

Types dérivés



Déf et réutilisation



Validation



Type Abstrait



Modul



Exp Ratio



Université Mohammed Premier
École Nationale des Sciences Appliquées
Oujda



Chapitre 3 : Schéma (XSD)

8 avril 2021

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

Plan du cours

- ➊ Dérivée des types simples
- ➋ Définition Local/globale et la réutilisation
- ➌ Validation
- ➍ Type Abstrait
- ➎ Modularité
- ➏ Expressions rationnelles

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

List

- L'opérateur xsd :list définit un nouveau type simple dont les valeurs sont les listes de valeurs du type simple donné par l'attribut itemType.
 - La liste de valeur est séparée par des espaces.
 - Les valeurs du type simple donné par itemType ne peuvent pas comporter de caractères d'espacement.

```
<!-- Type pour les listes d'entiers -->
<xsd:simpleType name="IntList">
    <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
<!-- Type pour les listes de 5 entiers -->
<xsd:simpleType name="IntList5">
    <xsd:restriction base="IntList">
        <xsd:length value="5"/>
    </xsd:restriction>
</xsd:simpleType>
```

Union

- L'opérateur xsd :union définit un nouveau type simple dont les valeurs sont celles des types listés dans l'attribut memberTypes.

```
<xsd:attribute name="maxOccurs" type="IntegerOrUnbounded"/>
<xsd:simpleType name="IntegerOrUnbounded">
    <xsd:union memberTypes="xsd:nonNegativeInteger Unbounded"/>
</xsd:simpleType>
<xsd:simpleType name="Unbounded">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="unbounded"/>
    </xsd:restriction>
</xsd:simpleType>
```

Union

```
<xsd:simpleType name="IntegerOrUnbounded">
    <xsd:union memberTypes="xsd:nonNegativeInteger">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="unbounded"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

Extension

- L'idée générale de l'extension est de **rajouter du contenu et des attributs**.
- L'extension s'applique aux **types simples** et aux **types complexes** mais elle donne toujours un **type complexe**.
- L'extension d'un type est introduite par l'élément **xsd :extension** dont l'attribut **base** donne le nom du type de base.
- Le contenu de l'élément **xsd :extension** explicite le contenu et les attributs à ajouter au type de base.
- L'élément **xsd :extension** est enfant :
 - **xsd :simpleContent**
 - ou **xsd :complexContent**, lui-même enfant de l'élément **xsd :complexType**.

Extension

Type Simple

- L'extension d'un type simple permet uniquement d'ajouter des attributs pour donner un type complexe à contenu simple.
- Lors d'une extension d'un type simple, l'élément xsd :extension est toujours enfant d'un élément xsd :simpleContent.
- Les déclarations des attributs qui sont ajoutés sont placées dans le contenu de l'élément xsd :extension.

Extension

Type Simple

```
<xsd:complexType name="Price">
    <xsd:simpleContent>
        <xsd:extension base="xsd:decimal">
            <!-- Attribut ajouté -->
            <xsd:attribute name="currency" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="price" type="Price"/>

<price currency=<<Dollard>> 2000 </price>
```

Extension

Type Complexe à contenu simple

- Il est possible d'étendre un type complexe à contenu simple pour lui ajouter de nouveaux attributs. On obtient alors un nouveau type complexe à contenu simple qui possède les attributs du type de base et ceux déclarés par l'extension.
- L'extension d'un tel type est similaire à l'extension d'un type simple.

Extension

Type Complexe à contenu simple

```
<!-- Type de base complexe à contenu simple -->
<xsd:complexType name="Price">
    <xsd:simpleContent>
        <xsd:extension base="xsd:decimal">
            <!-- Attribut ajouté au type xsd:decimal -->
            <xsd:attribute name="currency" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>          <price currency=«Dollard» country=«USA»> 2000 </price>
</xsd:complexType>
<!-- Extension du type de base -->
<xsd:complexType name="LocalPrice">
    <xsd:simpleContent>
        <xsd:extension base="Price">
            <!-- Attribut ajouté au type Price -->
            <xsd:attribute name="country" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="price" type="LocalPrice"/>
```

Extension

Type Complexe à contenu complexe

- L'extension d'un type complexe à contenu complexe consiste à ajouter du contenu et/ou des attributs.
- Le contenu est ajouté après le contenu du type de base.
- L'ajout d'attribut est semblable au cas des types complexes à contenu simple.

Extension

Type Complexe à contenu complexe

