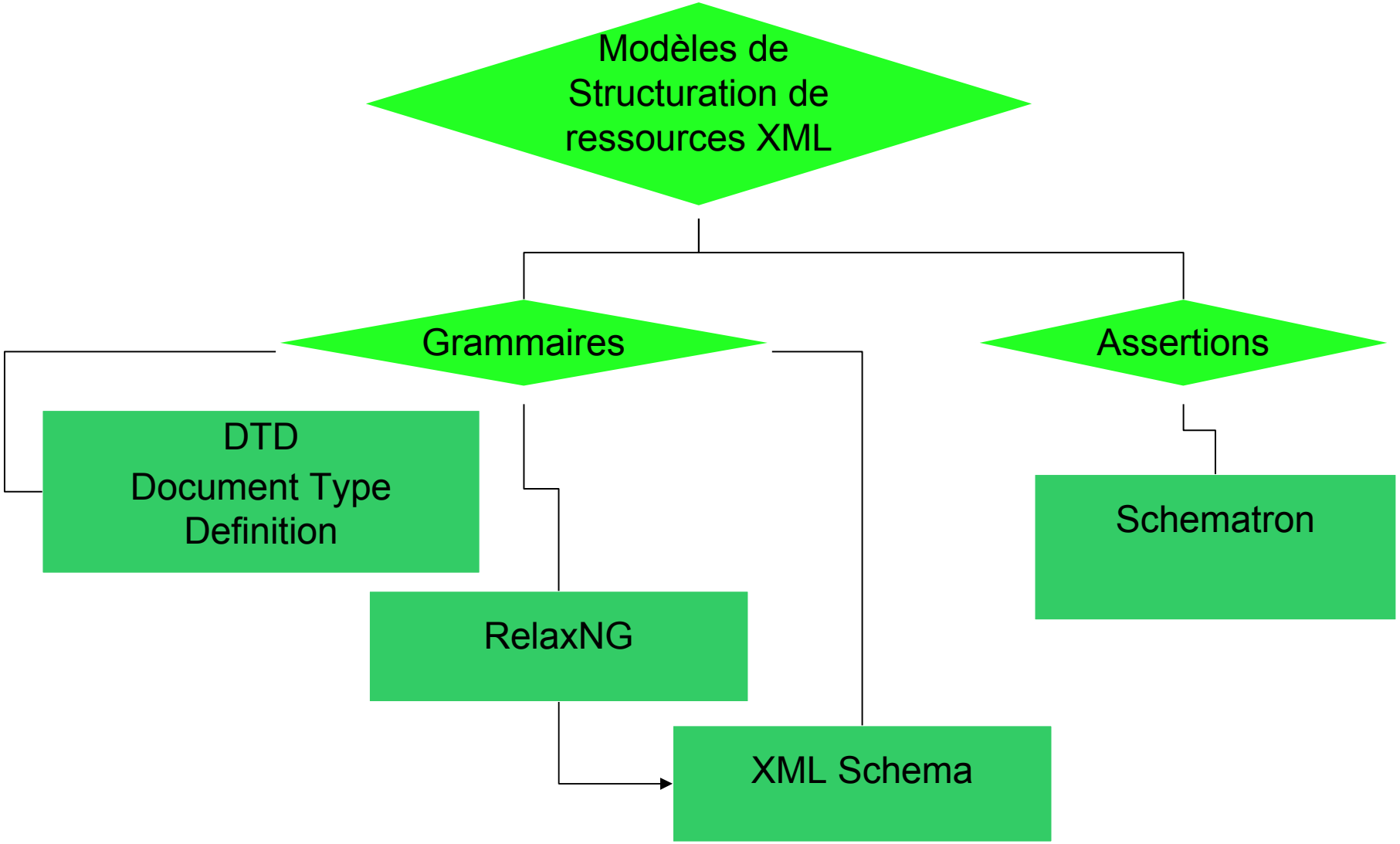
XML, langage de structuration

Modélisation structurelle en XML de *Messages* et de *Documents*

Introduction

- **Comprendre ce qu'est la modélisation de structures en XML nécessite la connaissance de son Histoire.**
  - **Au commencement était SGML, et les premiers documents XML ont dû se contenter des anciennes DTDs héritées de SGML.**
    - Avec leurs inconvénients :
      - » un langage différent nécessite un parseur différent,
      - » La notion d'espace de noms était philosophiquement étrangère aux concepteurs de SGML (qui n'envisageaient pas de diversité de noms de balises).
  - **À la façon démocratique du W3C, il se développa plusieurs initiatives pour de nouvelles expressions de modèles structurels**
    - Avec un ***typage de données*** plus puissant et évolutif,
    - Intégrant les notions d'***espace de noms***,
    - Utilisant XML pour définir les modèles eux-mêmes,
    - Apportant en plus la possibilité de contraintes partielles,.
    - Candidats pour s'intégrer aux recommandations du W3C.
    - – Indépendant de la forme sérialisée de XML,
      - » exploitant une structure logique basée sur les **XML InfoSet**.

Des candidats



Objectifs de modélisation pour XML

- **Définir un nouveau mécanisme de support à la modélisation**
  - **reprenant les acquis des DTD en termes de définition de modèles :**
    - Arbres d'objets typés et *valués,*
  - **permettant d'exprimer des contraintes fortes :**
    - Typage de données plus puissant et évolutif,
  - **utilisant XML pour définir les modèles eux-mêmes :**
    - Factorisation des outils et des méthodes,
- **Apporter en plus, de :**
  - **permettre de définir des contraintes incomplètes :**
    - Complémentarité aux DTD : outil de validation de données,
  - **s'intégrer à l'ensemble des spécifications W3C :**
    - Prise en compte des espaces de noms,
  - **se rendre indépendant du format XML sérialisé :**
    - validation appliquée à une structure logique basée sur les XML InfoSet.

