Architecture Orientée service SOA

Enseignante: Bènène Fradi

A.U 2023-2024

Objectifs du module

- Sensibiliser l'apprenant des défis de l'interopérabilité
- Maitriser les concepts liés de services web et technologies liées
- Construire et déployer des services web et leurs clients
- Familiariser l'apprenant avec le style d'architecture SOA
- Maitriser un outil de mise en œuvre SOA

Chapitre 1

XML
Extensible Markup Language

Objectifs

- Savoir le rôle de XML.
- Apprendre la structure et les règles syntaxiques d'un document XML.
- Comprendre la notion des espaces de nom.

Plan

- Introduction
- Structure de données
- Présentation de XML
- Structure d’un document XML
- Espace de noms XML

Introduction

- On doit organiser d'une certaine manière les données ce qui permet un traitement automatique de ces dernières plus efficace et rapide.

→ utilisation d’une structure de données.

Structure de données 1/3

- Structure de données:
  - Une organisation des informations.
  - est destinée à contenir des données, afin de leur donner une organisation permettant de simplifier leur traitement.

➡ baisser de manière significative la complexité d'une application informatique et diminuer le taux d'erreurs.

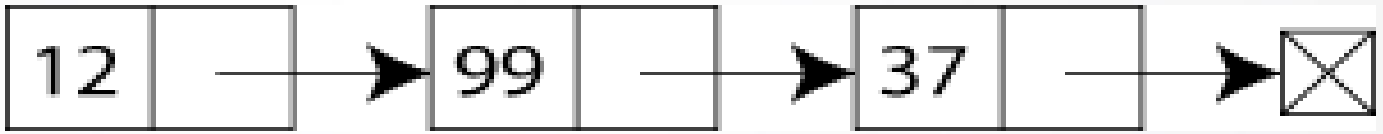
Structure de données 2/3

- Différentes structures de données existent:

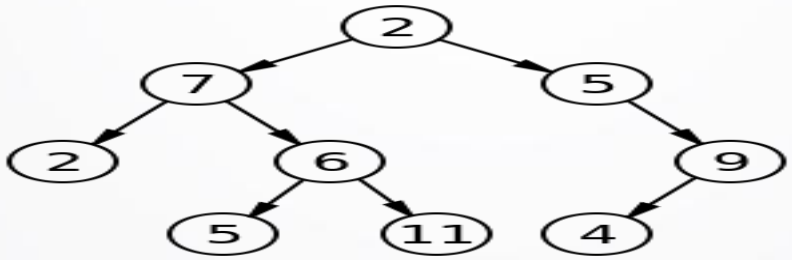
- Tableau:

Valeur	45	154	58	78	31	5	74
Index	0	1	2	3	4	5	6

- Liste chainée:



- Arbre:



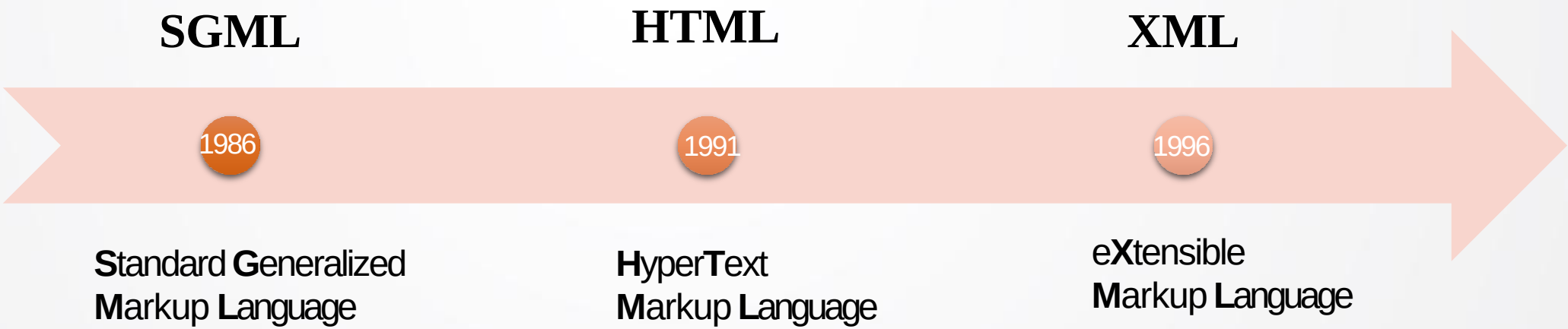
Structure de données 3/3

- Les documents structurés sont des documents qui contiennent de l'information à propos de leurs **structures logiques** et **physiques:**

- Structure physique : apparence visuelle (texte sur deux colonnes, texte justifié ou non, etc.)
- Structure logique : organisation du contenu intellectuel du document (chapitre, section, sous-section, etc.)

Langages de description de document structuré 1/3

- Les langages les plus couramment utilisés permettant d'encoder un document structuré à l'aide des balises sont:



Langages de description de document structuré 2/3

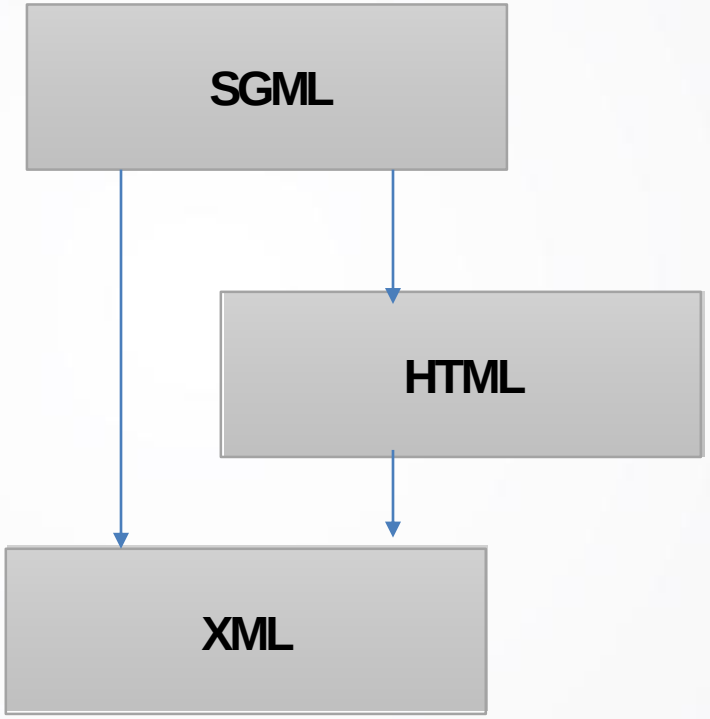
- SGML est un langage servant à préciser la structure d'un document quelconque. Il est compréhensible mais il était inadapté à l'écriture de documents pour internet. Il a donc été nécessaire d'en dériver le langage HTML.[1]

- HTML est le standard du développement web mais il n'est ainsi pas possible de définir autre chose qu'une page Web. [1]

- Le XML est un dérivé du SGML. Il tente à être plus souple que HTML et plus simple que SGML.

Langages de description de document structuré 3/3

- Sépare les données la structure des données et la mise en forme
  - Syntaxe complexe



- Présentation des documents sur le web 1986
- Mélange les données et la mise en forme
  - Non flexible, figé

- Structuration, échange des documents
  - Plus simple que SGML
  - Plus souple que HTML