```
<!-- Type de base -->
<xsd:complexType name="Name">
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<!-- Extension du type de base -->
<xsd:complexType name="Fullname">
    <xsd:complexContent>
        <xsd:extension base="Name">
            <xsd:sequence>
                <!-- Ajout de l'élément title après firstname et lastname -->
                <xsd:element name="title" type="xsd:string"/>
            </xsd:sequence>
                <!-- Ajout de l'attribut id -->
                <xsd:attribute name="id" type="xsd:ID"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

Restriction

- L'idée générale de la restriction est de définir un nouveau type dont les contenus au sens large sont des contenus du type de base.
- Elle est introduite par l'élément xsd :restriction dont l'attribut base donne le nom du type de base.
- La restriction s'applique aux types simples et aux types complexes mais elle prend des formes différentes suivant les cas.
- Dans le cas d'un type simple, l'élément xsd :restriction est enfant direct de l'élément xsd :simpleType.
- Dans le cas d'un type complexe, il est enfant d'un élément xsd :simpleContent ou xsd :complexContent, lui-même enfant de l'élément xsd :complexType.

Restriction

Type Simple

- La restriction des types simples est effectuée par l'intermédiaire de facettes qui imposent des contraintes aux contenus.
- Toutes les facettes ne s'appliquent pas à tous les types simples

Restriction

Facette

- **enumeration** : permet d'énumérer explicitement les valeurs autorisées.
- **Pattern** : permet de donner une expression rationnelle pour contraindre les valeurs. Elle ne s'applique pas uniquement aux types dérivés de string mais à tous les types simples.
- **length, minLength et maxLength** : donnent une longueur fixe, longueurs minimale et maximale. Elles s'appliquent aux types dérivés de xsd :string ainsi qu'aux types construits avec l'opérateur xsd :list.
- **minInclusive, minExclusive, maxInclusive et maxExclusive** : donnent des valeurs minimale et maximale en incluant ou non la borne donnée. Ces facettes s'appliquent à tous les types numériques ainsi qu'à tous les types de date et d'heure.
- **fractionDigits et totalDigits** : Ces deux facettes fixent respectivement le nombre maximal de chiffres de la partie fractionnaire (à droite de la virgule) et le nombre maximal de chiffres en tout. Ces deux facettes s'appliquent uniquement aux types numériques dérivés de xsd :decimal.

Restriction

Type Simple

- Les restrictions par énumération ou par motif se combinent avec un ou logique. Le contenu doit être une des valeurs énumérées ou il doit être conforme à un des motifs.
- Au contraire, les autres restrictions comme minInclusive et maxInclusive se combinent avec un et logique. Le contenu doit vérifier toutes les contraintes pour être valide

Restriction

Type Simple

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

Restriction

Type Complexe à Contenu Simple

- Les types complexes à contenu simple sont toujours obtenus par extension d'un type simple en lui ajoutant des attributs.
- La restriction d'un de ces types peut porter sur le type simple du contenu ou/et sur les attributs.
 - Il est possible de remplacer le type du contenu par un type obtenu par restriction.
 - Il est aussi possible de changer le type d'un attribut ou de modifier son utilisation.
- La restriction d'un type complexe à contenu simple donne toujours un **type complexe à contenu simple**.
- L'élément xsd :restriction contient un élément xsd :simpleType qui donne explicitement le nouveau type du contenu.