Pour mémoire, les DTDs

Le langage de **Document Type Definitions** est spécifique : (i.e. différent de SGML et XML)

Il s'inscrit soit dans des ressources externes soit dans

Un exemple commenté de DTD :

```
<!ELEMENT titre (#PCDATA)>
<!ELEMENT paragraphe (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT description (paragraphe+)>
<!ELEMENT opération (#PCDATA)>
<!ELEMENT finition (#PCDATA)>
<!ELEMENT temps (opération?,finition?)>
<!ELEMENT imageRef (#PCDATA)>
<!ELEMENT outil (#PCDATA)>
<!ATTLIST outil nécessité CDATA #IMPLIED>
<!ELEMENT ustensiles (ustensile+)>
<!ELEMENT référence (titre, paragraphe+)>
<!ATTLIST référence href CDATA #REQUIRED >
<!ELEMENT matière (#PCDATA)>
<!ATTLIST matière ref CDATA #IMPLIED>
<!ELEMENT matériaux (matière+)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT actions (paragraphe*|action)>
<!ELEMENT biblio (référence+)>
<!ELEMENT gamme (titre, description, temps, imageRef?, temps?,
outils?, matériaux?)>
```

#PCDATA signifie texte littéral Parsé

+ indique une ou plusieurs occurences

? indique un ou 0 occurences

( ) indique un contenu

, indique une succession

#IMPLIED indique une possibilité

CDATA signifie texte littéral Non parsé

#REQUIRED indique une exigence

***** indique 0 ou plusieurs occurences

| indique un choix

Liste d'attributs d'un élément

Modélisation en Schematron

Soit une structure à définir pour un exemple de ressource XML :

```
<Person Title="Mr">
   <Name>Eddie</Name>
   <Gender>Male</Gender>
</Person>
```

On souhaite que soit respectée la bonne application des règles suivantes :

* l'élement de contexte (Person) doit avoir un attribut "Title",
* l'élement de contexte doit contenir deux éléments "Name" et "Gender',
* l'élement "Name" doit précéder l'élement "Gender",
* Si l'attribut Title a la valeur value 'Mr‘, Gender doit avoir la valeur 'Male'.

ces assertions seraient exprimées en Schematron :

```
<assert test="@Title">The element Person must have a Title attribute.</assert>
<assert test="count(*) = 2 and count(Name) = 1 and count(Gender)= 1">The element Person should
have the child elements Name and Gender.</assert>
<assert test="*[1] = Name">The element Name must appear before element Gender.</assert>
<assert test="(@Title = 'Mr' and Gender = 'Male') or @Title != 'Mr'">If the Title is "Mr" then the gender of
the person must be "Male". </assert>
```

le texte contenu dans les balises sert de message envoyé en cas de non respect d'une assertion.

Modélisation en RelaxNG

à l'instar des DTDs et de XML Schema , une ressource Relax NG définit une grammaire.

Comme XML Schéma, RelaxNG est un langage XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
     xmlns:a="http://www.monChampLexical.com/commentaire">
     <start>
     <element name="personne">
  <a:documentation>commentaire</a:documentation>
     <optional><element name="titre"><text /></element></optional>
<element name="nom"><text /></element>
<element name="prenom"><text /></element>
<element name="adresse"><text /></element>
</element>
</start>
</grammar>
```

Le même avec la forme compacte (non XML) de relaxNG :

```
personne {
     element titre {text}?,
     element nom {text},
     element prenom {text},
     element adresse {text},
}
```

Modélisation en XML Schema

XML Schema est une recommandation du W3C destinée à remplacer les DTDs, ce qui était nécessaire pour pouvoir exprimer des caractéristiques introduites par le métalangage XML que SGML et son langage de DTD fermé ne permettaient pas.

- XML ayant pour vocation d'être extensible, le langage XML Schema de description de contenu de documents XML est lui-même un document XML défini par un schéma, dont les balises de définition s'auto-définissent.
(Il faut observer que pour être mathématiquement cohérente les recommandations de langages du W3C ont toujours une racine auto-définie de façon récursive).

- La recommandation du W3C 1.0 se compose d'un document de présentation (non normatif), d'un document précisant comment définir la structure, et d'un document précisant comment définir les données.

Terminologie de XML Schema

Selon la terminologie de XML Schema, toute balise est un « **élément** ».

Le nom de l'élément racine d'un document XML Schema est « **schema** ».

cet élément contient des balises principales nommées,

« **element** »,

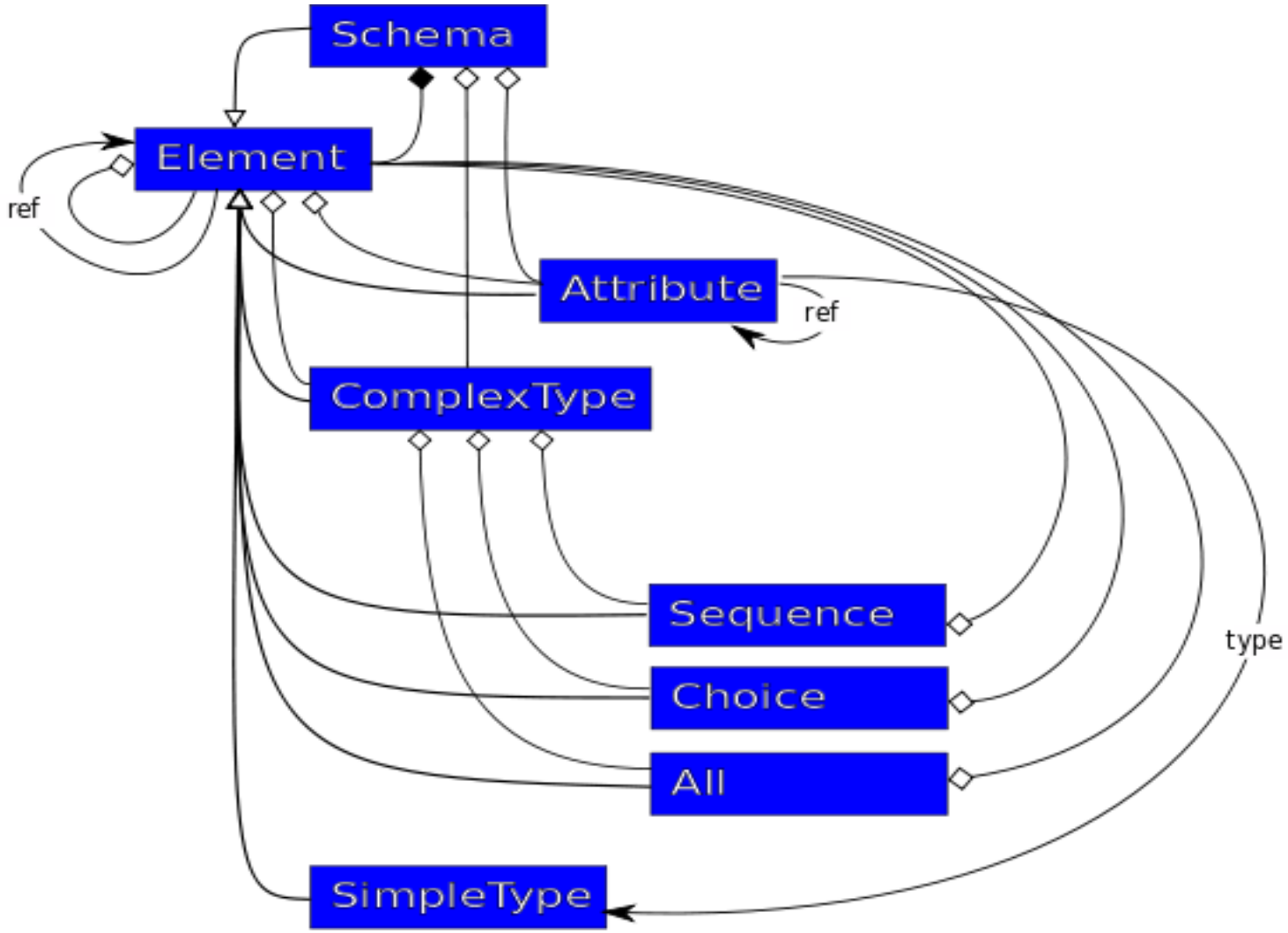
« **attribute** ».

Pour définir qu'une balise XML peut contenir un choix de balises ou une séquence de balises ou tout un ensemble de balises, les schemas introduisent la notion de « **complexType** » contenant un ou plusieurs éléments « **sequence** », « **choice** » ou « **all** », qui contiennent les éléments en question.

Les cardinalités possibles sont établies par des attributs « **minoccurs** » et « **maxoccurs** ».

Pour mettre en facteur commun des descriptions de hierarchie, XML Schema comporte des éléments « **simpleType** », « **complexType** », nommés qu'un élément peut référer.

Ontologie de XML Schema



Bootstrap de XML Schema

XML Schema est auto-décrit en XML Schema, publié par le W3C, à l'adresse: http://www.w3.org/2001/XMLSchema.xsd