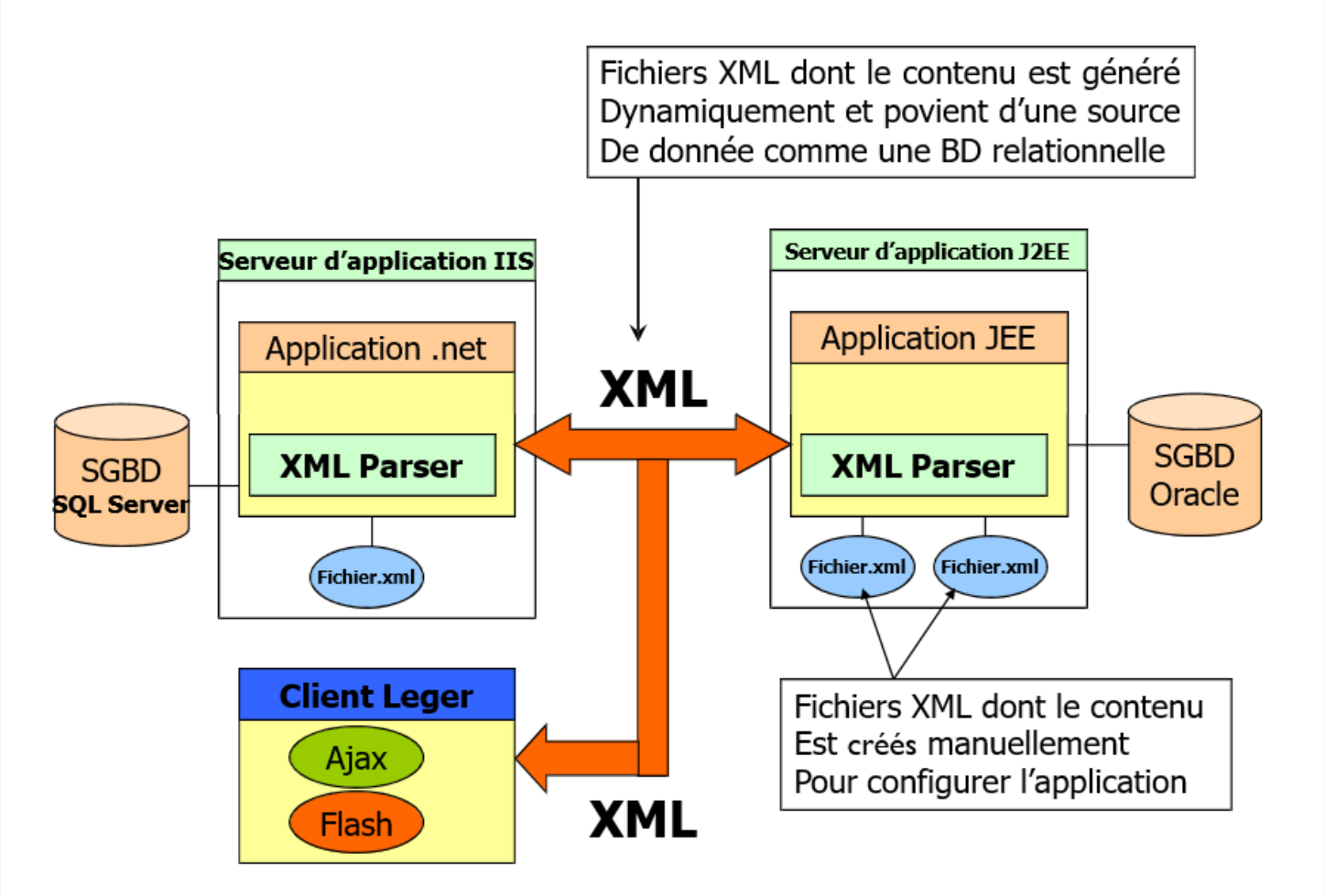
Présentation de XML

- eXtensible Markup Language.
- Langage de balises.
- Recommandation de W3C.
- XML décrit, structure, échange des données.
- Archiver des données.

Pourquoi utiliser XML?

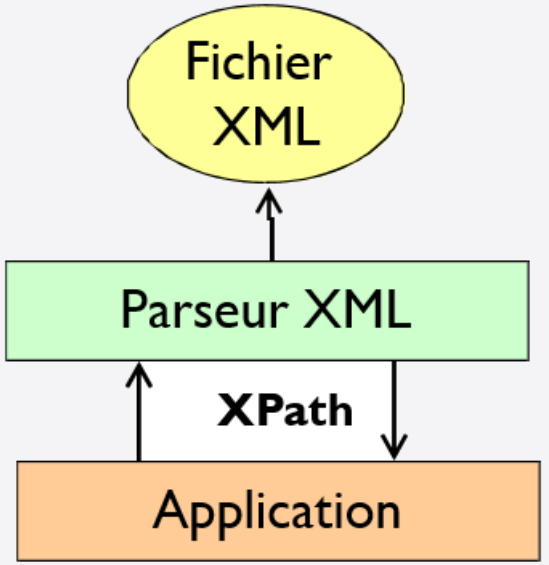
- Lisible : texte balisé avec marquage.
- Extensible : supporte les évolutions applicatives.
- Mise en forme avec des feuilles de style.
- Un méta langage permettant la définition de langages adaptés à des besoins variés.
- Supporté par les grands constructeurs: IBM, Microsoft .net, SUN, etc.

XML est au cœur des systèmes d'informations



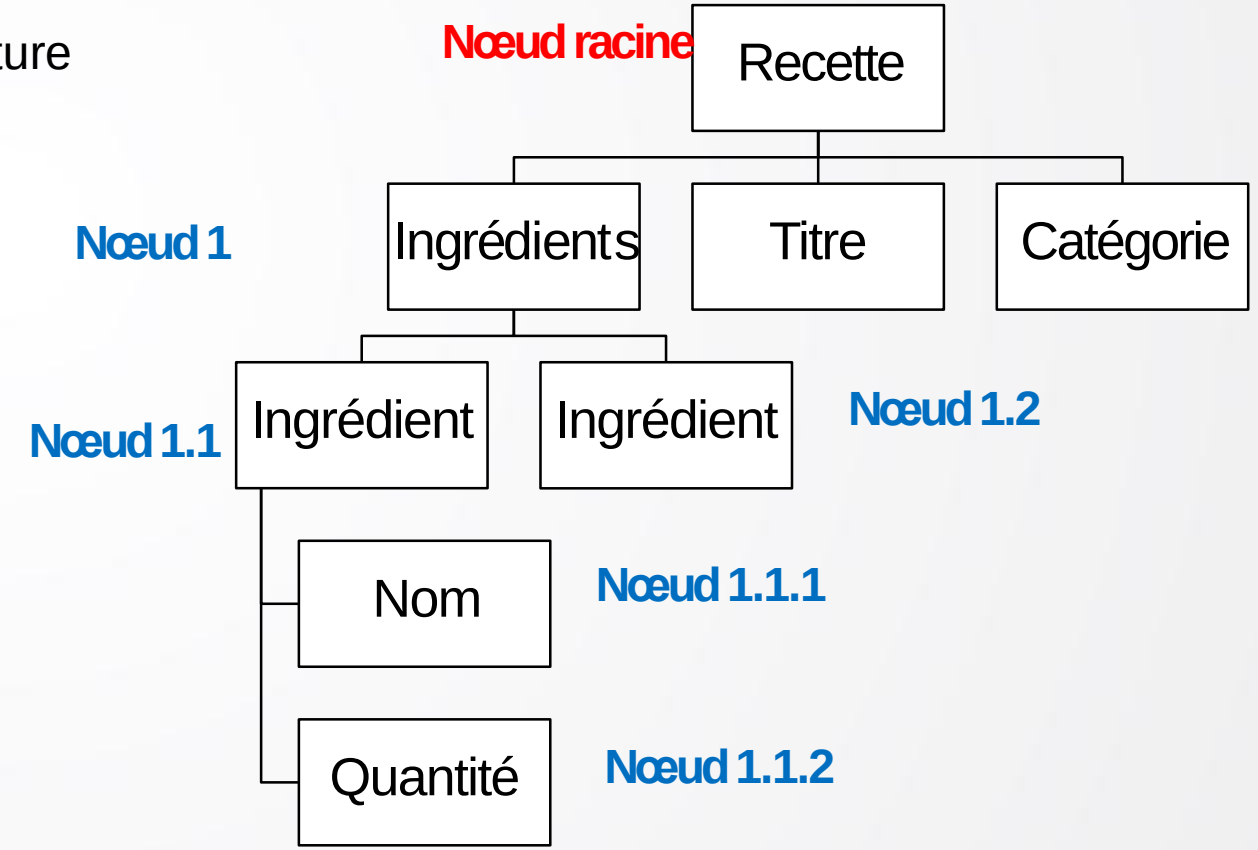
Parseur XML ?

- Le parseur XML permet de créer une structure hiérarchique contenant les données contenues dans le document XML.
- Il existe deux types de parseurs XML:
  - DOM (Document Object Model) : permet d'accéder et d'agir d'une manière directe sur le contenu et la structure de l'arbre XML.
  - SAX (Simple API for XML) : permet de réagir sur le contenu et la structure d'une document XML pendant une lecture séquentielle.



