# Bases de données et web – CM5

L3 Informatique & L3 MIASHS

# XSLT

**XSLT** est un langage spécialisé dans la transformation de document XML en tout autre type de document.

Les principaux principes de XSLT :

- Un programme XSLT, appelé aussi feuille de style, est composé de règles de transformations, appelés aussi des ***templates***.
- Un template est appliqué à un type de nœud spécifique du document XML en entrée et produit une partie de la sortie attendue. Une fois que tous les templates ont été *exécutés*, la sortie générée est la transformation complète.
- Le modèle d'exécution est le suivant : initialement, un template est appliqué au nœud racine du document en entrée. De là, le programme continu son exécution en suivant la suite logique d'appels de templates. Il est également possible de mettre tout le code de la transformation dans le template du nœud racine.

***Remarque*** : Une feuille de style XSLT est lui-même un document XML. L'espace de nom de la bibliothèque de fonctions XSLT est `xsl:`.

# Hello World! en XSLT

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version
       ="1.0">
4
5      <xsl:output methode="xml" encoding="utf-8"/>
6
7      <xsl:template match="/">
8          <hello>world</hello>
9      </xsl:template>
10
11 </xsl:stylesheet>
```

La structure générale de la feuille de style :

- une racine `<xsl:stylesheet>`. L'URL après `xmlns:xsl` est l'adresse de la bibliothèque de fonctions XSLT;
- des déclarations, comme par exemple ici `<xsl:output>`. Ici on indique que l'on va générer du XML encodé en utf-8;
- des règles de transformation, ici `<xsl:template>`. Ici le template agit sur la racine du document (`match="/"`) et va simplement créer `<hello>world</hello>`.

# Utilisation d'une feuille de style

Une feuille de style peut être utilisée :

- par un programme utilisant une bibliothèque de fonctions XSLT;
- en ligne de commande, par exemple avec `xsltproc`;
- dans un contexte de publication sur le web. Soit côté serveur par un script CGI, php, . . . . Soit côté client par un moteur de transformation XSLT, intégré dans la plus part des navigateurs web.

```php
<?php

// Chargement du XML.
$xml = new DOMDocument;
$xml->load('collection.xml');

// Chargement du XSL.
$xsl = new DOMDocument;
$xsl->load('collection.xsl');

// Configuration du transformateur.
$proc = new XSLTProcessor;
$proc->importStyleSheet($xsl); // Attachement des règles xsl.

// Affichage de la transformation.
echo $proc->transformToXML($xml);