```
<xs:element name="schema" id="schema">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:openAttrs">
    <xs:sequence>
     <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xs:include"/>
      <xs:element ref="xs:import"/>
      <xs:element ref="xs:redefine"/>
      <xs:element ref="xs:annotation"/>
     </xs:choice>
     <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="xs:schemaTop"/>
      <xs:element ref="xs:annotation" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:sequence>
    <xs:attribute name="targetNamespace" type="xs:anyURI"/>
    <xs:attribute name="version" type="xs:token"/>
    <xs:attribute name="finalDefault" type="xs:fullDerivationSet" use="optional" default=""/>
    <xs:attribute name="blockDefault" type="xs:blockSet" use="optional" default=""/>
    <xs:attribute name="attributeFormDefault" type="xs:formChoice" use="optional" default="unqualified"/>
    <xs:attribute name="elementFormDefault" type="xs:formChoice" use="optional" default="unqualified"/>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute ref="xml:lang"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

Notion d’espace de noms

- Un **espace de nom** est aux expressions informatiques ce qu'un **champ lexical** est aux langues vivantes.
- Un espace de nom (**namespace**) :
  - permet de garantir l'unicité d'un vocabulaire,
  - est dépositaire du modèle (de nommage) via l'association d'une URI qui documente et versionne ce dernier,
  - XML schema donne le choix de conception pour guider la réutilisabilité des élements qu'il déclare, soit avec une « forme qualifiée » ou une « forme non qualifiée ».

<ns:noeud> ... </ns:noeud> est une forme qualifiée,

<noeud> ... </noeud> ne l'est pas s'il n'y a pas de déclaration implicite.

=> importance de l'utilisation des espaces de noms !

Champs lexicaux : Espaces de Noms

- Un schéma fournit :
  - des définitions de types,
  - des déclarations d'éléments et d'attributs,

    appartenant à un espace de noms
- Chaque instance de document :
  - utilise un ou plusieurs espaces de noms,
  - peut associer chaque espace de nom à un schéma.

```
<po:purchaseOrder xmlns:po="http://www.example.com/PurchaseOrder"
  xsi:schemaLocation="http://www.example.com/PurchaseOrder po.xsd">
  …
</po:purchaseOrder>
```

XML Schema : approche par l'exemple 1 : un document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Etude
  xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation= "schemaEtude.xsd">

</Etude>
```

Le schéma de validation est associé via un attribut de la racine

XML Schema : approche par l'exemple 1 : le schéma

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

  <xs:element name="Etude"></xs:element>

</xs:schema>
```

La balise "Etude" est décrite comme un « élément »

XML Schema : approche par l'exemple 2 : le document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Etude
  xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation= "schemaOuvrage.xsd">
      <titre>étude de la situation économique du Lousdem en
         2011</titre>
      <Situation>la situation économique du
         Lousdem</Situation>
      <Problématique>la problématique économique du
         Lousdem</Problématique>
      <Résolution>solutions pour l'économie du Lousdem
      </Résolution>
      <Informations> à propos du Lousdem</Informations>
</Etude>
```

La balise « Etude » emboîte une sequence ordonnée de cinq balises

XML Schema : approche par l'exemple 2 : le schéma

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

  <xs:element name="Etude">

      <xs:complexType>

            <xs:sequence>

                  <xs:element name= "titre"/>

                  <xs:element name= "Situation"/>

                  <xs:element name= "Problématique"/>

                  <xs:element name= "Résolution"/>

                  <xs:element name= "Information"/>

            </xs:sequence>

      </xs:complexType>

  </xs:element>

</xs:schema>
```

L'élément « Etude » a un type complexe
pour êtrre composée d'une sequence ordonnée
de cinq éléments

XML Schema : approche par l'exemple 3 : le document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Etude
  xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation= "schemaOuvrage.xsd">
    <titre>étude économique du Lousdem en 2011</titre>
    <Situation date="2010-07-25>
          <titre>contexte de l'étude</titre>
          <paragraphe>Blah...</paragraphe>
          <paragraphe>Blah...</paragraphe>
    </Situation>
    <Problématique>
    </Problématique>
    <Résolution>
    </Résolution>
    <Informations>
    </Informations>
</Etude>
```

La balise « Situation » emboîte une sequence ordonnée de cinq balises

XML Schema : approche par l'exemple 3 -1 : le schéma

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

  <xs:element name="Etude">

      <xs:complexType>
            <xs:sequence>
                  <xs:element name= "titre"/>
                  <xs:element name= "Situation">
                        <xs:complexType>
                        <xs:sequence>
                        <xs:element name= "titre"/>
                        <xs:element name= "Paragraphe" maxOccurs="unbounded">
                        </xs:sequence>
                        <xs:complexType>
                  </xs:element>
                  <xs:element name= "Problématique"/>
                  <xs:element name= "Résolution"/>
                  <xs:element name= "Information"/>
            </xs:sequence>
      </xs:complexType>

  </xs:element>

</xs:schema>
```