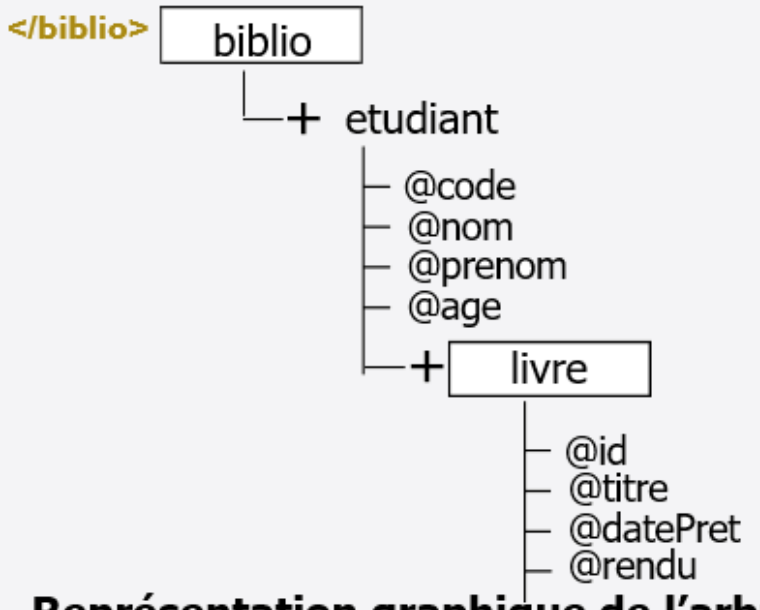
Arborescence XML

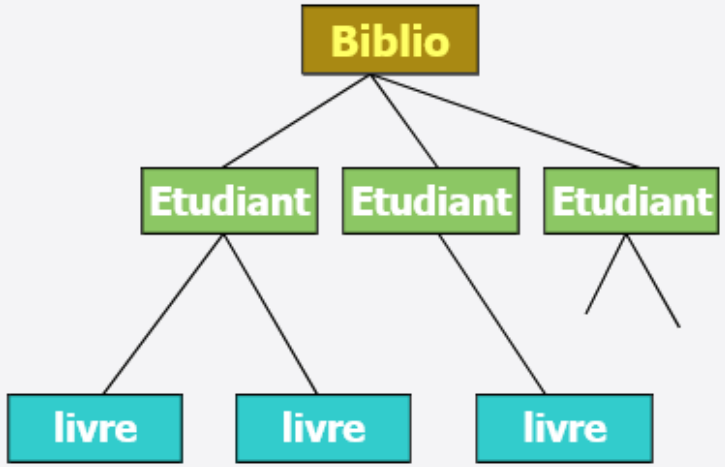
- Un document XML est composé de plusieurs nœuds.
- L'arborescence d'un document XML est la structure hiérarchique des nœuds.



Exemple de document XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<biblio>
   <etudiant code="1" nom="A" prenom="B" age="23">
      <livre id="534" titre="java" datePret="2006-11-12" rendu="oui"/>
      <livre id="634" titre="XML" datePret="2006-11-13" rendu="non"/>
   </etudiant>
   <etudiant code="2" nom="C" prenom="D" age="22">
      <livre id="33" titre="E-Commerce" datePret="2006-1-12" rendu="non"/>
   </etudiant>
</biblio>
```

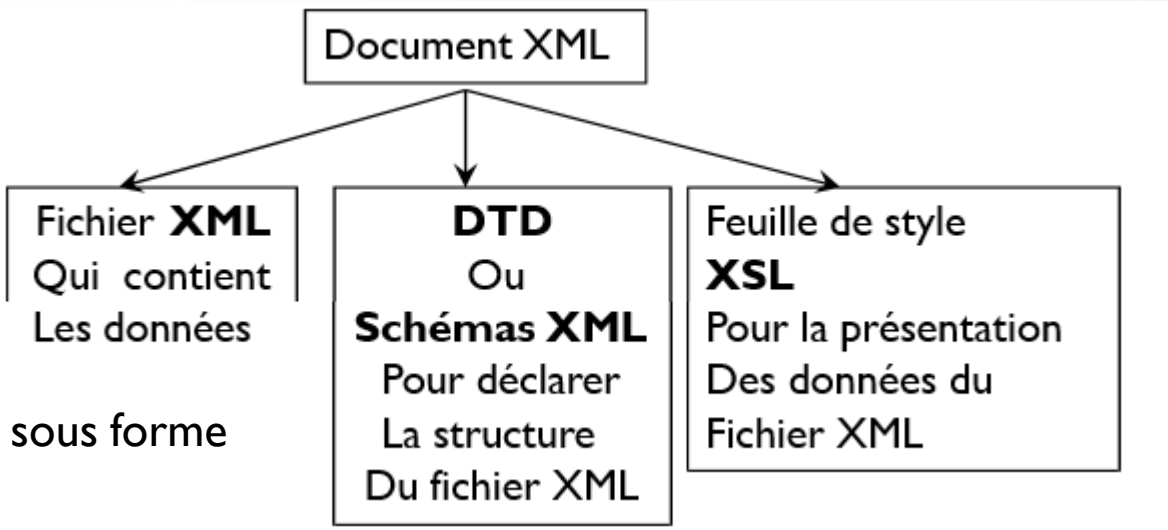




Représentation graphique de l'arbre XML

Structure d'un document XML

Un document XML se compose de 3 fichiers



- Le fichier XML stocke les données du document sous forme d'un arbre
- DTD (DataType Definition) ou Schémas XML définit la structure du fichier XML
- La feuille de style définit la mise en forme des données de la feuille xml

Structure d'un document XML 1/8

- Un document XML comporte :

  -une prologue.

  -l'arbre des éléments.

  -éventuellement des commentaires

Structure d'un document XML 2/8

- **La prologue**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

La prologue XML:

- est une instruction de traitement destinée à l'application chargée du traitement du document XML
- est facultative, mais fortement conseillée
- décrit:
  - la version du langage XML → version="1.0"
  - le codage des caractères (par défaut UTF-8) → encoding="UTF-8"
  - La dépendance à des document extérieurs → standalone="yes"

Structure d'un document XML 3/8

- **Les nœuds XML**

Il existe trois types de nœuds XML:

1. Les éléments
2. Les attributs
3. Les entités

Structure d'un document XML 4/8

1 Les éléments

- Un élément s'ouvre et se ferme par une balise
- Le nom de l'élément est repris dans la balise ouvrante et dans la balise fermante.

<categorie>Dessert**</categorie>**

Balise ouvrante

Balise fermante

Structure d'un document XML 5/8

2 Les attributs

- L'attribut se trouve dans la balise ouvrante d'un élément
- L'attribut n'est pas repris dans la balise fermante
- Un élément peut contenir plusieurs attributs
- Un même attribut ne peut pas être présent qu'une seule fois dans un élément
- L'ordre des attributs n'a pas d'importance au sein d'un élément
- La valeur de l'attribut est indiquée entre guillemets

```
<quantite unite ="g" >100</quantite>
```

Structure d'un document XML 6/8

2 Les entités

- Certains caractères ont un sens particulier en XML (caractères speciaux)

Exemple: >, &, "

- Les entités ont été prédéfinies afin de pouvoir utiliser les caractères réservés

- Une entité est une chaîne de caractère commençant par & et se terminant par ;&entite;
- Une entité est remplacée par la chaîne de caractère qu'elle représente.

Structure d'un document XML 7/8

Les entités prédéfinies

Caractère	Entité
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

Exemple:

<message>salaire < 1000</message>

Structure d'un document XML 8/8

- **Les commentaires**

```
<!-- This is a comment -->
```

Les commentaires sont ignorés lors de l'interprétation du document XML.

Les règles syntaxiques

- Un document XML a un seul élément racine.
- Un élément peut:
  - Être vide

```
<vide/>
```

  - Contenir une chaîne de caractères

```
<categorie>Dessert</categorie>
```

  - Contenir des éléments fils (qui doivent être correctement imbriqués)

```
<ingredient>
   <nom>beurre</nom>
   <quantite>100</quantite>
</ingredient>
```

- XML est sensible à la casse

<Categorie>incorrect</categorie>

Document XML bien formé

XML

```
<produit quantite= "80">
      <id> 123</id>
<produit>
```

Document XML **bien formé**

Respect de la syntaxe XML

Document XML bien formé

- Pour qu'un document soit bien formé, il doit obéir à 4 règles :
  - Un document XML ne doit posséder qu'une seule racine
  - Tous les éléments doivent être fermés
  - Les éléments contenus et contenant doivent être imbriqués.
  - La valeurs des attributs s'écrit entre guillemets

Document XML bien formé

- Un document XML ne doit posséder qu'un seul élément racine qui contient tous les autres.
  - Un document XML est un arbre.

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  TSIG Etudes
</formation>
<durée>
  1755 heures
</durée>
```

Document bien formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée>
    1755 heures
  </durée>
</formation>
```

Document XML bien formé

- Tous les éléments doivent être fermés
  - A chaque balise ouvrant doit correspondre une balise fermante.
  - A défaut, si un élément n'a pas de contenu, il est possible

d'agréger la balise fermante à la balise ouvrant en terminant celle-ci par un "slash".

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755">
</formation>
```

Document bien formé:

