntaxe

```
<xs:complexContent>
  <xs:restriction base="typdebase">
    (annotation?, (group | all | choice | sequence)?,
                  ((attribute | attributeGroup)*, anyAttribute?))
  </xs:restriction>
</xs:complexContent>
```

schema8.xsd

```
<xs:complexType name="personne">
  <xs:sequence>
    <xs:group ref="nom" />
    <xs:element minOccurs="0" ref="datenaissance" />
    <xs:element minOccurs="0" ref="datedeces" />
  </xs:sequence>
  <xs:attribute ref="id" />
</xs:complexType>
```

```
<xs:element name="personnage">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="personne">
        <xs:sequence>
          <xs:group ref="nom" />
        </xs:sequence>
        <xs:attribute ref="id" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="auteur">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="personne">
        <xs:sequence>
          <xs:group ref="nom" />
          <xs:element ref="datenaissance" />
        </xs:sequence>
        <xs:attribute ref="id" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Création de types complexes à contenu mixte

Élément contenant des éléments et du texte

Équivalent à un type complexe auquel on ajoute du texte

Syntaxe

Ajout de l'attribut `mixed="true"` à `<xs:complexType>`

Remarque

Il n'y a aucun contrôle sur le contenu textuel pouvant s'insérer partout.

Méthode de dérivation

La dérivation d'un modèle de contenu mixte est identique à celle d'un modèle de contenu complexe

Dérivation

Dérivation par extension : modèle mixte $\leftrightarrows$ modèle complexe

- complexe $\rightarrow$ mixte : interdit
  l'extension ne doit ajouter des éléments qu'après la structure, or elle autoriserait du texte dans la structure de base (non modifiable).
- mixte $\rightarrow$ complexe : interdit
  idem, les nœuds textuels deviendraient interdits dans le type de base.

Dérivation par restriction : modèle mixte $\leftrightarrows$ modèle complexe

- complexe $\rightarrow$ mixte : interdit
  le type ne serait plus une restriction, mais une extension puisqu'autorisant les nœuds textuels.
- mixte $\rightarrow$ complexe : autorisé
  cette dérivation permet d'interdire le contenu textuel (`mixed="false"`), ou encore, par exemple, d'interdire tous les éléments pour n'autoriser que le texte.

Création de types complexes à contenu vide

Élément n'ayant que des attributs

Équivalent à un élément avec un modèle complexe sans élément, ou avec un modèle simple contenant une chaîne de caractères de longueur 0.

Exemples

```
<xs:simpleType name="typeVideSimple">
  <xs:restriction base="xs:string">
    <xs:enumeration value="" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="typeVideComplexe" />

<xs:element name="eltVideSimple" type="typeVideSimple" />
<xs:element name="eltVideComplexe" type="typeVideComplexe" />
```

Déclaration d'une XSD

Déclarer l'utilisation du XMLSchema de XMLSchema

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

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

Utilisation d'un espace de noms

Principe de l'espace de noms

Associer des éléments et des attributs à des URI.
Les éléments de même nom avec des URIs différentes sont différents.
Permet aux documents XML d'inclure différents schémas tout en réglementant l'utilisation des balises.

Inclusion d'éléments SVG ou MATHML dans un document HTML.

Syntaxe : l'attribut `xmlns`

- `xmlns="URI"` tout élément non préfixé appartient à cet espace de noms par défaut
- `xmlns:prefixe="URI"` tout élément ou attribut prefixé appartient à cet espace de noms

Les attributs ne connaissent pas l'espace de noms par défaut
$\Rightarrow$ un attribut non préfixé n'a pas d'espace de noms

Emplacement des schémas 1 / 2

xsi:noNamespaceSchemaLocation

Généralement spécifié dans l'élement racine du document XML, il fournit l'URL du schema par défaut lorsque les espaces de noms ne sont pas utilisés.

Exemple

```
<bibliotheque  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="file:/home/nicolas/Cours/.../schema1.xsd">
...
</bibliotheque>
```

Emplacement des schémas 2 / 2

xsi:schemaLocation

Permet par des couples URI/URL de spécifier pour les espaces de noms où trouver les schémas correspondant.

Exemple

```
<bibliotheque   xmlns="http://ma.biblio.org/"
                xmlns:html="http://www.w3.org/1999/xhtml"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://ma.biblio.org/  file:/home/nicolas/mabiblio.xsd
                       http://www.w3.org/1999/xhtml  http://www.w3.org/1999/xhtml.xsd">
 ...
</bibliotheque>
```

Ce que nous n'avons pas vu...

- déclaration d'un espace de nom (targetNamespace)
- inclusion et redéfinition de schémas
- les clés/références
- fonctionnalités orientées objets (substitution et contrôle de dérivation)
- ...

En résumé

- Préciser quelle doit être la racine du document ✓ (via les groupes)
- Préciser le type d'un attribut ou d'un élément ✓
- Spécifier précisément l'arité d'un élément ✓
- Préciser différemment la nature d'un élément en fonction de sa position dans le fichier XML ✓ ✗
- Utiliser la grammaire en tant que document XML (utilisable en entrée d'un processeur XSL) ✓
- Conditionner la valeur d'un nœud text ✗