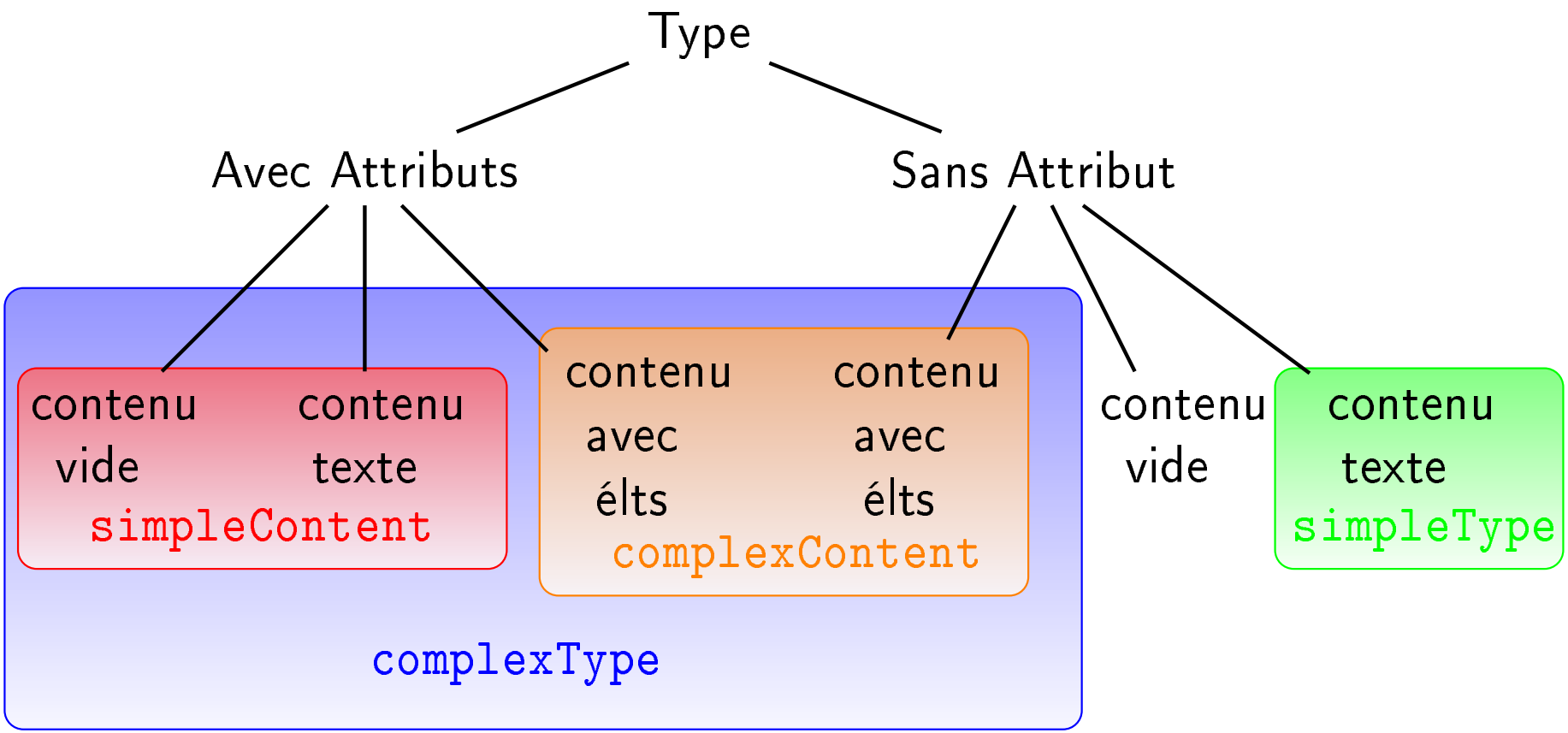
XML Schema (2)

XML, un langage d'arbres

Master Pro ILI - Master recherche SIA

Année 2013-14

Types et modèles de contenus



Dérivation de types simples

Définition de nouveaux types simples à partir de types prédéfinis

- pour des attributs
- ou pour des éléments à contenu simple.

