XSLT

Langage de transformation d'arbre

Yves Bekkers

Mise à jour : 24 mars 2010

Plan

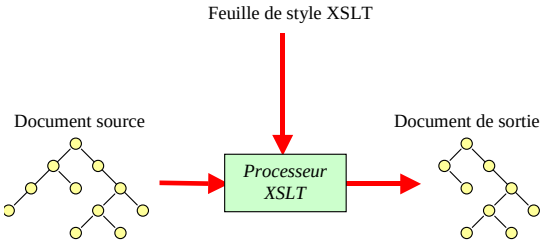
Transformation de documents

- XSL (e***X****tensible* ***S****tylesheet* ***L****anguage*)
- Deux normes indépendantes
  - XSLT : langage de transformation
    - XPath : langage pour adresser les nœuds d'un arbre
  - XSL-FO : langage de formatage
    - Permet de spécifier un formatage plus fin que celui que l'on obtient à l'aide de HTML+CSS

XSLT le langage de transformation

- Un langage déclaratif (Turing complet !)
  - avec une syntaxe XML !
- Les programmes XSLT s'appellent des *feuilles de styles*
  - Mais c'est beaucoup plus puissant que CSS
  - Exprime une transformation d'arbre en un autre arbre
- Modèle de calcul
  - Utilise une technique de *filtrage* à base de *motifs* (patterns) *et de modèles* (template) décrits dans des *règles* (template rules) pour transformer des arbres

XSLT = Transformation d'arbre



XSLT premier exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
     "http://www.w3.org/1999/XSL/Transform">
   <xsl:output method= "html" version="1.0"
     encoding="UTF-8"/>
    <xsl:template match="/">
        <html>
            <HEAD>
                <TITLE>Bonjour</TITLE>
            </HEAD>
            <BODY>
               <h1>Bonjour !</h1>
            </BODY>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

XSLT est un langage XML

- Les instructions sont des éléments XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
     "http://www.w3.org/1999/XSL/Transform">
   <!-- Format de sortie -->
   <xsl:output method="xml" version="1.0"
     encoding="UTF-8" indent="yes"/>

   <!-- ... règles XSLT ... -->

</xsl:stylesheet>
```


XSLT un espace de noms

- Espace de nom XSLT
  - `http://www.w3.org/1999/XSL/Transform`
  - Préfixe recommandé `xsl:`

Prélude d'une feuille de style

Elément `<xsl:stylesheet>`

- Élement 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

Elément `<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

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 (é ...)
    - (*sortie par défaut si XSL reconnaît l'arbre de sortie HTML4*)
  - text : tout autre format textuel :
    - du code Java, format Microsoft RTF, LaTex

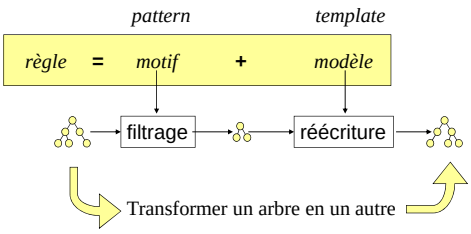
Parcours/transformation d'arbre

- Règle de réécriture : *template rules*
  - `<xsl:template>`
- Spécifier un parcours de l'arbre d'entrée
  - `<xsl:apply-templates>`
  - `<xsl:for-each>`
- Obtenir une valeur dans l'arbre source
  - `<xsl:value-of>`
  - les crochets dans un attribut

```
<a href="{@src}">
```


Règles de réécriture

template rules



Élément `<xsl:template>`

- Règle de réécriture *motif* + *modèle*

```
<xsl:template match="motif">
   … modèle …
</xsl:template>
```

  - Attribut `match` : expression XPATH
    - Un motif pour filtrer l'arbre d'entrée
  - Contenu de l'élément `<xsl:template>` :
    - Un modèle de sous-arbre en sortie
- Un programme XSLT est un ensemble de règles

Premier exemple complet

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

   <xsl:output method="html"/>

   <xsl:template match="/">
      <HTML>
         <HEAD>
            <TITLE>Welcome</TITLE>
         </HEAD>
         <BODY>
            Welcome!
         </BODY>
      </HTML>
   </xsl:template>
</xsl:stylesheet>
```

motif
« *filtre la racine du document d'entrée* »

règle

modèle
« *Document html*»

Contenu de l'élément `<xsl:template>`

- Un élément `<xsl:template>` contient
  - Le modèle de texte HTML (ou XML ou texte simple)
    - *le texte XML doit être bien formé*
  - Des instructions XSLT (mélangées au modèle de sortie)
    - *pour générer un texte en relation avec le contenu du document source*
    - *pour extraire des informations du document source*

