Université de Nice-Sophia Antipolis — Jeudi 7 juin 2007
Polytech' CiP1 — Durée: 3h

**Création et Manipulation de documents**

**(Hélène Renard / Sylvain Schmitz)**

**Travaux Dirigés – Séance n$^o$13**

---

# 1 Objectifs du TD

L'objectif de cette séance est de vous familiariser avec XSLT et les transformations d'un format XML vers un autre.

**Mise en place** Copiez le répertoire `~schmitz/xslt` dans `CMDocs` et placez-vous dedans.

# 2 Transformations XSLT

Nous avions vu que les outils de manipulation de chaînes de caractères tels que `sed` et `awk` n'étaient pas très simples d'utilisation pour manipuler des documents XML. En pratique, l'adoption d'XML pour les formats d'échanges de données a crée un besoin pour des outils spécialisés dans son traitement. Les feuilles de style XSLT (`http://www.w3.org/TR/xslt`) comblent en bonne partie ce manque.

## 2.1 Exemple

On souhaite opérer à la traduction du corps d'un document XHTML vers le corps d'un document *OpenDocument* et inversement comme suit :

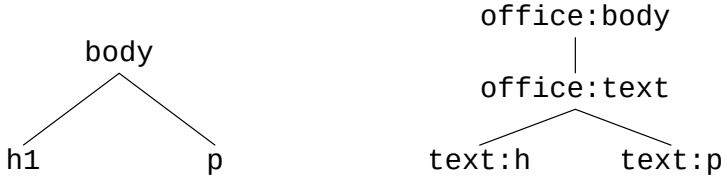
```
<body>
  <h1>Titre</h1>
  <p>...</p>
</body>
```

```
<office:body>
  <office:text>
    <text:h text:outline-level="1">Titre</text:h>
    <text:p>...</text:p>
  </office:text>
</office:body>
```

**Exercice n°1 :** Écrivez la traduction de XHTML vers *OpenDocument* du document `exo1.xhtml` à l'aide de `sed`. La traduction dans le sens inverse vous paraît-elle aisée ?

## 2.2 Principe de XSLT

Au lieu de procéder à des remplacements du *texte*, une transformation XSLT opère à des remplacements dans l'*arbre* XML :



Une feuille XSLT est principalement constituée de *templates*, qui décrivent comment traiter chaque nœud de l'arbre XML du document à transformer. La transformation ci-dessus s'écrit par exemple à l'aide de trois templates, un pour le nœud `body`, un pour le nœud `h1`, et un pour le nœud `p` :

```
<xsl:template match="body">
  <office:body>
    <office:text>
      <xsl:apply-templates/>
    </office:text>
  </office:body>
</xsl:template>

<xsl:template match="h1">
  <text:h text:outline-level="1">
    <xsl:apply-templates/>
  </text:h>
</xsl:template>

<xsl:template match="p">
  <text:p>
    <xsl:apply-templates/>
  </text:p>
</xsl:template>
```

Dans les trois cas, chaque fois que l'on rencontre un nœud `body`, `h1` ou `p`, on le transforme et on applique la transformation *récursivement* avec `<xsl:apply-templates/>` : si l'on avait écrit

```
<xsl:template match="body">
  <office:body>
    <office:text>
    </office:text>
  </office:body>
</xsl:template>
```

alors le document de sortie ne serait que le fragment XML

```
<office:body>
  <office:text>
  </office:text>
</office:body>
```

car les nœuds `h1` et `p` qui sont dans `body` n'auraient pas été visités.

## 2.3 Une feuille XSLT

Une feuille de style XSLT vraiment minimale ne comprend que

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

Cette feuille contient les templates implicites

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
```

```
</xsl:template>

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

qui sont appliqués à n'importe quel nœud de l'arbre XML et se contentent de s'appliquer récursivement et de copier le texte sur la sortie.

## 2.4 Appliquer une feuille XSLT

En ligne de commande, la commande `xsltproc` permet d'appliquer une feuille XSLT `feuille.xsl` à des fichiers XML :

`xsltproc feuille.xsl` *fichiers XML*

**Exercice n°2 :** Appliquez la feuille de style `minimal.xsl` aux fichiers `exo1.xhtml` et `exo2.xhtml`. Le traitement est plus long dans le deuxième cas car `exo2.xhtml` déclare proprement sa DTD, et `xsltproc` vérifie alors que `exo2.xhtml` est valide. Comment éviter cette validation ? Comment écrire le résultat dans le fichier `out.xml` ?

### Espaces de nom

**Exercice n°3 :** Le fichier `exo3.xsl` contient seulement les trois templates vus précédemment. Appliquez-le aux fichiers `exo1.xhtml` et `exo2.xhtml`.

