

Maîtriser XML et XSLT

Formation

Le langage de marquage XML

- XML: eXtensible Markup Language
- Standard (recommandation W3C, www.w3.org) pour
 1. documents structurés : héritier de SGML
 2. documents Web: généralisation de HTML
- XML facilite
 1. l'échange de données sur le Web
 2. l'intégration d'applications Web
 3. l'interrogation du Web

LeWorldWide Web Consortium (W3C)

- 400 partenaires industriels, parmi lesquels Oracle, IBM, Compaq, Xerox, Microsoft, etc..
- Laboratoires de recherche: MIT pour les États Unis, INRIA pour l'Europe, université Keio (Japon) pour l'Asie
- Objectif: définir un modèle pour faciliter l'échange de données sur le Web

XML = Modèle Documents Structurés

- XML est “compatible” avec SGML (standard pour documents structurés)
- l'édition de documents XML est simple (un éditeur texte standard suffit)
- la structure d'un document peut être prédéfinie par une grammaire (DTD) et analysée par un parseur
- le contenu d'un document est séparée de sa présentation: feuille de style XSL

Règles à respecter

Exemple 1

Un document XML doit comporter un ou plusieurs éléments.

Bien formé

```
<text>Ceci est un document XML</text>
```

Document XML bien formé comportant un élément

Bien formé

```
<text>Ceci est un  
<doctype>document XML</doctype>  
</text>
```

Document XML bien formé comportant plusieurs éléments

Mal formé

```
??? Ceci est un document XML ???
```

Un document XML doit comporter au moins un élément

Règles à respecter

Exemple 2

Il y a exactement un élément appelé élément racine ou élément document.

Bien formé

```
<book>Ceci est un livre</book>
```

`<book>` est l'élément racine

Bien formé

```
<list>  
<item>Item 1</item>  
<item>Item 2</item>  
<item>Item 3</item>  
</list>
```

`<list>` est l'élément racine

Mal formé

```
???  
<item>Item 1</item>  
<item>Item 2</item>  
<item>Item 3</item>  
???
```

Seul un élément racine est autorisé

Règles à respecter

Exemple 3

Le nom de la balise de fin d'un élément doit correspondre à celui de la balise de début.

Les noms tiennent compte des majuscules et des minuscules

Mal formé

```
<list>
<item>Voiture</itm>
<item>Avion</ITEM>
<item>Train</item>
</list>
```

<item> - </itm> et <item> - </ITEM> ne correspondent pas

Règles à respecter

Exemple 4

Les éléments délimités par les balises de début et de fin doivent s'imbriquer correctement les uns dans les autres.

Mal formé

```
<text>  
    <bold><italic>XML</bold></italic>  
</text>
```


Règles à respecter

Exemple 5

Chaque élément comporte une balise de fin ou adopte la forme spéciale.

Il n'y a aucune différence entre `<AAA></AAA>` et `<AAA/>` en XML.

Mal formé

```
<description>
Il y a des pommes <color>jaunes<color> et <color>rouges</color>.
</description>
```

Règles à respecter

Exemple 6

- Les noms d'éléments peuvent comporter des lettres, des chiffres, des tirets, des traits de soulignement, des deux-points ou des points.
- Le caractère deux-points (:) ne peut être utilisé que dans le cas particulier où il sert à séparer des espaces de noms.
- Les noms d'éléments commençant par xml, XML ou une autre combinaison de la casse de ces lettres sont réservés à la norme XML.

Document comportant des caractères autorisés

Bien formé

```
<permittedNames>
  <name/>
  <xsl:copy-of/>
  <A_long_element_name/>
  <A.name.separated.with.full.stops/>
  <a123323123-231-231/>
  <_12/>
</permittedNames>
```

Ce document comporte plusieurs erreurs.

Mal formé

```
<forbiddenNames>
  <A;name/>
  <last@name>
  <@#$$%^()&+?=/>
  <A*2/>
  <lex/>
</forbiddenNames>
```

Les noms ne peuvent pas commencer par xml

Mal formé

```
<forbiddenNames>
  <xmlTag/>
  <XMLTag/>
  <XmLTag/>
  <xMlTag/>
  <xmLTag/>
</forbiddenNames>
```

Chapitre 2: Structure des documents XML

2.1 Structure d'un document XML

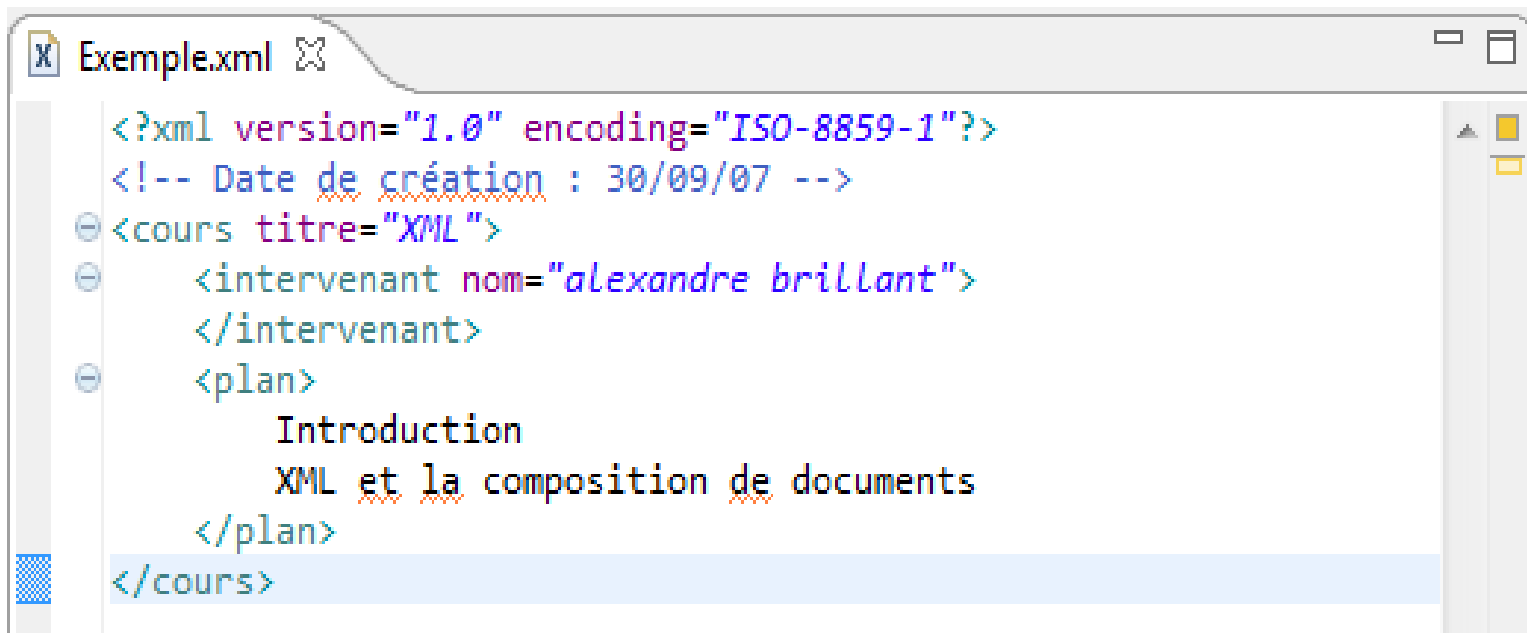
- *L'en-tête : le prologue*
- *Les instructions de traitement*
- *Les commentaires*
- *La déclaration du type de document*
- *Les noeuds élément*
- *Les attributs d'un élément*
- *Choix entre éléments et attributs*
- *Les noeuds textes*
- *Les entités du document*
- *Quelques règles de syntaxe*

2.2 Les espaces de noms

- *Application des espaces de noms dans un document XML*
- *Utilisation des espaces de noms dans un document XML*
- *L'espace de noms explicite*
- *La suppression d'un espace de noms*

2.1 Structure d'un document XML

Commençons par prendre un exemple simple de document XML :

A screenshot of a text editor window titled 'Exemple.xml'. The editor displays an XML document with the following content:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Date de création : 30/09/07 -->
<cours titre="XML">
  <intervenant nom="alexandre brillant">
  </intervenant>
  <plan>
    Introduction
    XML et la composition de documents
  </plan>
</cours>
```

 The XML is color-coded: opening and closing tags are in blue, attributes and values are in red, and text content is in black. The 'XML' attribute value is in quotes. The 'intervenant' tag has a 'nom' attribute. The 'plan' tag contains two lines of text: 'Introduction' and 'XML et la composition de documents'. The 'cours' tag has a 'titre' attribute. The document is enclosed in a root tag 'cours'.

Nous allons maintenant décortiquer la syntaxe et en comprendre les tenants et les aboutissants.

2.1 Structure d'un document XML

L'en-tête : le prologue

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

- Première ligne d'un document XML(pas d'espaces entre le début du document et cet élément) servant à donner les caractéristiques globales du document, c'est-à-dire :
 - La version XML, soit 1.0 ou 1.1
 - Le jeu de caractères employé (*encoding*).
Indique à l'interpréteur XML [Parser] le jeu de caractères à utiliser.
Lorsque l'encodage n'est pas précisé, c'est le standard UTF-8 qui est employé.

Rem: le prologue n'est pas obligatoire , défaut (version 1.0 et encoding UTF-8)

2.1 Structure d'un document XML

Les instructions de traitement

- Elles servent à donner à l'application qui utilise le document XML des informations.
- Un cas typique est l'utilisation avec les navigateurs Mozilla Firefox ou Internet Explorer pour effectuer la transformation d'un document XML en document XHTML affichable avec l'instruction :

```
<?xml-stylesheet type="text/xsl" href="affichage.xsl"?>
```

2.1 Structure d'un document XML

Les commentaires

- Ils se positionnent n'importe où après le prologue et peuvent figurer sur plusieurs lignes.