?>
```

# L'élément <xsl:template>

```
 7  <xsl:template match="livre">
 8      Le titre du livre est : <xsl:value-of select="titre"/>
 9
10      <h2>Liste des auteurs</h2>
11      <ul>
12          <xsl:apply-templates select="auteurs/nom"/>
13      </ul>
14  </xsl:template>
```

Un template est composé de :

- une expression XPath qui détermine les nœuds auxquels sera appliqué le template. L'expression XPath est la valeur de l'attribut `match`;
- le corps du template. C'est ici qu'est indiqué ce qui va être généré à partir des nœuds trouvés par le template.

# XPath et XSLT

Le rôle d'une expression XPath de l'attribut `match` est particulier à XSLT. L'expression décrit les nœuds qui peuvent être sélectionnés lors de l instanciation du template. Ces expressions sont appelées des ***patterns***. Elles doivent répondre aux prérequis suivants :

- Un pattern doit toujours faire référence à un ensemble de nœuds. Par exemple : `<xsl:template match="1">` est incorrecte.
- Un pattern doit être facile à lire *à l'œil nu*. Par exemple : `<xsl:template match="preceding::*[12]>"` est correcte mais difficile à comprendre. Le but du pattern d'un template est notamment de comprendre rapidement à quoi il fait référence.

Un pattern est n'importe quelle expression XPath utilisant uniquement les axes `child` et `@` ainsi que l'abréviation `//`. Les prédicats sont autorisés.

# Des exemples de pattern

Rappel : un pattern est interprété comme étant les nœuds auxquels s'applique le template. Exemples :

- `<xsl:template match="B">` s'applique à n'importe quel élément B.
- `<xsl:template match="A/B">` s'applique à n'importe quel élément B, enfant d'un élément A.
- `<xsl:template match="@att1">` s'applique à n'importe quel attribut nommé `att1`, peu importe qui est son parent.
- `<xsl:template match="A//@att1"` s'applique à n'importe quel attribut nommé `att1`, si son parent est un descendant d'un élément A.

Étant donné un arbre XML $T$, un pattern $P$ correspond à un nœud $N$ si il existe un nœud $C$ (le nœud du contexte actuel) dans $T$ tel que $N \in P(T, C)$.

# Le contenu du corps d'un template

Essentiellement, le corps d'un `<xsl:template>` est constitué de :

- Des littéraux éléments et texte. Exemple : `<h2>Auteurs</h2>`. Cela génère dans la sortie de la transformation un élément `h2`, avec un nœud enfant de type texte ayant pour valeur `Auteurs`.
- Des valeurs et des éléments provenant du document en entrée. Exemple : `<xsl:value-of select="title"/>`. Cela génère dans la sortie de la transformation le résultat de l'expression XPath `title`.
- Des appels à d'autres templates. Exemple `<xsl:apply-templates select="auteurs"/>`. Cela a pour effet d'appeler les templates pour lesquels le pattern correspondant à des nœuds `auteurs`. Les templates sont appelés pour chaque nœud trouvé par l'expression XPath `auteurs`.

***Remarque*** : il existe de nombreuses autres fonctionnalité en XSLT (boucles, tests, . . . ). Pour l'instant, nous nous concentrons sur la base.

# Instantiation de <xsl:template>

Les grands principes :

- Les éléments littéraux (ceux qui n'appartiennent pas à l'espace de nom XSLT) et les données brutes (texte) sont simplement copiés dans la sortie.
- Le nœud contexte. Un template est toujours instancié par rapport au nœud contexte provenant du document en entrée.
- Les expressions XPath. Toutes les expressions XPath relatives sont évaluées dans le template par rapport au nœud contexte actuel (à l'exception de `<xsl:for-each>`).
- Les appels à `<xsl:apply-templates>`. Ces appels trouvent et instancient un template correspondant pour chaque nœud trouvé par la requête XPath de l'attribut `select`.
- Les appels aux templates nommés. Ces appels sont remplacés par l'instanciation directe de ces templates.

# L'élément <xsl:apply-templates>

Exemple : `<xsl:apply-templates select="auteurs/nom"/>`

**select** est une expression XPath qui, si relative, est interprétée par rapport au nœud du contexte actuel. ***Remarque*** : la valeur par défaut de `select` est `child::node()`.

```
7  <xsl:template match="livre">
8      <ul><xsl:apply-templates select="auteurs/nom"/></ul>
9  </xsl:template>
10
11 <xsl:template match="nom">
12     <li><xsl:value-of select="."/></li>
13 </xsl:template>
```

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <livre>
4      <auteurs>
5          <nom>Serge</nom>
6          <nom>Ioana</nom>
7      </auteurs>
8  </livre>
```

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ul><li>Serge</li><li>Ioana</li></ul>
```

# Le modèle d'exécution d'XSLT

Une feuille de style XSLT est composée d'un ensemble de templates. La transformation d'un document entrée $I$, est exécutée comme suit :

- Le moteur de transformation considère la racine $R$ de $I$ et sélectionne le template qui s'applique à $R$.
- Le corps du template est copiée dans la sortie $O$.
- Ensuite, le moteur de transformation considère tous les appels à `<xsl:apply-templates>` qui ont été copiés dans $O$ et évalue les expression XPath des attributs `select` prenant comme nœud contexte la racine $R$.
- Pour chaque nœud resultant de l'expression XPath, un template correspondant est appelé et son exécution remplace l'appel `<xsl:apply-templates>` initialement inscrit.
- Le procédé se répète, étant donné que de nouveaux `<xsl:apply-templates>` ont pu être inséré par les actions précédentes.
- La transformation est terminée lorsque la sortie $O$ ne contient plus aucune instruction **xsl:**.

# Exemple de transformation

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <livres>
4      <titre> Web Data Model</titre>
5      <auteurs>
6          <nom>Serge</nom>
7          <nom>Ioana</nom>
8      </auteurs>
9      <contenu>
10         <chapitre id="1">
11             XML data model
12         </chapitre>
13         <chapitre id="2">
14             XPath
15         </chapitre>
16     </contenu>
17 </livres>
```

```
<xsl:output methode="html" encoding="utf-8"/>
<xsl:template match="/">
    <html>
        <head>
            <title>
                <xsl:value-of select="livres/titre"/>
            </title>
        </head>
        <body>

            <xsl:apply-templates select="livres"/>
        </body>
    </html>
</xsl:template>

<xsl:template match="livres">
    Titre : <xsl:value-of select="title"/>

    <h2>Liste des auteurs</h2>
    <ul>
        <xsl:apply-templates select="auteurs/nom"/>
    </ul>
</xsl:template>

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