Il existe 3 méthodes pour dériver un nouveau type simple

- la restriction (sur les valeurs)
- la liste : pour créer des listes de valeurs d'un même type
- l'union de plusieurs types simples déjà existants

Dérivation de types simples

- Un nouveau type simple se définit à l'intérieur d'une balise `<xs:simpleType>`.
- définition globale : la balise peut se trouver à un niveau haut et porter un attribut `name`.
  Le type est ainsi nommé et réutilisable.
- définition locale (sur un élément ou un attribut) : on parle d'un type anonyme.

Dérivation par restriction

La dérivation par restriction :

- ajout de contraintes sur un type de base ;
- type de base défini
  - soit à l'aide de l'attribut `base` de la balise `<xs:restriction>` ;
  - soit défini localement par une balise `<simpleType>` en première position de la balise `<xs:restriction>`.

Dérivation par restriction

```
<xs:simpleType name="mesEntiers">
  <xs:restriction base="positiveInteger">
    <xs:minInclusive value="1" />
    <xs:maxInclusive value="10" />
  </xs:restriction>
</xs:simpleType>
<xs:element name="enfant">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nom" type="token" />
      <xs:element name="prenom" type="token" />
      <xs:element name="age" type="mesEntiers" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Dérivation par restriction

Ajout de contraintes sur le type de base :

- on appelle les contraintes des facettes ;
- les facettes sont définies pour chaque type de base.

  On n'applique pas les même facettes sur les types de chaînes de caractères et sur les types numériques.

Facettes : token

Facettes disponibles pour des chaînes de caractères compactées :

- `xs:length` pour spécifier la longueur en nombre de caractères (pas très utile, en général on préfère les facettes suivantes). Suivant le type de chaînes de caractères, il peut s'agir d'une longueur exprimée en octets ;
- `xs:maxLength` ou `xs:minLength`

```
<xs:simpleType name="chaineNonVide">
  <xs:restriction base="token">
    <xs:minLength value="1" />
  </xs:restriction>
</xs:simpleType>
```

Facettes : token

- `xs:enumeration` avec l'attribut `value` pour énumérer la liste des valeurs possibles ;

```
<xs:simpleType name="type_transport">
  <xs:restriction base="token">
    <xs:enumeration value="vélo" />
    <xs:enumeration value="métro" />
    <xs:enumeration value="bus" />
  </xs:restriction>
</xs:simpleType>
<xs:element name="transport">
  <xs:complexType>
    <xs:attribute name="modeTransport"
        type="type_transport" />
  </xs:complexType>
</xs:element>
```

Facettes : flottants et dates

Facettes disponibles sur des nombres flottants, ou des dates :

- `xs:enumeration`
- `xs:maxExclusive` et `minExclusive`
- `xs:maxInclusive` et `minInclusive`

```
<xs:simpleType name="auMoinsCent">
  <xs:restriction base="integer">
    <xs:minInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Facettes : integer

Facettes disponibles sur des entiers et types dérivés :

- les mêmes que précédemment, plus
- `xs:totalDigits` : nombre maximal de chiffres

```
<xs:simpleType name="tel">
  <xs:restriction base="nonNegativeInteger">
    <xs:totalDigits value="10" />
  </xs:restriction>
</xs:simpleType>
```

Facettes : decimal

Facettes disponibles sur des décimaux :

- les mêmes que précédemment, plus
- `xs:fractionDigits` : nombre maximal de chiffres après la virgule

```
<xs:simpleType name="noteSur20">
  <xs:restriction base="decimal">
    <xs:totalDigits value="4" />
    <xs:fractionDigits value="2" />
    <xs:maxInclusive value="20" />
    <xs:minInclusive value="0" />
  </xs:restriction>
</xs:simpleType>
```

Dérivation par restriction en chaîne

- les facettes doivent être de plus en plus restrictives
- une seule facette ne peut pas être redéfinie : `xs:length`
- on peut empêcher de modifier une facette par dérivation successive en appliquant sur la facette l'attribut `fixed="true"`
- `xs:enumeration` et `xs:pattern` ne peuvent pas être fixés.

