

XML

Quelques bases

Responsable : Patricia Serrano Alvarado
Master INFO 1^{ère} année

Plan

- Introduction générale
- Documents XML
- DTD (Définition de type de document)
- Schéma XML

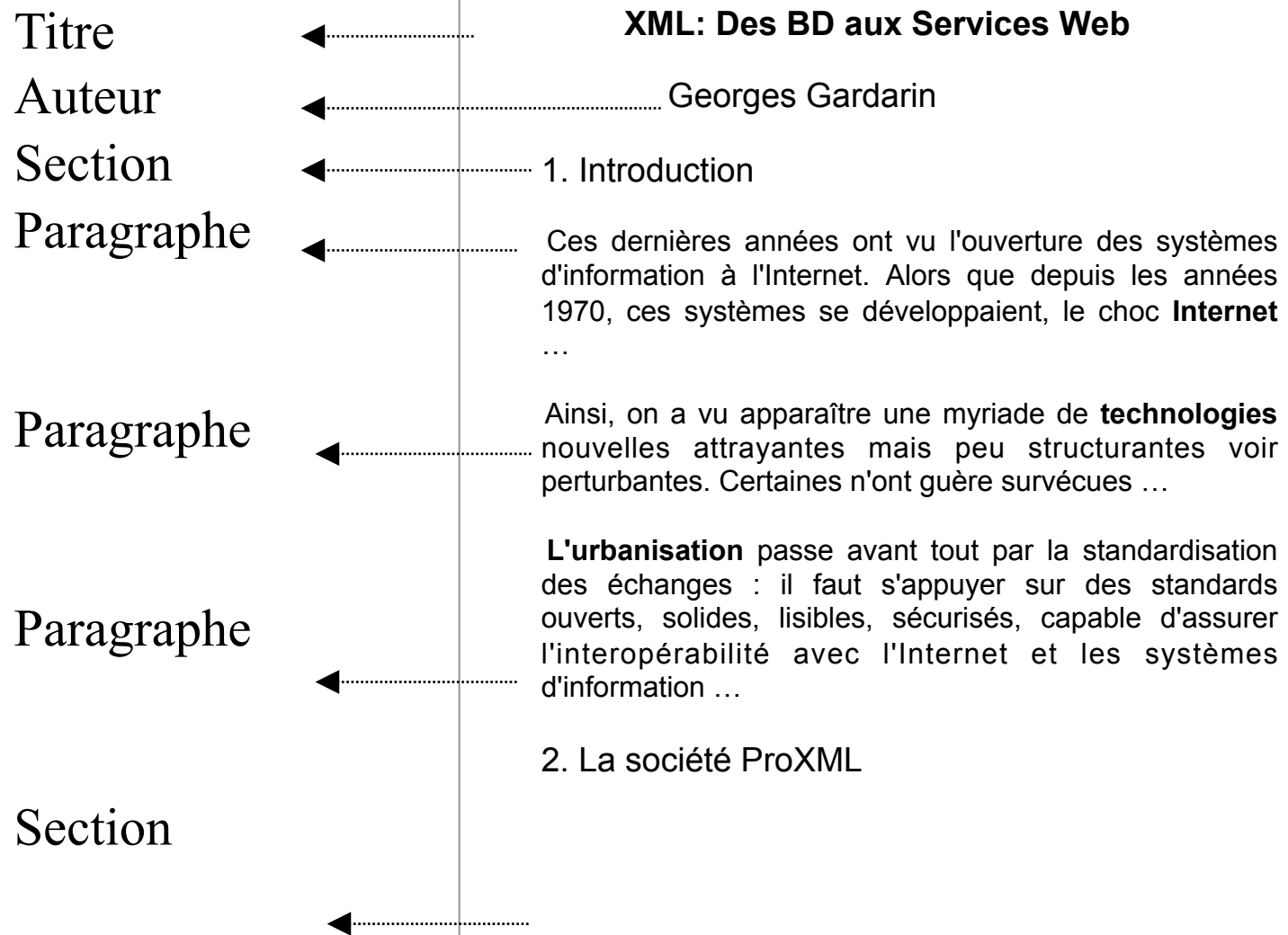
1. Origine et objectifs

- La gestion de données s'est divisée en deux branches vers 1970:
 - BD structurées (Relationnel, Objet)
 - Tables objet-relationnel
 - Langage de requêtes SQL
 - Gestion de documents (GED)
 - Documents balisés
 - Recherche d'information par mots-clés
 - Moteurs de recherche (e.g., Google)
- XML est issu de la Gestion de Documents (GED)

GED

- Séparation du **fond** de la **forme**.
 - Forme = présentation à partir de la structure (style)
 - Fond = structure + données (contenu)
- Langage à balises
 - Encadrement des champs par des balises (*tags*) début et fin
- Multiples précurseurs dont les plus connues :
 - **SGML** pour la structuration
 - **HTML** pour la présentation
- Approches mélangeant parfois le fond et la forme !

Présentation et Structuration



Vue Balisée

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

```
<Livre>
```

```
<Titre>XML : Des BD aux Services Web</Titre>
```

```
<Auteur>Georges Gardarin</Auteur>
```

```
<Section titre="Introduction">
```

```
<Paragraphe>Ces dernières années ont vu l'ouverture des systèmes d'information à l'Internet.  
Alors que depuis les années 1970, ces systèmes se développaient, le choc Internet ...</  
Paragraphe>
```

```
<Paragraphe>Ainsi, on a vu apparaître une myriade de technologies nouvelles attrayantes  
mais peu structurantes voir perturbantes. Certaines n'ont guère survécues ...</  
Paragraphe>
```

```
<Paragraphe>L'urbanisation passe avant tout par la standardisation des échanges : il faut  
s'appuyer sur des standards ouverts, solides, lisibles, sécurisés, capable d'assurer  
l'interopérabilité avec l'Internet et les systèmes d'information ...</Paragraphe>
```

```
</Section>
```

```
<Section titre="La Société ProXML">...</Section>
```

```
</Livre>
```

Les balises peuvent porter
plus ou moins de sémantique

World Wide Web Consortium (W3C)

- W3C - Fondé en 1994
- Consortium industriel international accueilli par différents sites
 - CSAIL (*MIT Computer Science and Artificial Intelligence Laboratory*) aux Etats-Unis
 - ERCIM (*European Research Consortium for Informatics and Mathematics*) en Europe
 - Keio University au Japon
- 316 membres enregistrés sur la page Web officielle en août 2011

W3C : dans quel but ?

- Accroître le potentiel du Web
- Développer des protocoles communs
- Assurer l'inter-opérabilité sur le Web entre les différents systèmes
- Stock d'informations sur les standards et les normes pour développeurs et utilisateurs
 - Code référence pour présenter et promouvoir les différents standards
 - Prototypes variés et exemples d'applications

XML

write once, *publish everywhere*

XML

- e**X**tensible **M**arkup **L**anguage.
- Recommandation W3C
- XML 1.0 - 10 février 1998
- XML 1.1 - 04 février 2004.
- Développé par le groupe de travail XML en 1996 (auparavant SGML, Standard Generalized Markup Language, 1986).
- Propose un format d'échange de documents et données structurés à base de balises (*tags*).

Objectifs de XML

- XML doit être facilement utilisable sur le Web
- XML doit supporter une grande variété d'applications
- XML doit être compatible avec SGML
- Il doit être facile d'écrire des programmes qui traitent des documents XML
- Les documents XML doivent être lisibles et raisonnablement clairs
- La conception de XML doit être menée rapidement
- La description de XML doit être formelle et concise
- Les documents XML doivent être faciles à créer
- La concision du balisage XML est peu importante

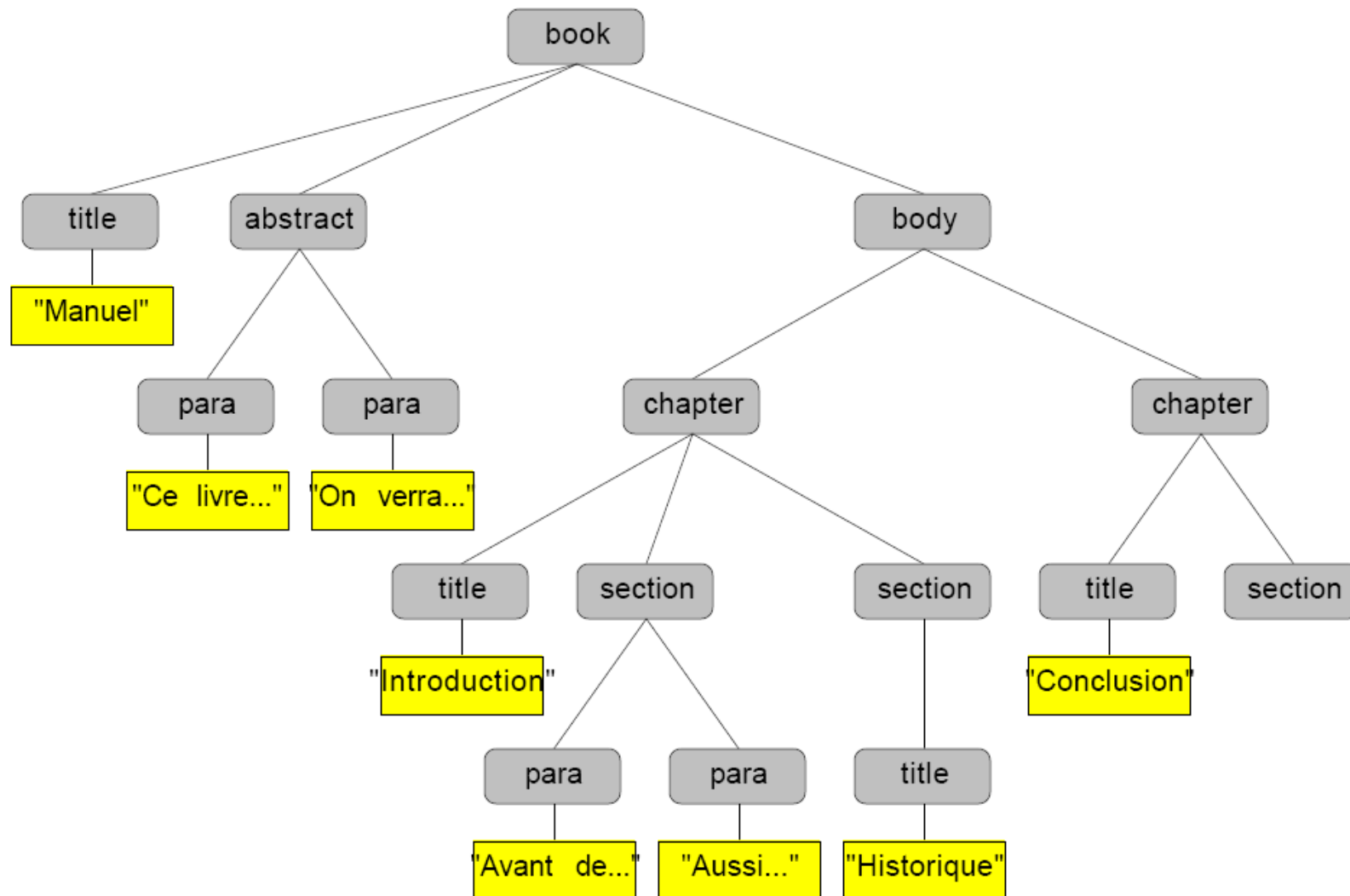
Structure, contenu et présentation

- Trois aspects dans les documents
 - Le contenu
 - La structure logique
 - La présentation
- XML permet de représenter les contenus textuels et la structure logique
 - Les autres contenus sont des ressources externes (photos, vidéo, sons...)
 - La présentation est décrite par des moyens complémentaires (CSS, XSL)
 - La présentation peut changer, indépendamment des contenus et de la structure