```
<!-- Date de création : 30/09/07 -->
```

Rem: les caractères -- sont interdits comme commentaires

2.1 Structure d'un document XML

La déclaration du type de document

- Cette déclaration optionnelle sert à attacher une grammaire de type DTD (*Document Type Definition*) à votre document XML.
- Elle est introduite avant la première balise (**racine**) de votre document sous cette forme : `<!DOCTYPE racine SYSTEM "URI vers la DTD">`
- **racine** est le premier élément (la première balise).
- L'URI peut être absolue(Ex: URL) ou relative au document.
- Exemple: `<!DOCTYPE cours SYSTEM "cours.dtd">`

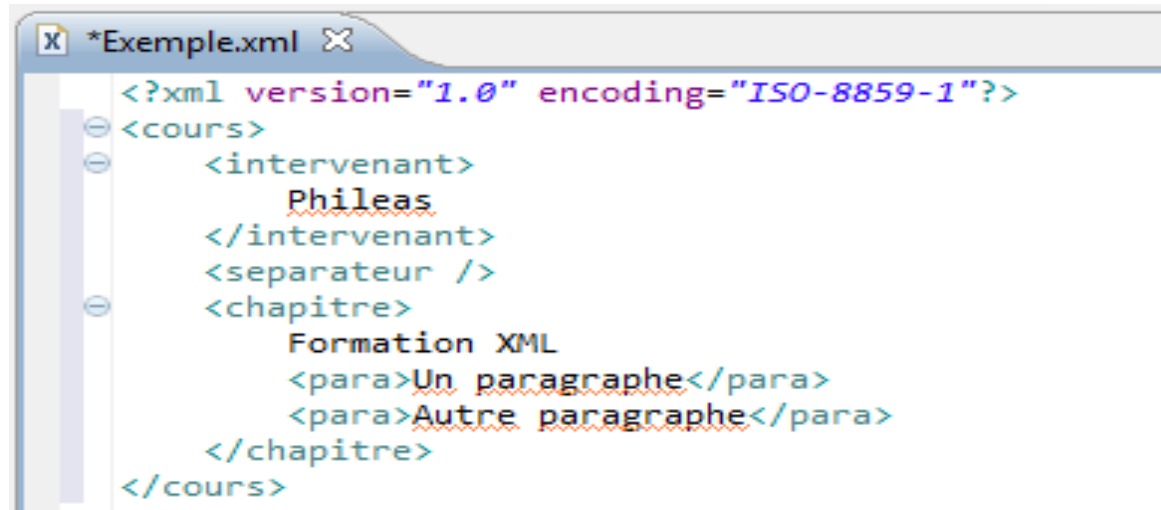
2.1 Structure d'un document XML

Les nœuds élément

- Les éléments gèrent la structuration des données d'un document XML, un peu à la manière des répertoires qui servent à l'organisation des fichiers.
- *Pour décrire ce que contiennent les éléments, on parle de modèle de contenu. On trouve :*
 - Rien : il n'y pas de contenu, l'élément est vide.
 - Du texte : nous détaillerons par la suite cette notion.
 - Un ou plusieurs éléments
 - Un mélange de textes et d'éléments (forme rare)

2.1 Structure d'un document XML

Exemple:

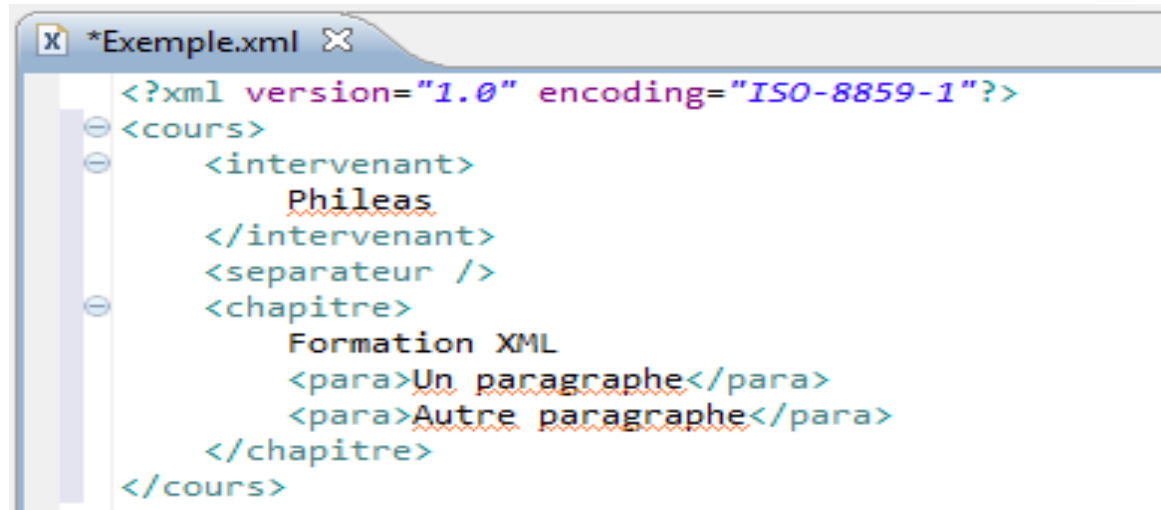


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
  <intervenant>
    Phileas
  </intervenant>
  <separateur />
  <chapitre>
    Formation XML
    <para>Un paragraphe</para>
    <para>Autre paragraphe</para>
  </chapitre>
</cours>
```

- **cours** : élément racine contenant trois éléments fils : intervenant, separateur et chapitre ;
- **intervenant** : élément contenant du texte ;
- **separateur** : élément sans contenu ;
- **chapitre** : élément contenant du texte et des éléments fils para ;
- **para** : élément contenant du texte.

2.1 Structure d'un document XML

Exemple:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
  <intervenant>
    Phileas
  </intervenant>
  <separateur />
  <chapitre>
    Formation XML
    <para>Un paragraphe</para>
    <para>Autre paragraphe</para>
  </chapitre>
</cours>
```

Si maintenant nous nous penchons sur la syntaxe, nous avons donc :

- **<element>** : balise ouvrante.
- **</element>** : balise fermante.
- **<element/>** : balise ouverte et fermée. C'est l'équivalent de **<element></element>**. Elle désigne donc un élément vide.

- **EXERCICE 1**

2.1 Structure d'un document XML

Les attributs d'un élément

- Un attribut est un couple (clé, valeur) associé à la définition d'un élément.
- Il est localisé dans la balise ouvrante de l'élément.
- Un élément peut donc avoir de 0 à n attributs **uniques**.

Voici un exemple de document XML avec des attributs :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
  <intervenant nom="fog" prenom="phileas" />
  <introduction />
  <chapitre numero="1">
    Formation XML
    <paragraphe>Détails du format</paragraphe>
  </chapitre>
</cours>
```

2.1 Structure d'un document XML

Choix entre éléments et attributs

- L'attribut peut sembler superflu. En effet, ce qui s'écrit avec des attributs peut également l'être en s'appuyant uniquement sur des éléments.

Exemple :

- Cas avec attributs :

```
<personne nom="brillant" prenom="alexandre"/>
```

- Cas sans attribut :

```
<personne>
```

```
  <nom>brillant</nom>
```

```
  <prenom>alexandre</prenom>
```

```
</personne>
```

2.1 Structure d'un document XML

Choix entre éléments et attributs (suite...)

Règles simples pour déterminer s'il est préférable d'utiliser un attribut ou un élément.

- Lorsqu'une valeur est de taille modeste, a peu de chance d'évoluer vers une structure plus complexe, et n'est pas répétée, alors l'attribut peut tout à fait convenir.
- Dans tous les autres cas, **l'élément** reste incontournable.

- **EXERCICE 2**

2.1 Structure d'un document XML

Les nœuds textes

- Dans un document XML, ce qui est appelé donnée est le texte qui est associé à l'attribut(=valeur), ou à l'élément(=contenu).
- Dans le vocabulaire propre à DOM (*Document Object Model*), la donnée apparaît comme un nœud texte.
- Certains caractères sont réservés à la syntaxe XML, il faut être vigilant lors de l'écriture des données.

Exemple:

```
<calcul>  
    if ( a<b et b>c) ...  
</calcul>
```

<b et **b>** n'est pas une balise mais fait partie des données liées à l'élément calcul.

2.1 Structure d'un document XML

Les nœuds textes (suite...)

Pour résoudre ce problème, nous disposons d'entités prédéfinies.

Voici la liste des entités prédéfinies :

- < équivalent de < (*less than*) ;
- > équivalent de > (*greater than*) ;
- & équivalent de & (*ampersand*) ;
- " équivalent de " (*quote*) ;
- ' équivalent de ' (*apostrophe*).

L'exemple précédent peut donc être correctement réécrit :

```
If (a&lt;b et b&gt;c)
```

2.1 Structure d'un document XML

Les nœuds textes (suite...)

- Les entités prédéfinies présentes en trop grand nombre dans un même bloc peuvent alourdir inutilement le document.
- Dans le cas du contenu textuel d'un élément (et uniquement dans ce cas), nous disposons des sections CDATA (*Character Data*).
- Cette section doit être considérée comme un bloc de texte dont les caractères seront pris tel quel par le parseur jusqu'à la séquence de fin `]]>`.

Exemple:

```
<![CDATA[  
    <element>C'est un document XML  
    </element>  
]]>
```

- Dans cet exemple, **<element>** n'est pas considéré comme une balise de structuration, mais comme du texte.