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
</formation>
```

Document XML bien formé

- Les éléments contenus et contenant doivent être imbriqués.
  - Tous les éléments fils doivent être contenus dans leur père.
  - Si un document XML est un arbre, un élément est une branche.

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
  <module id="100">
  <sequence id="525">
  </module>
  </sequence>
</formation>
```

Document bien formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
  <module id="100">
    <sequence id="110">
    </sequence>
  </module>
</formation>
```

Document XML bien formé

- La valeurs des attributs s'écrit entre guillemets.

Document mal formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur=1755 />
  <module id=100>
  </module>
  <module id=110>
  </module>
</formation>
```

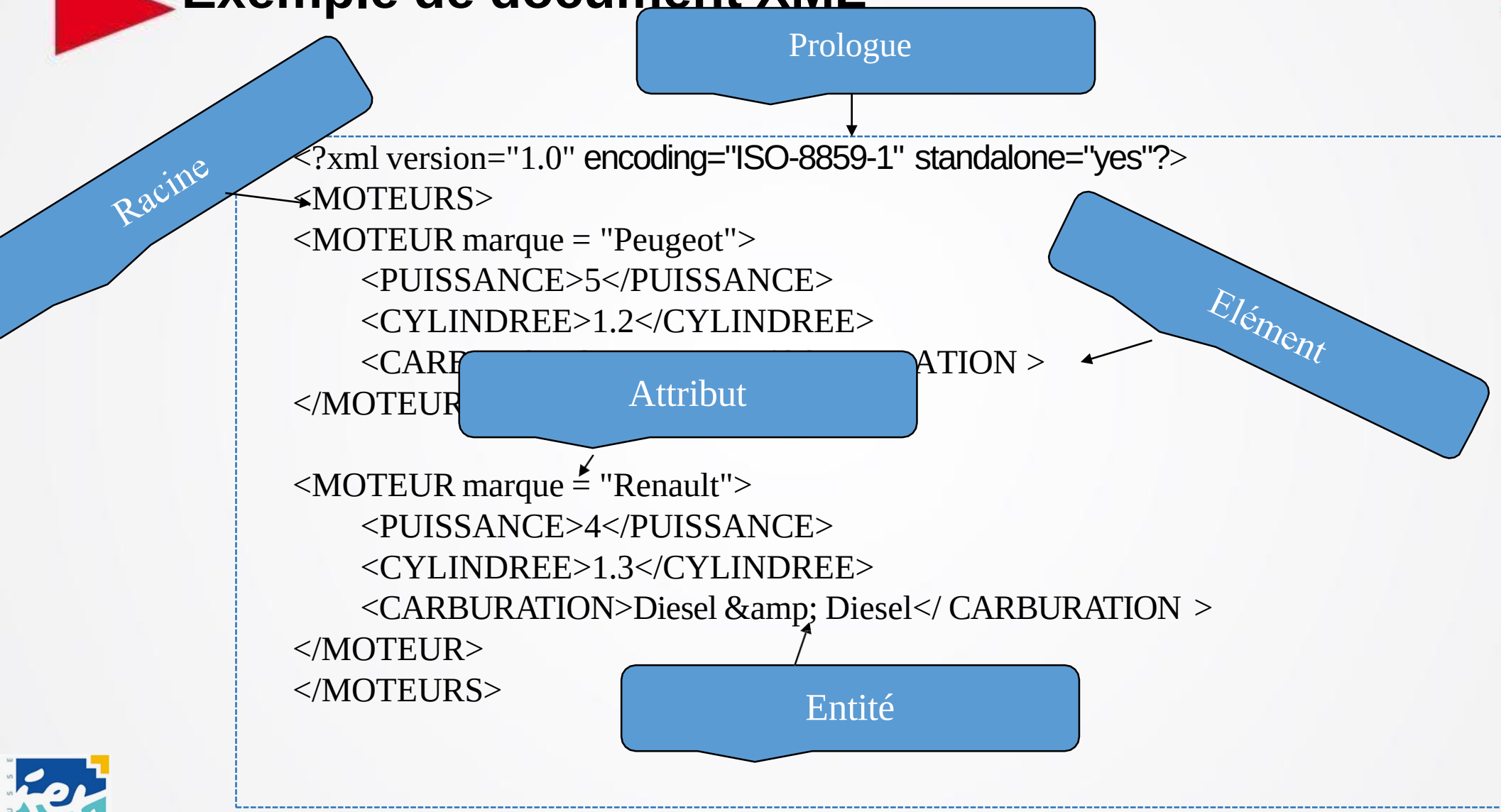
Document bien formé :

```
<?xml version="1.0" ?>
<formation>
  <nom>
    TSIG Etudes
  </nom>
  <durée valeur="1755" />
  <module id="100">
  </module>
  <module id="110">
</module>
</formation>
```

Document XML valide

- Un document XML valide est un document XML bien formé qui possède une DTD (*Document Type Definition*) ou un schéma XML
- La DTD est une série de spécifications déterminant les règles structurelles des documents xml qui lui sont associés.
  - La DTD spécifie :
    - Le nom des balises associées à tous les éléments,
    - Pour chaque balise, les attributs possibles et leur type,
    - Les relations contenant-contenu entre les éléments et leur cardinalité,
    - Les entités (raccourcis) internes et externes

Exemple de document XML



Grammaire

1 DTD

- Une DTD (Document Type Definition) est une grammaire qui permet de définir une structure

  type de document XML.

2 XSD

- XML Schema est un langage de description de format de document XML permettant de

  définir la structure et le type de contenu d'un document XML.[2]

- Cette définition permet notamment de vérifier la validité de ce document.

DTD

Déclaration des éléments

- La déclaration d 'une nouvelle balise se fait grâce à l 'instruction ELEMENT. La syntaxe est:

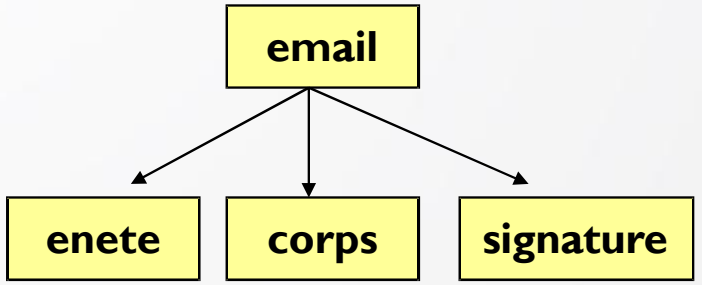
  <!ELEMENT nom contenu>

- Le Contenu peut être choisi parmi cinq modèles:
  - Modèle texte #PCDATA
    - exemple <!ELEMENT paragraphe (#PCDATA)>

    Pour déclarer une balise <paragraphe> qui reçoit du texte

    <paragraphe>Texte</paragraphe>
  - Modèle séquence d 'éléments fils :
    - Une séquence définit les éléments fils autorisés à apparaître dans une balise.
    - Exemple : **<!ELEMENT email (entete, corps, signature)>**

```
<email>
   <entete>….</entete>
   <corps>…..</corps>
   <signature>……</signature>
</email>
```



Déclaration des éléments

- Modèle séquence d ’éléments fils :
  - Chaque nom d ’élément peut être suffixé par un indicateur d ’occurrence.
  - Cet indicateur permet de préciser le nombre d ’instances de l ’élément fils que l ’élément peut contenir. Ces indicateurs sont au nombre de 3.
    - element**?** : Indique que l ’élément peut apparaître zéro ou une fois;
    - element**+** : Indique que l ’élément peut apparaître une ou plusieurs fois;
    - element***** : Indique l ’élément peut apparaître zéro, une ou plusieurs fois.
  - L ’absence de cet indicateur d ’occurrence indique que l ’élément peut apparaître une seule fois au sein de son élément parent.
  - On peut utiliser les parenthèses pour affecter un indicateur d ’occurrence à plusieurs élément

Déclaration des éléments

- Modèle séquence d ’éléments fils : (suite)
  - Exemple:

    <!ELEMNT livre (preface,introduction,(intro-chap,contenu-chap)+)>

    Nous pourrons alors écrire :

    <livre>

    <preface>…..</preface>

    <introduction> …. </introduction>

    <intro-chap> …..</intro-chap>

    <contenu-chap> …. </contenu-chap>

    <intro-chap> …..</intro-chap>

    <contenu-chap> …. </contenu-chap>

    </livre>
  - Le choix le plus libre qu ’une définition de document puisse accorder correspond à la définition suivante :

    <!ELEMENT paragraphe (a | b | c)*>

Déclaration des éléments

- Modèle mixte :
  - Le modèle mixte naît de l 'utilisation conjointe de #PCDATA et d 'éléments fils dans une séquence.
  - Un tel modèle impose d 'utiliser un ordre libre et de disposer #PCDATA en première position. De plus l 'indicateur d 'occurrence doit être *.
  - Exemple : <!ELEMENT p(#PCDATA | u | i)* >
  - En complétant ces déclarations par les deux suivantes :

    <!ELEMENT u (#PCDATA)>

    <!ELEMENT i (#PCDATA)>

  la balise p peut être alors utilisée :

  Dans son livre<u> "la huitième couleur" </u>,Tirry Pratchet nous raconte les aventures du premier touriste du Disque-Monde , le dénommé Deux Fleurs

Déclaration des éléments

- Modèle Vide.
  - Le mot EMPTY permet de définir une balise vide.
  - La balise <br> de HTML reproduite en XML en constitue un parfait exemple :

<!ELEMENT livre EMPTY >

  - Une balise vide ne pourra en aucun cas contenir un élément fils.
  - Un élément vide peut avoir des attributs.