L'élément «Situation»
a un type complexe
pour être composé
d'un titre et de paragraphes

XML Schema : approche par l'exemple 3 -2 : le schéma

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

 <xs:element name="Etude">
     <xs:complexType>
          <xs:sequence>
               <xs:element name="titre"/>
               <xs:element name="Situation" ref="sectionType"/>
               <xs:element name="Problématique" ref="sectionType"/>
               <xs:element name="Résolution" ref="sectionType"/>
               <xs:element name="Information" ref="sectionType"/>
          </xs:sequence>
     </xs:complexType>
     <xs:attribute name="date" type="xs:date"/>

  </xs:element>

  <xs:complexType name="sectionType">
     <xs:sequence>
          <xs:element name="titre"/>
          <xs:element name="Paragraphe" maxOccurs="unbounded">
     </xs:sequence>

  </complexType>

</xs:schema>
```

Les éléments «Situation», « problématique », « Résolution », « Information » Référencent la même structure de type complexe pour être composés d'un titre et de paragraphes

Diagramme XML Spy

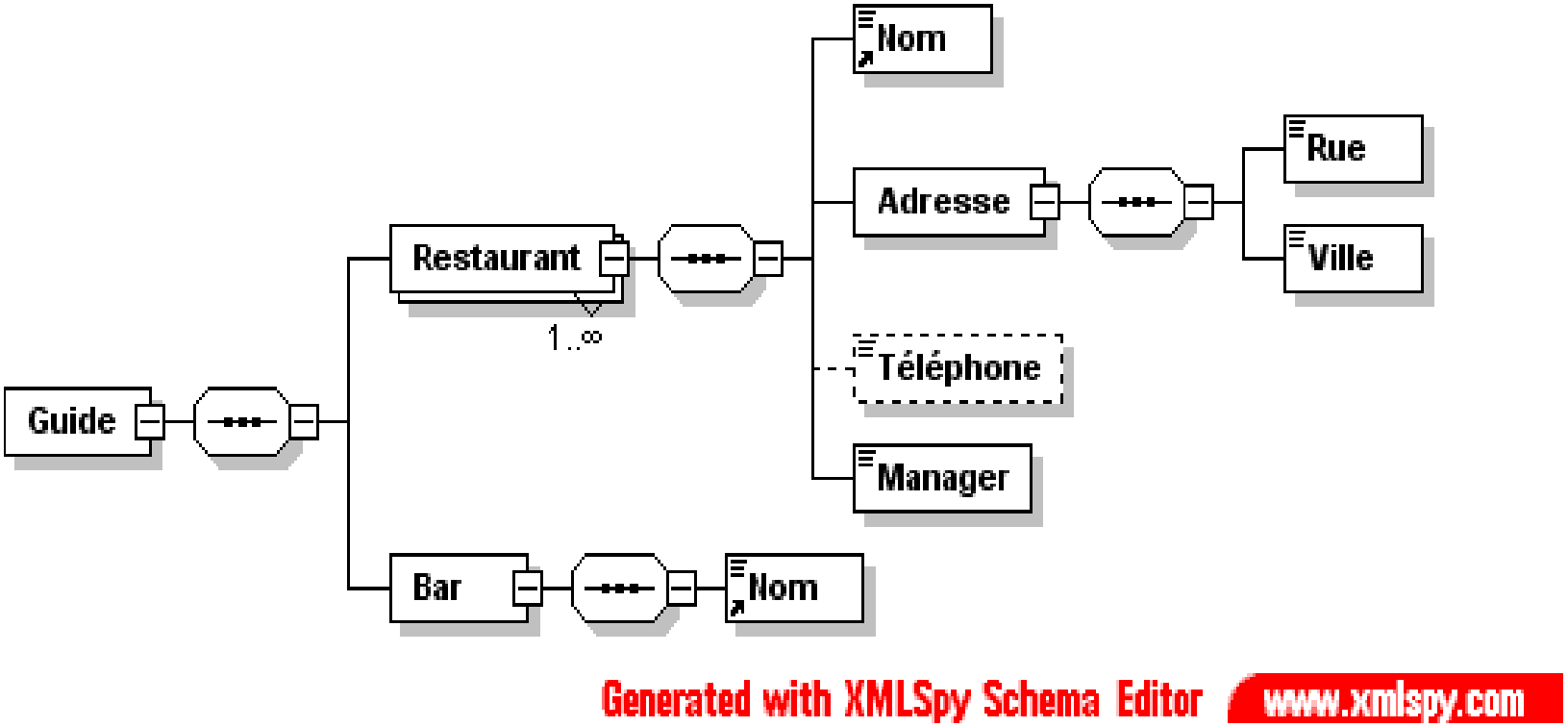
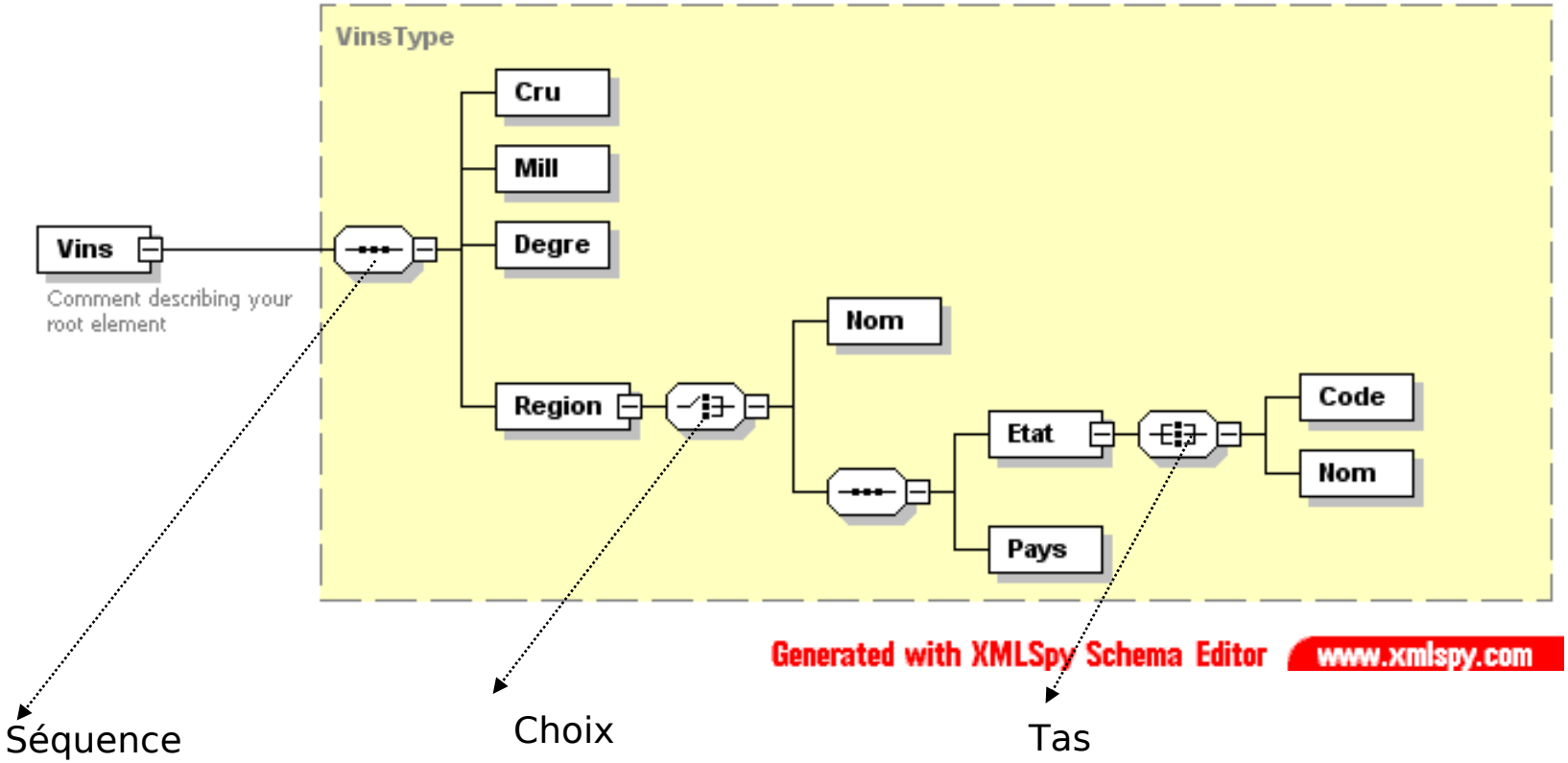


Diagramme XML Spy



Question de qualification

- La qualification des éléments dans des espaces de noms dépend des choix de conception d'un XML Schema :
  - Un même document XML peut être validé par deux ressource différentes de Schéma XML de même qu'un même schéma peut valider des documents de structure différentes.. .