Exemple

```
<xs:simpleType name="noteSur20">
  <xs:restriction base="decimal">
    <xs:totalDigits value="4" fixed="true" />
    <xs:fractionDigits value="2" fixed="true" />
    <xs:maxInclusive value="20" />
    <xs:minInclusive value="0" fixed="true" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="noteSur10">
  <xs:restriction base="noteSur20">
    <xs:maxInclusive value="10" />
  </xs:restriction>
</xs:simpleType>
```

Cherchez l'erreur

```
<xs:simpleType name="noteSur30">
  <xs:restriction base="noteSur20">
    <xs:maxInclusive value="30" />
  </xs:restriction>
</xs:simpleType>
```

Cherchez l'erreur

```
<xs:simpleType name="noteSur30">
  <xs:restriction base="noteSur20">
    <xs:maxInclusive value="30" />
  </xs:restriction>
</xs:simpleType>
```

Ce n'est pas une restriction de passer d'un intervalle [0,20] à [0,30].
Cette dérivation n'est donc pas autorisée.

La facette whiteSpace

Définit la politique de gestion des espaces :

- peut prendre l'une des valeurs : preserve, replace, ou collapse
- est utilisée pour qualifier `string` (valeur preserve)
- puis par restriction `normalizedString` (valeur replace) : le processeur XML remplacera les fins de ligne, retours chariot et tabulations par des espaces
- puis toujours par restriction `token` (valeur collapse) : idem + le processeur XML supprimera les espaces consécutifs et les espaces de début et de fin de chaîne.

La facette pattern

Contraint une valeur à respecter une expression régulière

- les expressions régulières sont exprimées comme en Perl
- mais n'utilisent pas les caractères $ et ^ de début et fin de ligne
- ^ permet de définir des exclusions

```
<xs:simpleType name='better-us-zipcode'>
  <xs:restriction base='string'>
    <xs:pattern value='[0-9]{5}(-[0-9]{4})?'/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="UKPostcode">
  <xs:restriction base="token">
    <xs:pattern value="[A-Z]{2}\d\s\d[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
```

Quelques jokers Perl

Quelques rappels...

- `.` un caractère (quelconque)
- `\d` équivalent à [0-9]
- `\D` équivalent à ^ [0-9] (tout sauf un digit)
- `\S` non espace ou tabulation
- `\w` un mot (caractères alphanumériques et '`_`')
- `\W` pas un mot
- `\n` nouvelle ligne
- `\r` retour chariot
- `\t` tabulation
- `\s` espace, tabulation

La facette pattern

- si plusieurs facettes `pattern` sont posées sur un même type, alors elles forment une disjonction ;
- si plusieurs facettes `pattern` sont posées successivement par dérivation, alors elles s'appliquent successivement (elles forment une conjonction)

Dérivation par listes

- définit une liste d'items séparés par des espaces.
- balise `<xs:list>` avec l'attribut `itemType`.
  exemples :

```
<xs:simpleType name="listeEntiers">
  <xs:list itemType="xs:integer"/>
</xs:simpleType>
<xs:simpleType name="maListe">
  <xs:list itemType="monType"/>
</xs:simpleType>
```

- comment définiriez-vous une liste de token de taille 10 (n'accepte pas plus de 10 mots) ?