Structure et éléments

- Un document est essentiellement représenté en XML comme une **structure logique arborescente**
- Les éléments sont les constituants « logiques » du document
- Les utilisateurs peuvent définir leurs propres éléments :
 - Manuel, Titre, Auteur, Résumé, Chapitre, Section, Paragraphe, Note, Exemple, etc.
 - Pièce, Personnage, Acte, Scène, Réplique, etc.
 - LivreCuisine, Plat, Recette, Ingrédient, Temps, Préparation, Étape, etc.
 - etc.
- Les éléments ne sont pas prédéfinis, mais choisis en fonction du type de document à représenter
- Les éléments terminaux de la structure sont des éléments vides ou des chaînes de caractères
- L'ensemble de la structure est ordonné

Structure et éléments - exemple



Attributs

- Les attributs donnent des précisions sur les éléments et leur contenu
 - Exemples : langue, statut, sous-type, etc.
- Les attributs ne sont pas prédéfinis, mais choisis en fonction du type de document à représenter
 - Exceptions : xml:lang, xml:space, xml:ns
- Un attribut a un nom et une valeur
- Un élément peut avoir zéro, un ou plusieurs attributs
- Les attributs d'un même élément n'ont pas d'ordre

Identificateurs et références

- Deux types d'attributs permettent des relations non hiérarchiques dans la structure du document :
 - ID : identificateur unique dans le document (ex : **label**)
 - IDREF : référence à un élément portant un attribut de type ID (ex : **item**)

- Exemple

<Para>...voir <RefBib **item**="bib1"/> en fin d'ouvrage...</Para>

...

<Biblio>

 <BiblItem **label**="bib1">

 <Auteur> J. Dupont </Auteur><Titre>...</Titre>

 </BiblItem>

...

</Biblio>

Langage de balisage

- Le contenu est structuré en éléments qualifiés par des attributs avec des valeurs
- Chaque élément est représenté par une paire de balises et son contenu :

```
<chapitre>...</chapitre>
```

- Les balises ouvrantes portent les attributs :

```
<chapitre version="provisoire" date="16/06/03">
```

- L'imbrication et l'ordre des éléments reflètent la structure :

```
<ol xml:lang="fr">  
  <li>Des balises décrivent la structure</li>  
  <li>Structure arborescente</li>  
</ol>
```

Contraintes syntaxiques

- Tout élément doit avoir une balise ouvrante et une balise fermante :

`<aaa>...</aaa>`

- Raccourci d'écriture pour les éléments vides :

`` ou ``

- Les paires de balises ouvrantes/fermantes doivent être imbriquées :

- correct

`<aaa>...<bbb>...</bbb>...</aaa>`

- Faux

`<i>......</i>...`

- Un document possède une racine et une seule
- Tous les attributs doivent avoir une valeur
- Majuscules et minuscules sont différents

En-tête

■ Ligne d'en-tête

- Un document XML est composé d'une ou plusieurs lignes d'en-tête (ce n'est pas une obligation mais conseillée).
- La ligne d'entête minimale doit se trouver sur le premier caractère de la première ligne. Elle commence par `<?xml` et termine par `?>`.
- Elle contient au moins un attribut **“version”** (valeur “1.0” ou “1.1”) qui référence la version du standard XML utilisé par le fichier.
- Elle peut également permettre de préciser l'encodage utilisé (**encoding**).
- Mais aussi de savoir si le document est autonome ou pas i.e., s'il y a lien vers d'autres documents (**standalone**).

Exemple

```
<?xml version="1.0" encoding='ISO-8859-1' standalone='yes' ?>
```

Codage

Encodage	Description	Encodage	Description
US-ASCII Anglais		ISO-8859-8	ASCII plus hébreu
UTF-8 (par défaut)	Unicode compressé	ISO-8859-9	Latin-5, turc
UTF-16	UCS compressé	ISO-8859-10	Latin-6, ASCII plus langues nordiques
ISO-10646-UCS-2	Unicode brut	ISO-8859-11	ASCII plus thaï
ISO-10646-UCS-4	UCS Brut	ISO-8859-13	Latin-7, ASCII plus langues baltes
ISO-8859-1	Latin-1, Europe occidentale	ISO-8859-14	Latin-8, ASCII plus gallos et gaëlique
ISO-8859-2	Latin-2, Europe orientale	ISO-8859-15	Latin-9, Latin-10, Europe occidentale
ISO-8859-3	Latin-3, Europe méridionale	ISO-2022-JP	Japonais
ISO-8859-4	Latin-4, Europe septentrionale	ISO-2022-CN	Chinois
ISO-8859-5	ASCII plus cyrillique	KOI6-R	Russe
ISO-8859-6	ASCII plus arabe	ISO-2022-KR	Coréen
ISO-8859-7	ASCII plus grec		

Diverses

- Des commentaires peuvent être insérés, mais pas dans les balises :
`<!-- declarations for <head> & <body> -->`
- Pour rendre le code XML lisible, on le formate souvent avec des sauts de ligne et des espaces.
- Les sauts de ligne sont équivalents à des espaces
- Les espaces sont interprétés par l'application
- Le document peut contenir des attributs `xml:space` (valeur **default** ou **preserve**)
- Un document XML est bien formé (well-formed), s'il respecte les règles de la syntaxe XML
- Un processeur XML doit s'arrêter lorsqu'il rencontre une erreur

Le nom des éléments

- Un **nom XML** peut contenir n'importe quel caractère alphanumérique (lettres standard de "A" à "Z" et de "a" à "z" et les chiffres de "0" à "9") et les trois caractères de ponctuation suivants : " _ " (souligné), " - " (trait d'union), " . " (point).
- Il peut commencer par des lettres ou " _ ". Il ne peut pas commencer par un chiffre, le " - " ou " . "
- Les noms sont sensibles à la casse (" Nom " et " nom " sont deux noms XML différents).
- Un nom XML ne peut pas commencer par la chaîne " xml " (quelque soit la casse) qui est utilisée pour les noms réservés du standard.
- Il n'y a pas de limite de longueur.
- Le caractère " : " (deux points) est autorisé, mais il est réservé pour une utilisation avec les **espaces de noms**. Les autres caractères de ponctuation sont interdits (y compris les différents types d'espace, retours chariot, etc).

Examples

```
<?xml version="1.0"?>
<memo xml:lang="fr">
  <to>J. Smith</to>
  <from>P. Brown</from>
  <date>20/01/2000</date>
  <body>
    <p>meeting confirmed
      <em>tomorrow</em> 10am
    </p>
  </body>
</memo>
```

```
<?xml version="1.0"?>
<customer>
  <name>...</name>
  <order>...</order>
  <order>...</order>
</customer>
```

```
<?xml version="1.0"?>
<mfrac>
  <mn> 1 </mn>
  <mrow>
    <mn> 2 </mn>
    <mo> &InvisibleTimes; </mo>
    <mi> n </mi>
  </mrow>
</mfrac>
```

```
<?xml version="1.0"?>
<par>
  <video id="a" src="movie1"/>
  <seq>
    <audio src="audio1"/>
    <audio begin="5s" src="audio2"/>
  </seq>
</par>
```

Instructions de traitement

- Une instruction de traitement contient une chaîne de caractères qui ne sera pas interprétée comme du XML. Cette chaîne est destinée à être traitée par une application.
- Elle débute par <?cible et termine par ?> où “cible” est un nom XML (autre que “xml” ou toute autre version avec des casses différentes) représentant l’application pour laquelle la chaîne est destinée. Elles peuvent être placées n’importe où (sauf dans une balise XML).

```
<?xml version=' 1.0' ?>
<racine>
  <?php
    function maFonc($x) {
      =24;
      "bonjour";
      ($i=1;$i<=10;$i=$i+1) {
        =x+1; }
      x;}
  ?>
</racine>
```

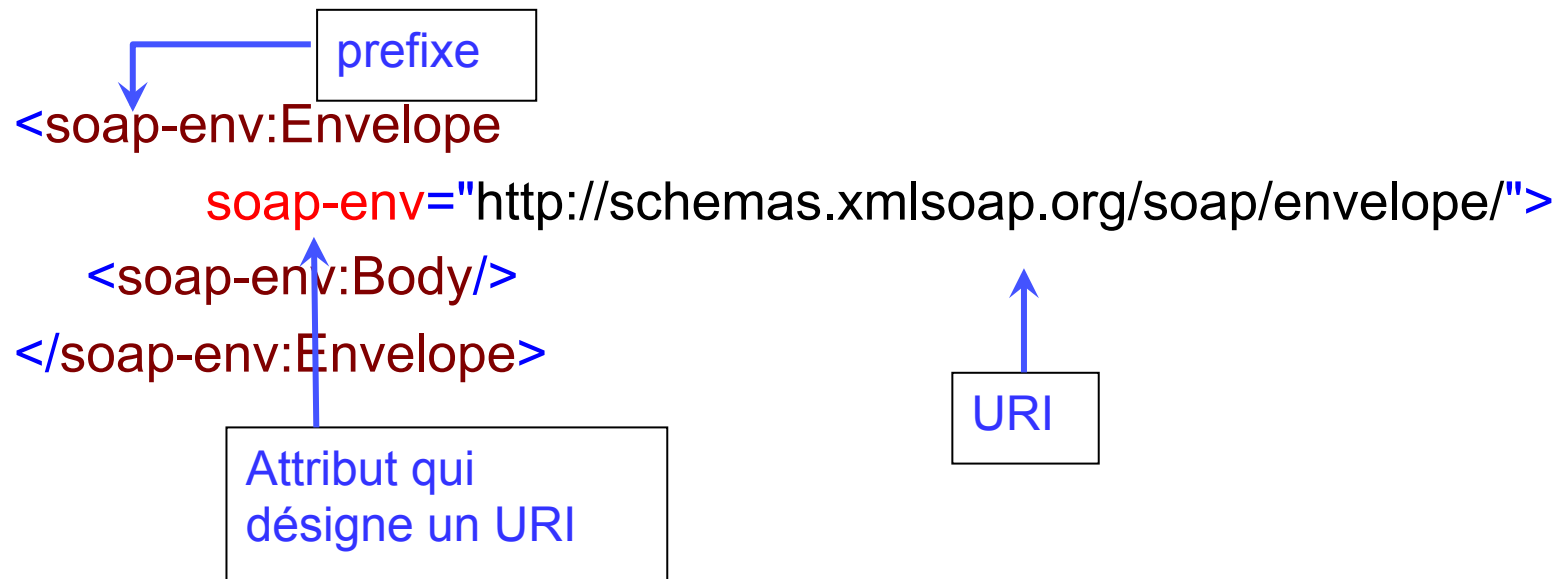

Un document XML bien formé

Un document XML bien formé est un document XML respectant les règles énoncées précédemment.

- Les éléments sont correctement imbriqués ;
- Les éléments comportent bien des balises ouvrante et fermante ;
- L'élément racine est unique ;
- Les identificateurs d'éléments et d'attributs sont des noms XML;
- Les valeurs des attributs sont entre guillemets ou apostrophes ;
- Un élément ne doit pas avoir deux attributs de même nom ;
- Les commentaires et instructions de traitement ne doivent pas apparaître à l'intérieur d'une balise ;
- Aucun caractère ' < ' ou ' & ' non échappé ne doit apparaître dans les données textuelles d'un élément ou d'un attribut.