```
1  <html>
2      <head>
3          <meta http-equiv="Content-Type" content="text/html; charset=utf
       -8">
4          <title> Web Data Model</title>
5      </head>
6      <body>
7          Titre : <h2>Liste des auteurs</h2>
8          <ul>
9              <li>Serge</li>
10             <li>Ioana</li>
11         </ul>
12     </body>
13 </html>
```

# Les éléments enfants de <xsl:stylesheet>

- **import, include** : importe les templates provenant d'autres programmes XSLT.
- **ouput** : détermine le format de sortie de la transformation. Par défaut, c'est le format XML qui est utilisé.
- **param** : définit ou importe des paramètres.
- **variable** : définit une variable.
- **template** : définit un template.

l'élément `<xsl:template>` est probablement le plus important pour nous dans ce cours. C'est dans le corps des templates que nous allons préciser notre transformation. Nous allons voir que le code d'une transformation peut être assez évolué et peut utilisé des éléments de programmation classique (for, if, . . . ).

# Les éléments import et include

Pour importer des templates provenant d'autres fichiers sources XSLT :

```
<xsl:import href="lib_templates.xsl"/>
```

Les templates sont alors importés depuis la feuille de style externe (ici `lib_templates.xsl`). L'élément `<xsl:import>` doit être placé en ***premier*** dans la feuille de style de la transformation.

L'élément `<xsl:include>` fonctionne de la même manière :

```
<xsl:include href="lib_templates.xsl"/>
```

La différence entre import et include se fait au niveau des règles de priorité des templates importés. Include considère les templates importés comme des templates appartenant à la feuille de style actuelle, donc avec la même priorité. Import associe une priorité inférieure aux templates importés par rapport aux templates locaux.

Si plus d'un seul template correspond à la même expression XPath (par exemple provenant d'un `select`) seul le template avec la priorité la plus élevé sera exécuté.

# L'élément output

L'élément output permet de spécifier la format de sortie de la transformation. Il permet également de donner des détails vis à vis du format de sortie, par exemple si la sortie doit être indentée, l'encodage, . . . :

```
<xsl:output method="html" encoding="iso-8859-1" doctype-public="-//W3C//
DTD HTML 4.01//EN" doctype-system="http://www.w3.org/TR/html4/strict.dtd
" indent="yes"/>
```

- **method** : peut être soit ***xml*** (valeur par défaut), soit ***html***, soit ***text***.
- **encoding** : l'encodage souhaité pour la sortie.
- **doctype-public** et **doctype-system** : permet l'ajout de la déclaration d'un DTD dans la sortie.
- **indent** : spécifie si la sortie doit être indentée ou non (par défaut, ***no***).

***Remarque*** : la format de sortie ***text*** donne la liberté de générer de ce que l'on veut. Cela pourrait être du php, javascript, du texte, etc . . .

# La gestion de conflit entre templates

- Les templates importés ont une priorité plus faible que les templates locaux.
- C'est le template avec la plus haute priorité qui prévaut. Il est possible de fixer une priorité avec l'attribut `priority` associé à l'élément <`xsl:template`>. Si aucune priorité n'est spécifiée, une priorité par défaut est donné au template. Cette priorité par défaut dépend de la précision de la requête XPath (attribut `match`), plus la requête est précise plus la priorité est haute.
- Si il y a encore des conflits après cela, cela génère une erreur.
- Si aucun template ne correspond à la requête XPath (provenant par exemple d'un `select`) c'est un template ***par défaut*** (invisible pour le programmeur) qui est appelé

Les templates par défaut possède un code source caché. Il existe un template par défaut pour les nœuds éléments et un template par défaut pour les nœuds attributs et texte.

# Les templates par défaut

Le template par défaut s'appliquant aux nœuds éléments est le suivant :

```
7  <xsl:template match="*|/">
8      <xsl:apply-templates select="node()"/>
9  </xsl:template>
```

Le template par défaut s'appliquant aux nœuds attributs et textes est le suivant :

```
11 <xsl:template match="@*|text()">
12     <xsl:value-of select="."/>
13 </xsl:template>
```

Ce qu'il faut retenir c'est que la valeur du nœud contexte actuel est copiée dans le document de sortie.

***Remarque*** : si en TP vous avez des données supplémentaires qui s'affichent à la fin de votre sortie générée par la transformation, c'est que vous avez probablement un `select` qui ne correspond à aucun `match` dans le code source de votre transformation.

# Variables globales et paramètres

Déclaration d'un paramètre globale :

```
<xsl:param name="nom" select="'John Doe'"/>
```

Déclaration d'une variable globale :