Restriction

Type Complexe à Contenu Simple

```
<!-- Type de base -->
<xsd:complexType name="Base">
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute name="format" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
<!-- Restriction du type de base -->
<xsd:complexType name="Derived">
    <xsd:simpleContent>
        <xsd:restriction base="Base">
            <!-- Nouveau type pour le contenu du type Derived -->
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="32"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="base" type="Base"/>
<xsd:element name="derived" type="Derived"/>
```

Restriction

Type Complexe à Contenu Complexé

- La restriction d'un type complexe permet d'imposer des contraintes aussi bien au contenu qu'aux attributs.
- La restriction doit rester fidèle au principe que tous les contenus possibles du type restreint doivent être valides pour le type de base. Il est, par exemple, possible de changer le type d'un élément en un type restreint ou de changer le nombre d'occurrences d'un éléments ou d'un bloc avec les attributs minOccurs et maxOccurs.
- Les restrictions portant sur les attributs sont identiques à celles possibles pour un type complexe à contenu simple.

Restriction

Type Complexe à Contenu Complexé

```
<!-- Type de base -->
<xsd:complexType name="Name">
    <xsd:sequence>
        <!-- Nombre illimité d'occurrences de l'élément firstname -->
        <xsd:element name="firstname" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>
<!-- Restriction du type Name -->
<xsd:complexType name="Shortname">
    <xsd:complexContent>
        <xsd:restriction base="Name">
            <xsd:sequence>
                <!-- Nombre limité d'occurrences de l'élément firstname -->
                <xsd:element name="firstname" type="xsd:string" maxOccurs="1"/>
                <xsd:element name="lastname" type="xsd:string"/>
            </xsd:sequence>
            <!-- Attribut id obligatoire -->
            <xsd:attribute name="id" type="xsd:ID" use="required"/>
        </xsd:restriction>
    </xsd:complexContent>
```

Restriction

Type Complexe à Contenu Complexé

```
<!-- Type de base -->
<xsd:complexType name="Number">
    <xsd:choice>
        <xsd:element name="integer" type="xsd:integer"/>
        <xsd:element name="float" type="xsd:float"/>
        <xsd:element name="double" type="xsd:double"/>
    </xsd:choice>
</xsd:complexType>
<!-- Restriction du type de base -->
<xsd:complexType name="Float">
    <xsd:complexContent>
        <xsd:restriction base="Number">
            <xsd:choice>
                <!-- Suppression de l'élément integer -->
                <xsd:element name="float" type="xsd:float"/>
                <xsd:element name="double" type="xsd:double"/>
            </xsd:choice>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

Définition Local/globale

Écriture à plat

- Toutes les définitions et déclarations sont directement au niveau 0 de l'élément.

```
<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 Local/globale

Écriture en poupées russes

```
<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>
```

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>
```

```
<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>
```

Réutilisation

Groupe d'attribut

```
<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>
```

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

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

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
- xsi:noNamespaceSchemaLocation

Spécifier l'emplacement des schémas

xsi :schemaLocation

- Réservé à une suite 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.
- Pour voir si un schéma décrit un espace de noms, il suffit de regarder l'attribut "**targetNamespace**" de l'élément **xs :schema**

Spécifier l'emplacement des schémas

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

Spécifier l'emplacement des schémas

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>
```

Espace de nom

Mon schéma décrit-il un espace de noms ?

Présence d'un attribut targetNamespace

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

Espace de nom

elementFormDefault = "unqualified"

- La valeur par défaut de l'attribut elementFormDefault est unqualified.
- Quand la valeur de elementFormDefault est unqualified, seuls les éléments définis globalement sont dans l'espace de noms cible.
- Les autres éléments sont sans espace de noms.
- Dans le schéma suivant, l'élément global name est dans l'espace de noms identifié par l'URI `http://www.omega-one.org/carton/` alors que les deux éléments locaux `firstname` et `lastname` sont sans espace de noms.

Espace de nom

elementFormDefault = "unqualified"