```
<element name="purchaseOrder"
type="po:PurchaseOrderType"/>
 <element name="comment"  type="string"/>
 <complexType name="PurchaseOrderType">
  <sequence>
   <element name="shipTo"    type="po:USAddress"/>
   <element name="billTo"    type="po:USAddress"/>
   <element ref="po:comment" minOccurs="0"/>
   <!-- etc. -->
  </sequence>
  <!-- etc. -->
 </complexType>
 <complexType name="USAddress">
  <sequence>
   <element name="name"   type="string"/>
   <element name="street" type="string"/>
   <!-- etc. -->
  </sequence>
 </complexType>
```

```
<element name="purchaseOrder" type="po:PurchaseOrderType"/>
 <element name="shipTo"  type="po:USAddress"/>
 <element name="billTo"  type="po:USAddress"/>
 <element name="comment" type="string"/>
 <element name="name" type="string"/>
 <element name="street" type="string"/>
 <complexType name="PurchaseOrderType">
  <sequence>
   <element ref="po:shipTo"/>
   <element ref="po:billTo"/>
   <element ref="po:comment" minOccurs="0"/>
   <!-- etc. -->
  </sequence>
 </complexType>
 <complexType name="USAddress">
  <sequence>
   <element ref="po:name"/>
   <element ref="po:street"/>
   <!-- etc. -->
  </sequence>
 </complexType>
```

Avantages et inconvénients des éléments globaux

- Les deux schémas précédents peuvent sembler équivalents
  - Le premier déclare deux éléments globaux : purchaseOrder et comment
  - Le second déclare plus d'élements globaux :
    - PurchaseOrder
    - ShipTo
    - BillTo
    - Comment
    - Name street
- Mais...
  - Les élements globaux sont considérés comme des racines potentielles pour un document XML !
  - Les déclarations locales (i.e. Au sein de la déclaration d'un élement) restent "anonymes" et leur portée contextuelle,
  - Les déclarations globales sont visibles de tous et peuvent générer des conflits !

Faut-il qualifier ou non ?

- Plus il y a d'élements globaux, plus le risque d'entrer en conflit avec un autre XML Schema est grand :
  - Si la définition fait partie d'un ensemble, il est conseillé de qualifier pour éviter les conflits,
  - Si elle représente un modèle unique de documents, avec peu, voir pas, d'utilisations de vocabulaires externes :
    - Il est possible de ne rien qualifier,

Dans le doute, éviter de multiplier les élements globaux et qualifier uniquement la racine .

Espace de nom cible

- Lorsque l'on crée un XML Schema, il faut lui associer un espace de nom cible :
  - Il identifiera de facon unique le modèle,
  - Syntaxe : targetNameSpace="http://www.domain.name/qualifier"
  - Tout document se réclamant instance de ce modèle doit importer cet espace de noms.
- **Si l'on veut qualifier les élements et/ou les attributs, il faut indiquer :**
  - ElementFormDefault = "true|false"
  - AttributeFormDefaut = "true|false"
- **Syntaxe complète :**

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:po="http://www.example.com/PO1"
    targetNamespace="http://www.example.com/PO1"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
```

Qualification explicite

- Exemple de document utilisant une forme qualifiée explicite :

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
                   orderDate="1999-10-20">
   <apo:shipTo country="US">
      <apo:name>Alice Smith</apo:name>
      <apo:street>123 Maple Street</apo:street>
      <!-- etc. -->
   </apo:shipTo>
   <apo:billTo country="US">
      <apo:name>Robert Smith</apo:name>
      <apo:street>8 Oak Avenue</apo:street>
      <!-- etc. -->
   </apo:billTo>
   <apo:comment>Hurry, my lawn is going wild!</apo:comment>
   <!-- etc. -->
</apo:purchaseOrder>
```

Qualification implicite

- Exemple de document utilisant une forme qualifiée implicite :

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/PO1"
               orderDate="1999-10-20">
   <shipTo country="US">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <!-- etc. -->
   </shipTo>
   <billTo country="US">
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <!-- etc. -->
   </billTo>
   <comment>Hurry, my lawn is going wild!</comment>
   <!-- etc. -->
</purchaseOrder>
```