<livre id="534" titre="java" datePret="2006-11-12" rendu="oui"/>

- Modèle différent.
  - Ce dernier modèle permet de créer des balises dont le contenu est totalement libre.
  - Pour utiliser ce modèle, on utilise le mot ANY
  - Exemple : **<!ELEMENT disque ANY>**

l 'élément disque peut être utilisé librement.

Déclaration d ’attributs

- La déclaration d ’attributs dans une DTD permet de préciser quels attributs peuvent être associés à un élément donnée.
- Ces attributs reçoivent de plus une valeur par défaut.
- La syntaxe de déclaration d ’un attribut est :

  <!ATTLIST nom-balise nom-attribut type-attribut présence >
- Il est possible de déclarer plusieurs attribut au sein d ’une même balise ATTLIST.
- Exemples:

  <!ELEMENT personne EMPTY>

  <!ATTLIST personne nom CDATA #REQUIRED >

  <!ATTLIST personne prenom CDATA #REQUIRED >

  OU:

  <!ELEMENT personne EMPTY>

  <!ATTLIST personne

  nom CDATA #REQUIRED prenom CDATA #REQUIRED>

Déclaration d ’attributs

- Les types d’attributs les plus courants sont:
  - Type **CDATA** : signifie que la valeur de l ’attribut doit être une chaîne de caractères;
  - Type **NMTOKEN** : signifie que la valeur de l ’attribut doit être une chaîne de caractères ne contenant pas d’espaces et de caractères spéciaux;
  - Type **ID** : Ce type sert à indiquer que l'attribut en question peut servir d'*identifiant* dans le fichier XML. Deux éléments ne pourront pas posséder le même attribut possédant la même valeur.
  - Type énuméré : une liste de choix possible de type (A|B|C) dans laquelle A, B, C désignent des valeurs que l ’attribut pourra prendre.
    - Une valeur booléenne peut donc être représentée par en XML par les deux déclarations suivantes:

```
<!ELEMENT boolean EMPTY >
<!ATTLIST boolean value (true | false ) 'false'>
```

Déclaration d ’attributs

- Le terme Présence permet de définir comment doit être gérée la valeur de l ’attribut. Il existe 4 types de présences :
  - La valeur par défaut de l ’attribut(comme dans le cas de l ’exemple de l ’élément boolean)
  - **#REQUIRED** Indique que l ’attribut doit être présent dans la balise et que sa valeur doit être obligatoirement spécifiée.
  - **#IMPLIED** Indique que la présence de l ’attribut est facultative;
  - **#FIXED** "valeur " Fixe la valeur de l ’attribut.

Déclaration d ’entités

- Une entité définit un raccourcis vers un contenu qui peut être soit une chaîne de caractères soit le conteu d’un fichier.
- La déclaration générale est : <!ENTITY nom type>
- Exemple d ’entité caractères:
  <!ENTITY **eacute** "é">
  c ’est comme si je définie une constante eacaute qui représente le caractère « é ».
- Exemple d’entité chaîne de caractères:
  - <!ENTITY **ADN** "Acide désoxyribonucléique">
  - Pour utiliser cette entité, on écrit : **&ADN;**
- Exemples d ’entité relative au contenu d ’un fichier:
  <!ENTITY autoexec SYSTEM "file:/c:/autoexec.bat " >
  <!ENTITY notes SYSTEM "../mesnotes/notes.txt" >
  Pour utiliser une entité dans un document XML, on écrit:
  &nom-entité;

Exemple 1

- Un opérateur Télécom fournit périodiquement, à l'opérateur de réglementation des télécoms ANRT un fichier XML qui contient les clients de cet opérateur. Chaque client possède plusieurs abonnements et chaque abonnement reçoit plusieurs factures. Un exemple de fichier XML correspondant à ce problème est le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<operateur>
    <client code="1" nom="Ahmed">
        <abonnement num="123" type="GSM" dateAb="2006-1-11">
            <facture numFact="432" dateFact="2006-2-11" montant="350.44" reglee="oui"/>
            <facture numFact="5342" dateFact="2006-3-11" montant="450" reglee="oui"/>
           <facture numFact="5362" dateFact="2006-4-13" montant="800.54" reglee="non"/>
        </abonnement>
        <abonnement num="2345" type="FIXE" dateAb="2006-12-1">
            <facture numFact="5643" dateFact="2007-1-12" montant="299" reglee="oui"/>
            <facture numFact="6432" dateFact="2007-2-12" montant="555" reglee="non"/>
        </abonnement>
    </client>
    <client code="2" nom="abbassi">
        <abonnement num="7543" dateAb="2006-12-1" type="GSM">
            <facture numFact="7658" dateFact="2007-2-12" montant="350.44" reglee="oui"/>
            <facture numFact="7846" dateFact="2007-3-12" montant="770" reglee="non"/>
        </abonnement>
    </client>
</operateur>
```

Schémas XML

Présentation de XSD

- Alternative au DTD
- Recommandations W3C
- Issu de XML

=> Tous les outils (validateurs, parseurs, processeurs, ...) mais également les langages (XSLT, XPath, ...) qui permettent de travailler les documents XML sont utilisables sur des XSD.

DTD (Document Type Definition)

```
<!ELEMENT boutique (telephone*)>
<!ELEMENT telephone (marque, modele)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT modele (#PCDATA)>
```

```
<?xml version = "1.0" ?>
<boutique>
 <telephone>
  <marque>Samsung</marque>
  <modele>Galaxy S5</modele>
 </telephone>
 <telephone>
  <marque>Apple</marque>
  <modele>iPhone 6</modele>
 </telephone>
</boutique>
```

DTD vs XSD

DTD	XSD
Nouveau langage → Syntaxe particulière	Langage issu de XML → Syntaxe XML
Types de données limités → PCDATA , CDATA	Types de données plus riches → integer, byte, string, float, …
Nombre d'occurrence très général → *, + et ?	Nombre d'occurrence plus précis [1,100] ,]2,100]
Aucune contrainte sur le contenu des éléments et attributs	Définition des contraintes sur le contenu des éléments/attributs → mot de passe de longueur 8 email contenant le caractère @
	Extensible

Apports des schémas

- Conçu pour pallier aux déficiences pré citées des DTD, XML

  Schema propose, en plus des fonctionnalités fournies par les DTD, des nouveautés :
  - Le typage des données est introduit, ce qui permet la gestion de booléens, d'entiers, d'intervalles de temps... Il est même possible de créer de nouveaux types à partir de types existants,
  - La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément,
- Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif.
- Le support des espaces de nom.
- Les Schémas XML sont des documents XML ce qui signifie d'un parseur XML permet de les manipuler facilement

Vers un document XML valide

XML

```
<produit quantite= "80">
    <id> 123</id>
<produit>
```

Document XML **bien formé**

Respect de la syntaxe XML

XSD

Document XML **valide**

Respect des règles XSD

Structure d'un schéma XML

- Un document schema XML est défini dans un fichier dont l'extension est ***.xsd**
- Comme tout document XML, un schéma XML commence par la prologue XML et a un élément racine
- L'élément **<xs:schema>** est la racine de tout document Schema XML

Fichier XSD

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

Déclaration des éléments

```
<xs:element name="theName" type="theType" />
```

Valeur par défaut

```
<xs:element name="firstName" type="xs:string" default="Mickael" />
```

L'attribut default de l'élément *firstName* précise une valeur au cas où elle serait absente

Valeur fixée

```
<xs:element name="firstName" type="xs:string" fixed="Mickael" />
```

L'attribut fixed de l'élément *firstName* précise une valeur et ne peut être modifiée

Déclarations d'attributs

- Un attribut, dans un schéma, se déclare avec la balise **<xsd:attribute>**.
- Un attribut ne peut être que de type simple
- Exemple:

```
<xsd:element name="contacts" type="typeContacts"/>
<xsd:element name="remarque" type="xsd:string"/>
    <!-- déclarations de types ici -->
<xsd:complexType name="typeContacts">
     <!-- déclarations du modèle de contenu ici -->
   <xsd:attribute name="maj" type="xsd:date" />
</xsd:complexType>
```

- Ici on déclare que contacts est un élément de type complexe et qu'il possède un attribut maj de type date

Contraintes d'occurrences pour les attribus

- L'élément **attribute** d'un Schema XML peut avoir trois attributs optionnels :
  - **use** : indique le présence , il peut prendre pour valeur required(obligatoire), optional(facultatif) ou prohibited (ne doit pas apparaitre)
  - **default** : pour indiquer la valeur par défaut
  - **fixed** : indique l'attribut est renseigné, la seule valeur que peut prendre l'attribut déclaré est celle de l'attribut **fixed**. Cet attribut permet de "réserver" des noms d'attributs pour une utilisation future, dans le cadre d'une mise à jour du schéma.
- Exemple :

```
<xsd:attribute name="maj" type="xsd:date" use="optional"
   default="2003-10-11" />
```

Déclaration des attributs

```
<xs:attribute name="theName" type="theType" use="required" />
```

(required ou optional)

Valeur par défaut

```
<xs:attribute name="remark" type="xs:string" default="Remarque à préciser" />
```

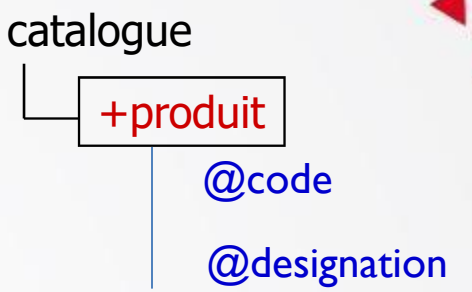
L'attribut default de l'attribut *remark* précise une valeur au cas où elle serait absente

Valeur fixée

```
<xs:attribute name="remark" type="xs:string" fixed="Remarque à préciser" />
```

L'attribut fixed de l'attribut *remark* précise une valeur et ne peut être modifiée

Premier Exemple schéma xml

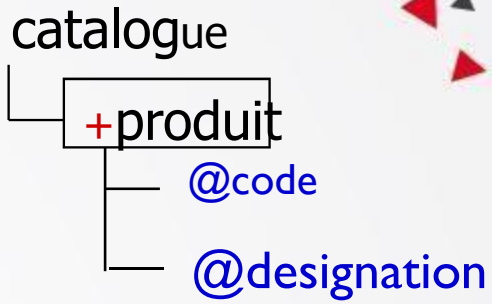



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="catalogue">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="produit" type="T_PRODUIT"
                             maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

 <xsd:complexType name="T_PRODUIT">
    <xsd:attribute name="code" type="xsd:int" use="required"/>>
    <xsd:attribute name="designation" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

Fichier XML respectant le schéma précédent




```
<?xml version="1.0" encoding="UTF-8"?>
<catalogue
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:noNamespaceSchemaLocation="file:/C:/m/catalogue.
  xsd">
  <produit code="1" designation="PC HP 650"/>
  <produit code="2" designation="Imprimante HP 690"/>
</catalogue>
```

Déclaration d'élément ne contenant que du texte avec un (ou plusieurs) attribut(s)

- Un tel élément est de type complexe, car il contient au moins un attribut.
- Afin de spécifier qu'il peut contenir également du texte, on utilise l'attribut **mixed** de l'élément **<xsd:complexType>**.
- Par défaut, **mixed="false"**; il faut dans ce cas forcer **mixed="true"**.
- Par exemple:

```
<xsd:element name="elt">
 <xsd:complexType mixed="true">
  <xsd:attribute name="attr" type="xsd:string" use="optional" />
 </xsd:complexType>
</xsd:element>
```

Les types de données 1/2

- Les attributs sont de types simples
- Les éléments sont de:
  - **Types simples**

    un élément de type simple ne peut comporter ni attributs, ni de sous éléments

  - **Types complexes**

    un élément de type complexe permet d'exprimer des sous éléments et permet également d'y associer des attributs

Les types de données 2/2

Eléments de type simple

```
<film/>
```

```
<film>Gladiator</film>
```

Eléments de type complexe

```
<film type="action " />
```

```
<film type= "action ">
      Gladiator
</film>
```

```
<film>
   <annee>2000</annee>
  <realisateur>R.Scott</realisateur>
</film>
```

```
<film type= "action " >
   <annee>2000</annee>
  <realisateur>R.Scott</realisateur>
</film>
```

Dérivation

- Les types simples et complexes permettent déjà de faire plus de choses que les déclarations dans le langage DTD.
- Il est possible de raffiner leur déclaration de telle manière qu'ils soient
  - une "restriction"
  - ou une extension d'un type déjà existant, en vue de préciser un peu plus leur forme.
- Nous allons nous limiter dans ce cours d'initiation à la restriction des types simples.

Les types simples 1/8

Les principaux types simples prédéfinis

- xs:int
- xs:boolean
- xs:string
- xs:long
- xs:float
- xs:positiveInteger : 1, 2, …
- xs:negativeInteger : …, -2, -1
- xs:nonNegativeInteger : 0, 1, 2, …
- xs:nonPositiveInteger : …, -2, -1, 0
- xs:unsignedLong : 0, 1, … 18446744073709…

Les types simples: restriction

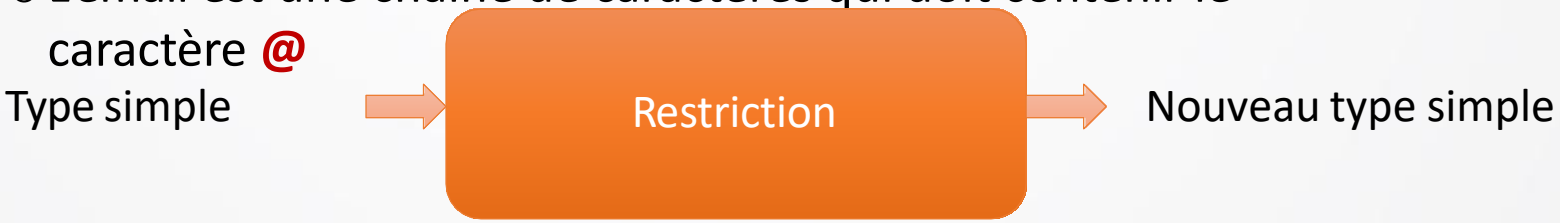
Les types simples dérivés

- On peut créer de nouveaux types simples en dérivant des types simples existants (prédéfinis ou dérivés)

- Un nouveau type simple peut être défini par **restriction** ou **extension**

Exemple:

- L'age est un entier compris entre **1** et **100**
- L'email est une chaîne de caractères qui doit contenir le caractère **@**



Les types simples: restriction

Les types simples dérivés

- Les restrictions sur les types simples permettent de dériver de nouveaux types à partir de types existants

- Les restrictions passent par l'utilisation des **facettes**

- Une facette permet de définir des contraintes sur le nouveau type à créer

Les types simples: restriction

Les types simples dérivés

- La création de nouveaux types simples est réalisée avec la balise **<xs:simpleType>**

```
<xs:simpleType name="newType" >
        ...
</xs:simpleType>
```

- La restriction est exprimée avec la balise **<xs:restriction>**

```
<xs:simpleType name="newType" >
        <xs:restriction base="type" >
                ...
        </xs:restriction>
</xs:simpleType>
```

Nouveau Type simple

Type simple de départ

Les types simples: restriction

Les principales facettes

- **Facette length**

```
<xs:element name=" password" type ="passwordType" />
<xs:simpleType name ="passwordType" >
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
</xs:simpleType>
```

- **Facette minLength, maxLength**

```
<xs:element name="password"  type="passwordType" />
<xs:simpleType name =" passwordType" >
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
</xs:simpleType>
```

Les types simples: restriction

Les principales facettes

- **Facette minLInclusive, minExclusive, maxInclusive, maxExclusive**

```
<xs:element name=" age" type ="ageType" />
<xs:simpleType name="ageType">
   <xs:restriction base="xs:int">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="100" />
   </xs:restriction>
</xs:simpleType>
```

Les types simples: restriction

Les principales facettes

- **Facette enumeration**

```
<xs:element name=" sexe" type ="sexeType" />
<xs:simpleType name="sexeType" >
  <xs:restriction base="xs:string">
    <xs:enumeration value="homme" />
    <xs:enumeration value="femme" />
    <xs:enumeration value="indéterminé" />
  </xs:restriction>
</xs:simpleType>
```

Trois valeurs sont autorisées

Les types simples: restriction

Les principales facettes

- **Facette pattern**