Espace de noms

- Concept utilisé pour gérer l'ambiguïté des éléments issus de différents langages, applications, etc.
- Un espace de nom assigne des éléments et des attributs à des URI (*Uniform Resource Identifier*)
- Tous les éléments des applications XML différentes sont assignés à un URI différent



DTD

(Définition de Type de Document)

XML : un langage pour créer des langages

- XML contient deux langages :
 - Un langage de balisage des instances de documents
 - Un langage de définition de types de documents
- XML est un langage qui permet de créer des langages de balisage
 - Création de langages adaptés au type d'information à décrire
 - Exemples : XHTML, MathML, SVG, SMIL, etc.

Types de documents

- Un type de document est un modèle qui représente un ensemble (un type) de documents utilisant tous :
 - Les mêmes éléments
 - Les mêmes attributs avec les mêmes valeurs possibles
 - Les mêmes relations structurales entre éléments
 - Les mêmes associations d'attributs aux éléments
- Un type de document peut être défini par une DTD

DTD (Définition de Type de Document)

- Une Définition de Type de Document (DTD) spécifie
 - Les éléments utilisables dans les documents de ce type
 - Les noms et valeurs possibles des attributs utilisables
 - Quels attributs peuvent être associés à quels éléments
 - Les règles d'assemblage des éléments dans la structure
 - Les entités utilisables
- Une DTD définit les contraintes qui s'appliquent aux documents d'un type donné

Exemple de DTD

<!ELEMENT personne (nom, profession*)>

<!ELEMENT nom (prénom, nom_famille)>

<!ELEMENT prénom (#PCDATA)>

<!ELEMENT nom_famille (#PCDATA)>

<!ELEMENT profession (#PCDATA)>

Explication de l'exemple

- La DTD peut être stockée dans un fichier différent du document qu'elle décrit. L'extension .dtd est assez courante.
- Dans l'exemple, chaque ligne est une déclaration d'élément. Mettre une seule déclaration par ligne est assez courant.

<!ELEMENT personne (nom, profession*)>

- Cette déclaration indique que chaque élément personne doit contenir exactement un sous-élément nom, suivi par 0 ou plusieurs éléments profession. Ceci est appelé une **séquence**. Une séquence a un ordre spécifique.

<!ELEMENT nom (prénom, nom_famille)>

- Cette déclaration indique que chaque nom doit contenir un prénom et un nom-famille.

<!ELEMENT prénom (#PCDATA)>

<!ELEMENT nom_famille (#PCDATA)>

<!ELEMENT profession (#PCDATA)>

- Ces trois déclarations indiquent que le prénom, nom_famille et profession doivent contenir du #PCDATA (*Parser Character Data*), c'est-à-dire du texte brut sans aucune balise ou sous-élément.

Déclaration d'élément

■ Nombre de sous éléments

- ? autorise zéro ou un élément
- * autorise zéro plusieurs éléments
- + autorise un ou plusieurs éléments

■ Choix de sous éléments

<!ELEMENT chiffre (zero|un|deux|trois)>

- Dans cet exemple, chaque élément chiffre peut contenir exactement un des sous éléments zero, un, deux ou trois

■ Contenu mixte

<!ELEMENT paragraphe (#PCDATA|nom|profession|note|date)*>

- Dans cet exemple, un élément paragraphe peut contenir du texte mais également de sous éléments nom, profession, note, date, dans n'importe quel ordre. Dans un contenu mixte #PCDATA doit toujours être le premier de la liste.

Déclaration d'élément

■ Éléments vides (EMPTY)

```
<image source="bus.jpg" width="152" height="345" alt="Alain Turing debut  
devant un bus/">
```

Cet élément n'a pas des éléments, il a que des attributs. Il doit être déclaré comme suit :

```
<!ELEMENT image EMPTY>
```

■ Éléments vagues (ANY)

```
<!ELEMENT image ANY>
```

- Cette déclaration indique qu'un élément image peut contenir n'importe quel contenu. A utiliser avec précaution.

Déclarations d'attribut

- La déclaration d'attributs se fait avec ATTLIST

```
<!ATTLIST image source CDATA #REQUIRED  
alt CDATA #IMPLIED>
```

- Cette déclaration indique qu'un élément image a un attribut appelé source et un autre appelé alt. Contrairement à alt, l'attribut source doit obligatoirement apparaître dans un document conforme à cette DTD. Les valeurs des deux attributs sont une donnée textuelle.

- Attribut par défaut

- #IMPLIED. L'attribut est optionnel.
- #REQUIRED. L'attribut est obligatoire.
- #FIXED. La valeur de l'attribut est fixe et non modifiable.
- Littéral. La vraie valeur par défaut en tant que chaîne entre guillemets
 - <!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Types d'attributs

- Dix types d'attributs existent dans XML
 - CDATA.
 - Une valeur d'attribut CDATA peut contenir n'importe quelle chaîne de caractères.
 - NMTOKEN
 - La valeur d'un attribut de ce type est une unité lexicale nominale XML qui ressemble à un nom XML, sauf que tous les caractères peuvent être le premier caractère (i.e., 12, .csrc).
 - NMTOKENS
 - La valeur de cet attribut contient une ou plusieurs unités lexicales nominales XML séparées par des blancs.
 - Énumération
 - C'est le seul type qui n'est pas représenté par un mot clé XML. Le modèle de contenu est une liste de tous les valeurs possibles pour un attribut.
- <!ATTLIST date jour (lundi|mardi|mercredi|jeudi|vendredi|samedi|dimanche)
#REQUIRED>

Types d'attributs

■ ENTITY

- Un attribut de ce type contient un nom d'une entité non analysée déclarée n'importe où dans la DTD.

`<!ATTLIST image source ENTITY #REQUIRED>`

■ ENTITIES

- Un attribut de ce type contient les noms séparés par des blancs, de plusieurs entités non analysées, déclarées n'importe où dans la DTD

■ ID

- Un attribut ID doit contenir un nom XML unique dans le document.

`<!ATTLIST employe numero_secu ID #REQUIRED>`

■ IDREF

- Un attribut IDREF fait référence à un attribut de type ID d'un élément du document.

■ IDREFS

- Un attribut IDREFS contient une liste de noms XML séparés par des blancs, chacun d'eux devront faire référence à un attribut de type ID d'un élément du document.

Types de DTD

- Interne

- La DTD est incorporée au document XML

- Externe

- La DTD existe dans un fichier différent que celui du document XML
- Possibilité de combiner DTD interne et externe

- 2 types de DTD externe

- Privé -> accessible uniquement en local
- Public -> disponibles publiquement grâce à un URI (Uniform Resource Identifier)

Déclaration de DTD interne

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE biblio[
  <!ELEMENT biblio (livre)*>
  <!ELEMENT livre (titre, auteur, nb_pages)>
    <!ATTLIST livre
      type (roman | nouvelles | poemes | théâtre) #IMPLIED
      lang CDATA "fr"
    >
  <!ELEMENT titre (#PCDATA)>
  <!ELEMENT auteur (#PCDATA)>
  <!ELEMENT nb_pages (#PCDATA)>
]>
<biblio>
  <livre type="roman">
    <titre>El coronel no tiene quien le escriba</titre>
    <auteur>Gabriel García Marquez</auteur>
    <nb_pages>100</nb_pages>
  </livre>
</biblio>
```

Déclaration de DTD externe privé

Fichier bibliographie.dtd

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

<!ELEMENT biblio (livre*)>
<!ELEMENT livre (titre, auteur, nb_pages)>
  <!ATTLIST livre
    type (roman | nouvelles | poemes | théâtre)
      #IMPLIED
    lang CDATA "fr"
  >
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT nb_pages (#PCDATA)>
```

Document XML

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

<!DOCTYPE biblio SYSTEM "bibliographie.dtd">

<biblio>
  <livre type="roman">
    <titre>El coronel no tiene quien le escriba</titre>
    <auteur>Gabriel García Marquez</auteur>
    <nb_pages>100</nb_pages>
  </livre>
</biblio>
```


Déclaration de DTD externe publique

- `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
- Après PUBLIC
 - Identifiant de la DTD
 - -
 - Propriétaire de la DTD
 - W3C
 - Nom
 - XHTML 1.0 Strict
 - Langue
 - EN
 - URI
 - "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"

Limites de la DTD

- Il y beaucoup de choses que la DTD ne dit pas, particulièrement
 - Quel est l'élément racine du document
 - Combien d'instances de chaque élément apparaissent dans le document
 - À quoi ressemblent les données textuelles des éléments
 - Quel est le sens d'un élément
 - Pas de support pour définir des espaces de noms

- Une DTD n'est pas faite en XML
 - Langage spécifique

Schéma XML

Schéma XML

- Définit

- Les éléments possibles dans un document XML
- Les attributs associés à ces éléments
- La structure du document
- Les types de données

- Présente de nombreux avantages

- Spécifié en XML donc pas de nouveau langage
- Structures de données avec types de données
- Extensibilité par héritage et ouverture
- Analysable à partir d'un parseur XML standard
- Facilité de conception par modules
- Support des espaces de noms...

Objectifs des schémas

- Reprendre les acquis des DTD
 - Plus riche et complet que les DTD
- Permettre de typer les données
 - Éléments simples et complexes
 - Attributs simples
- Permettre de définir des contraintes
 - Existence, obligatoire, optionnel
 - Domaines, cardinalités, références
 - Patterns, ...
- S'intégrer à la galaxie XML
 - Espace de noms

Le modèle des schémas

- Déclaration d' éléments et d' attributs
 - Nom
 - Typage similaire à l'objet
- Spécification de types simples
 - Grande variété de types
- Génération de types complexes
 - Séquence (Sequence)
 - Choix (Choice)
 - Tas (All)

Commandes de base xsd

- **Attribut « type »** : les multiples types de base (appelés types intégrés)
 - entier, réel, string, time, date, ID, IDREF, ...,
 - extensibles par des contraintes
- **Élément « element »** : association d'un type à une balise
 - attributs : name, type, ref, minOccurs, maxOccurs, mixed, fixed, default ...
- **Élément « attribute »** : association d'un type à un attribut
 - attributs : name, type, fixed, default, id
- **Élément « complexType »** : une composition de types
 - définit une agrégation d'éléments typés

Structure de base

- Comme tout document XML, un schéma XML commence par un prologue, et a un élément racine nommé « schema »

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!-- déclarations d'éléments, d'attributs et de types ici -->
</xs:schema>
```


Déclaration d' éléments et d' attributs

■ Éléments simples

- Chaque élément est associé à un type de données quelconque

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
  elementFormDefault="qualified" attributeFormDefault="unqualified">
```