Notion de type

XML Schema offre le moyen de définir des types pour les utiliser dans des modèles de contenus ou des déclarations d'attributs

Un type est identifié et soit référencé par son nom, soit directement défini (type anonyme)

```
<simpleType name="cpType">
   <restriction base="string">
      <pattern value="\d{5}"/>
   </restriction>
</simpleType>
```

type Simple

```
<complexType name="caType">
  <sequence>
    <element ref="ADRESSE" minOccurs="1" />
  </sequence>
  <attribute
    name="DATE-CREATION"
    type="date"/>
  <attribute name="DATE-MAJ" type="date"/>
</complexType>
```

type Complexe

Les types s'utilisent dans les déclarations d'éléments ou d'attributs :

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

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

Toutes les définitions de types se font par référence ou par construction à partir de types existants, d'où les deux possibilités :

- **Extension** : ajout d'informations à un type existant,
- **Restriction** : contrainte d'un type existant aux valeurs qu'il peut prendre.

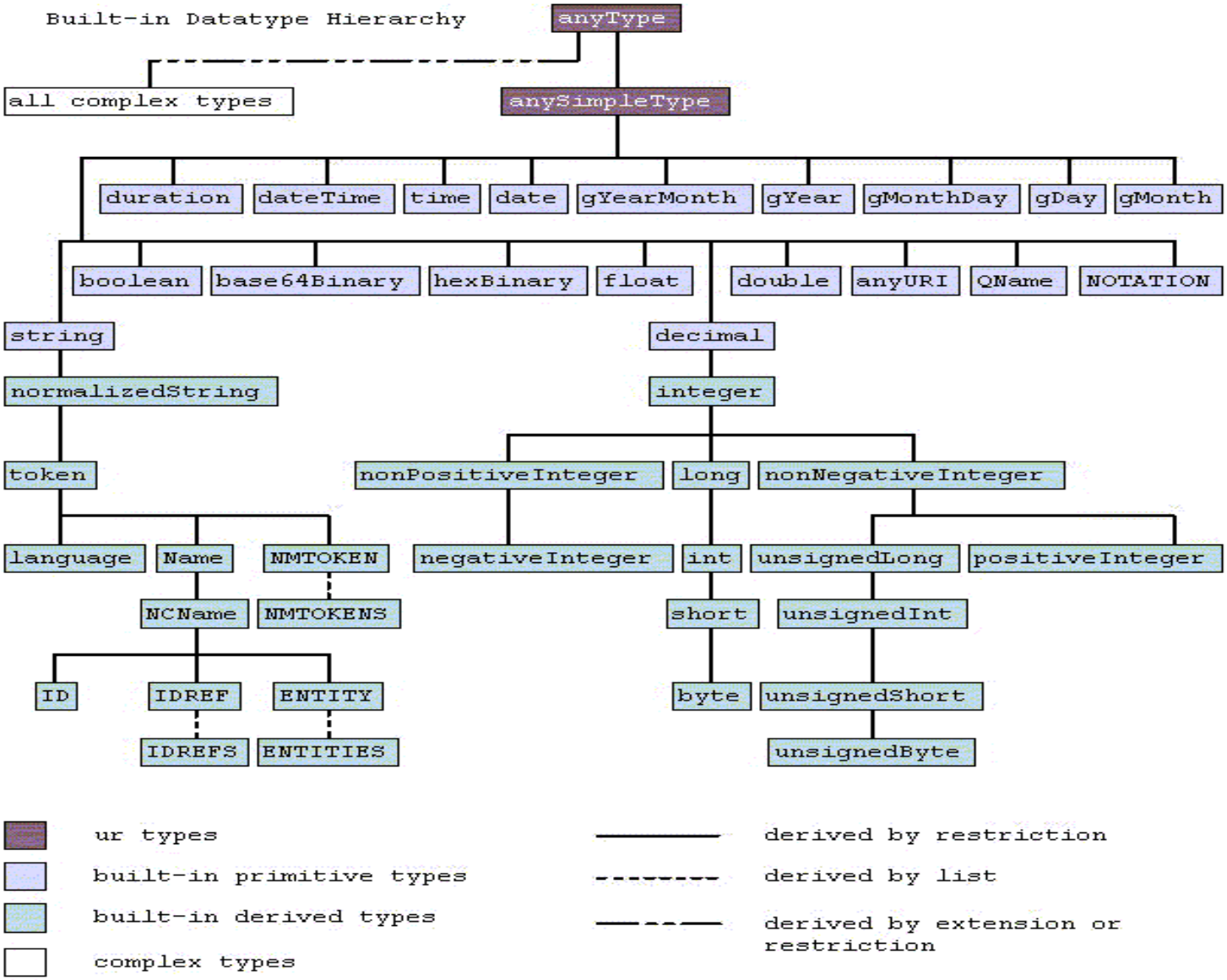
Types et dérivation

- Apport de l'approche "Objet",
- Modularité optimisée,
- Dérivations de types,
  - Simples et complexes,
  - Par extension (ajout d'éléments, d'attributs),
  - Par restriction (interdiction d'éléments, d'attributs, ajout de contraintes, *facets* sur les types simples).
- Des types abstraits
  - Utilisés comme des interfaces.

```
<complexType name="CONTACT.COM">
<complexContent>
 <extension base="contactType">
  <sequence>
   <element name="DEMANDE"
        type="string"/>
   <element ref="RESP.INT" />
  </sequence>
 </extension>
</complexContent>
</complexType>
```

```
<simpleType name="telNumType">
    <restriction base="string">
        <length value="14"/>
    </restriction>
</simpleType>
```

Les types simples prédéfinis



Les types simples (2)

- long
  - -1, 12678967543233
- unsignedLong
  - 0, 12678967543233
- short
  - -1, 12678
- unsignedShort
  - 0, 12678
- decimal
  - -1.23, 0, 123.4, 1000.00
- float
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- double
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- boolean
  - true, false 1, 0
- time
  - 13:20:00.000, 13:20:00.000-05:00
- dateTime
  - 1999-05-31T13:20:00.000-05:00
- duration
  - P1Y2M3DT10H30M12.3S
- date
  - 1999-05-31
- gMonth
  - --05--
- gYear
  - 1999

Les types simples (3)