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- unqualified est la valeur par défaut de elementFormDefault --&gt;
&lt;xsd:schema targetNamespace="http://www.omega-one.org/~carton/"
  elementFormDefault="unqualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"&gt;
  &lt;xsd:element name="name"&gt;
    &lt;xsd:complexType&gt;
      &lt;xsd:sequence&gt;
        &lt;xsd:element name="firstname" type="xsd:string"/&gt;
        &lt;xsd:element name="lastname" type="xsd:string"/&gt;
      &lt;/xsd:sequence&gt;
    &lt;/xsd:complexType&gt;
  &lt;/xsd:element&gt;
&lt;/xsd:schema&gt;
&lt;?xml version="1.0" encoding="UTF-8"?&gt;
&lt;tns:name xmlns:tns="http://www.omega-one.org/~carton/"&gt;
  &lt;firstname&gt;Gaston&lt;/firstname&gt;
  &lt;lastname&gt;Lagaffe&lt;/lastname&gt;
&lt;/tns:name&gt;</pre>

```

Espace de nom

Élément qualifié

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.omega-one.org/~carton/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string" form="qualified"/>
        <xsd:element name="lastname" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<tns:name xmlns:tns="http://www.omega-one.org/~carton/">
  <tns:firstname>Gaston</tns:firstname>
  <lastname>Lagaffe</lastname>
</tns:name>

```

Espace de nom

- Lorsqu'un élément, un attribut, un groupe d'éléments, un groupe d'attributs ou encore un type défini globalement est **référencé** par un **attribut ref ou type**, la valeur de cet attribut doit contenir le nom **qualifié**.
- Ceci oblige à associer un préfixe à l'espace de noms cible et à l'utiliser pour qualifier l'élément ou le type référencé.
- Les éléments, attributs, groupes et types doivent être nommés avec un nom non qualifié quand ils sont déclarés ou définis. Ils sont à ce moment implicitement qualifiés par l'espace de nom cible.

Espace de nom

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.omega-one.org/~carton/"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.omega-one.org/~carton/">
  <!-- Référence au type Name par son nom qualifié -->
  <!-- Le nom name de l'élément déclaré n'est pas qualifié -->
  <xsd:element name="name" type="tns:Name" />

  <!-- Le nom Name du type défini n'est pas qualifié -->
  <xsd:complexType name="Name">
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Espace de nom

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.omega-one.org/~carton/"
  xmlns:tns="http://www.omega-one.org/~carton/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- Déclaration des éléments globaux name et tree -->
  <xsd:element name="name" type="xsd:string"/>
  <!-- Référence au type Tree par son nom qualifié -->
  <xsd:element name="tree" type="tns:Tree"/>
  <xsd:complexType name="Tree">
    <xsd:sequence>
      <!-- Référence à l'élément global name par son nom qualifié -->
      <xsd:element ref="tns:name"/>
      <!-- Référence récursive au type Tree par son nom qualifié -->
      <!-- Le nom child de l'élément déclaré n'est pas qualifié -->
      <xsd:element name="child" type="tns:Tree" minOccurs="0" maxOccurs="2"
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  <tns:tree
    xmlns:tns=
    "http://www.omega-one.org/~carton/">
    <tns:name>Racine</tns:name>
    <child>
      <tns:name>Fils Gauche</tns:name>
      <child>
        <tns:name>Petit fils</tns:name>
        </child>
      </child>
      <child>
        <tns:name>Fils Droit</tns:name>
      </child>
    </tns:tree>
  
```

Espace de nom

L'espace de noms par défaut égal à l'espace de noms cible

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.omega-one.org/~carton/"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.omega-one.org/~carton/"
  xmlns:tns="http://www.omega-one.org/~carton/"/>
  <!-- Référence au type Name par son nom qualifié -->
  <xsd:element name="name" type="Name" />

  <!-- Définition du type Name -->
  <xsd:complexType name="Name">
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

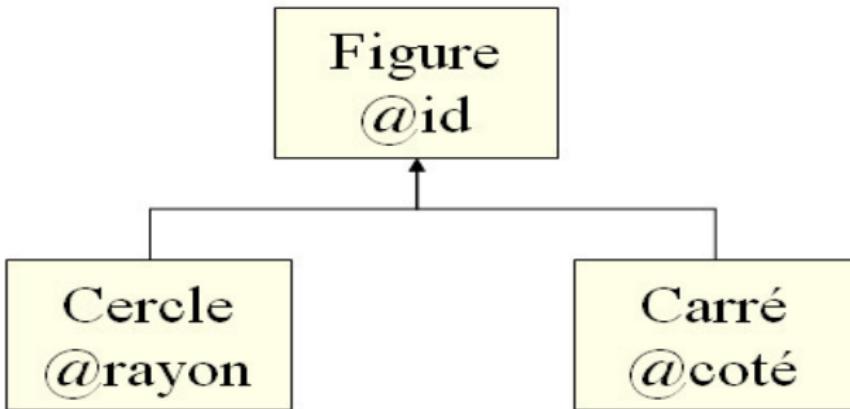
⑤ Modularité

⑥ Expressions rationnelles

Type Abstrait

- Un type complexe peut être déclaré abstrait en donnant la valeur true à l'**attribut abstract** de l'élément **xsd :complexType**.
- Un type simple déclaré avec **xsd :simpleType** ne peut pas être **abstrait**.
- Un type déclaré **abstrait** peut être utilisé pour dériver d'autres types mais il **ne peut pas être instancié**.
- Lorsqu'un type est déclaré abstrait dans un schéma, celui-ci peut encore être utilisé dans la déclaration d'un élément. En revanche, l'élément ne pourra pas avoir ce type dans un document.
- Un document valide doit nécessairement opérer une **substitution de type** par l'intermédiaire de l'attribut **xsi :type** ou une substitution d'élément par l'intermédiaire d'un groupe de substitution.

Héritage et validation



Héritage et validation