```
<xsl:variable name="pi" select="3.14159"/>
```

Les paramètres (globaux) permet de définir la valeur par défaut d'un paramètre. Nous verrons cela plus en détails avec les templates ***nommés***.

Les variables (globales) permettent de définir une constante. Les variables ne sont pas modifiables après leur déclaration.

# Les templates nommés

```
3  <livres>
4      Texte 1
5      <livre>
6          <auteurs>
7              <nom>Serge</nom>
8              <nom>Ioana</nom>
9          </auteurs>
10     </livre>
11     Texte 2
12     <livre>
13         <auteurs>
14             <nom>Jean</nom>
15             <nom>Dupont</nom>
16         </auteurs>
17     </livre>
18     Texte 3
19 </livres>
```

```
 7  <xsl:template name="afficher">
 8      <xsl:value-of select="position()"/> : <xsl:value-of select="."/>
 9  </xsl:template>
10
11  <xsl:template match="/livres/*">
12      <xsl:call-template name="afficher"/>
13  </xsl:template>
14
15  <xsl:template match="text()">
16      <xsl:call-template name="afficher"/>
17  </xsl:template>
```

```
1 :  Texte 1
2 :  Serge Ioana
3 :  Texte 2
4 :  Jean Dupont
5 :  Texte 3
```

Les templates nommés ont le même rôle que les fonctions dans un langage de programmation. Un template nommé est déclaré avec l'attribut `name` (et non pas `match`). Il est ensuite appelé avec `<xsl:call-template>` où l'attribut `name` est le nom du template que l'on souhaite invoquer.

***Remarque*** : l'appel à un template nommé ne change pas le nœud contexte.

# Les paramètres

```
19 <xsl:template name="afficher-message">
20     <xsl:param name="message" select="'vide'"/>
21     <xsl:value-of select="position()"/> : <xsl:value-of select="$message
       "/>
22 </xsl:template>
23
24 <xsl:template match="*">
25     <xsl:call-template name="afficher-message">
26         <xsl:with-param name="message" select="'Nœud élément'"/>
27     </xsl:call-template>
28 </xsl:template>
```

Le mécanisme est similaire à `<xsl:apply-templates>` :

- **param** : détermine le paramètre reçu par le template.
- **with-param** : définit la valeur du paramètre à passer au template.

***Remarque*** : on accède à la valeur d'un paramètre ou d'une variable avec le symbole `$`.

# Les instructions conditionnelles : <xsl:if>

```
7  <xsl:template match="film">
8      <xsl:if test="annee > 1970">
9          <xsl:copy-of select="."/>
10     </xsl:if>
11 </xsl:template>
```

L'élément <xsl:copy-of> effectue une copie de tout le sous-arbre alors que l'élément <xsl:copy> effectue une copie le nœud sans ses descendants.

Dans ce template, si l'année du film est strictement supérieure à 1970, alors on copie tout le sous-arbre (à partir du nœud contexte actuel, c'est à dire la balise `film`).

En XSLT, il n'y a pas *directement* de condition ***else***. Pour effectuer un ***if-elseif-else***, il faut utiliser l'élément `xsl:choose`, qui permet également de faire un ***switch***.

# Les instructions conditionnelles : <xsl:choose>

```
13  <xsl:choose>
14      <xsl:when test="annee mod 4">non</xsl:when>
15      <xsl:when test="annee mod 100">oui</xsl:when>
16      <xsl:when test="annee mod 400">peut-être</xsl:when>
17      <xsl:otherwise>par défaut</xsl:otherwise>
18  </xsl:choose>
19
20  <xsl:value-of select="count(a)"/> item<xsl:if test="count(a) > 1">s</xsl:
        if>
```

L'instruction `xsl:choose` permet d'effectuer un branchement conditionnel dans l'exécution de la transformation. Le premier <xsl:when> évalué à vrai est exécuté (les autres sont ignorés). Si aucun <xsl:when> n'est vrai, c'est <xsl:otherwise> qui est alors exécuté.

# Les boucles

L'élément <xsl:for-each> permet d'effectuer une boucle sur un ensemble de nœuds. Son utilisation peut être vue comme une alternative à <xsl:apply-templates>.

- L'ensemble de nœuds sur lequel l'itération va être réalisée est obtenu par une expression XPath (attribut `select`).
- Chaque nœud de l'ensemble devient à son tour le nœud contexte de la boucle for (qui remplace alors temporairement le nœud contexte du template).
- Le corps de <xsl:for-each> est instancié pour nœud de l'ensemble.

L'utilisation de <xsl:for-each> ne requiert pas l'appel à un template pour itérer sur un ensemble de nœud. Son utilisation est donc un peu plus simple. Cependant, la boucle est directement intégrée au corps du template, ce qui *découpe* un peu moins le code. L'idéal est une bonne combinaison des deux.