- **EXERCICE 3**

2.1 Structure d'un document XML

Quelques règles de syntaxe

Ces règles de syntaxe sont à respecter impérativement pour qu'un document XML soit bien formé.

- Le nom d'un élément ne peut commencer par un chiffre.
- Si le nom d'un élément est composé d'un seul caractère il doit être dans la plage [a-zA-Z] ou _ ou :.
- Avec au moins 2 caractères, le nom d'un élément peut contenir _, -, . et : plus les caractères alphanumériques (attention, le caractère : est réservé à un usage avec les espaces de nom que nous aborderons par la suite).
- Tous les éléments ouverts doivent être fermés.
- Un élément parent est toujours fermé après la fermeture des éléments fils.
- Ref: <http://www.w3.org/TR/2006/REC-xml-20060816/#sec-well-formed>

2.1 Structure d'un document XML

Quelques conventions de nommage

Voici quelques conventions souvent employées dans les documents XML :

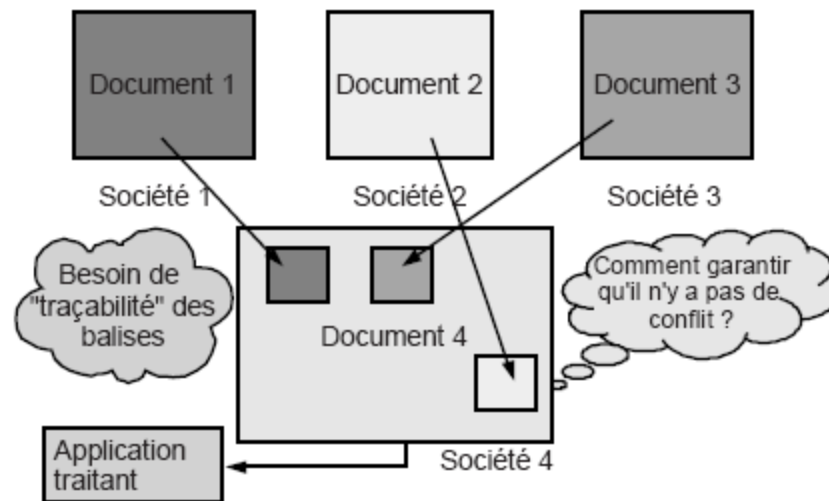
- Employer des minuscules pour les attributs et les éléments.
- Éviter les accents dans les noms d'attributs et d'éléments pour des raisons de compatibilité avec les outils du marché qui proviennent souvent d'un univers anglo-saxon.
- Préférer les guillemets délimitant les valeurs d'attribut.
- Séparer les noms composés de plusieurs mots par les caractères -, _, . ou une majuscule.

2.2 Les espaces de noms

Les espaces de noms sont un concept très commun en informatique.

- Par exemple, dans le langage de programmation Java, les *packages* ***servent à délimiter*** la portée d'une classe.

Application des espaces de noms dans un document XML



2.2 Les espaces de noms

Application des espaces de noms dans un document XML(suite...)

- Pour délimiter la portée d'une balise, d'un attribut ou d'une valeur d'attribut, nous disposons d'espaces de noms (*namespace*).
- L'utilisation des espaces de noms garantit une forme de traçabilité de la balise et évite les ambiguïtés d'usage.
- Pour que les espaces de noms aient un sens, il faut pour chacun d'eux un identifiant unique.
- Cet identifiant unique peut être simplement l'URL, puisqu'il ne peut y avoir qu'un propriétaire pour une URL donnée.

Attention: *L'URL ne signifie pas qu'il doit y avoir un document sur votre serveur HTTP*

2.2 Les espaces de noms

Vocabulaire : qualification des éléments

- Un élément qui est connu dans un espace de noms est dit **qualifié** ; dans le cas contraire, il est dit **non qualifié**.

2.2 Les espaces de noms

L'espace de noms par défaut

Un premier usage consiste à utiliser simplement l'espace de noms par défaut.

- Ce dernier est précisé par un pseudo-attribut **xmlns**.
- La valeur associée sera une URL garantissant l'unicité de l'espace de noms.
- L'espace de noms par défaut s'applique à l'élément où se situe sa déclaration et à tout son contenu.

Exemple:

```
<chapitre xmlns="http://www.masociete.com">  
  <paragraphe>  
    ...  
  </paragraphe>  
</chapitre>
```

Ici l'élément chapitre est dans l'espace de noms
<http://www.masociete.com>.

C'est également le cas de l'élément paragraphe, puisqu'il est dans l'élément chapitre.

2.2 Les espaces de noms

L'espace de noms par défaut(suite...)

- Nous pouvons changer l'espace de noms par défaut même dans les éléments enfants : dans ce cas, une règle de priorité est appliquée.

Attention, les espaces de noms ne sont pas imbriqués ; on ne peut appliquer qu'un seul espace de noms à la fois.

Exemple:

```
<chapitre xmlns="http://www.masociete.com">  
  <paragraphe xmlns="http://www.autresociete.com">  
    ...  
  </paragraphe>  
</chapitre>
```

L'élément paragraphe n'appartient pas à l'espace de noms <http://www.masociete.com> mais uniquement à l'espace de noms <http://www.autresociete.com>.

2.2 Les espaces de noms

L'espace de noms explicite

- Pour disposer de davantage de souplesse dans ces espaces et pouvoir également les appliquer aux attributs et valeurs d'attributs, la syntaxe introduit la notion de préfixe.
- On déclare un préfixe comme un pseudo-attribut commençant par **xmlns:prefixe**.
- Une fois déclaré, il est employable uniquement dans l'élément le déclarant et dans son contenu.

Exemple:

```
<p:resultat xmlns:p="http://www.masociete.com">  
</p:resultat>
```

L'élément résultat est dans l'espace de noms <http://www.masociete.com> grâce au préfixe **p**.

2.2 Les espaces de noms

L'espace de noms explicite(suite...)

On peut déclarer et utiliser plusieurs espaces de noms grâce aux préfixes.

Exemple: `<p:res xmlns:p="http://www.masociete.com" xmlns:p2="http://www.autresociete.com">
 <p2:res>
 </p2:res>
</p:res>`

Le premier élément **res** est dans l'espace de noms <http://www.masociete.com> alors que l'élément **res** à l'intérieur est dans l'espace de noms <http://www.autresociete.com>.

2.2 Les espaces de noms

La suppression d'un espace de noms

- Aucun espace de noms n'est utilisé lorsqu'il n'y a pas d'espace de noms par défaut ni de préfixe.

Exemple:

```
<p:element xmlns:p="http://www.masociete.com">  
  <autreelement />  
</p:element>
```

L'élément **element** est dans l'espace de noms `http://www.masociete.com` alors que l'élément **autreelement**, qui n'est pas préfixé, n'a pas d'espace de noms.

2.2 Les espaces de noms

La suppression d'un espace de noms (suite...)

- Pour supprimer l'action d'un espace de noms il suffit d'utiliser la valeur vide "", ce qui revient à ne pas avoir d'espace de noms.

Exemple:

```
<element xmlns="http://www.masociete.com">
  <autreelement xmlns="">
    .. Aucun d'espace de noms
  </autreelement>
  <encoreunelement>
    ... Espace de nom par défaut
  </encoreunelement>
</element>
```

L'élément **element** est dans l'espace de noms <http://www.masociete.com> alors que l'élément **autreelement** n'est plus dans un espace de noms.

L'élément **encoreunelement** se trouve également dans l'espace de noms <http://www.masociete.com>, de par l'espace de noms de son parent.

- **EXERCICE 4**

2.2 Les espaces de noms

Application d'un espace de noms sur un attribut

- Les espaces de nom peuvent s'appliquer via un préfixe sur un attribut ou une valeur d'attribut.
- Cet emploi peut servir à :
 - contourner la règle qui veut que l'on ne puisse pas avoir plusieurs fois un attribut de même nom sur une déclaration d'élément.
 - lever l'ambiguïté sur une valeur d'attribut

Exemple:

```
<livre xmlns:p="http://www.imprimeur.com" p:quantite="p:50lots">  
  <papier type="p:A4" />  
</livre>
```

Dans cet exemple, nous avons qualifié l'attribut quantité ainsi que les valeurs d'attribut 50lots et A4.

Chapitre 3: Validation des documents XML

3.1 Rôle de la validation dans l'entreprise

3.2 La première forme de validation par DTD

- *La définition d'un élément*
- *La définition d'un attribut*
- *La définition d'une entité*

3.2 La validation par un schéma W3C

- Les différentes formes de type
- Les définitions globales et locales
- L'assignation d'un schéma à un document XML
- Les catégories de type simple
- L'utilisation des types complexes
- Les définitions d'éléments
- Réutilisation des définitions
- L'utilisation des clés et références de clés
- Relations entre schémas

3.1 Rôle de la validation dans l'entreprise

- La validation va renforcer la qualité des échanges en contraignant l'émetteur de données et le consommateur de données à vérifier la cohérence des données structurées en XML.
- Par cohérence, il faut entendre :
 - à la fois le vocabulaire (éléments, attributs et espaces de noms),
 - l'ordre,
 - et les quantités.
- La plupart des outils, et notamment les parseurs XML, proposent des outils de validation.
- Les parseurs courants supportent une ou plusieurs formes de grammaires.
 - **Les DTD** (*Document Type Definition*), sont présentes dans la plupart des outils.
 - **Les schémas W3C**, une forme de grammaire plus moderne et plus complexe.

3.2 La première forme de validation par DTD

- Une DTD (*Document Type Definition*)
 - Avantage : rapide à écrire
 - Inconvénient: pauvre en possibilités de contrôle (typage de données, par exemple).
- Une DTD peut être interne ou externe au document XML.