- gYearMonth
  - 1999-02
- gDay
  - ---31
- gMonthDay
  - --05-31
- Name
  - shipTo
- QName
  - po:Address
- NCname
  - Address
- AnyURI
  - http : // ww.example.com/,
  - http : ww.example.com/doc.html#ID5
- language
  - en-GB, en-US, fr
- ID
  - "A212"
- IDREF
  - "A212"
- IDREFS
  - "A212" "B213"
- ENTITY
- ENTITIES
- NOTATION
- NMTOKEN, NMTOKENS
  - US
  - Brésil Canada Mexique

Exemples de simpleTypes

- Decrivons les différents types d'unités de mesure (gramme, kilogramme, litre respectivement 3, 2, 2 digits) :

```
<xs:simpleType name=''gramme''>
    <xs:restriction base=''xs:integer''><xs:length
     value=''3''/></xs:restriction>
</xs:simpleType>
<xs:simpleType name=''kilogramme''>
    <xs:restriction base=''xs:integer''><xs:length
     value=''2''/></xs:restriction>
</xs:simpleType>
<xs:simpleType name=''litre''>
    <xs:restriction base=''xs:integer''><xs:length
     value=''2''/></xs:restriction>
</xs:simpleType>
```

Liste et union de types simples

- list : pour structurer un contenu textuel comme une liste :

```
<simpleType name="ListePrenom">
    <list itemType="string"/>
</simpleType>
```

  – **Sur lequel on peut imposer des contraintes :**

```
<simpleType name="ListePrenom3">
    <restriction base="ListePrenom">
        <length value="3" />
     </restriction>
</simpleType>
```

- union : pour fusionner plusieurs types simples :

```
<simpleType name="moneyInternational">
   <union memberTypes="Euros Dollars" />
</simpleType>
```

<listePrenom>Pierre Paul Jacques</listePrenom>

Dérivation de types simples

Contraintes sur type simple prédéfini :

- Utilisation de facettes :
  - Taille fixe, minimale, maximale, nombre de chiffres après la virgule,
  - Domaine de valeurs (bornes min et max) pour les types ordonnés,
  - Expressions régulières.
- Exemple :

```
<xsd:simpleType name=''RepereFonctionnel">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[A-Z]{3}[0-9]{3}[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Exemple de dérivation de type simple

- Descriptions des types d'unités de mesure possibles avec les abréviations communes (gr pour gramme, kg pour kilogramme, l pour litre ...) en utilisant le mécanisme de dérivation :

```
<xs:simpleType name=''unite">
    <xsd:restriction base="xs:string">
       <xs:enumeration value=''gr''/>
       <xs:enumeration value=''kg''/>
       <xs:enumeration value=''l''/>
    </xs:restriction>
</xs:simpleType>
```

Les types complexes

- Définition d'objets complexes
  - **<sequence>** : collection ordonnée d'éléments typés
  - **<all>** : collection non ordonnée d'éléments typés
  - **<choice>**: choix entre éléments typés
- Caractérisation du modèle de contenu
  - **Mixed**
  - **Nillable** (balise vide autorisée)
  - **Empty** (balise vide obligatoire)

Exemple

```
<xsd:complexType name="USAddress">
 <xsd:sequence>
 <xsd:element name="name"
 type="xsd:string"/>
 <xsd:element name="street"
 type="xsd:string"/>
 <xsd:element name="city"
 type="xsd:string"/>
 <xsd:element name="state"
 type="xsd:string"/>
 <xsd:element name="zip"
 type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="country"
 type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

Dérivation de types complexes

- Définition de sous-types par héritage :
  - Par extension : ajout d'informations
  - Par restriction : ajout de contraintes
- Possibilité de contraindre la dérivation :
  - **Final** : interdit la dérivation du type
  - **Block** : empêche la substitution de types

- Exemple :

```
<complexType name="AdressePays">
  <complexContent>
    <extension base="Adresse">
      <sequence>
        <element name="pays"
  type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Exemple de dérivation de complexType

- Description d'une sorte d'ingrédient par extension en ajoutant une information de type conservation :

```
<xs:simpleType name="conservation" type="xs:string"/>
<xs:complexType name="ingredient2">
    <xs:complexContent>
            <xs:extension base="ingredient">
                <xs:element type="conservation"/>
            </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Les attributs

- En XML schema il est possible (comme avec les DTD) de déclarer des attributs qui sont aussi soumis au processus de typage :
  - Un attribut peut être déclaré au sein de l'élement (de manière anonyme):

```
<xs:element ....
 <xs:attribute name="att" type="xs:type"/>
</xs:element>
```

  - Sa déclaration peut être factorisée, via des groupes :

```
<xs:element ....
   <xs:attributeGroup ref="monAtt"/>
</xs:element>
<xs:attributeGroup name="monAtt">
   <xs:attribute name="att" type="xs:type"/>
</xs:attributeGroup>
```

Les patrons, dits « patterns »

- Contraintes sur type simple prédéfini,
- Utilisation d'expression régulières,
  - Similaires à celles de Perl, Python, Tcl, Java, etc.

- Exemple

```
<xsd:simpleType name=« RepereFonctionnel">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="[A-Z]{3}[0-9]{3}[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Exemple de pattern

- L'utilisation des expressions régulières permet de définir de manière plus ou moins précise les contenus possibles pour un type. Décrivons le type temperature pour qu'il n'accepte que la forme nn,nnC° :

```
<xs:simpleType name="temperature">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{2}(,[0-9]{2})?C°"/>
    </xs:restriction>
</xs:simpleType>
```

- De même pour le type gramme :

```
<xs:simpleType name="gramme">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{2}(,[0-9]{2})?gr"/>
    </xs:restriction>
</xs:simpleType>
```

Modalités de réutilisation

- Référence locale à un élément global :
  - <element ref="Nom" /> (Nom défini au premier niveau)
  - Importe l'élément et son type
- Définition de modèles réutilisables dans un schéma :
  - group
  - attributeGroup
- import de types associés à un espace de noms :
  - <import nameSpace = "http:// ..."
  - schemaLocation = "http:// ..." />
- Inclusion ou extension d'un schéma :
  - <include schemaLocation="http:// ..."/>
  - <redefine schemaLocation="http:// ..."/>
  - .... Extensions ...
  - </redefine>

Exemple de réutilisation

- Réutilisation d'un schema existant déclarant des ingrédients :