```
  <xs:element name="Commentaire" type="xs:string"/>
```

```
</xs:schema>
```

■ Éléments complexes

- Peuvent contenir des éléments enfants ou des attributs et peuvent être de type complexe

■ Attributs

- Ne contient pas d'éléments ni d'autres attributs
- Sont uniquement de type simple
- Sont déclarés après les types complexes (i.e. sequence, choice, all)

Les types intégrés (1)

- string
 - Confirmez que ceci est électrique
- normalizedString
 - Confirmez que ceci est électrique
- token
 - Confirmez que ceci est électrique
- byte
 - -1, 126
- unsignedByte
 - 0, 126
- base64Binary
 - GpM7
- hexBinary
 - 0FB7
- integer
 - -126789, -1, 0, 1, 126789
- positiveInteger
 - 1, 126789
- negativeInteger
 - -126789, -1
- nonNegativeInteger
 - 0, 1, 126789
- nonPositiveInteger
 - -126789, -1, 0
- int
 - -1, 126789675
- unsignedInt
 - 0, 1267896754

Les types intégrés (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 (AAAA-MM-JJ)
- gMonth
 - --05--
- gYear
 - 1999

Les types intégrés (3)

- gYearMonth
 - 1999-02
- gDay
 - ---31
- gMonthDay
 - --05-31
- Name
 - shipTo
- QName
 - po:USAddress
- NCName
 - USAddress
- anyURI
 - <http://www.example.com/>
 - <http://www.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

Déclaration d' éléments et d' attributs

- La déclaration d'un élément contenant du texte et un attribut utilise l'attribut « mixed = "true" »

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

- L'attribut mixed permet de spécifier si entre les enfants d'un élément de type complexe le texte est permis (par défaut sa valeur est "faux")

Les types complexes

■ Définition de types complexes

- **sequence** : collection ordonnée d'éléments typés
- **all** : collection non ordonnée d'éléments typés
- **choice** : choix entre éléments typés

■ Exemple

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"  
attributeFormDefault="unqualified">
```

```
<xs:complexType name="AdresseFR">  
  <xs:sequence>  
    <xs:element name="nom" type="xs:string"/>  
    <xs:element name="rue" type="xs:string"/>  
    <xs:element name="ville" type="xs:string"/>  
    <xs:element name="codep" type="xs:decimal"/>  
  </xs:sequence>  
  <xs:attribute name="pays" type="xs:NMTOKEN" fixed="FR"/>  
</xs:complexType>
```

```
</xs:schema>
```

Déclaration du type
complexe AdresseFR

Déclaration par référencement

- Recommandation->créer les éléments de type simple et ensuite ceux de type complexe
- Dans la déclaration d'un type complexe on peut faire référence à un élément de type simple ou complexe préalablement défini
 - Plus facile à maintenir
 - Optimise la réutilisation de types
- Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="nom" type="xs:string"/>
  <xs:element name="codep" type="xs:decimal"/>
  <xs:complexType name="AdresseFR">
    <xs:sequence>
      <xs:element ref="nom"/>
      <xs:element ref="codep"/>
    </xs:sequence>
    <xs:attribute name="pays" type="xs:NMTOKEN" fixed="FR"/>
  </xs:complexType>
</xs:schema>
```

Déclaration du type
complexe AdresseFR

Héritage de types

- Définition de sous-types par héritage
 - Par extension : ajout d'informations
 - Par restriction : ajout de contraintes à l' espace de valeurs des types simples ou complexes à l' aide des facettes

- Exemple qui étend le type AdresseFR définit précédemment

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

- L' élément complexContent définit des extensions ou des restrictions sur un type complexe

Les facettes

- Permettent de placer une contrainte sur l'ensemble des valeurs que peut prendre un type de base
- Il existe un nombre important de facettes qui permettent de
 - fixer, restreindre ou augmenter la longueur minimale ou maximale d'un type simple
 - énumérer toutes les valeurs possibles d'un type
 - prendre en compte des expressions régulières
 - fixer la valeur minimale ou maximale d'un type
 - fixer la précision du type...

Les facettes

- length
 - Restriction du nombre de caractères
- minLength
 - Minimum nombre de caractères
- maxLength
 - Maximum nombre de caractères
- pattern
 - Restriction à l'aide d'expressions régulières
- enumeration
 - spécifie un ensemble de valeurs
- whiteSpace
 - Les espaces (tabulations, retour de chariot et nouvelle ligne) sont soit préservés, remplacés ou collapsés

Les facettes

- **maxInclusive**
 - Les valeur possibles sont inférieurs ou égale à la limite donnée
- **maxExclusive**
 - Les valeur possibles sont inférieurs à la limite donnée
- **minInclusive**
 - Les valeurs possibles sont supérieures ou égales à la limite donnée
- **minExclusive**
 - Les valeur possibles sont supérieures à la limite donnée
- **totalDigits**
 - Délimite le nombre de décimales
- **fractionDigits**
 - Délimite le nombre de fractions

Les facettes - exemples

- Type simple, « MonEntier » limité aux valeurs comprises entre 0 et 99 inclus

```
<xsd:simpleType name="monEntier">  
  <xsd:restriction base="nonNegativeInteger">  
    <xsd:maxExclusive value="100" />  
  </xsd:restriction>  
</xsd:simpleType>
```

- Type simple « jour » permettant de limiter les valeurs de certains attributs

```
<xsd:attribute name="jour" type="typeJourSemaine" use="required" />  
<xsd:simpleType name="typeJourSemaine">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="lundi" />  
    <xsd:enumeration value="mardi" />  
    <xsd:enumeration value="mercredi" />  
    <xsd:enumeration value="jeudi" />  
    <xsd:enumeration value="vendredi" />  
    <xsd:enumeration value="samedi" />  
    <xsd:enumeration value="dimanche" />  
  </xsd:restriction>  
</xsd:simpleType>
```

Les facettes - exemples

- Type simple permettant de définir les valeurs possibles de la température chez l'humain en degrés Celsius

```
<xs:simpleType name="tempType">  
  <xs:restriction base="xs:decimal">  
    <xs:totalDigits value="4"/>  
    <xs:fractionDigits value="1"/>  
    <xs:minInclusive value="36.4"/>  
    <xs:maxInclusive value="40.5"/>  
  </xs:restriction>  
</xs:simpleType>
```

Les facettes

- Les contraintes peuvent être définies à l'aide d'**expressions régulières**

Motif	Signification
[]	spécification d'intervalle (par ex: [a-z] indique une lettre dans l'intervalle a à z)
[^]	spécification d'un caractère qui n'est pas un qui se trouve à l'intérieur des crochets
\w	lettre ou chiffre; équivaut à [0-9A-Za-z]
\W	ni lettre ni chiffre
\s	caractère espace; équivaut à [\t\n\r\f]
\S	caractère non espace
\d	chiffre; équivaut à [0-9]
\D	non-chiffre
*	zero, 1 ou n occurrences de ce qui précède
+	1 ou n occurrences de ce qui précède
{n}	une séquence de n occurrences de ce qui précède
{m,n}	au moins m et au plus n occurrences de ce qui précède
?	Au plus une occurrence de ce qui précède; équivaut à {0,1}
	alternative: soit ce qui précède soit ce qui suit
()	groupe
.	n'importe quel caractère
Etc	etc

Les facettes

- Exemples de restriction avec des expressions régulières
- ISBN – numéro à 10 chiffres

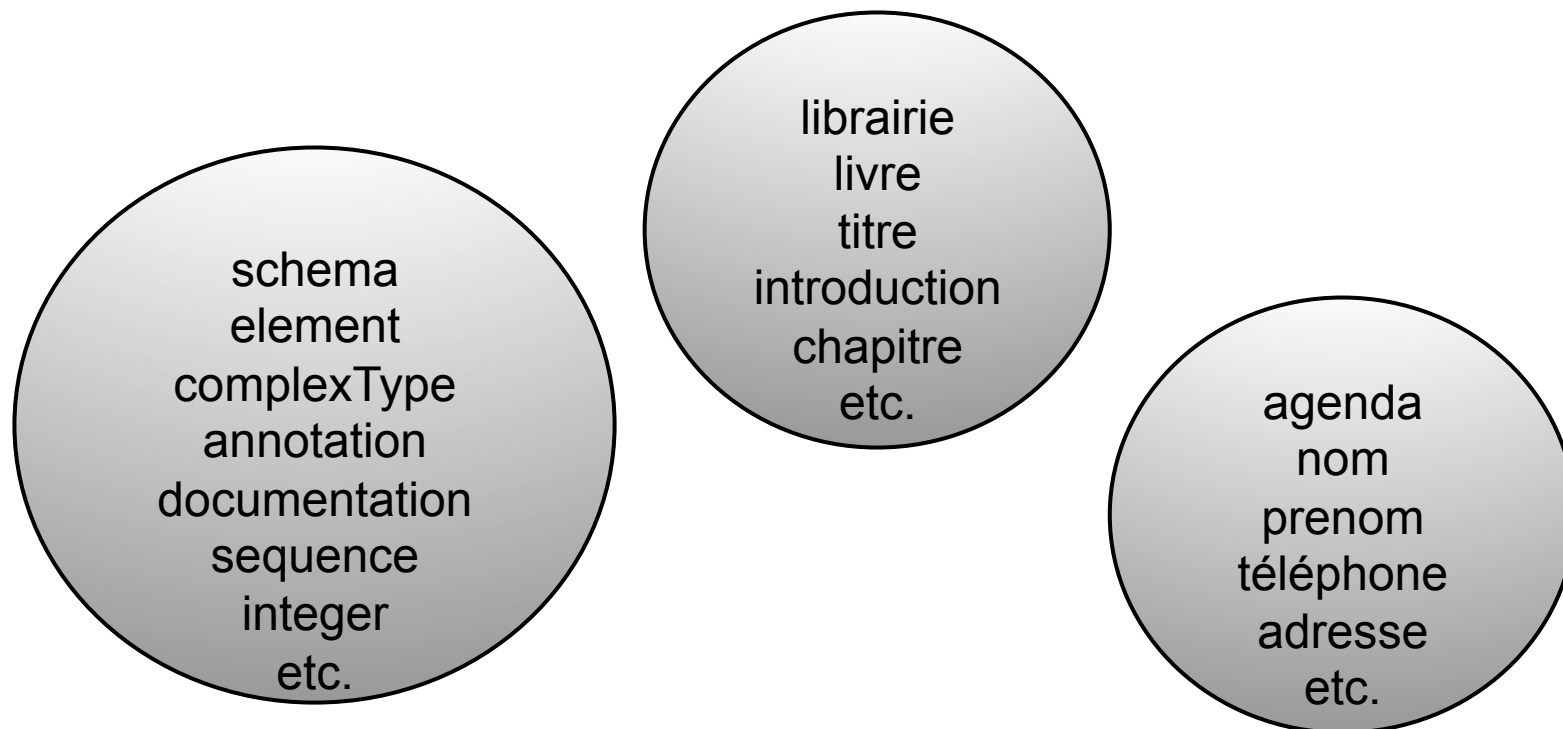
```
<xs:simpleType name="typeISBN">  
  <xs:restriction base="xs:string">  
    <xs:pattern value="[0-9]{10}" />  
  </xs:restriction>  
</xs:simpleType>
```

- Adresse électronique simple – deux chaînes d'au moins un caractère, entre les deux le caractère @

```
<xs:simpleType name="typeAdresseElectronique">  
  <xs:restriction base="xs:string">  
    <xs:pattern value="(.)+@(.)+" />  
  </xs:restriction>  
</xs:simpleType>
```

Espaces de nom

- Chaque schéma XML doit au moins avoir 2 espaces de noms
 - XMLSchema
 - Schéma cible (targetNamespace)



Espaces de nom

- Trois manières de définir les espaces de nom
 1. *Faire de XMLSchema l'espace de nom par défaut*
 2. et qualifier explicitement toutes les références aux éléments de targetNamespace
 3. *Faire de targetNamespace l'espace de nom par défaut*
 4. et qualifier explicitement toutes les références aux éléments de XMLSchema
 5. *Ne pas avoir de espace de nom par défaut*
 6. et qualifier explicitement les éléments d XMLSchema et de targetNamespace

Espaces de nom

■ Trois manières de définir les espaces de nom

- Faire de XMLSchema l' espace de nom par défaut
- et qualifier explicitement toutes les références aux éléments de targetNamespace

- Faire de targetNamespace l' espace de nom par défaut
- et qualifier explicitement toutes les références aux éléments de XMLSchema

- Ne pas avoir de espace de nom par défaut
- et qualifier explicitement les éléments d XMLSchema et de targetNamespace

Espaces de nom – type 1

- XMLSchema -> défaut
- targetNamespace -> qualifié
- Avantages
 - Si le schéma utilise plusieurs espaces de nom, ceci est une manière cohérente de les nommer tous
- Desavantages
 - Les schémas sans targetNamespace doivent nomer XMLSchema