Par exemple, un document XML ayant une DTD externe cours.dtd, située Dans le même répertoire que notre document XML (accès relatif), se présente sous la forme :

```
<?xml version="1.0"?>
<!DOCTYPE cours SYSTEM "cours.dtd">
<cours>
    ...
</cours>
```

3.2 La première forme de validation par DTD

La définition d'un élément

- L'élément (ou balise) est exprimé par l'instruction ELEMENT suivie du nom de l'élément que l'on souhaite décrire et de son contenu.
- Voici une synthèse de cette syntaxe :

```
<!ELEMENT unNom DEF_CONTENU>
```

3.2 La première forme de validation par DTD

- *La définition d'un élément (suite...)*

Quelques exemples :

```
<!ELEMENT personne (nom_prenom | nom)>  
<!ELEMENT nom_prenom (#PCDATA)>  
<!ELEMENT nom (#PCDATA)>
```

Cela nous autorise deux documents XML, soit :

```
<personne>  
  <nom_prenom>Brillant Alexandre</nom_prenom>  
</personne>
```

ou bien

```
<personne>  
  <nom>Brillant</nom>  
</personne>
```

L'opérateur de choix, `|`, indique que l'un ou l'autre de deux éléments doit être présent.

L'opérateur de suite (ou séquence), `,`, indique que les deux éléments doivent être présents.

3.2 La première forme de validation par DTD

- *La définition d'un élément (suite...)*
- Les contenus (élément ou groupe d'éléments) peuvent être quantifiés par les opérateurs *, + et ?.
- Ces opérateurs sont liés au concept de cardinalité. Lorsqu'il n'y a pas d'opérateur, la quantification est de 1 (donc toujours présent).
- Voici le détail de ces opérateurs :
 - * : 0 à n fois ;
 - + : 1 à n fois ;
 - ? : 0 ou 1 fois.

3.2 La première forme de validation par DTD

- *La définition d'un élément (suite...)*

Quelques exemples :

<!ELEMENT plan (introduction?,chapitre+,conclusion?)>

L'élément plan contient un élément introduction optionnel, suivi d'au moins un élément chapitre et suivi par un élément conclusion optionnel.

<!ELEMENT chapitre (auteur*,paragraphe+)>

L'élément chapitre contient de 0 à n éléments auteur suivi d'au moins un élément paragraphe.

<!ELEMENT livre (auteur?,chapitre)+>

L'élément livre contient au moins un élément, chaque élément, étant un groupe d'éléments où l'élément auteur, est optionnel et l'élément chapitre est présent en un seul exemplaire.

3.2 La première forme de validation par DTD

La définition d'un attribut

- Les attributs sont précisés dans l'instruction **ATTLIST**.
- Cette dernière, étant indépendante de l'instruction **ELEMENT**, on précise à nouveau le nom de l'élément sur lequel s'applique le ou les attributs.
- On peut considérer qu'il existe cette forme syntaxique :

nom TYPE OBLIGATION VALEUR_PAR_DEFAULT

3.2 La première forme de validation par DTD

La définition d'un attribut(suite...)

Le **TYPE** peut être principalement :

- CDATA : du texte (*Character Data*) ;
- ID : un identifiant unique (combinaison de chiffres et de lettres) ;
- IDREF : une référence vers un ID ;
- IDREFS : une liste de références vers des ID (séparation par un blanc) ;
- NMTOKEN : un mot (donc pas de blanc) ;
- NMTOKENS : une liste de mots (séparation par un blanc) ;
- Une énumération de valeurs : chaque valeur est séparée par le caractère |.

3.2 La première forme de validation par DTD

La définition d'un attribut(suite...)

L'OBLIGATION ne concerne pas les énumérations qui sont suivies d'une valeur par défaut.

Dans les autres cas, on l'exprime ainsi :

- #REQUIRED : attribut obligatoire.
- #IMPLIED : attribut optionnel.
- #FIXED : attribut toujours présent avec une valeur.

3.2 La première forme de validation par DTD

La définition d'un attribut(suite...)

Quelques exemples :

```
<!ATTLIST chapitre  
    titre CDATA #REQUIRED  
    auteur CDATA #IMPLIED>
```

L'élément chapitre possède ici un attribut titre obligatoire et un attribut auteur optionnel.

```
<!ATTLIST crayon  
    couleur (rouge|vert|bleu) "bleu">
```

L'élément crayon possède un attribut couleur dont les valeurs font partie de l'ensemble rouge, vert, bleu.

- **Tuto 3: Création d'une DTD**
- **EXERCICE 1**
- **EXERCICE 2**

3.3 La validation par un schéma W3C

- *Introduction*
- *Les définitions globales et locales*
- *L'assignation d'un schéma à un document XML*
- *Les catégories de type simple*
- *L'utilisation des types complexes*
- *Les définitions d'éléments*
- *Réutilisation des définitions*
- *L'utilisation des clés et références de clés*
- *Relations entre schémas*

3.3 La validation par un schéma W3C

Introduction

Limitations des DTD

1. Premièrement, les DTD ne sont pas au format XML.
2. Deuxièmement, les DTD ne supportent pas les « espaces de nom »
3. Troisièmement, le « typage » des données est extrêmement limité.

Apports des schémas

1. Le typage des données est introduit.
2. Le support des espaces de nom.
3. Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif
4. Les schémas sont très facilement concevables par modules.

3.3 La validation par un schéma W3C

Les premiers pas

- Le but d'un schéma est de définir une classe de documents XML.
- Il permet de décrire les autorisations d'imbrication et l'ordre d'apparition des éléments et de leurs attributs, tout comme une DTD.

Structure de base

- Un premier point est qu'un fichier Schema XML est un document XML.
- Comme tout document XML, un Schema XML commence par un prologue, et a un élément racine.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- déclarations d'éléments, d'attributs et de types ici -->
</xsd:schema>
```

- L'élément racine est l'élément xsd:schema.
- Tout élément d'un schéma doit commencer par le préfixe xsd.

3.3 La validation par un schéma W3C

Type simple et type complexe

- Type simple permettent de donner un type aux :
 - Éléments qui ne contiennent que du texte et sans d'attribut
 - Exemple `<prenom>Yves</prenom>`
 - Attributs (leur contenu n'est que textuel !)
- Type complexe permettent de donner un type aux :
 - Éléments qui ne contiennent que du texte mais avec attributs
 - Exemple `<ville codePostal="35000">Rennes</ville>`
 - Éléments qui contiennent d'autres éléments, à contenu mixte ou non

3.3 La validation par un schéma W3C

- *Les définitions globales et locales*
- Les composants globaux apparaissent au premier niveau au sein de l'élément `<xsd:schema>`
 - Ils sont toujours nommés (`xsd:... name="..."`)
 - Leur nom doit être unique au sein de leur type de composant
- Les composants locaux
 - Leur nom a une portée locale au type complexe dans lequel ils sont définis
 - Types simples et types complexes définis localement sont anonymes (ils ne peuvent être réutilisés)

3.3 La validation par un schéma W3C

Types simples

- Les types simples peuvent être
 - Primitifs (ne dérivant pas d'un autre)
 - Dérivés d'un autres type simple
 - Par une *liste* : séquence de types séparés par des blancs
 - Par une *union* : union d'autres types simples
 - Par une *restriction* :
 - length, minLength, maxLength (longueur de listes)
 - enumeration (liste de valeurs)
 - pattern (expression régulière à la Perl)
 - whitespace (préserver, remplacer, réduire les espaces)
 - minInclusive,maxInclusive (intervalles bornés de valeurs)

3.3 La validation par un schéma W3C

Types simples prédéfinis

- Ils peuvent être primitifs ou dérivés
- Il y en a une cinquantaine
- Exemples :

string	negative-integer
normalized-string	date
name	time
NCName	datetime
language	duration
float	ID
double	IDREF
long	IDREFS
int	boolean
short	hexBinary
positive-integer	anyURI

3.3 La validation par un schéma W3C

La création de nouveaux types simples (Restriction d'un type)

Exemple : restriction de la longueur d'une chaîne

```
<xs:simpleType name="chaîne32">  
  <xs:restriction base="xs:string">  
    <xs:maxLength value="32"/>  
  </xs:restriction>  
</xs:simpleType>
```

3.3 La validation par un schéma W3C

La création de nouveaux types simples (Restriction d'un type)

Exemple : **restriction du type** `date`

```
<xs:simpleType>  
  <xs:restriction base="xs:int">  
    <xs:pattern value="0{1,2}10?" />  
  </xs:restriction>  
</xs:simpleType>
```

PS : Pour exp reg voir Annexe1

3.3 La validation par un schéma W3C

La création de nouveaux types simples (Restriction d'un type)

Exemple : Entier borné

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


3.3 La validation par un schéma W3C

La création de nouveaux types simples (Restriction d'un type)

Exemple : Type énuméré - 1

```
<xs:simpleType name="couleurType">  
  <xs:restriction base="xs:string">  
    <xs:enumeration value="bleu"/>  
    <xs:enumeration value="blanc"/>  
    <xs:enumeration value="rouge"/>  
  </xs:restriction>  
</xs:simpleType>
```

3.3 La validation par un schéma W3C

La création de nouveaux types simples (la construction de liste)

- Les valeurs d'une liste sont séparées par un blanc et il est impossible d'utiliser un autre séparateur.

Exemple 1:

```
<xs:simpleType>  
    <xs:list itemType="xs:int"/>  
</xs:simpleType>
```

La liste 10 20 333 4 3 est donc bien cohérente avec ce type.

3.3 La validation par un schéma W3C

La création de nouveaux types simples (La construction de liste)

Exemple 2: On peut écrire également une forme plus complexe en précisant de nouveaux types simples :