```
<xs:element name="email" type ="emailType" />
<xs:simpleType name="emailType">
  <xs:restriction base="xs:string">
    <xs:pattern value= " [a-z]+@[a-z]+ " />
  </xs:restriction>
</xs:simpleType>
```

Toutes les chaînes de caractères de type *emailType* doivent respecter ce pattern

[a-z]*	0 ou plusieurs lettre(s)
([a-z][A-Z])+	1 ou plusieurs paires de lettres min et maj sToP, Stop,STOP,stop
male|female	Liste de choix
[a-zA-Z0-9]{8}	8 caractères (chiffre, lettre min, lettre maj)

Types complexes

- Un élément de type simple ne peut contenir de sous-élément ou des attributs.
- Un type complexe est un type qui contient soit des éléments fils, ou des attributs ou les deux.
- On peut alors déclarer,
  - Des séquences d'éléments,
  - Des types de choix

Les types complexes 1/6

- Un élément **de type complexe** peut contenir d'autres **éléments et / ou des attributs**

- **Quatre** combinaisons d'éléments complexes sont à distinguer
  - **Eléments vides** qui ne contiennent que **des attributs**
  - **Eléments de type simple** qui contiennent **des attributs**
  - **Eléments** qui peuvent contenir **des sous éléments**
  - **Eléments** qui contiennent **des attributs** et **des sous éléments**

- La création d'un éléments de type complexe est réalisée avec la balise **<xs:complexType>**

```
<xs:complexType name="newType" >
        ...
</xs:complexType>
```

Types complexes: 2/6

- Nous savons déjà comment, dans une DTD, nous pouvons déclarer un élément comme pouvant contenir une suite de sous-éléments, dans un ordre déterminé.
- Il est bien sûr possible de faire de même avec un schéma.
- On utilise pour ce faire l'élément **xsd:sequence**, qui reproduit l'opérateur, du langage DTD. Exemple:

```
<xsd:complexType>
 <xsd:sequence>
  <xsd:element name="nom" type="xsd:string" />
  <xsd:element name="prénom" type="xsd:string" />
  <xsd:element name="dateDeNaissance"
  type="xsd:date"/>
  <xsd:element name="adresse" type="xsd:string" />
  <xsd:element name="email" type="xsd:string" />
  <xsd:element name="tel" type="numéroDeTéléphone"
  />
 </xsd:sequence>
</xsd:complexType>
```

Les types complexes 3/6

Eléments qui peuvent contenir des sous éléments

```
<person>
  <name>…</name>
  <firstName>…</firstName>
  <old>…</old>
  <email>…</email>
</person>
```

sequence exprime que les sous éléments doivent apparaître **dans l'ordre** spécifié

```
<xs:element name="person" type="personType " />
<xs:complexType name="personType " >
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="old" type="ageType" />
      <xs:element name="email" type="emailType" />
   </xs:sequence>
 </xs:complexType>
```

Les types complexes 4/6

Eléments qui peuvent contenir des sous éléments

Indicateurs d'ordre

- XSD permet d'exprimer trois sortes d'indicateurs d'ordre:
  - sequence
  - all
  - choice

- **all** tous les sous éléments peuvent apparaître dans **n'importe quel ordre**

```
<xs:complexType name= "personType">
  <xs:all>
      <xs:element name="name" type="xs:string" />
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="old" type="ageType" />
      <xs:element name="email" type="emailType" />
  </xs:all>
</xs:complexType>
```

```
<person>
  <name>…</name>
  <email>… </email>
  <old>…</old>
  <firstName>…</firstName>
</person>
```

Les types complexes 5/6

Eléments qui peuvent contenir des sous éléments

Indicateurs d'ordre

- **choice** exprime qu'**un seul** élément parmi tous les sous éléments peut apparaître

```
<xs:complexType name= "personType">
  <xs:choice>
      <xs:element name="name" type="xs:string" />
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="old" type="ageType" />
      <xs:element name="email" type= "emailType" />
  </xs:choice>
</xs:complexType>
```

```
<person>
   <name>…</name>
</person>
```

Les types complexes 6/6

Eléments qui peuvent contenir des sous éléments

Indicateurs d'occurence

- maxOccurs : précise le nombre d'occurrence maximum
- minOccurs : précise le nombre d'occurrence minimum

- Si les valeurs de maxOccurs ou minOccurs ne sont pas explicitement précisées, la valeur par **défaut est de 1**

- Pour définir une valeur infinie, fixer la valeur à **unbounded**

```
<xs:complexType name= "personType">
 <xs:sequence>
     <xs:element name="name" type="xs:string" />
     <xs:element name="firstName" type="xs:string" />
     <xs:element name="old" type="ageType" minOccurs="0" />
     <xs:element name="email" type= " emailType" minOccurs="2"
                                                 maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

old est un élément optionnel

Espace de noms XML 1/6

Problème

```
<employe>
  <id>E0000001</id>
  <nom>Smith</nom>
  <prenom>John </prenom>
</employe>
```

```
<departement>
   <id>D001</id>
   <nom>Marketing</nom>
</departement>
```

Fusion des 2 documents

```
<entreprise>
 <departement>
   <id>D001</id>
   <nom>Marketing</nom>
   <employe>
     <id>E0000001</id>
     <nom>Smith</nom>
     <prenom>John </prenom>
   </employe>
 </departement>
</entreprise>
```

Confusion sur le sens des éléments id et nom

Espace de noms XML 2/6

Objectif **Distinguer les éléments et les attributs de différentes documents XML qui ont le même nom**

Solution Utiliser les espaces de noms XML

```
<emp:employe>
  <emp:id>E0000001</emp:id>
  <emp:nom>Smith</emp:nom>
  <emp:prenom>John </emp:prenom>
</emp:employe>
```

```
<dep:departement>
   <dep:id>D001</dep:id>
   <dep:nom>Marketing</dep:nom>
</dep:departement>
```

Fusion des 2 documents

```
<dep:departement>
   <dep:id>D001</dep:id>
   <dep:nom>Marketing</dep:nom>
   <emp:employe>
      <emp:id>E0000001</emp:id>
      <emp:nom>Smith</emp:nom>
      <emp:prenom>John </emp:prenom>
   </emp:employe>
</dep:departement>
```

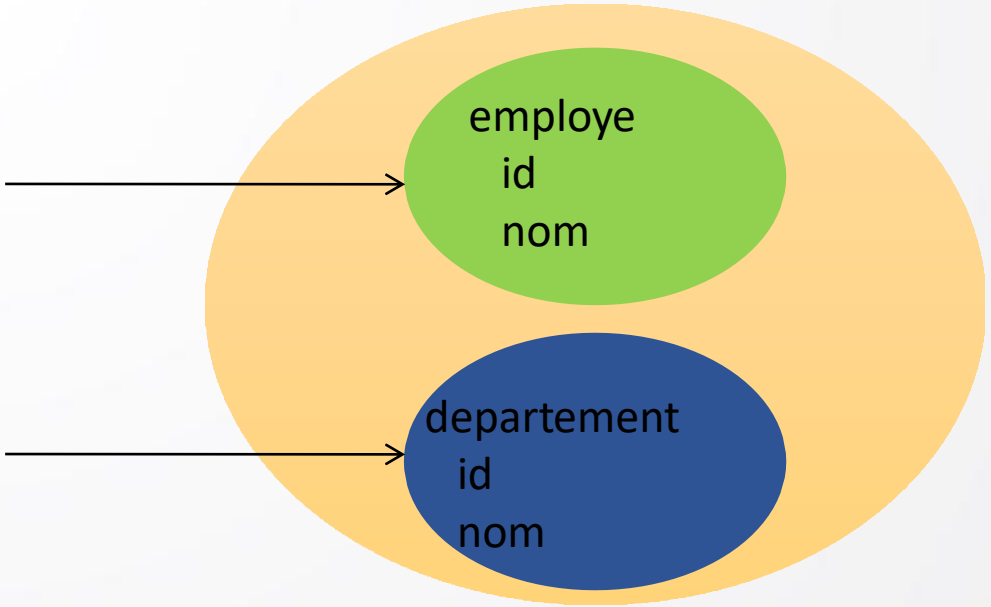
Espace de noms XML 3/6

Déclaration des espaces de noms

- Un espace de nom associe un préfixe à un URI
- L'URI (Uniform Resource Identifier) sert à identifier un espace de noms
- Le préfixe est une chaîne utilisée pour référencer l'espace de nom dans un fichier XML.