La feuille `exo3.xsl` ne fonctionne pas sur `exo2.xhtml` car les éléments d'`exo2.xhtml` sont tous implicitement dans l'espace de nom `http://www.w3.org/1999/xhtml`. Pour correspondre à un élément du document `exo2.xhtml`, il faut donc préciser l'espace de nom utilisé en ajoutant la déclaration

```
xmlns:xhtml="http://www.w3.org/1999/xhtml"
```

à l'élément racine `xsl:stylesheet` et écrire des templates de la forme :

```
<xsl:template match="xhtml:p">
  <text:p>
    <xsl:apply-templates/>
  </text:p>
</xsl:template>
```

Le fichier `arbre.xsl` permet de traiter `exo2.xhtml`.

**Sélectionner des nœuds** Le fichier `arbre.xsl` contient en particulier un template de la forme

```
<xsl:template match="xhtml:html">
  <office:document-content office:version="1.0">
    <xsl:apply-templates select="xhtml:body"/>
  </office:document-content>
</xsl:template>
```

où l'on sélectionne explicitement le nœud `body` pour l'application récursive des templates par `xsl:apply-templates`.

**Exercice n°4 :** Comment faire pour ignorer le contenu de l'élément `head` du document `exo2.xhtml` sans sélectionner explicitement `body` ?

# 3 Aperçu d'XPath

Les expressions qui permettent de sélectionner à quel(s) nœuds s'applique un `xsl:template` ou un appel récursif `xsl:apply-templates` sont des expressions XPath (`http://www.w3.org/TR/xpath`). Ce standard est assez riche, mais pour une utilistion courante, les quelques éléments qui suivent suffisent.

## 3.1 Chemins

Les chemins XPath fonctionnent un peu comme l'adressage dans l'arborescence des fichiers, avec « `.` » pour désigner l'élément courant, « `/` » pour trouver les éléments fils, et « `..` » pour remonter au père ; le chemin « `a//b` » trouve tous les éléments `b` descendants de l'élément `a`. Par exemple, les éléments du fichier `exo1.xhtml` sont identifiés par les chemins `/body/h1` et `/body/p` depuis la racine « `/` ».

À l'intérieur d'un template, le chemin part depuis le nœud courant. Par exemple, dans

```
<xsl:template match="xhtml:body">
  <office:body>
    <office:text>
      <xsl:apply-templates select="../xhtml:head/xhtml:title"/>
      <xsl:apply-templates/>
    </office:text>
  </office:body>
</xsl:template>

<xsl:template match="xhtml:title">
  <text:h>
    <xsl:apply-templates/>
  </text:h>
</xsl:template>
```

on applique récursivement les templates pour le titre du document XHTML, que l'on place dans un élément `text:h` au début du document *OpenDocument* obtenu.

## 3.2 Axes

Les chemins que nous venons de voir sont en fait des notations abrégées pour les différents *axes* de recherche : « `..` » correspond à l'axe `parent::`, « `//` » à l'axe `descendant::`. Des généralisations de ces axes existent, comme `descendant-or-self::`, `ancestor::`, `ancestor-or-self::`, et ainsi de suite.

Enfin, on peut tester l'existence d'un frère avant ou après l'élément atteint par `preceding-sibling::` et `following-sibling::` respectivement. Ainsi, le chemin `preceding-sibling::a/b` trouve un élément `b` fils d'un élément `a` qui précède immédiatement le nœud courant.

## 3.3 Prédicats

Chaque étape d'un chemin `a/b/c/d` peut recevoir un prédicat entre crochets qui doit être satisfait pour que l'étape soit validée. Par exemple

```
a[position()=last()-1]/b[../e]/c[@href]/d[@id='foo']
```

vérifiera

- que le nœd `a` est l'avant-dernier élément `a` sous l'élément courant,
- qu'il a un fils `b` avec un élément `e` pour frère,
- que ce `b` a pour fils un nœud `c` avec un attribut `href`, et
- que ce nœud `c` a un fils `d` avec un attribut `id` avec la valeur `foo`.

### 3.4 Choix multiples

L'étoile « * » a le sens habituel puisqu'elle accepte n'importe quel élément. On peut l'utiliser par exemple pour trouver tous les éléments `a` qui suivent l'élément courant ou qui sont plus bas par `following-sibling::*/descendant-or-self::a`

De même, « `@*` » acceptera n'importe quel attribut. Enfin, deux chemins différents peuvent être regroupés par un signe « `|` », comme dans `a/b/c|d/b/e`.

### 3.5 Exercices

**Exercice n°5 :** Ajoutez un nouveau template pour le premier élément `h1` d'un document XHTML à `arbre.xsl`. Ce template va de plus copier le titre du document XHTML, de telle sorte que l'on obtienne le titre

```
<text:h text:outline-level="1">Exercice 2 : Titre 1</text:h>
```

à partir d'`exo2.xhtml`. Pour écrire « : », vous aurez besoin d'écrire du texte sous XSLT, ce qui se fait par un élément `xsl:text` contenant le texte à insérer, donc ici

```
<xsl:text> : </xsl:text>
```

Donnez deux expressions XPATH différentes pour faire cet exercice.