```
<xs:simpleType>
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:int">
        <xs:minInclusive value="40"/>
        <xs:maxInclusive value="50"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:list>
</xs:simpleType>
```

- Dans cet exemple, nous autorisons une liste d'entiers bornés entre 40 et 50 (par exemple 40 43 50 44).

3.3 La validation par un schéma W3C

La création de nouveaux types simples (l'union)

- L'union sert à combiner plusieurs types simples ; il suffira qu'une valeur corresponde à l'un des types pour que la valeur soit correcte.

3.3 La validation par un schéma W3C

La création de nouveaux types simples (l'union)

```
<xs:simpleType>
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:int"/>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:boolean"/>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

3.3 La validation par un schéma W3C

L'utilisation des types complexes (Élément `<xs:complexType>`)

- Déclaration de *type complexe*

```
<xs:complexType name="..." mixed="...">  
  <!-- un modèle de contenu -->  
  <!-- des déclarations d'attributs -->  
</xs:complexType>
```

1. Déclaration d'attribut

```
<xs:attribute name="..." type="..." use="...">
```

- L'attribut `type` fait référence à un type simple
- L'attribut `use` prend une valeur parmi
 - `required`, `optional`, `prohibited`

2. Modèle de contenu ... à suivre ...

3.3 La validation par un schéma W3C

L'utilisation des types complexes

(Modèle de contenu pour éléments)

3.3 La validation par un schéma W3C

La représentation de l'élément vide avec attributs

- L'attribut rend la structure de l'élément plus complexe. Son typage est donc également sous cette forme.

Voici un exemple :

```
<xs:element name="img">  
  <xs:complexType>  
    <xs:attribute name="src" type="xs:string"/>  
  </xs:complexType>  
</xs:element>
```

Cette balise s'exprimera donc dans un document XML sous cette forme :

```

```


3.3 La validation par un schéma W3C

La représentation de l'élément avec contenu simple et attributs

Exemple:

```
<xs:element name="auteur">  
  <xs:complexType>  
    <xs:simpleContent>  
      <xs:extension base="xs:string">  
        <xs:attribut name="nom" type="xs:string"/>  
      </xs:extension>  
    </xs:simpleContent>  
  </xs:complexType>  
</xs:element>
```

- On peut le lire de la façon suivante : l'élément auteur est un type complexe dont le contenu simple (typé xs:string) a été étendu pour lui ajouter un attribut nom.

3.3 La validation par un schéma W3C

La représentation de l'élément avec contenu complexe et attributs

Prenons l'exemple suivant :

```
<xs:element name="auteur">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="prenom" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:token"/>
  </xs:complexType>
</xs:element>
```

- Dans ce cas, l'élément auteur contient un élément nom suivi d'un élément prenom et possède un attribut id.

3.3 La validation par un schéma W3C

La représentation d'un contenu mixte

- Un contenu mixte sert à faire cohabiter du texte et des éléments.
- On positionne un attribut **mixed** dans un type complexe pour obtenir l'effet recherché.

Un exemple :

```
<xs:complexType name="personType" mixed="true">
  <xs:sequence>
    <xs:element name="nom" type="xs:string"/>
    <xs:element name="prenom" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="employe" type="personType"/>
```

On pourrait donc, par exemple, écrire ce document XML :

```
<employe>Bonjour <nom>MrDupont</nom>,<prenom>Jean</prenom> de Paris</employe>
```

3.3 La validation par un schéma W3C

Contenu complexe Expressions régulières

Trois opérateurs de composition

- Élément `<xs:sequence>` séquence d'éléments
- Élément `<xs:choice>` choix d'éléments
- Élément `<xs:all>` permutation d'éléments

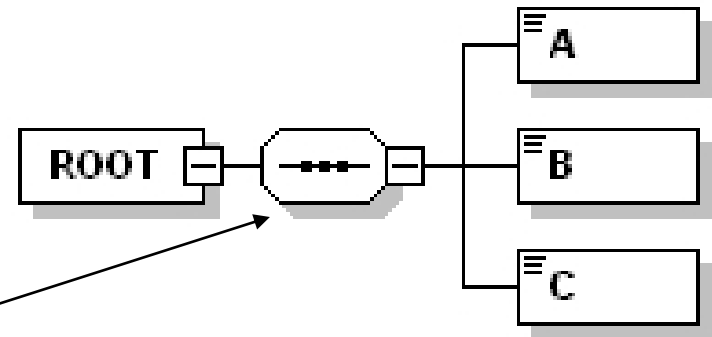
3.3 La validation par un schéma W3C

Premier connecteur : la séquence

- La séquence caractérise des éléments fils présents dans un ordre donné.

- Exemple:

- XMLSchema



```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:sequence>
      <xs:element name="A" type="xs:string"/>
      <xs:element name="B" type="xs:string"/>
      <xs:element name="C" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Dans cet exemple, l'élément ROOT contient nécessairement 3 éléments fils de contenu simple A, B et C.

3.3 La validation par un schéma W3C

Premier connecteur : la séquence (suite...)

- Nous aurions pu également l'écrire avec un type global (par exemple **rootType**) :
- XMLSchema

```
< xs:complexType name="rootType">
  <xs:sequence>
    <xs:element name="A" type="xs:string"/>
    <xs:element name="B" type="xs:string"/>
    <xs:element name="C" type="xs:string"/>
  </xs:sequence>
</ xs:complexType >
<xs:element name="ROOT" type="rootType"/>
```

DTD

```
<!ELEMENT ROOT (A,B,C) >
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

3.3 La validation par un schéma W3C

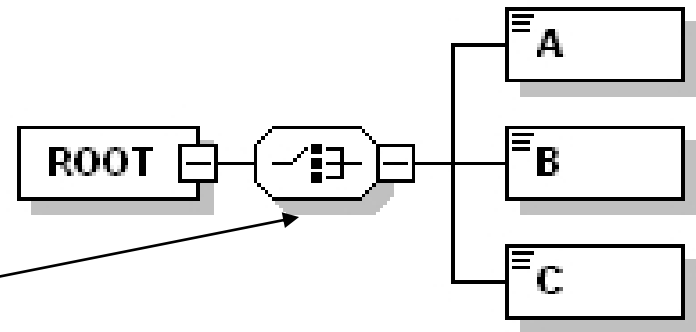
Deuxième connecteur : le choix

- DTD

```
<!ELEMENT ROOT (A|B|C) >
```

- XMLSchema

```
<xs:element name="ROOT">  
  <xs:complexType mixed="false">  
    <xs:choice>  
      <xs:element name="A" type="xs:string"/>  
      <xs:element name="B" type="xs:string"/>  
      <xs:element name="C" type="xs:string"/>  
    </xs:choice>  
  </xs:complexType>  
</xs:element>
```



Ce qui autorisera un élément **A** ou bien un élément **B** ou bien un élément **C** dans l'élément **ROOT**.

3.3 La validation par un schéma W3C

Dernier connecteur : tout

- Le connecteur **all** est propre au schéma et caractérise un ensemble d'éléments de présence obligatoire mais sans contrainte sur l'ordre (toutes les permutations sont donc valides).
- Exemple

```
<xs:element name="plan">
  <xs:complexType>
    <xs:all>
      <xs:element name="auteur" type="xs:string"/>
      <xs:element name="chapitres" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Dans cet exemple l'élément plan contiendra les éléments auteur et chapitres dans n'importe quel ordre.

3.3 La validation par un schéma W3C

La limitation des quantités d'éléments : les cardinalités

- Tout comme dans les DTD, les cardinalités sont possibles sur les éléments de connecteur ou sur les connecteurs eux-mêmes (jouant le rôle des parenthèses dans une DTD).
- Ces cardinalités sont positionnées par les attributs **minOccurs** et **maxOccurs**. L'infini est caractérisé par la chaîne **unbounded**.
- Pour faire le parallèle avec les DTD, nous retrouvons l'équivalent des opérateurs ?, + et * avec :
 - **?** : minOccurs="0" maxOccurs="1" ;
 - **+** : minOccurs="1" maxOccurs="unbounded" ;
 - ***** : minOccurs="0" maxOccurs="unbounded".

Lorsqu'on ne précise rien, minOccurs et maxOccurs ont la valeur 1.

3.3 La validation par un schéma W3C

La limitation des quantités d'éléments : les cardinalités

Exemple sur un élément :

```
<xs:element name="plan">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="auteur" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="chapitre" type="xs:string" minOccurs="2" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Dans cet exemple, l'élément plan contient un élément auteur suivi d'au moins 2 éléments chapitre.