```
<xs:schema.. xmlns:recette="http:// recette.cuisine.org"..>
<import nameSpace = "http:// recette.cuisine.org"
schemaLocation =
   "http:// recette.cuisine.org/modeles/ingredients.xsd"/>
...
<xs:element name= "plat">
<xs:complexType>
    <xs:sequence>
    <xs:element name="ingredient1" type="recette:ingredient"/>
    </xs:sequence>
    </xs:complexType>
</xs:element>
...
</xs:schema>
```

Autres fonctionnalités

- Wildcards
  - any, anyAttribute,

  autorise l'introduction de n'importe quel élement d'un namespace donné.
- Annotations
  - Pour documenter un schéma,.
- Unicité et références
  - unique,
  - key et keyRef permet de définir une clé nommée de référencement mutuel (autre que ID IDREF).
    (tentative maladroite de construction de graphes sémantiques)
- Types et éléments abstraits
  - Pour différentes modalités d'extensions.
- Substitution Groups
  - Pour autoriser le remplacement d'éléments par d'autres.

Autres possibilités ...

- Mixed Content
- Empty Content
- anyType
- Annotations
- Attribute Groups
- Nil Values
- Specifying Uniqueness
- Defining Keys & their References
- Namespaces, Schemas & Qualification
- Target Namespaces & Unqualified Locals
- Qualified Locals
- Global vs. Local Declarations
- Undeclared Target Namespaces
- Substitution Groups
- Abstract Elements & Types

Bilan

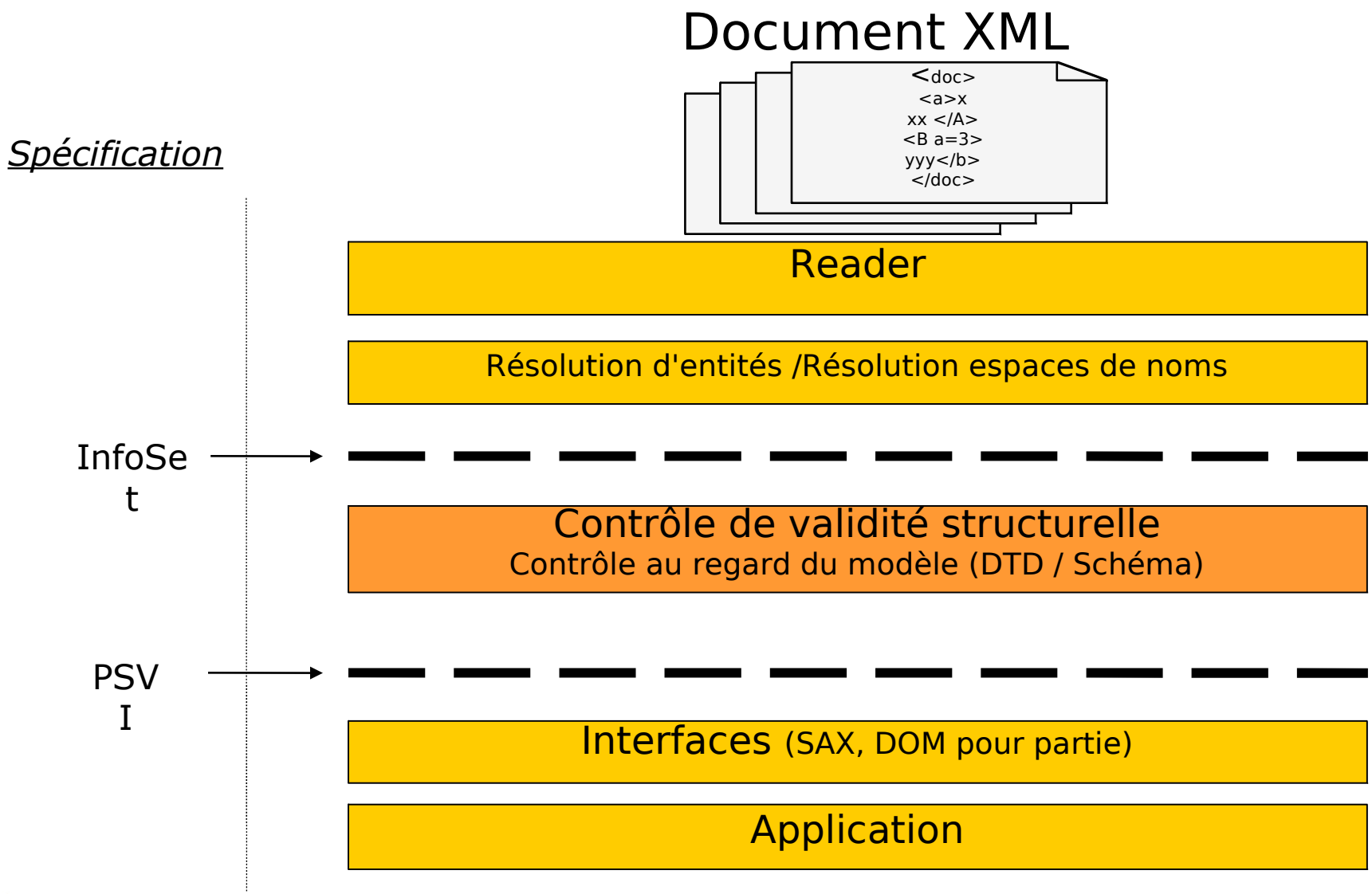
- Intérêt
  - Conception : modélisation de balisage avec factorisation
    - Lisibilité,
    - Compatibilité XML et espaces de noms,
    - Pouvoir de description des structures et des types,
    - Pouvoir d'expression de contraintes,
    - Très modulaire,
      - » Mécanisme de modularisation des modèles (inclusions et dérivation),
      - » Permet des architectures de modèles ouvertes!.
    - Utilisation d'XML comme seul langage,
      - » Toute l’ingénierie d'analyse syntaxique est réutilisable.
  - Exploitation : validation et utilisation multiples,
    - Coopération : plusieurs schémas sur une même instance (modularité),
    - Application spécifique : validation d'instances utilisée dans un contexte applicatif précis.

Bilan (suite)

- **Limites**
  - Spécification complexe,
  - Forme verbeuse,
  - Outils (de validation) difficiles à réaliser,
  - Contraintes sur les atomes (données), mais pas sur les inter-relations (ceci est l'objet de RDF,RDFS, OWL).
    - Quel est le rôle du modèle ? Quel est le rôle de l'applicatif ?
    - Trop de modèles sont exprimés en XMLSchema devraient faire l'objet de descriptions sémantiques.
  - Ne permet pas la fragmentation de fichiers de données.
    - Nécessité d'une coopération avec DTD ou utilisation de Xinclude
  - La compatibilité avec les DTD est incomplète.

- **Une autre orientation serait possible :**
  - ***La définition d'une spécification sémantique en RDF/OWL !***

La validation vue comme une transformation



Fin du module