```
 7  <xsl:template match="personne">
 8      <xsl:for-each select="enfant">
 9          <xsl:sort select="@age" order="ascending" date-type="number"/>
10
11          <xsl:value-of select="nom"/> : <xsl:value-of select="@age"/>
12      </xsl:for-each>
13  </xsl:template>
```

Cette boucle for avec itérer sur tous les nœud (sous-éléments de `personne`) nommés `enfant`. Cette itération va se faire par ordre croissant d'âge. À chaque tour de boucle, on va afficher le nom et l'âge de la personne.

L'élément `<xsl:sort>` peut être aussi utilisé comme enfant directe de l'élément `<xsl:apply-templates>`.

# Les variables

Une variable est une paire (nom, valeur). Cela peut être défini par :

- le résultat d'une expression XPath. Par exemple
  `<xsl:variable name="pi" select="3.141592"/>` ou
  `<xsl:variable name="a" select="//enfant"/>`;
- ou le contenu de la balise `<xsl:variable>`.
  `<xsl:variable name="b">Texte1</xsl:variable>`

***Remarque*** : une variable possède une portée (ses frères/sœurs, et ses descendants) et ne peut pas être défini dans le bloc de code à sa portée.

# Autres fonctionnalités d'XSLT

Il existe d'autres fonctionnalités d'XSLT :

- Des éléments permettant de contrôler le texte en sortie : `<xsl:text>`, `<xsl:strip-space>`, `<xsl:preserve-space>` et `<xsl:normalize-space>`.
- Création d'éléments et attributs : `<xsl:element>` et `<xsl:attribute>`

Exemple de création d'éléments et d'attributs :

```
1 <xsl:element name="{concat('p',@age)}">
2     <xsl:attribute name="prenom">
3         <xsl:value-of select="nom"/>
4     </xsl:attribute>
5 </xsl:element>
```

# Quizz

Écrivez une transformation XSLT calculant le nième terme de la suite de Fibonacci.

$$F_0 = 0, F_1 = 1$$

et

$$F_n = F_{n-1} + F_{n-2}$$

pour $n > 1$.

```
 5  <xsl:template name="Fibon">
 6      <xsl:param name="n1" select="0"/>
 7      <xsl:param name="n2" select="1"/>
 8      <xsl:param name="num"/>
 9
10      <xsl:value-of select="$n1"/>
11
12      <xsl:if test="$num &gt; 0">
13          <xsl:text>, </xsl:text>
14          <xsl:call-template name="Fibon">
15              <xsl:with-param name="n1" select="$n2"/>
16              <xsl:with-param name="n2" select="$n1 + $n2"/>
17              <xsl:with-param name="num" select="$num - 1"/>
18          </xsl:call-template>
19      </xsl:if>
20  </xsl:template>
21
22  <xsl:template match="/">
23      <result>
24          <xsl:call-template name="Fibon">
25              <xsl:with-param name="num" select="8"/>
26          </xsl:call-template>
27      </result>
28  </xsl:template>
```

# Quizz

Concevoir une feuille XSLT transformant un document de type actu vers XTML.
En particulier, ce traitement devra :

- regrouper les nouvelles par thème ;
- indiquer combien de brèves sont rédigées en français, combien en anglais ;
- insérer l'éventuelle photo associée à chaque brève ;
- transformer les urls disponibles en liens hypertextes, la phrase associée devenant le texte actif du lien.

```
1  <?xml version="1.0" ?>
2  <actu >
3      <breve theme="actu" langue="fr" date="7 décembre 2004">
4          <titre>Les députés renoncent au CV anonyme</titre>
5          <texte>
6              Les députés ont achevé lundi, après plus de deux semaines de
       débat marathon, l'examen du projet de loi de cohésion sociale. Ils
       ont abandonné le CV anonyme, mesure à laquelle le gouvernement et
      l'UMP étaient hostiles en l'état.
7          </texte>
8          <url href="http://www.reuters.fr/locales/c_newsArticle.jsp?type=
      topNews&localeKey=fr_FR&storyID=7007458">Reuters</url>
9          <url href="http://news.tf1.fr/news/economie/0,,3189581,00.html">
      TF1</url>
10     </breve>
11     <breve theme="actu" langue="fr" date="7 février 2006">
12         <titre>Mobilisation contre le CPE</titre>
13         <texte>
14             Manifestations en ce moment dans toutes les grandes villes
```

Correction : voir `actu.xsl` (sur Moodle).