3.3 La validation par un schéma W3C

La limitation des quantités d'éléments : les cardinalités

- Exemple sur un connecteur :

```
<xs:element name="plan">
  <xs:complexType>
    <xs:sequence minOccurs="2" maxOccurs="3">
      <xs:element name="auteur" type="xs:string"/>
      <xs:element name="chapitre" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Avec cet exemple, nous contrôlons que l'élément plan contient entre 2 et 3 suites d'éléments auteur et chapitre.

3.3 La validation par un schéma W3C

Définition d'un attribut dans un type complexe

- L'attribut implique la présence d'un type complexe.
- Il est toujours placé en dernière position.
- L'attribut en lui-même, ne contenant que du texte, est un type simple.
- L'attribut peut être global et donc réutilisable au sein de plusieurs définitions de type complexe.

3.3 La validation par un schéma W3C

Définition d'un attribut dans un type complexe

- Exemple :

```
<xs:element name="personne"  
  <xs:complexType>  
    ...  
    <xs:attribute name="nom" type="xs:string"/>  
  </xs:complexType>  
</xs:element>
```

- Dans cet exemple, nous associons à l'élément `personne` l'attribut `nom`. L'attribut `nom` est local à l'élément `personne` ; il ne peut pas être employé dans un autre élément.

3.3 La validation par un schéma W3C

Définition d'un attribut dans un type complexe

- Pour créer un attribut réutilisable pour des définitions de type complexe, il faut le rendre global en le positionnant sous la racine schema.
- L'attribut ref sert à désigner la définition d'un attribut global.

3.3 La validation par un schéma W3C

Définition d'un attribut dans un type complexe

- Exemple :

```
<xs:schema ...>
  <xs:attribute name="nom">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="5"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <complexType name="monType">
    <xs:attribute ref="nom"/>
  </complexType>
</xs:schema>
```

- Dans cet exemple, l'attribut nom est déclaré globalement et est utilisé dans la définition d'un type complexe monType.

3.3 La validation par un schéma W3C

Limitation de l'attribut : les cardinalités (use)

- Voici un exemple :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="a">
    <xs:complexType>
      <xs:attribute name="t1" use="required" type="xs:int"/>
      <xs:attribute name="t2" use="optional" type="xs:string" default="valeur"/>
      <xs:attribute name="t3" use="required" type="xs:token" fixed="autre"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


3.3 La validation par un schéma W3C

Les groupes d'attributs

- Des définitions d'attributs communes à plusieurs définitions d'éléments peuvent être concentrées dans des groupes d'attributs.
- Exemple :

```
<xs:attributeGroup name="RGB">  
  <xs:attribute name="rouge" type="xs:byte" use="required"/>  
  <xs:attribute name="vert" type="xs:byte" use="required"/>  
  <xs:attribute name="bleu" type="xs:byte" use="required"/>  
</xs:attributeGroup>
```

Pour faire référence à ce groupe d'attributs, il suffit d'insérer l'instruction **<xs:attributeGroup ref="RGB"/>** à l'endroit où nous souhaitons utiliser ces trois attributs.

3.3 La validation par un schéma W3C

Cas des groupes d'éléments

- Un exemple :

```
<xs:group name="monGroupe">
  <xs:sequence>
    <xs:element name="contact" type="xs:string"/>
    <xs:element name="note" type="xs:string"/>
  </xs:sequence>
</xs:group>
<xs:element name="liste1">
  <xs:complexType maxOccurs="unbounded">
    <xs:group ref="monGroupe"/>
  </xs:complexType>
</xs:element>
```

- Nous avons globalement défini le groupe monGroupe qui a été intégré à la définition complexe de l'élément liste1.

4 XSL

- XSL se compose de deux parties:
 - XSLT: un langage pour transformer des documents XML
 - XPath : une langue pour la navigation dans des documents XML
- **Objectif** : référencer noeuds (éléments, attributs, commentaires, ...) dans un document XML.

4.1 XPATH

Présentation

- Le langage XPATH offre un moyen d'identifier un ensemble de nœuds dans un document XML.
- Toutes les applications ayant besoin de repérer un fragment de document XML peuvent utiliser ce langage.
- Les feuilles de style XSL, les pointeurs XPOINTER et les liens XLINK utilisent de manière intensive les expressions XPATH.
- XPATH est un premier pas vers un langage d'interrogation d'une base de données XML (XQuery).

4.1 Path

Pour voir comment fonctionne XPath, nous allons voir:

1. comment XPATH utilise un modèle de représentation arborescente d'un document XML
2. Les expressions Xpath ou comment cheminer dans un tel arbre, avec la notion de ***chemin de localisation*** (*axes de localisation, les filtres, et les prédicats.*)

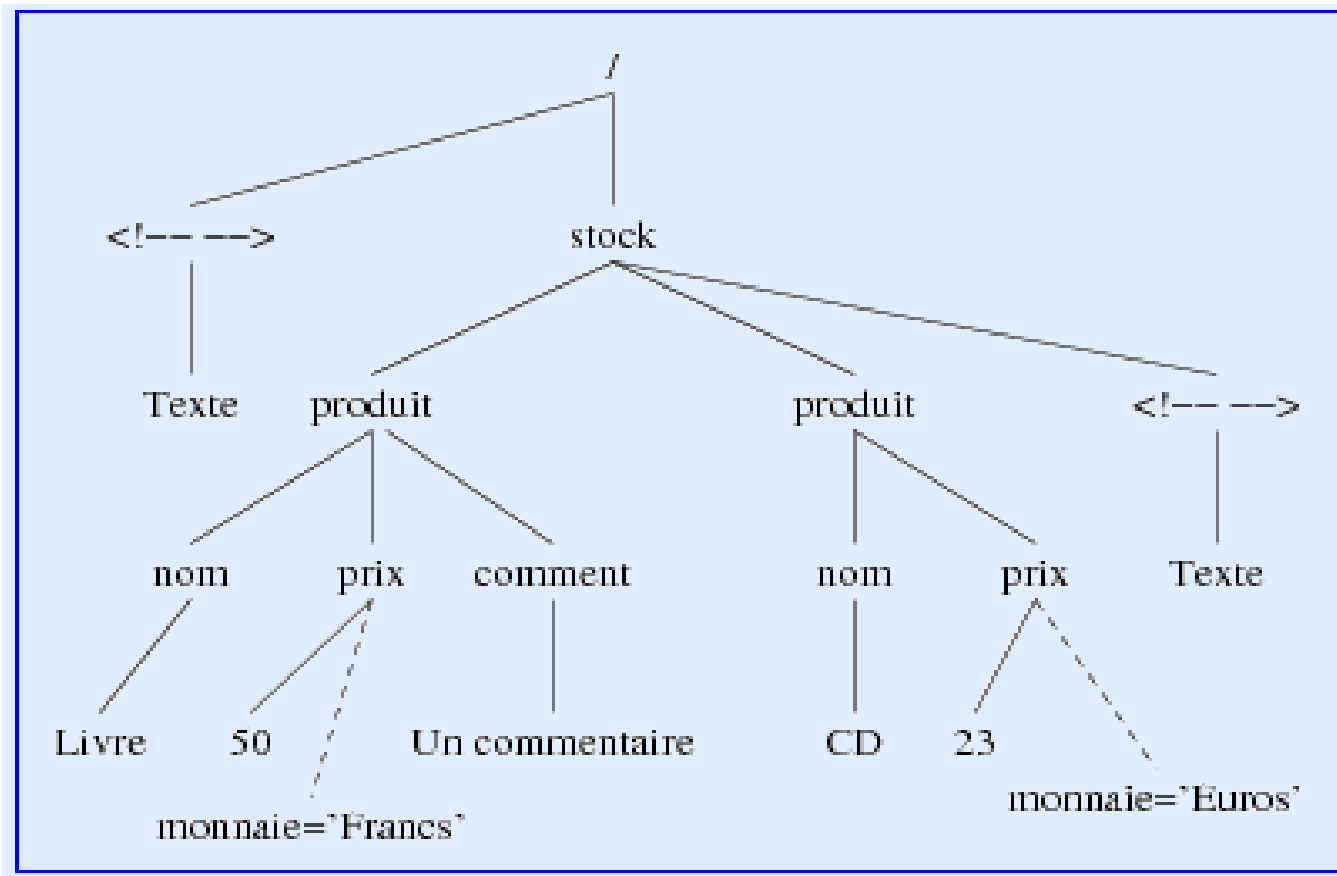
4.1 XPATH

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Texte -->
<?play audio armide.avi?>
<stock>
  <produit>
    <nom> Livre </nom>
    <prix monnaie="Francs"> 50 </prix>
    <comment> Un commentaire </comment>
  </produit>
  <produit>
    <nom> CD </nom>
    <prix monnaie="Euros"> 23 </prix>
  </produit>
  <!-- Texte -->
</stock>
```

4.1 XPATH

Modèle arborescent d'un document XML vu par Xpath



4.1 XPATH

Les expressions

- La forme générale d'une *expression XPATH* est

sélecteur1/sélecteur2/... (exp. relative)
/sélecteur1/sélecteur2/... (exp. absolue)

- Chaque **sélecteur** sélectionne un ensemble de nœuds en fonction du résultat du sélecteur précédent.
- L'ensemble **initial** est soit le *nœud courant* (forme relative) soit *la racine* (forme absolue).
- Exemple: **/stock/produit/comment**

4.1 XPATH

Les sélecteurs de nœuds

Une étape de positionnement est défini par un **axe** et un **test**:

1. **l'axe** sélectionne un ensemble de nœuds par rapport à leur position absolue ou relative à un autre nœud.
2. **le test** est évalué pour chaque nœud dans la sélection.

Formule: `axe::test/axe::test/.../axe::test`

Exemples: `/child::film/descendant::acteurs/child::@nom`

4.1 XPATH

Les axes de recherche

- Un axe de localisation représente une première approximation de ce que l'on veut.
- Un axe se base sur la notion de voisinage du nœud contexte :
 - les enfants,
 - les frères, les ascendants, etc. ..on choisit ce qui se rapproche le plus du node-set souhaité, quitte ensuite à filtrer les nœuds en trop.
- L'approximation doit toujours se faire par excès, puisqu'il est possible de filtrer, mais pas d'ajouter.

4.1 XPATH

Les axes: Père/Fils

- L'axe le plus utilisé est celui du fils child:: suivi d'un test.