URI: http://employe.com
Préfixe: emp

URI: http://departement.com
Préfixe: dep



Espace de noms XML 4/6

Déclaration des espaces de noms

- La déclaration de l'espace de noms se fait au moyen de l'attribut **xmlns**

```
<element xmlns:prefix="URI">
```

```
<emp:employe xmlns:emp="http://emloye.com">
  <emp:id>E0000001</emp:id>
  <emp:nom>Smith</emp:nom>
  <emp:prenom>John </emp:prenom>
</emp:employe>
```

```
<dep:departement xmlns:dep="http://departement.com">
   <dep:id>D001</dep:id>
   <dep:nom>Marketing</dep:nom>
</dep:departement>
```

Espace de noms XML 5/6

Déclaration des espaces de noms

```
<emp:employe xmlns:emp="http://emloye.com">
  <emp:id>E0000001</emp:id>
  <emp:nom>Smith</emp:nom>
  <emp:prenom>John </emp:prenom>
</emp:employe>
```

```
<dep:departement xmlns:dep="http://departement.com">
  <dep:id>D001</dep:id>
  <dep:nom>Marketing</dep:nom>
</dep:departement>
```

Fusion des 2 documents

```
<entreprise xmlns:dep="http://departement.com"
            xmlns:emp="http://emloye.com" >
 <dep:departement>
  <dep:id>D001</dep:id>
  <dep:nom>Marketing</dep:nom>
  <emp:employe>
    <emp:id>E0000001</emp:id>
    <emp:nom>Smith</emp:nom>
    <emp:prenom>John </emp:prenom>
  </emp:employe>
 </dep:departement>
</entreprise>
```

Espace de noms XML 6/6

Déclaration des espaces de noms

Espace de nom pour XSL

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        …
</xsl:stylesheet >
```

Espace de nom pour XSD

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

Espace de noms XSD 1/4

XML Schema Namespace

http://www.w3.org/2001/XMLSchema

schema
element
complexType
string
integer
boolean

Les éléments et les types appartenant au XML Schema Namespace sont utilisés pour écrire un document XSD

Document XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns="http://www.departement.org"
                    targetNamespace="http://www.departement.org">
   <xs:element name="departement" type="depType"/>
   <xs:complexType name="depType">
      <xs:sequence>
                    <xs:element name="id" type="xs:integer"/>
                    <xs:element name="nom" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Les nouveaux types et élément déclarés dans le document XSD appartiennent à un nouveau espace de nom: c'est le tragetNamespace

Target Namespace

http://www.departement.org

departement
id
nom
departemenType

Le targetNamesapce est utilisé par le fichier XML pour la validation

Document XML

```
<departement>
   <id>D001</id>
   <nom>Marketing</nom>
</departement>
```

Espace de noms XSD 2/4

Espaces de noms XSD

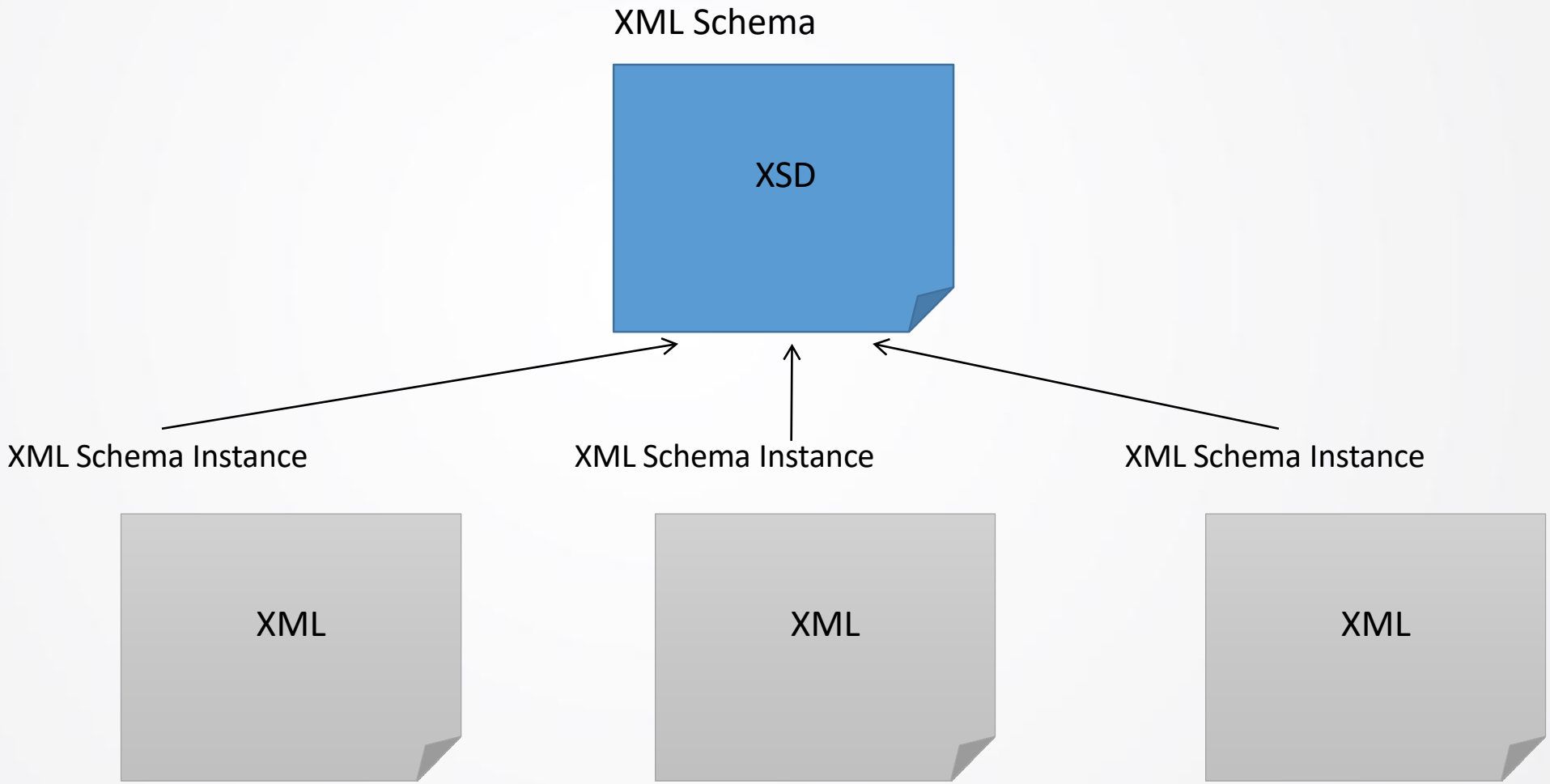
Fichier XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:dep="http://www.departement.org"
           targetNamespace="http://www.departement.org" >
          <xs:complexType name="depType" >
             <xs:sequence>
                  <xs:element name="id" type="xs:integer"/>
                 <xs:element name="nom" type="xs:string"/>
          </xs:sequence>
          </xs:complexType>
          <xs:element name="departement" type=" dep:depType "/>
</xs:schema>
```

xmlns:xs	▪espace de nommage des éléments et types XSD
xmlns:dep	▪espace de nommage des nouveaux types définis par le programmeur
targetNamespace	▪espace de nommage du schema XSD cible ▪C'est l'espace de noms qui sera référé par le fichier XML

Espace de noms XSD 3/4

Association d'un fichier XML à un fichier XSD



Espace de noms XSD 4/4

Association d'un fichier XML à un fichier XSD

Fichier XML

```
<departement xmlns="http://www.departement.org"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.departement.org departement.xsd">
```

xmlns	▪Espace de noms des éléments utilisés dans le fichier XML
xmlns:xsi	▪Il s'agit d'une instance de schema XSD
xsi:schemaLocation	▪Localiser le document XSD

En résumé

- XML est un langage de structuration de données
- Un document XML est structuré à l'aide d'éléments et d'attributs
- Un document XML doit respecter les règles syntaxiques pour qu'il soit bien formé
- XSD est un langage permettant la définition de la structure d'un document XML
- XSD offre une richesse de types
  - ✓ Types simples, types complexes
  - ✓ Restriction, extension
- L'association d'un fichier XML à un fichier XSD passe par l'utilisation des espaces de noms.

Références

- [1]http://www.gchagnon.fr/cours/xml/base.html

- [2]http://fr.wikipedia.org/wiki/XML_Schem
- http://www.liafa.jussieu.fr/~carton/Enseignement/XML/Cours/Schemas/

- http://www.grappa.univ-lille3.fr/~torre/Enseignement/Cours/XML/xmlschema.php

- http://www.teluq.ca/inf6450/mod1/chapitre4.xml