```
1 <schema xmlns="http://www.w3.org/2001/XMLSchema"
2       xmlns:po="http://www.example.com/PO1"
3       targetNamespace="http://www.example.com/PO1">
4
5     <element name="purchaseOrder" type="po:PurchaseOrderType"/>
6     <element name="comment" type="string"/>
7
8     <complexType name="PurchaseOrderType">
9       <sequence>
10        <element name="shipTo" type="po:USAddress"/>
11        <element name="billTo" type="po:USAddress"/>
12        <element ref="po:comment" minOccurs="0"/>
13        <!-- etc. -->
14      </sequence>
15      <!-- etc. -->
16    </complexType>
17
18    <complexType name="USAddress">
19      <sequence>
20        <element name="name" type="string"/>
21        <element name="street" type="string"/>
22        <!-- etc. -->
23      </sequence>
24    </complexType>
25    <!-- etc. -->
26  </schema>
27
```

Espaces de nom – type 2

- XMLSchema -> qualifié
- targetNamespace -> défaut
- Avantages
 - Pas de changement concernant XMLSchema pour les schémas sans targetNamespace
- Désavantages
 - Les schémas utilisant plusieurs espaces de nom seront qualifiés et cela pourrait porter à confusion avec l'utilisation de l'espace de nom par défaut

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3           xmlns="http://www.example.com/PO1"
4           targetNamespace="http://www.example.com/PO1">
5
6     <xs:element name="purchaseOrder" type="PurchaseOrderType"/>
7     <xs:element name="comment" type="xs:string"/>
8
9     <xs:complexType name="PurchaseOrderType">
10      <xs:sequence>
11        <xs:element name="shipTo" type="USAddress"/>
12        <xs:element name="billTo" type="USAddress"/>
13        <xs:element ref="comment" minOccurs="0"/>
14        <!-- etc. -->
15      </xs:sequence>
16      <!-- etc. -->
17    </xs:complexType>
18
19    <xs:complexType name="USAddress">
20      <xs:sequence>
21        <xs:element name="name" type="xs:string"/>
22        <xs:element name="street" type="xs:string"/>
23        <!-- etc. -->
24      </xs:sequence>
25    </xs:complexType>
26    <!-- etc. -->
27  </xs:schema>
```

Espaces de nom – type 3

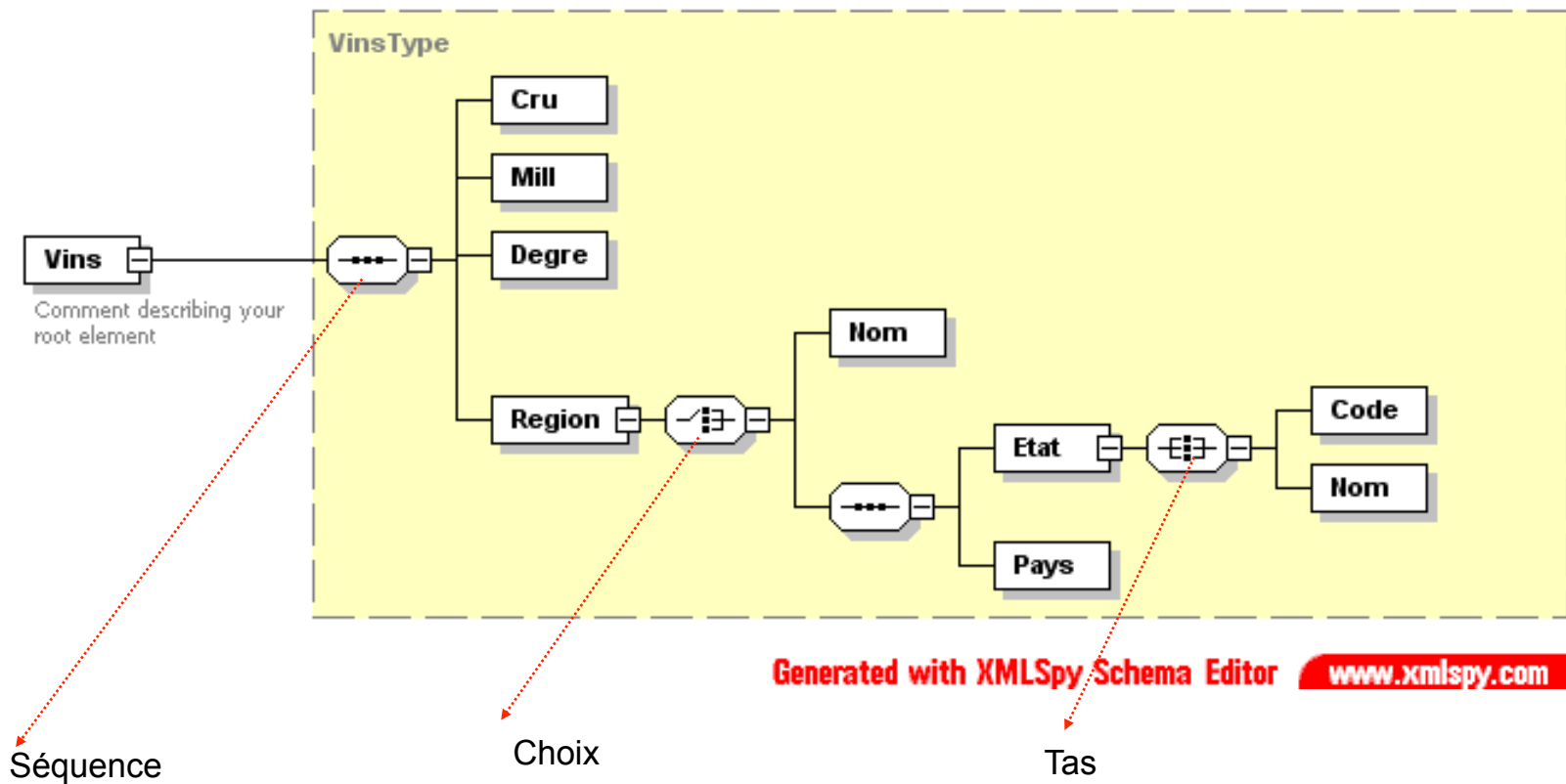
- Pas de espace de nom par défaut
- Qualification de XMLSchema et de targetNamespace
- Avantages
 - Manière cohérente de rédiger tous les schémas même avec l'utilisation de plusieurs espaces de nommage
- Désavantages
 - La rédaction et la lecture des schémas peut devenir lourd

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3           xmlns:po="http://www.example.com/PO1"
4           targetNamespace="http://www.example.com/PO1">
5
6   <xs:element name="purchaseOrder" type="po:PurchaseOrderType"/>
7   <xs:element name="comment" type="xs:string"/>
8
9   <xs:complexType name="PurchaseOrderType">
10    <xs:sequence>
11      <xs:element name="shipTo" type="po:USAddress"/>
12      <xs:element name="billTo" type="po:USAddress"/>
13      <xs:element ref="po:comment" minOccurs="0"/>
14      <!-- etc. -->
15    </xs:sequence>
16    <!-- etc. -->
17  </xs:complexType>
18
19  <xs:complexType name="USAddress">
20    <xs:sequence>
21      <xs:element name="name" type="xs:string"/>
22      <xs:element name="street" type="xs:string"/>
23      <!-- etc. -->
24    </xs:sequence>
25  </xs:complexType>
26  <!-- etc. -->
27 </xs:schema>
```

Lier un fichier XML à un schéma

- Utiliser le préfixe xmlns dans l'élément racine
- Pour référencer les éléments de XMLSchema
`<xmlns:xs="http://www.w3.org/2001/XMLSchema">`
- Si on a déposé un schéma à l'adresse
`http://www.monsite.org/collection_schemas/biblio`
- On peut le référencer par
`<biblio xmlns="http://www.monsite.org/collection_schemas/biblio">`
- Pour une référence locale, on utilise l'attribut
noNamespaceSchemaLocation du schéma XMLSchema-
instance,
`<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="biblio10.xsd">`

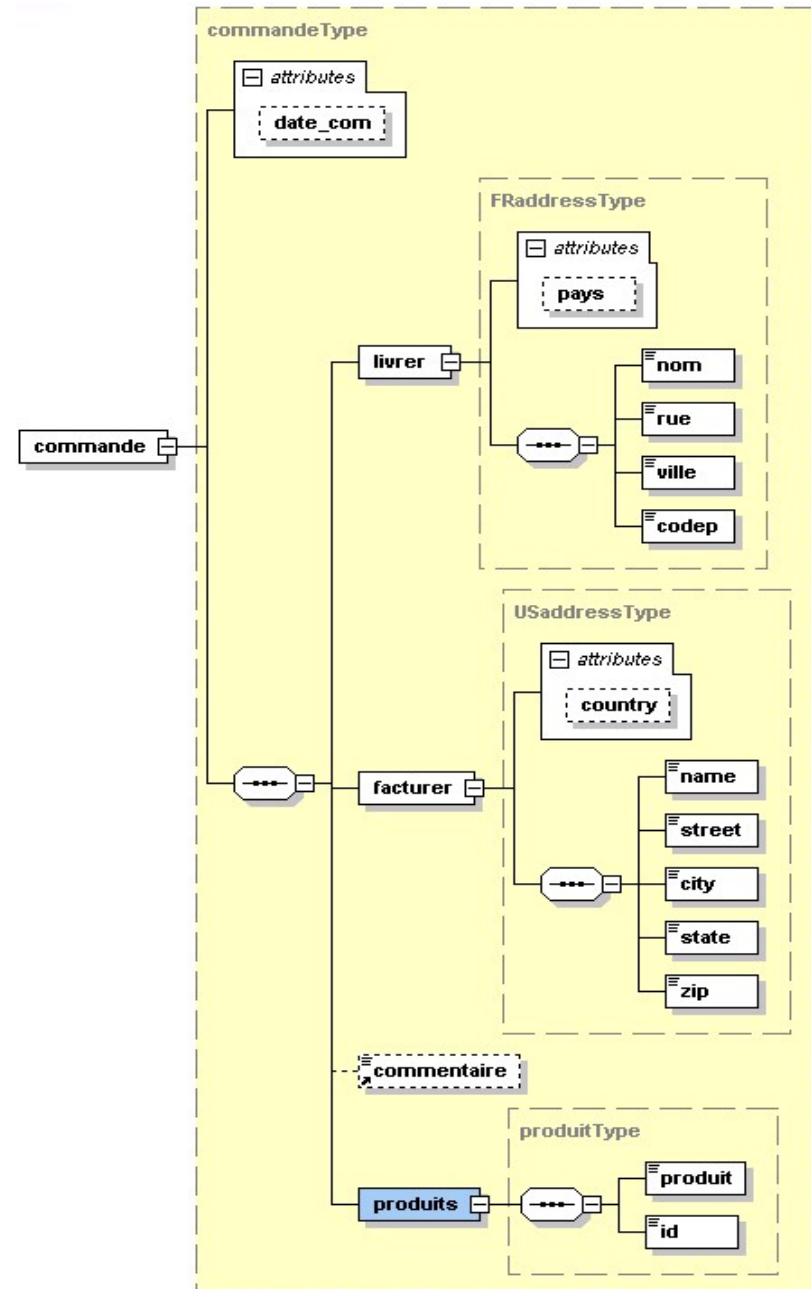
Diagramme de type (XML Spy)