/child::* (le **root element**) raccourci **/***

/child::DEBUT (récupère le premier élément DEBUT après la racine)

s'écrit

/DEBUT

4.1 XPATH

Exemple: Pour la suite les éléments sélectionnés apparaîtront en rouge

XPath : `/ROOT/AA`

`<ROOT>`

`<AA>`

`<BB/>`

`</AA>`

`<AA>`

`<BB/>`

`<CC/>`

`</AA>`

`</ROOT>`

4.1 XPATH

- L'axe parent s'écrit **parent::**. Raccourci « .. »

XPath : **/ROOT/*/BB/..**

<ROOT>

<AA>

<BB/>

</AA>

<EE>

<BB/>

<CC/>

</EE>

</ROOT>

4.1 XPATH

- Se sélectionner soi-même s'écrit **self::node()** mais on lui préfère le raccourci « . »

XPath : `/ROOT/AA/.`

`<ROOT>`

`<AA>`

`<BB/>`

`</AA>`

`<EE>`

`<BB/>`

`<CC/>`

`</EE>`

`<ROOT>`

4.1 XPATH

- L'axe descendant sélectionne tous les nœuds spécifiés contenus dans le nœud original. Il s'écrit **descendant::**

XPath : `/ROOT/ descendant::*`

`<ROOT>`

`<AA>`

`<BB>`

`<CC/>`

`</BB>`

`<DD/>`

`</AA>`

`<AA>`

`<BB/>`

`</AA>`

`</ROOT>`

4.1 XPATH

- Il existe une variante permettant de sélectionner en plus le noeud à l'origine **descendant-or-self::**, le raccourci pour cela est **//**

XPath : `/ROOT/descendant-or-self::*` ou `/ROOT//*`

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<BB/>

</AA>

</ROOT>

4.1 XPATH

- L'axe des ancêtres fonctionne dans le sens inverse. Il s'écrit **ancestor::** comme pour descendant, il existe une version **ancestor-or-self::**

XPath : `/ROOT/*/*/ ancestor::*`

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<BB/>

</AA>

</ROOT>

4.1 XPATH

- Par l'axe **preceding::**, on sélectionnera tous les noeuds précédant - hors ancêtres - le noeud en lecture.

XPath : `//EE/preceding ::*`

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<EE/>

</AA>

</ROOT>

4.1 XPATH

- Par l'axe **following::**, on sélectionnera tous les noeuds suivant - hors descendants et ancêtres - le noeud en lecture.

XPath : **//BB/following::***

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<EE/>

</AA>

</ROOT>

4.1 XPATH

Il existe aussi la notion de « frères » en Xpath:

- les « frères » précédents par l'axe **preceding-sibling::**
- les « frères » suivants par l'axe **following-sibling::**

XPath : `//BB/following-sibling ::*`

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<EE/>

</AA>

</ROOT>

4.1 XPATH

- Pour sélectionner des attributs par XPath, on utilisera l'axe **attribute::*** dont le raccourci est **@***.

XPath : `//@*`

`<ROOT>`

`<AA>`

`<BB test1='1' >`

`<CC/>`

`</BB>`

`<DD/>`

`</AA>`

`<AA test2='1' >`

`< BB/>`

`</AA>`

`</ROOT>`

4.1 XPATH

- XPath permet de faire l'union entre deux XPath par l'opérateur « | ».

XPath : **//DD|//EE**

<ROOT>

<AA >

<BB>

<CC/>

</BB>

<DD/>

</AA>

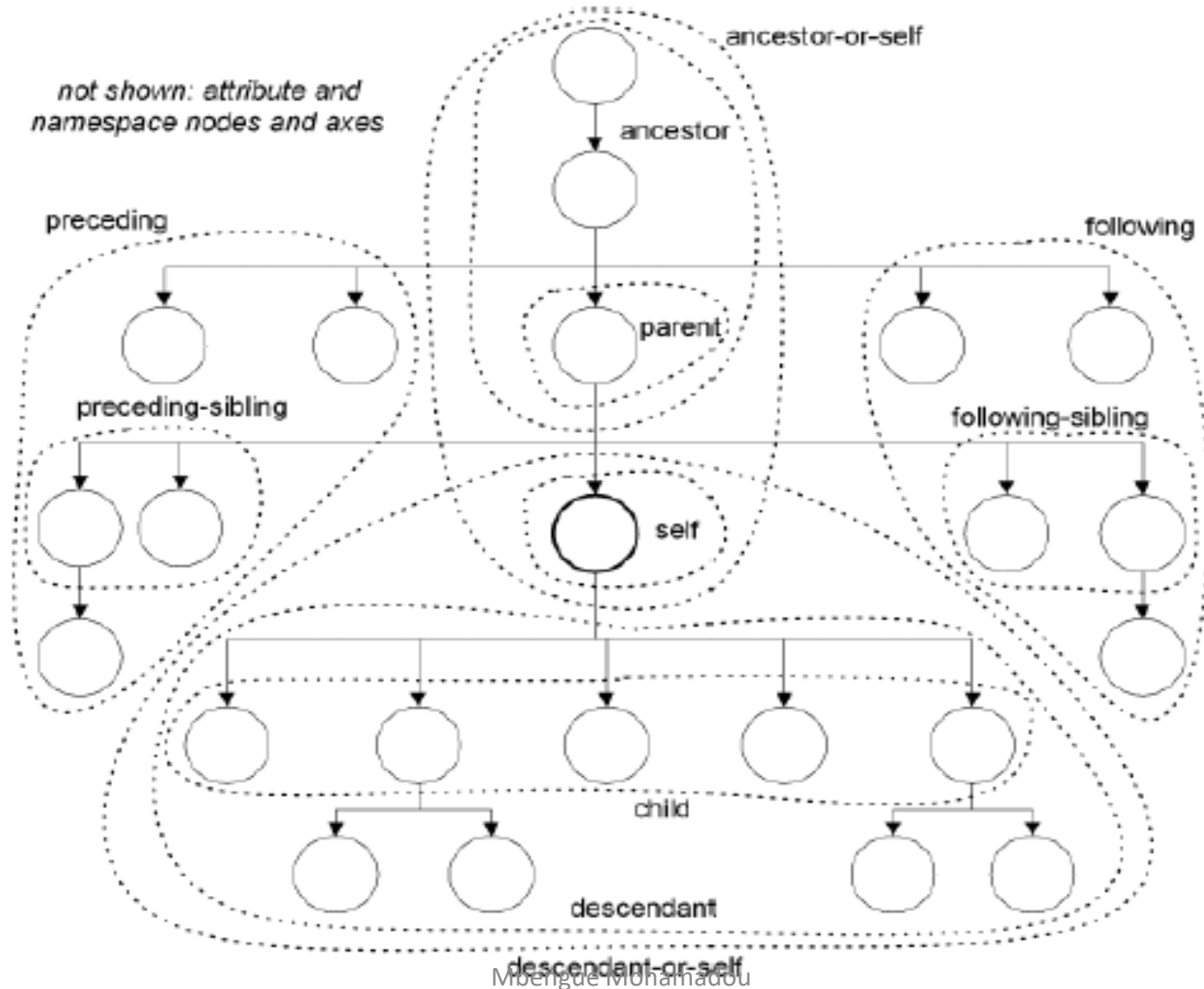
<AA>

<EE/>

</AA>

</ROOT>

4.1 XPATH



4.1 XPATH

Les tests(prédicats)

- Un prédicat commence par [et se termine par].
- Un prédicat peut contenir des XPath et ainsi d'autres prédicats.
- Un prédicat est une condition, on peut le comparer à la clause **WHERE** en SQL.

4.1 XPATH

Les tests: Opérateurs

Les opérateurs booléens :

Les booléens sont `true()` et `false()`. Le noeud vide, la chaîne vide et zéro sont convertis en `false()`.

- NON : c'est une fonction en XPath, **`not(...)`**, elle englobe la partie sur laquelle porte la négation ;
- OU : **`or`** ;
- ET : **`and`** ;
- EGAL et DIFFERENT : `=` et `!=` attention `!=` n'est pas la négation de `=`, comme cela sera détaillé plus tard.
- COMPARETEURS D'ORDRE : `<=`, `<`, `>=`, `>`.

4.1 XPATH

Les tests: Opérateurs

Les opérateurs numériques:

Si une de ces opérations est effectuée sur une chaîne de caractères la valeur renvoyée est **NaN**(Not a Number).

- **ADDITION** : + ;
- **SOUSTRACTION** : -, attention sur l'opérateur de soustraction, il faut toujours le faire précéder et suivre d'un espace sinon l'expression peut être confondue avec un nom d'élément ;
- **MULTIPLICATION** : * ;
- **DIVISION**: div ;
- **MODULO**: mod.

4.1 XPATH

Exemple sur les tests:

- Lors de test entraînant une comparaison tout nœud est converti en sa valeur textuelle.
- Celle-ci pourra être considérée comme un nombre (si sa forme le permet) ou une chaîne de caractères.
- **Pour la suite les éléments sélectionnés apparaîtront en rouge.**

4.1 XPATH

Test de valeur

- Il porte sur la valeur textuelle du noeud sélectionné. On ne peut comparer que des types simples.
- On notera ici l'importance du self::node() et de son raccourci «.».

Xpath : `//AA[.=1]`

`<ROOT>`

`<AA>1</AA>`

`<AA>2</AA>`

`</ROOT>`

Xpath : `//AA[. > 1]`

`<ROOT>`

`<AA>1</AA>`

`<AA>2</AA>`

`</ROOT>`

4.1 XPATH

Test de position

- La fonction position() renvoie la position du noeud en lecture dans son contexte parent.
- La numérotation des positions en XPath commence à 1.