```
<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>
```

Héritage et validation

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

Héritage et validation

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

Éléments abstraits

Groupe 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.

Plan du cours

① Dérivée des types simples

② Définition Local/globale et la réutilisation

③ Validation

④ Type Abstrait

⑤ Modularité

⑥ Expressions rationnelles

Modularité

- il est possible d'importer d'autres schémas dans un schéma à l'aide des éléments **xsd :include** et **xsd :import**.
- L'élément **xsd :include** est employé lorsque l'espace de noms cible est identique pour le schéma importé.
- L'élément **xsd :import** est employé lorsque l'espace de noms cible du schéma importé est différent de celui qui réalise l'import.
- Les deux éléments **xsd :include** et **xsd :import** possèdent un attribut schemaLocation pour donner l'URL du schéma.
- L'élément **xsd :import** a, en outre, un attribut namespace pour spécifier l'URI qui identifie l'espace de noms cible du schéma importé.

Modularité

Le schéma à l'adresse `http://www.w3.org/2001/XMLSchema` contient les définitions des quatre attributs particuliers `xml:lang`, `xml:space`, `xml:base` et `xml:id` de l'espace de noms XML.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.omega-one.org/~carton/">
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/XMLSchema"/>
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:simpleContent>
        <!-- Le contenu est purement textuel -->
        <xsd:extension base="xsd:string">
          <!-- L'élément name a les attributs xml:lang, xml:space ... -->
          <xsd:attributeGroup ref="xml:specialAttrs"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Plan du cours

- ➊ Dérivée des types simples
- ➋ Définition Local/globale et la réutilisation
- ➌ Validation
- ➍ Type Abstrait
- ➎ Modularité
- ➏ Expressions rationnelles

Expressions rationnelles

- Une expression rationnelle désigne un ensemble de chaînes de caractères.
- Les expressions rationnelles sont construites à partir des caractères et des opérateurs **d'union ' | ', de concaténation et de répétition '?' , '*' et '+'**.
- La concaténation n'est plus marquée par la virgule ','.
- Ces expressions rationnelles sont utilisées par la facette xsd :pattern pour définir des restrictions de types simples.
- Elles sont également utilisées, en dehors des schémas, par les fonctions XPath matches(), replace() et tokenize() ainsi que l'élément XSL xsl :analyze-string.

Expressions rationnelles

- La plupart des caractères se désignent eux-mêmes dans une expression. Ceci signifie que l'expression 'a' (sans les apostrophes) désigne l'ensemble réduit à l'unique chaîne 'a' (sans les apostrophes).
- Certains caractères, dits spéciaux, ont une signification particulière dans les expressions.
- La liste des caractères spéciaux comprend les caractères : '|', '?', '**', '+', ':', ',', '(', ')', '[', ']', "", "".
- Pour **supprimer le caractère spécial** d'un de ces caractères et l'inclure dans une expression, il faut le faire **précéder** du caractère d'échappement '\'.

Expressions rationnelles

Opérateurs

- Les principaux opérateurs des expressions rationnelles sont l'**union** désignée par le caractère '|', la **concaténation** notée par simple **juxtaposition** et les opérateurs de **répétition** '?', '*', '+'.
- Comme pour les expressions arithmétiques, on peut utiliser les parenthèses '(' et ')' pour former des groupes.