```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3
4  <!--Definition d'elements simples-->
5  <xsd:element name="USAddress" type="USAddressType"/>
6  <xsd:element name="FRAddress" type="FRAddressType"/>
7  <xsd:element name="commande" type="commandeType"/>
8  <xsd:element name="commentaire" type="xsd:string"/>
9
10 <!--Definition de types complexes-->
11 <xsd:complexType name="USAddressType">
12 <xsd:sequence>
13 <xsd:element name="name" type="xsd:string"/>
14 <xsd:element name="street" type="xsd:string"/>
15 <xsd:element name="city" type="xsd:string"/>
16 <xsd:element name="state" type="xsd:string"/>
17 <xsd:element name="zip" type="xsd:decimal"/>
18 </xsd:sequence>
19 <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
20 </xsd:complexType>
21
22 <xsd:complexType name="FRAddressType">
23 <xsd:sequence>
24 <xsd:element name="nom" type="xsd:string"/>
25 <xsd:element name="rue" type="xsd:string"/>
26 <xsd:element name="ville" type="xsd:string"/>
27 <xsd:element name="codep" type="xsd:decimal"/>
28 </xsd:sequence>
29 <xsd:attribute name="pays" type="xsd:NMTOKEN" fixed="FR"/>
30 </xsd:complexType>
31
32 <xsd:complexType name="produitType">
33 <xsd:sequence>
34 <xsd:element name="produit"/>
35 <xsd:element name="id"/>
36 </xsd:sequence>
37 </xsd:complexType>
38
39 <xsd:complexType name="commandeType">
40 <xsd:sequence>
41 <xsd:element name="livrer" type="FRAddressType"/>
42 <xsd:element name="facturer" type="USAddressType"/>
43 <xsd:element ref="commentaire" minOccurs="0"/>
44 <xsd:element name="produits" type="produitType"/>
45 </xsd:sequence>
46 <xsd:attribute name="date_com" type="xsd:date"/>
47 </xsd:complexType>
48
49 </xsd:schema>
50

```



Comment concevoir DTD/Schéma ?

- A la main
 - Syntaxe complexe, devient illisible
- Interface graphique IDE
 - Partir d'un fichier d'exemples
 - Générer un premier schéma via l'outil
 - Modifier le schéma graphiquement
- A partir de UML
 - Décrire données avec UML
 - Générer un modèle logique hiérarchique

Validation de documents XML-DTD

The screenshot shows the 'DTD and Schema Validator - Check for XML grammar - Mozilla Firefox' window. The address bar displays 'http://www.validome.org/grammar/validate/'. The page title is 'XML - DTD and Schema Validator'. The 'validome' logo is visible, along with navigation links: Forum, Resources, Sidebar, Contact, About, Weblog. The main form has two sections: 'Source' and 'Source code'. The 'Source' section includes a 'URL' field with 'http://', an 'Upload' button, and a 'Parcourir...' button. The 'Source code' section is a large text area. At the bottom, there are radio buttons for 'View Sourcecode', 'DTD', and 'Schema', and a 'Validate' button.

Validator for your XML DTD / Schema

Checks syntax and structure of your XML DTD or Schema according to [W3C approved recommendation](#). The validator provides check of the document you specify.

Please note: The validator does NOT check SGML DTDs, but only XML grammars. URL of your document must be available online, if not please use the textarea or upload.

This service provides grammar-validation only. In order to validate other documents, please use the appropriate services.

Other Validators:

- [HTML / XHTML / WML](#) Validates HTML, XHTML and WML documents. » [WML / XHTML / HTML Validator](#) «
- [XML](#) Allows validation of XML 1.0 documents according to W3C specifications. » [XML Validator](#) «
- [RSS / Atom](#) Check your feeds with the » [RSS and Atom validator](#) «
- [Google Sitemap](#) The » [Google Sitemap\(s\) validator](#) « helps you checking XML conformity and other specific requirements

↑ v1.1.3 - 9.12.2006 ©validome.org - all rights reserved

Quelques outils de travail

Editeur	Outil	Support
Tibco	Turbo XML	DTD, XSL Schéma
Altova	XMLSpy	DTD, Schéma XSL, Xquery
SyncRO Ltd.	Oxygen	DTD, Schéma XSL, Xquery
Data Junction	XML Junction	Schéma
Insight Soft.	XMLMate	DTD, Schéma, XSL, XPath
XML Mind	XMLMind Editor	DTD, Schéma, XSL, XPath

Conclusion

- Limites et avantages de XML
- Limites
 - XML n'est qu'une syntaxe
 - XML ne porte aucune sémantique
 - Description de structures uniquement
 - Pas de types
- Avantages de XML
 - Productivité
 - Réutilisation
 - Pérennité
 - Intégrité
 - Partage
 - Portabilité
 - Extensibilité

Références

- W3C Recommendation 04 February 2004 :
<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>
- <http://www.w3.org/TR/xmlschema-0/>
- <http://www.w3.org/TR/xmlschema-1/>
- <http://www.w3.org/TR/xmlschema-2/>
- W3C Technical Reports and Publications : <http://www.w3.org/TR/>
- Site Web francophone sur XML : <http://xmlfr.org/>
- Site Web comment ça marche :
<http://www.commentcamarche.net/xml/xmlintro.php3>
- Elliotte Rusty Harold, W Scott Means. XML in a Nutshell. Ed O' Reilly, 2nd Edition, 2002.

XPath

Recommandation W3C

Version 1.0 - 16 novembre 1999

<http://www.w3.org/TR/1999/REC-xpath-19991116.html>

Une version française hébergée par XMLfr

<http://xmlfr.org/w3c/TR/xpath/>

Recommandation Candidate

Version 2.0 - 3 novembre 2005

<http://www.w3.org/TR/xpath20/>

Généralités

- XPath est un langage d'interrogation des documents XML.
- Il permet de sélectionner certaines parties d'un document XML : des sous arbres, des nœuds, des attributs, etc.
- XPath n'est pas un langage de la famille XML cependant il est central dans le monde XML
- XPath intervient comme brique de base dans :
 - XML Schéma (expression des contraintes d'unicité et de clefs),
 - XSLT
 - XQuery
 - XLink
 - XPointer
 - etc.

A quoi ça sert XPath ?

- XPath permet de désigner un ou plusieurs nœuds dans un document XML, à l'aide d'**expressions de chemin**.
 - *Sélection de nœuds* auxquels on souhaite appliquer une règle
`<xsl:apply-templates select="CINEMA/SALLE"/>`
 - *Sélection de règles*
`<xsl:template match="FILM/TITRE">`
 - *Extraction de valeurs*
`<xsl:value-of select="SALLE/@NO">`
 - *Prédicats de test*
`<xsl:if test=" ($titre = " or TITRE = $titre)
and ($seance = " or HEURE >=$seance)
and ($ville = " or VILLE = $ville)">`

Exemples pour XSLT

Arborescence vue par XPath (1)

- XPath crée un arbre à partir d'un document XML
- L'arborescence construite et vue par XPath est composée de sept types de nœuds.
 - **root** : le type racine de l'arbre (celle-ci n'est pas visible)
 - /
 - **element** : le type d'un nœud élément XML
 - <xxx>...</xxx>
 - **text** : le type d'un nœud text faisant partie d'un élément
 - ... bla bla ...
 - **attribute** : le type d'un nœud attribut d'élément XML
 - surface='12m2'

Arborescence vue par XPath (2)

- **namespace** : le type d'un nœud domaine nominal permettant de qualifier les noms d'attributs ou d'éléments intervenant dans certaines parties d'un document XML
 - `xmlns:txt="http://www.w3c.org/xml/schemas/Basic-text.dtd"`
- **processing-instruction** : ce type de nœud, en principe est adressé à une application cible
 - `<?cible arg2 arg2 ... ?>`
- **comment** : le type d'un nœud commentaire XML
 - `<!--...-->`

Lien de descendance entre les nœuds

Lien parent-enfant

- **root** : la racine de type **element** du document, les **processing-instruction** et les **comment** qui interviennent dans le prologue et après la fin de l'élément racine
- **element** : les nœuds de type element, **processing-instruction**, **comment** et **text** qui font partie du contenu de l'élément considéré. Les saut de ligne, les tabulations et les espaces donnent naissance à des nœuds **text**

Lien enfant-parent

- **attribute** : l'élément associé à l'attribut
- **namespace** : l'élément associé au namespace

Valeur textuelle des nœuds

- **root** : la concaténation de tous ses nœuds descendants pris dans l'ordre de lecture du document
- **element** : la concaténation des valeurs textuelles de tous ses descendants de type text (pas uniquement les enfants directs)
- **attribute** : la valeur de l'attribut
- **namespace** : la valeur de ce domaine nominal
- **processing-instruction** : la chaîne de caractères comprise entre le nom de l'instruction et le **?>** final
- **comment** : la chaîne de caractères contenue dans le commentaire
- **text** : la valeur de ce text

Un document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Fichier enseignants.xml