**Exercice n°6 :** Complétez l'exercice précédent en affichant aussi le nom de l'auteur du document du fichier `exo6.xhtml` en titre de niveau 2 du document *OpenDocument* obtenu :

```
<text:h text:outline-level="1">Exercice 6 : Titre 1</text:h>
<text:h text:outline-level="2">John Doe</text:h>
```

L'élément XSLT qui permet de copier une valeur est `xsl:value-of`, qui écrit son attribut `select` (revoyez son utilisation en section 2.3).

## 4 Aperçu d'XSLT

Le standard XSLT est lui aussi très riche, et est encore étendu par le standard EXSLT qui lui ajoute d'autres fonctionnalités. Nous ne verrons qu'un aperçu des possibilités offertes par ce langage de transformations, en sus des éléments `xsl:template`, `xsl:apply-templates`, `xsl:text` et `xsl:value-of` déjà entrevus.

### 4.1 Éléments et attributs

À la place d'écrire

```
<xsl:template match="xhtml:h1">
  <text:h text:outline-position="1">
    <xsl:apply-templates/>
  </text:h>
</xsl:template>
```

on pourrait écrire

```
<xsl:template match="xhtml:h1">
  <xsl:element name="text:h">
    <xsl:attribute name="text:outline-position">1</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

L'intérêt est de pouvoir manipuler plus facilement les noms de éléments et les valeurs des attributs. Par exemple, on peut alors traduire les titres XHTML en *OpenDocument* avec un seul template comme le suivant :

```
<xsl:template match="xhtml:h1|xhtml:h2|xhtml:h3|xhtml:h4|xhtml:h5|xhtml:h6">
  <text:h>
    <xsl:attribute name="text:outline-position">
      <xsl:value-of select="substring(name(.),2)"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </text:h>
</xsl:template>
```

Inversement, on peut traduire les titres *OpenDocument* en XHTML par le template

```
<xsl:template match="text:h[@text:outline-level]">
  <xsl:element name="xhtml:h{@text:outline-level}">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

## 4.2 Énoncés conditionnels

À l'intérieur d'un template, on peut choisir d'appliquer une transformation uniquement sous certaines conditions. Dans un élément `xsl:choose`, l'élément `xsl:when` contient un attribut `test` et n'applique son contenu que si le test est vrai. Le test correspond usuellement à un prédicat XPATH, mais peut utiliser des connecteurs logiques `not()`, `and` ou `or`. Par exemple, on peut mettre le paragraphe qui suit immédiatement un titre `h1` dans un style particulier par

```
<xsl:template match="xhtml:p">
  <text:p>
    <xsl:choose>
      <xsl:when test="preceding-sibling::xhtml:h1">
        <xsl:attribute name="text:style-name">Pchapeau</xsl:attribute>
      </xsl:when>
    </xsl:choose>
    <xsl:apply-templates/>
  </text:p>
</xsl:template>
```

Le test peut être complété pour tenir compte du cas où il n'y a pas d'élément `h1` dans le document, et dans ce cas considérer comme chapeau le premier paragraphe avec le test

```
(not(//xhtml:h1) and position()=1) or preceding-sibling::xhtml:h1
```

Si plusieurs possibilités sont possibles, on peut ajouter d'autres choix avec d'autres éléments `xsl:when`, et finir sur un `xsl:otherwise` pour les cas non traités. Quand un seul cas nous intéresse, comme dans l'exemple plus haut, il est plus simple d'utiliser l'élément `xsl:if`

```
<xsl:template match="xhtml:p">
  <text:p>
    <xsl:if test="preceding-sibling::xhtml:h1">
      <xsl:attribute name="text:style-name">Pchapeau</xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </text:p>
</xsl:template>
```

**Exercice n°7 :** Le fichier `exo7.xhtml` ne contient pas d'élément `h1`, et notre feuille XSLT n'affiche plus le titre et le nom de l'auteur. Modifiez `arbre.xsl` pour remédier à ce problème.

# 5 Mise en pratique

Le document `exo8.xhtml` contient une série de titre de niveaux 1 à 3. Notez que les titres sont proprement ordonnés, sans saut d'un titre de niveau 1 à un de niveau 3. De plus, une séquence de titres de niveau inférieur est toujours mise dans un élément `div`.

**Exercice n°8 :** On souhaite générer un document XHTML qui donne la table des matières d'`exo8.xhtml`. Le résultat souhaité est donné dans le fichier `out8.xhtml`. Complétez `exo8.xsl` pour générer la table des matières désirée.