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<BB/>

</AA>

</ROOT>

Xpath : **/ROOT/AA[position()=2]**

ou **/ROOT/AA[2]** (écriture raccourcie)
sélectionne le deuxième fils AA de ROOT

4.1 XPATH

Test de position 2

- Attention un XPath du type `//*[2]` ne renvoie pas le deuxième noeud de la sélection mais tous les noeuds sélectionnés précédemment qui sont des deuxièmes fils.

```
<ROOT>
  <AA>
    <BB>
      <CC/>
    </BB>
    <DD/>
  </AA>
  <AA>
    <BB/>
  </AA>
</ROOT>
```

Xpath : `//*[2]`

4.1 XPATH

Exemple de test avec des fonctions

last() : Récupérer le dernier fils d'un nœud :

```
<ROOT>
  <AA>
    <BB/>
  </AA>
  <AA>
    <BB/>
    <BB/>
    <BB/>
  </AA>
  <AA>
    <BB/>
    <BB/>
  </AA>
</ROOT>
```

Xpath : **/ROOT/AA/BB[position()=last()]**

4.2 XSLT

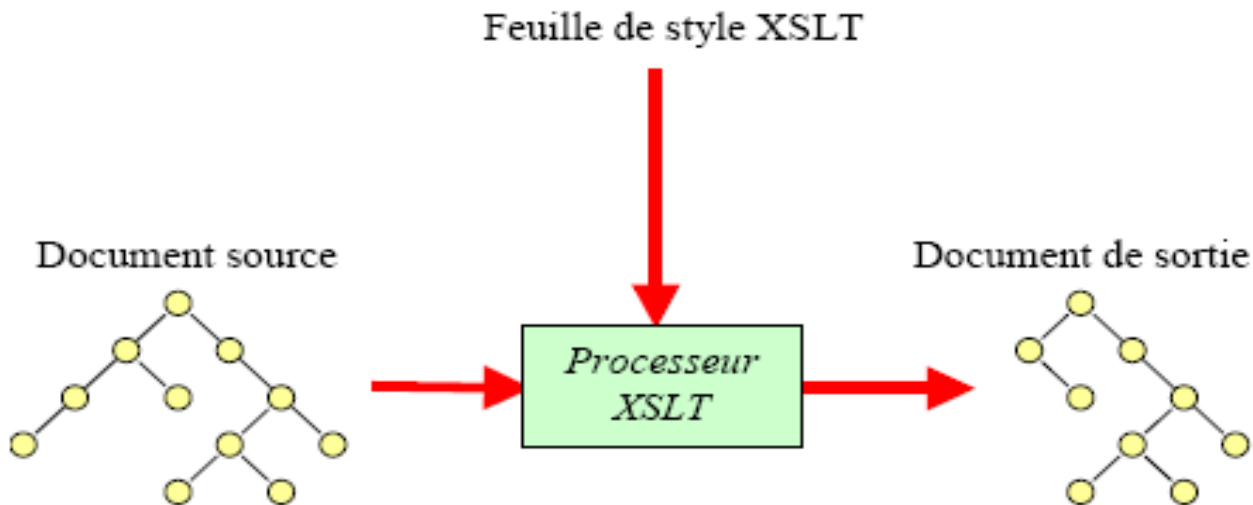
Présentation

- XSL est synonyme de e**X**tensible **S**tylesheet **L**anguage, et c'est un langage de feuille de style pour les documents XML.
- XSLT est synonyme de transformations XSL.
- XSLT est utilisé pour transformer un document XML dans un autre document XML, ou un autre type de document qui est reconnu par un navigateur, comme le HTML et XHTML.

4.2 XSLT

Présentation

XSLT = Transformation d'arbre



4.2 XSLT

Présentation

- Avec XSLT, vous pouvez ajouter / supprimer des éléments et des attributs vers un fichier de sortie.
- Vous pouvez également :
 - réorganiser
 - Effectuer des tri
 - prendre des décisions sur les éléments à masquer et d'afficher,
 - et beaucoup plus...

4.2 XSLT

Présentation

- XSLT utilise XPath pour trouver des informations dans un document XML.
- XPath est utilisé pour naviguer à travers les éléments et attributs dans les documents XML.

4.2 XSLT

Présentation

Comment ça marche?

- XSLT utilise XPath pour définir les parties du document source qui correspondent à un ou plusieurs modèles prédéfinis.
- Lorsqu'une correspondance est trouvée, XSLT va transformer la partie correspondante du document source dans le document résultat.

XSLT est une recommandation du W3C

4.2 XSLT

XSLT Navigateurs

- Les principaux navigateurs sont compatibles avec XML et XSLT.
 - **Mozilla Firefox** (version 3)
 - **Internet Explorer** (version 6)
 - **Google Chrome** (version 1)
 - **Opéra** (version 9)
 - **Apple Safari** (version 3)

4.2 XSLT

XSLT – Transformation

- L'élément racine du document qui déclare être une feuille de style XSL est **<xsl:stylesheet>** ou **<xsl:transform>**.
- **Exemple:**

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

Ou:

```
<xsl:transform version="1.0"  
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

4.2 XSLT

L'élément `<xsl:stylesheet>`

- Élément racine d'un document XSLT

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

- Attribut `version` : version de langage XSL (obligatoire)
- Attribut `xmlns:xsl` : espace de nom XSL

4.2 XSLT

L'élément `<xsl:output>`

- Format de sortie du document résultat

```
<xsl:output method="xml" version="1.0"  
            encoding="UTF 8" indent="yes"/>
```

- Attribut `method` : type du document en sortie
- Attribut `encoding` : codage du document
- Attribut `indent` : indentation en sortie

4.2 XSLT

L'élément `<xsl:output>`

Type de document en sortie

- Trois types de document en sortie
 - **xml** : vérifie que la sortie est bien formée
 - *(sortie par défaut)*
 - **html** : accepte les balises manquantes, génère les entités HTML (´ ...)
 - *(sortie par défaut si XSL reconnaît l'arbre de sortie HTML4)*
 - **text** : tout autre format textuel :
 - du code Java, format Microsoft RTF, LaTeX

4.2 XSLT

XSLT – Transformation

Exemple: Nous voulons transformer le document XML suivant ("cdcatalog.xml") en XHTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>
```

4.2 XSLT

Créer une feuille de style XSL

- Ensuite, vous créez une feuille de style XSL ("cdcatalog.xsl") avec un modèle de transformation:

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

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

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

4.2 XSLT

Lier la feuille de style XSL au document XML

- Ajouter la référence feuille de style XSL à votre document XML ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>
```

4.2 XSLT

L'élément `<xsl:template>`

- Une feuille de style XSL est composée d'une ou plusieurs ensemble de règles qui sont appelées modèles.
- L'élément `<xsl:template>` est utilisé pour construire des modèles.
- La valeur de l'attribut **match** est une expression XPath
- Exemple: **match** = "/" définit l'ensemble du document.

4.2 XSLT

L'élément `<xsl:value-of>`

- L'élément `<xsl:value-of>` est utilisé pour extraire la valeur d'un nœud sélectionné.

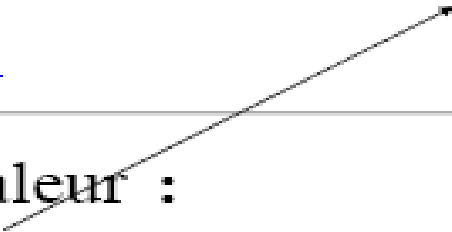
4.2 XSLT

L'élément `<xsl:value-of>`

Élément `<xsl:value-of>`

- Générer le contenu d'un élément

```
<xsl:template match="carteDeVisite">  
  <p>Nom : <xsl:value-of select="nom"/>  
  </p>  
</xsl:template>
```



- Sélection de la valeur :
 - attribut `select` : expression xpath
 - ici : le texte contenu dans l'élément `nom` de l'élément `carteDeVisite`

4.2 XSLT

L'élément `<xsl:value-of>`

Résultat de `<xsl:value-of>` et type nœud

- Le nœud sélectionné est un *élément*
 - Concaténation de tous les textes qui se trouvent comme contenu de cet élément et de ses descendants
- Le nœud est un nœud *text*
 - Texte du nœud lui même
- Le nœud est un *Attribut*
 - Valeur de l'attribut normalisée (pas d'espace de début et fin)
- Le nœud est une *Instruction de traitement*
 - Valeur de l'instruction de traitement
(sans les marques `<?` et `?>` et sans le nom)
- Le nœud est un *Commentaire*
 - Le texte du commentaire (sans les marques `<!--` et `-->`)

4.2 XSLT

L'élément `<xsl:value-of>`

Exemple 2

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Règle

```
<xsl:template match="note">  
  <xsl:value-of select="."/>  
</xsl:template>
```

- En sortie

```
enseigne XML au SEP
```

4.2 XSLT

L'élément `<xsl:value-of>`

Exemple 3

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Règle

```
<xsl:template match="note">  
  <xsl:value-of select="text()"/>  
</xsl:template>
```

- En sortie

```
enseigne
```

Seul le premier élément sélectionné est produit

4.2 XSLT

L'élément `<xsl:for-each>`

- Itération sur une ensemble de nœuds

```
<xsl:template match="/carnetDAdresse">  
  <xsl:for-each select="carteDeVisite">  
    <p><xsl:value-of select="nom"/></p>  
  </xsl:for-each>  
</xsl:template>
```

4.2 XSLT

XSLT <xsl:sort> élément

- Permet de trier l'ensemble des nœuds sélectionnés par les instructions avant de les traiter.
- Tri sur plusieurs critères (possible)

4.2 XSLT

L'élément `<xsl:if>`

- Conditionnelle

```
<xsl:for-each select="carteDeVisite">  
  <xsl:value-of select="nom"/>  
  <xsl:if test="position() != last()">,  
  </xsl:if>  
</xsl:for-each>
```

- Génère une virgule après chaque nom sauf pour le dernier