```
<COURS CODE="TC234">
```

```
  <SUJET>Publication XSLT</SUJET>
```

```
  <ENSEIGNANTS>
```

```
    <!-- Enseignant responsable -->
```

```
    <NOM>Amann</NOM>
```

```
    <NOM>Rigaux</NOM>
```

```
  </ENSEIGNANTS>
```

```
  <PROGRAMME>
```

```
    <SEANCE ID="1">Documents XML</SEANCE>
```

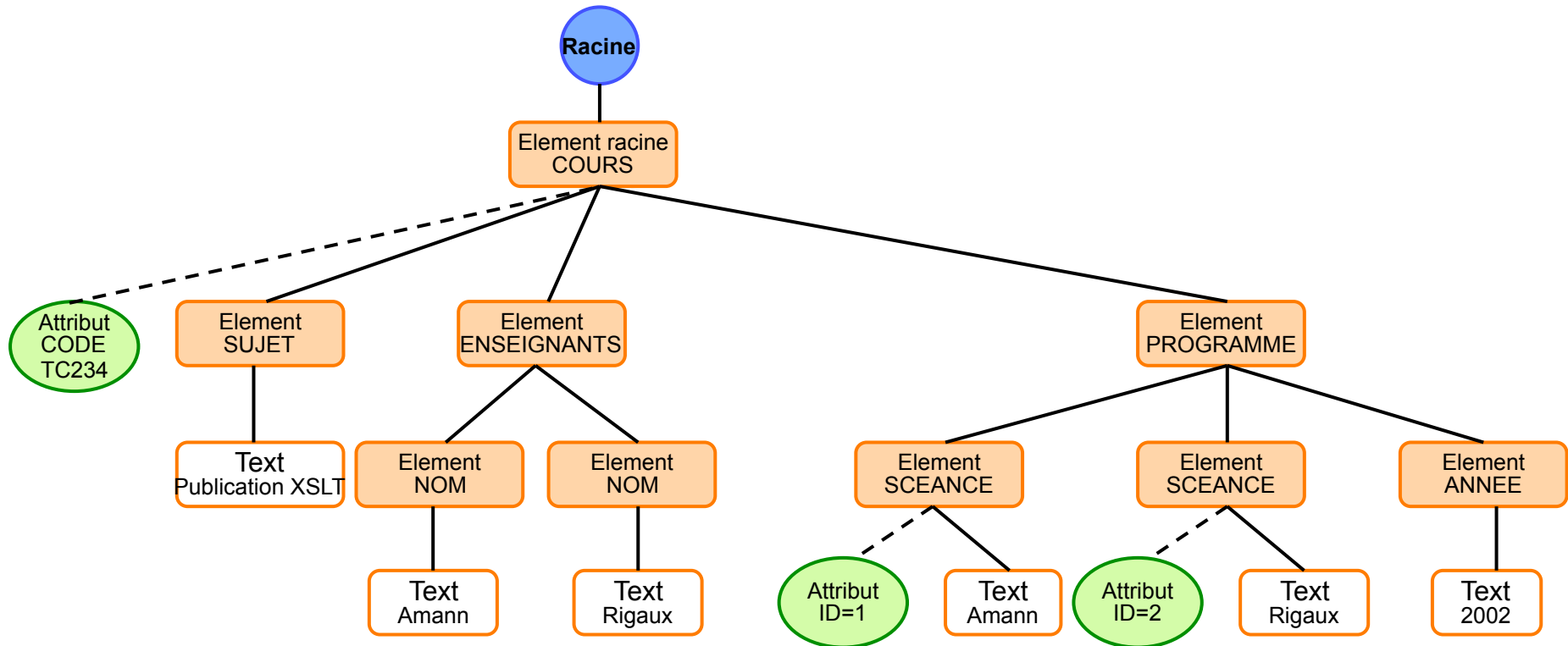
```
    <SEANCE ID="2">Programmation XSLT</SEANCE>
```