Dérivation par listes

```
<xs:simpleType name="listeDeDixTokens">
  <xs:restriction>
    <xs:simpleType> <!-- définition type de base -->
      <xs:list itemType="xs:NMTOKEN" />
    </xs:simpleType>
    <xs:length value="10" /> <!-- restriction -->
  </xs:restriction>
</xs:simpleType>

<xs:element name="liste10Tok" type="listeDeDixTokens" />
```

Dérivation par union

- l'union des espaces lexicaux de plusieurs types
- balise `xs:union` avec l'attribut `memberTypes` qui contient
  - la liste des types séparés par des blancs
  - ou des définitions locales à l'intérieur de `xs:union`
  - ou bien un mélange des deux.
- la sémantique du nouveau type est donnée par l'intersection des sémantiques
- on ne peut plus dériver un tel type par restriction qu'à l'aide des facettes communes.

Exemple

```
<xs:simpleType name="noteOuLettre">
    <xs:union memberTypes="noteSur20">
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:enumeration value="A" />
          <xs:enumeration value="B" />
          <xs:enumeration value="C" />
          <xs:enumeration value="D" />
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
</xs:simpleType>
```

Dérivation de type complexe

. . .à contenu simple à partir d'un type simple

- La seule dérivation possible est d'ajouter des attributs
- cela s'appelle une dérivation par extension
- les dérivations par extension s'appliquent uniquement aux types complexes
- elles ont une sémantique différente suivant que la dérivation s'applique aux contenus simples ou complexes.

Dérivation de type complexe

Exemple :

```
<xs:complexType name="noteSur20AMentionLibre">
  <xs:simpleContent>
    <xs:extension base="noteSur20">
      <xs:attribute name="mention" type="token" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Remarque : le type de base de l'extension est obligatoirement défini dans l'attribut base, il ne peut pas être défini localement.

Dérivation de type complexe

. . .à contenu simple à partir d'un autre type complexe à contenu simple

Dérivation par extension : le type `note` étant toujours à contenu simple, il peut être dérivé par extension de la même manière que `xs:noteSur20` (on ajoute de nouveaux attributs).

Dérivation par restriction :

- sur le contenu : ajout de facettes (comme pour un type simple)
- suppression d'attribut(s) : on pose l'attribut `use="prohibited"` sur le ou les attribut(s) qu'on veut exclure
- restriction du type d'un attribut (ajout de facettes).

On peut conjuguer ces trois restrictions.

Exemple

```
<xs:complexType name="noteSur10AMention">
  <xs:simpleContent>
    <xs:restriction base="noteSur20AMentionLibre">
      <xs:maxInclusive value="10" />
      <xs:attribute name="mention">
        <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:enumeration value="TBien" />
          <xs:enumeration value="Bien" />
          <xs:enumeration value="ABien" />
          <xs:enumeration value="Passable" />
        </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

Dérivation de type complexe

Dérivation par extension d'un type complexe à contenu complexe à partir d'un type complexe

Similaire aux cas précédents, mais sémantique différente :

- soit on veut dériver un nouveau type complexe en lui ajoutant des attributs $\Rightarrow$ on revient au cas précédent ;
- soit on veut dériver par extension pour enrichir la structure de nouveaux fils. On arrive ici dans des contraintes fortes.
  Dériver par extension, c'est :
  - inclure dans une séquence le type de base,
  - ajouter à sa suite son extension.

  Les contraintes :
  - les nouveaux sous-éléments sont à la fin,
  - ils sont reliés au type de base par une séquence.