- Exemple : L'expression $a(a|b)bc$ désigne l'ensemble contenant les deux chaînes aabc et abbc.

Expressions rationnelles

Opérateurs

- Les opérateurs '?', '*', '+' permettent de répéter des groupes.
- Ils utilisent une notation postfixée : ils se placent après leur opérande.
- Les accolades '{' et '}' permettent d'exprimer des répétitions dont le nombre est, soit un entier fixé, soit dans un intervalle donné.
 - Une expression de la forme $\text{expr}\{n\}$, où n est un entier, désigne l'ensemble constitué de toutes les chaînes obtenues en concaténant exactement n chaînes conformes à l'expression expr.
 - Une expression de la forme $\text{expr}\{m,n\}$, où m et n sont des entiers, désigne l'ensemble constitué de toutes les chaînes obtenues en concaténant un nombre de chaînes conformes à l'expression expr compris, au sens large, entre m et n.
 - L'entier n peut être omis pour donner une expression de la forme $\text{expr}\{m,\}$. Le nombre de répétitions doit seulement être supérieur à m.
 - L'expression $(a|b)\{6\}$ désigne, par exemple, les chaînes de longueur 6 formées de 'a' et de 'b'.
 - L'expression $(a|b)\{3,8\}$ désigne les chaînes de longueur comprise entre 3 et 8 formées de 'a' et de 'b'.

Expressions rationnelles

Ensembles de caractères

- Il est souvent nécessaire de désigner des ensembles de caractères pour construire des expressions rationnelles. Il existe plusieurs façons de décrire ces ensembles.
- Le caractère spécial '.' désigne tout caractère autre qu'un retour à la ligne.
- L'expression a.b désigne, par exemple, l'ensemble de toutes les chaînes de trois caractères dont le premier est 'a', le second n'est pas un retour à la ligne et le dernier est 'b'.
- L'expression .* désigne l'ensemble de toutes les chaînes ne contenant aucun retour à la ligne.

Expressions rationnelles

Ensembles de caractères

- Un ensemble fini de caractères peut simplement être décrit en donnant ces caractères encadrés par des crochets '[' et ']'. L'expression [aeiouy] désigne l'ensemble des voyelles minuscules et elle est équivalente à l'expression a|e|i|o|u|y.
- Cette syntaxe est pratique car elle permet d'inclure facilement un intervalle en plaçant un tiret '-' entre les deux caractères qui le délimitent.
- L'expression [0-9] désigne, par exemple, l'ensemble des chiffres. Il est possible de mettre des caractères et plusieurs intervalles.
- L'ordre des caractères entre les crochets est sans importance. Pour inclure un tiret '-' dans l'ensemble, il faut le placer en premier ou en dernier.

Expressions rationnelles

Ensembles de caractères

- Lorsque le premier caractère après le crochet ouvrant est le caractère '^', l'expression désigne le complémentaire de l'ensemble qui aurait été décrit sans le caractère ^. L'expression [^0-9] désigne, par exemple, l'ensemble des caractères qui ne sont pas un chiffre.
- Pour inclure un caractère '^' dans l'ensemble, il faut le placer à une autre place que juste après le crochet ouvrant.

Expressions rationnelles

Ensembles de caractères

- \s : caractères d'espacement, c'est-à-dire l'ensemble [\t \n \r]
- \S : caractères autres que les caractères d'espacement, c'est-à-dire l'ensemble [^ \t \n \r]
- \d : chiffres, c'est-à-dire [0-9]
- \D : caractères autres que les chiffres, c'est-à-dire [^0-9]
- \w : caractères alphanumériques et le tiret '-', c'est-à-dire [-0-9a-zA-Z]
- \W : caractères autres que les caractères alphanumériques et le tiret, c'est-à-dire [^ -0-9a-zA-Z]