```
    <ANNEE>2002</ANNEE>
```

```
  </PROGRAMME>
```

```
</COURS>
```

L'arbre XPath du document XML



Node-set

- Un **node-set** est un ensemble de nœuds : collection non ordonnée de nœuds d'arbre XML, sans doublons.
- Qu'un élément appartienne à un node-set n'implique pas que ça descendance en fasse nécessairement partie.
- L'opérateur "|" représente l'union ensembliste (opérateur **union** in XPath 2.0)
- `count()` renvoi le nombre d'éléments du node-set donné
- L'égalité "=" entre deux node-set repose sur la valeur textuelle des nœuds
- L'intersection et la différence entre deux node-sets, P et Q, est :
 - `P intersect Q` (XPath 2.0)
 - `P except Q` (Xpath 2.0)

XPath, un langage d'expressions

- Expressions sans argument de type node-set
 - Numériques (+, -, *, =, <, >, <=, >=, !=, floor, ceiling, etc.)
 - Chaînes de caractères (=, !=, string-length, concat, normalize-space, substring, contains, start-with, substring-after, substring-before, translate)
 - Booléennes (or, and, not)
- Expressions avec arguments de type node-set
 - Count(P)
- Expressions mixtes
 - Un argument est un node-set et l'autre un type simple
 - node-set = boolean, node-set = String, node-set = Number, node-set > String, etc.
- Chemins de localisation

Quelques particularités de casting

- node-set = vraie
 - si le nœud set n' est pas vide
- node-set = String est vraie
 - si le node-set contient au moins un nœud dont la valeur textuelle est égale de la String donnée
- node-set = Number est vraie
 - si node-set contient au moins un nœud dont la valeur textuelle convertie en nombre est égale au nombre donné
- node-set > String est vraie
 - si le node-set contient au moins un nœud dont la valeur textuelle convertie en nombre est supérieur à la String donnée convertie elle aussi en nombre

Chemins de localisation

- Trois concepts principaux
 - Nœud contexte
 - Nœud considéré comme le nœud courant. A partir de ce nœud on peut désigner ses voisins plus ou moins proches ou lointains dans toutes les directions (ascendants, descendants, frères, cousins, etc.)
 - Chemin de localisation
 - LocationPath = "/"?, LocationStep, ("/", LocationStep)*
Un chemin de localisation est une suite d'étapes de localisation séparées par de "/". Le "/" initial indique un chemin absolu ; en son absence, on a un chemin relatif
 - Évaluation du chemin de localisation
 - L'évaluation d'un chemin de localisation permet d'obtenir un node-set

Étape de localisation

- **Nœud courant** : c'est l'endroit dont on part.
À partir de là, on considère trois éléments :
 - un **axe** : la direction vers laquelle on se dirige à partir du nœud courant (vers le père, vers les fils, vers les frères de gauche, etc.) ;
 - un **filtre** (déterminant, NodeTest) : le type de nœuds qui nous intéresse dans l'axe choisi (des nœuds quelconques, des éléments quelconques ou un élément précis, des commentaires, etc.) ; l'ensemble de nœuds qui ne répondent pas au critère indiqué sont éliminés
 - un **prédicat** optionnel : des conditions supplémentaires pour sélectionner des nœuds parmi ceux retenus par le filtre dans l'axe.
- Ces trois éléments constituent une *étape*
 - LocationStep = Axe, ":", Filtre, Predicat*
- L'enchaînement de plusieurs étapes constitue une *chemin XPath* :
 - axe1::filtre1[prédicat1]/axe2::filtre2[prédicat2]

Exemple : parent::*/*[child::node()][position()=2]

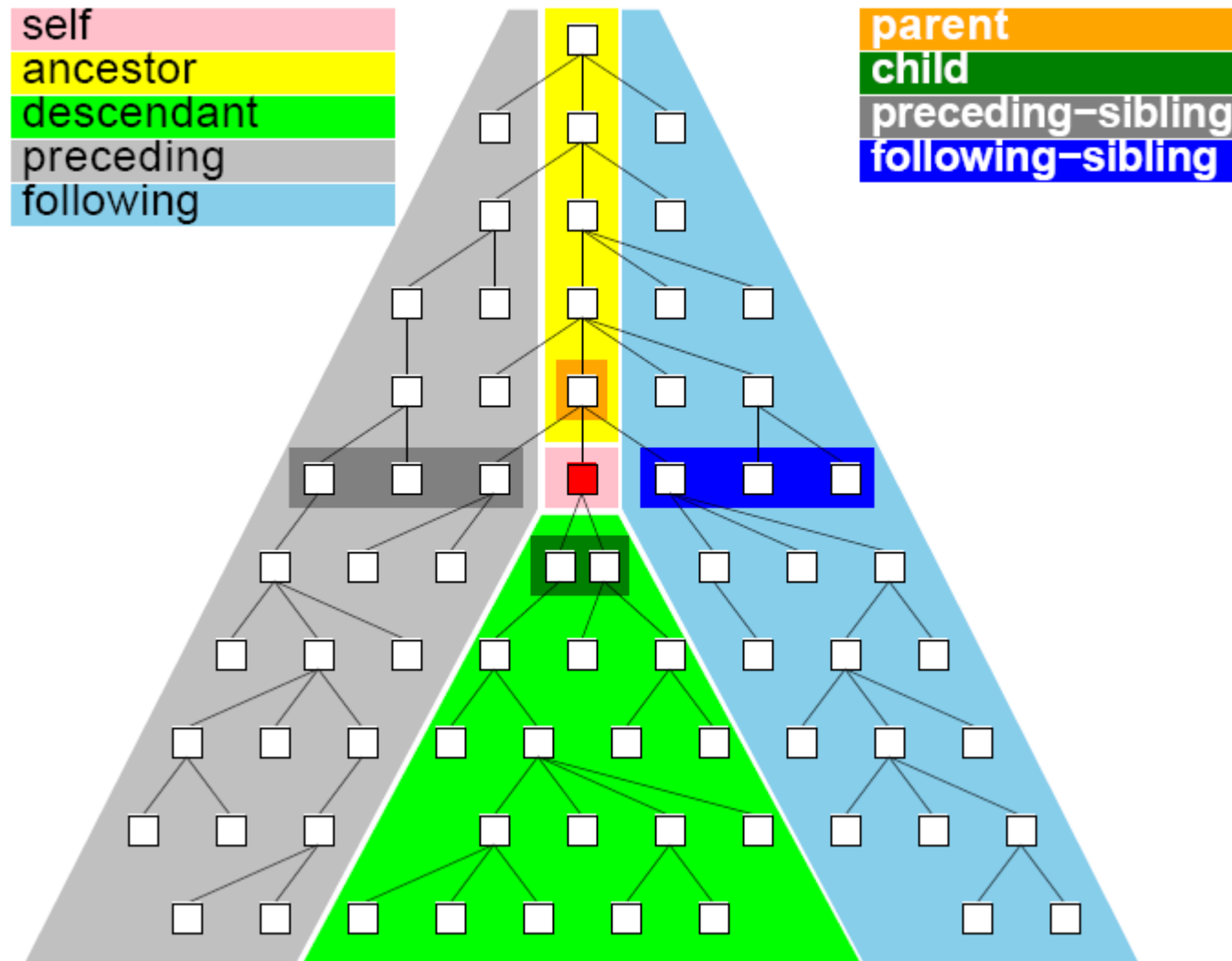
Axes de localisation (1)

Etant donné un nœud contexte, un axe est un ensemble de nœuds partageant une propriété commune vis-à-vis du nœud contexte

Liste des 13 axes :

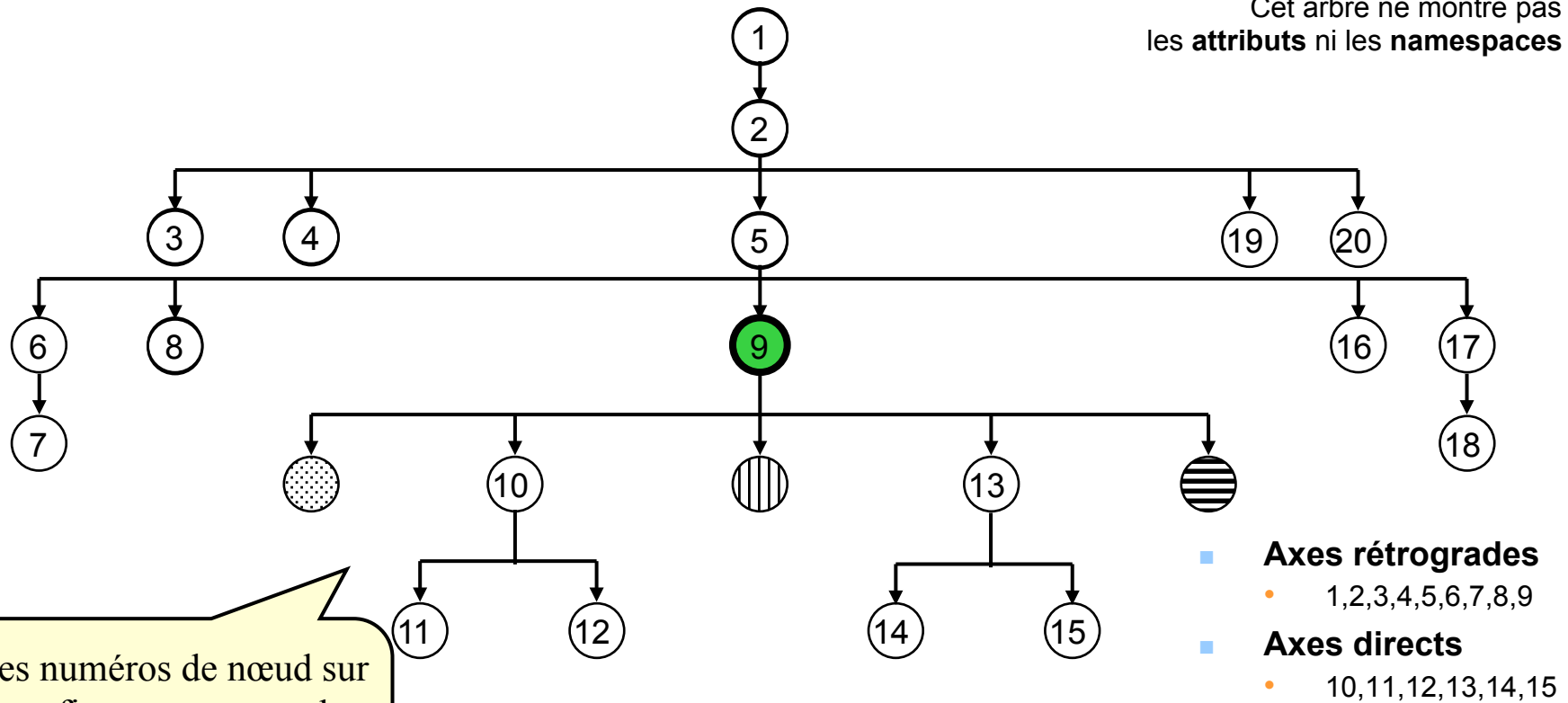
- **self** : le nœud courant lui-même ;
- **child** : les enfants du nœud courant ;
- **descendant, descendant-or-self** : tous les descendants du nœud courant (enfants, enfants des enfants, etc.) ;
- **parent** : le père du nœud courant ;
- **ancestor, ancestor-or-self** : les ancêtres du nœud courant (parent du parent, etc.) ;
- **attribute** : les attributs du nœud courant ;
- **preceding** : les nœuds précédents du nœud courant dans l'ordre de lecture du document (pas les ancêtres) ;
- **following** : les nœuds suivants du nœud courant, dans l'ordre de lecture du document (pas les descendants) ;
- **preceding-sibling, following-sibling** : les frères, précédant ou suivant, le nœud courant ;
- **namespace** : les espaces de noms.

Axes de localisation (2)



Représentation des axes de localisation

Cet arbre ne montre pas les **attributs** ni les **namespaces**



Nœud
contexte

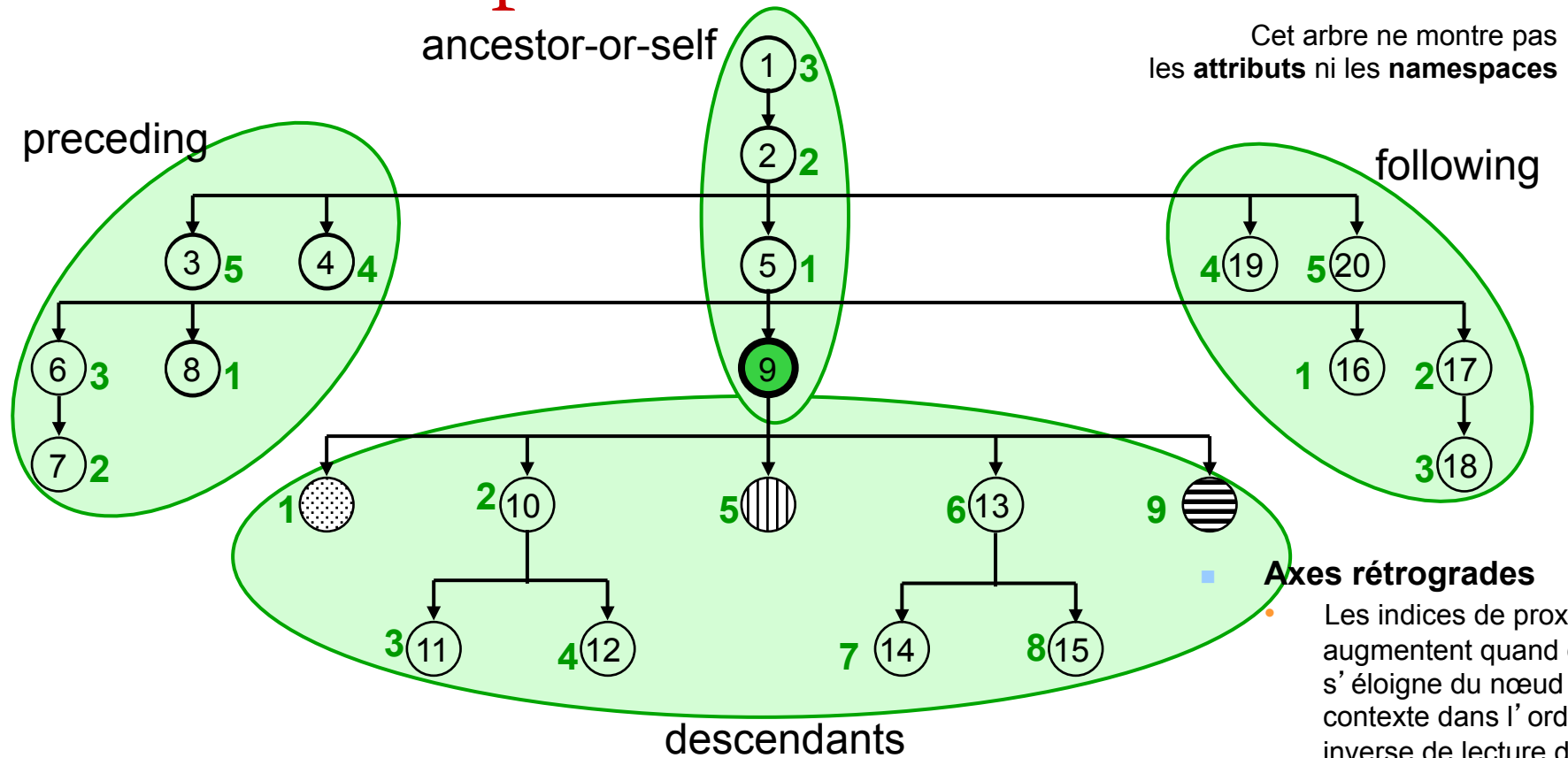
Nœud
processing-
instruction

Nœud
comment

Nœud
text

Nœud
element

Indices de proximité




Axes rétrogrades

Les indices de proximité augmentent quand on s'éloigne du nœud contexte dans l'ordre inverse de lecture du document

Axes directs


Les indices de proximité augmentent quand on s'éloigne du nœud contexte dans l'ordre de lecture du document

 Nœud contexte

 Nœud processing-instruction

 Nœud comment

 Nœud text

 Nœud element

Filtre (ou déterminant ou NodeTest)

- Étant donné un ensemble de nœuds fourni par un axe de localisation, le filtre es une fonction booléenne, qui, appliquée à un nœud de cet ensemble, dit si ce nœud doit rester ou non dans l'ensemble

Axe::Filtre

- Un filtre utilise la notion de **type principal de nœud**
 - Type de nœuds les plus fréquemment contenus dans l'axe de localisation
 - element, text, comment, processing-instruction, root, attribute

Formes de filtre

- `node()` : tous les nœuds
- `text()` : les nœuds textuels
- `*` : tous les éléments
- `nom` : les éléments portant ce *nom*
- `comment()` : les nœuds commentaires
- `processing-instruction('cible')` : les nœuds instructions, seulement les instructions *cible* si cet argument est fourni

Examples

- `child::figure`
- `child::*`
- `attribute::*`
- `child::text()`
- `child::comment()`
- `child::processing-instruction("play")`
- `child::node()`

Prédicats

- Un prédicat permet d'affiner le filtrage déjà effectué par le filtre sur l'axe de localisation, en éliminant de l'ensemble résultat les nœuds qui ne répondent pas à un certain critère
- Le prédicat est constitué d'une expression booléenne utilisant les connecteurs logiques and et or
[Expression-Booléenne]
- Les prédicats peuvent être appliqués en cascade
Axe::Filtre[Expression-Booléenne] [Expression-Booléenne]
...[Expression-Booléenne]

Exemples

■ Un prédicat peut

- Utiliser l'indice de proximité avec la fonction position()
 - `child::figure[position()=3]`
 - `child::figure[position()=last()]`
- Être constitué d'une étape de localisation
 - `child::figure[attribute::type='gif']`
- Avoir la forme [node-set]
 - `child::figure[attribute::scale]`
 - `child::figure[parent::paragraphe]`
 - `child::*[self::figure or self::image]`
- Avoir la forme [node-set=String]
 - `child::figure[attribute::scale="0.5"]`

Chemins de localisation

- Un chemin de localisation est composé d'une ou plusieurs étapes de localisation
 - Rappel
 - `LocationPath = "/"?, LocationStep, ("/", LocationStep)*`
- Il existe deux types
 - Chemin de localisation relatif
 - `child::chapitre/child::section`
 - Chemin de localisation absolu
 - `/child::chapitre/child::section`
- L'évaluation d'un chemin de localisation se fait de gauche à droite
- La lecture d'un chemin de localisation se fait de droit à gauche

Lecture d'un chemin de localisation (1)

■ Chemin de localisation sans prédicat

• $\text{etape1/etape2/.../etapeN}$
(F) (E) (S) (E) (S) ... (S) (E) (I)

- (I) se prononce « les »
- (E) se prononce « f qui sont les a » (f de filtre et a de axe)
- (S) se prononce « des »
- (F) se prononce « du nœud contexte ou racine »
- Exemple : $\text{child::chapitre/child::section/attribute::niveau}$
« Les niveaux qui sont les attributs des sections qui sont les enfants des chapitres qui sont les enfants du nœud contexte »

Lecture d'un chemin de localisation (2)

- Chemin de localisation avec prédicats
 - `child::chapitre/child::section[position()=2]/attribute::niveau`
« Les niveaux qui sont les attributs de certaines¹ sections qui sont des enfants des chapitres qui sont des enfants du nœud contexte »
¹uniquement celles qui ont la position 2 dans l'ensemble de sections dont il est question
- Chemin de localisation dans le prédicat
 - `/child::paragraphe[child::figure/attribut::scale]`
« les paragraphes (ayant un enfant figure ayant un attribut scale) qui sont les enfants de la racine du document »
 - `/child::paragraphe[child::figure/attribut::scale="0.5"]`
« les paragraphes (ayant un enfant figure ayant un attribut scale égale à 0.5) qui sont les enfants de la racine du document »

La syntaxe abrégée

- Notation plus simple mais pas aussi expressive que la notation étendue des chemins de localisation.
- L'équivalent étendu de certaines notations abrégées n'est pas toujours celui que l'on pense.

Forme longue	Abréviation
child::nom	nom
child::*	*
attribute::nom	@nom
attribute::*	@*
[position()=x]	[x]
self::node()	.
parent::node()	..
/descendant-or-self::node()	//

Exemples

Forme longue	Forme courte
child::figure	figure
child::text()	text()
/descendant-or-self::node()/child::figure	//figure
child::bloc/descendant-or-self::node()/child::figure	bloc//figure
child::bloc/[position=3]/child::figure[position()=1] [attribute::type='gif']	bloc[3]/figure[@type='gif'][1]
parent::node()/child::figure	../figure
self::node()/descendant-or-self::node()/ child::paragraphe	./paragraphe
/descendant-or-self::node()/child::*[not(child::*)]	//*[not(*)]