Dérivation de type complexe

```
<xs:complexType name="Tetape">
  <xs:sequence>
    <xs:element ref="periode" minOccurs="0"/>
    <xs:element ref="titre"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="suite">
  <xs:complexContent>
    <xs:extension base="Tetape">
      <xs:sequence>
        <xs:element ref="descriptif" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Dérivation de type complexe

Une instance de ce type devra avoir pour éléments fils :

```
<periode>...</periode>  (optionnel)
<titre>...</titre>
<descriptif>...</descriptif>  (optionnel)
```

Dérivation de type complexe (2)

```
<xs:complexType name="Tetape">
  <xs:choice>
    <xs:element ref="periode"/>
    <xs:element ref="titre"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="suite">
  <xs:complexContent>
    <xs:extension base="Tetape">
      <xs:choice>
        <xs:element ref="descriptif"/>
        <xs:element ref="contenu"/>
      </xs:choice>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Dérivation de type complexe (2)

Une instance de ce type devra avoir pour éléments fils l'une de ces quatre structures :

```
<periode> ...
</periode>
<descriptif> ...
</descriptif>

<titre> ...
</titre>
<contenu> ...
<contenu>
```

```
<titre> ...
</titre>
<descriptif> ...
</descriptif>

<periode> ...
</periode>
<contenu> ...
</contenu>
```

Dérivation de type complexe

Attention au cas de `xs:all` :

Compte-tenu des restrictions qui s'imposent quant à son utilisation, on ne peut pas dériver par extension un élément qui a en racine un connecteur `xs:all`.

Dérivation de type complexe

Dérivation par restriction d'un type complexe à contenu complexe à partir d'un type complexe

- reprendre la définition du type de base en enlevant les parties dont on ne veut plus.
- n'a d'intérêt que pour définir une hiérarchie, et dans le cadre de la redéfinition de types.
- Attention : un modèle défini par restriction doit être conforme à son type de base.
  Cela veut dire par exemple qu'on ne peut enlever que des sous-éléments optionnels.

Définir des identifiants(1)

En XML-Schema, on définit un identifiant par l'élément `xs:key`. Cet élément doit être posé dans un élément qui contiendra l'identifiant.

```
<xs:element name="oiseaux">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="fiche-observations"
                  type="TFiche"/>
      <!- le type TFiche indique notamment
      la présence d'un attribut id-obs ->
    </xs:sequence>
  </xs:complexType>
  <!- ICI ->
```

Définir des identifiants(2)

```
    <!- définition de l'identifiant ->
    <xs:key name="numFiche">
      <!- les éléments fiche-observations qui ->
      <!- se trouvent en-dessous de oiseaux ->
      <xs:selector xpath="./fiche-observations"/>
      <!- portent un attribut (@) de nom id-obs ->
      <!- et cet attribut est un identifiant ->
      <xs:field xpath="@id-obs"/>
    </xs:key>
</xs:element>
```

Définir des identifiants(3)

Et pour y faire référence, on utilise l'élément `xs:keyref`.
Maintenant, pour chaque espèce observée, on trouve un élément `aussi-dans` qui indique dans quelles autres observations on peut trouver cet oiseau.

```
    <!- à la suite de la clé->
    <xs:key name="numFiche">
      <xs:selector xpath="./fiche-observations"/>
      <xs:field xpath="@id-obs"/>
    </xs:key>
    <!- c'est ici qu'on définit qui va référencer numFiche -
    <xs:keyref name="refFiche" refer="numFiche">
      <!- ce sera dans l'élément aussi-dans ->
      <xs:selector  xpath=".//aussi-dans" />
      <!- et plus particulièrement l'attribut (@) ref-obs ->
      <xs:field xpath="@ref-obs" />
    </xs:keyref>
  </xs:element>
```

Définir des identifiants(4)

Ce qui donne (avec un ajout de l'élément `aussi-dans` à l'élément `espece`) :

```
<fiche-observations auteur="O.Peterson" id-obs="obs2">
<lieu>Painchonvalle</lieu>
<date>2006-12-02</date>
...
<espece nom="grive mauvis" nbre="20">
  <famille>Turdidés</famille>
  <description>en migration probablement.</description>
  <aussi-dans ref-obs="obs1" />
  <aussi-dans ref-obs="obs2" />
  <aussi-dans ref-obs="obs4" />
</espece>
...
</fiche-observations>
```