Expressions XPath

```
<xsl:template match="titre">
      <h1><xsl:value-of select="."/></h1>
</xsl:template>
```

Extraction du contenu de l'arbre en entrée

Second exemple - le carnet d'adresse

En entrée

```
<carnetDAdresse>
  <carteDeVisite>
    <nom>Bekkers</nom>
    ...
  </carteDeVisite >
  <carteDeVisite>
    <nom> Bartold </nom>
    ...
  </carteDeVisite >
  ...
</ carnetDAdresse >
```

En sortie

```
<html>
 <body>
 <h1>Liste des Noms</h1>
       <p>Nom : Bekkers</p>
       <p>Nom : Bartold</p>
       <p>Nom : Letertre</p>
       <p>Nom : Apolon</p>
 </body>
</html>
```

Second exemple (1)

```
<xsl:stylesheet version="1.0"
   xmlns:xsl=
      "http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" indent="no"
      encoding="iso-8859-1"/>

   <xsl:template match="/">
     <html>
       <xsl:apply-templates select="child::*"/>
     </html>
   </xsl:template>
```

Expression xpath

Modèle de sous-arbre

...

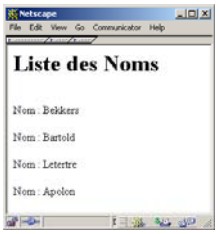
Second exemple (2)

```
    <xsl:template match="carnetDAdresse">
      <body>
        <h1>Liste des Noms</h1>
        <xsl:apply-templates select="child::*"/>
      </body>
    </xsl:template>

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

</xsl:stylesheet>
```


Résultat

- Pour un document source contenant 4 cartes de visite

```
<html>
 <body>
 <h1>Liste des Noms</h1>
       <p>Nom : Bekkers</p>
       <p>Nom : Bartold</p>
       <p>Nom : Letertre</p>
       <p>Nom : Apolon</p>
 </body>
</html>
```



Changement de contexte - Élément `<xsl:apply-templates>`

- Descente dans les fils d'un nœud

```
<xsl:template match="carnetDAdresse">
  <body>
    <h1>Liste des Noms</h1>
    <xsl:apply-templates
                  select="child::node()"/>
  </body>
</xsl:template>
```

Expression xpath

- Raccourci d'écriture
  - descente par défaut aux nœuds fils

```
<xsl:apply-templates/>
```


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`

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

- Le nœud selectionné 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 -->)

Exemple 1

- Arbre en entrée

```
<carteDeVisite>
  <nom>Bekkers</nom>
</carteDeVisite>
```

- Règle

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

- Arbre en sortie

```
<p>Nom : Bekkers</p>
```


Exemple 2

- Arbre en entrée

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

- Régle

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

- En sortie

```
enseigne XML au SEP
```


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

Exemple 4

- Arbre en entrée

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

- Règle

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

- En sortie

```
note
```


Exemple 5

- Arbre en entrée

```
4 cartes de visite : Bekkers, Bartold, Letertre, Apolon
```

- Règle

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

- En sortie

```
Bekkers
```

Seul le premier élément sélectionné est produit

Exemple 6

- Arbre en entrée

```
4 cartes de visite : Bekkers, Bartold, Letertre, Apolon
```

- Régle

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

- En sortie

```
BekkersBartoldLetertreApolon
```

Pour chaque carte de visite le template est appliqué

Règles par défaut

Règles par défaut (1)

Traverser la racine et tous les noeuds « élément »

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

Sortir les feuilles « texte » et les « attributs »

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


Règles par défaut (2)

- Commentaires et instructions de traitement

```
<xsl:template match="processing-
  instruction()|comment()"/>
```

- Ne rien faire

Feuille de style minimum

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

    <xsl:output method="text"/>

</xsl:stylesheet>
```

- Traverse tout l'arbre et sort les feuilles (contenu d'élément texte et valeur d'attribut)

Génération de contenu

Résultat littéral ou non ?

Méthodes de génération de contenu

- Deux méthodes de génération de contenu :
  1) Résultat litéral

```
<xsl:template match="subtitle">
  <h2><xsl:apply-templates/></h2>
</xsl:template>
```

  2) Résultat non litéral

(validation possible de la feuille de style)

```
<xsl:template match="subtitle">
  <xsl:element name="h2">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

Valeur d'attribut par résultat littéral évalué

Évaluation d'expression xpath en accolades dans les valeurs d'attribut

- Arbre en entrée

```
<a href="fic.txt"/>
```

- Template

```
<xsl:template match="a">
   <b id="{@href}"/>
</xsl:template>
```

- En sortie

```
<b id="fic.txt"/>
```


Résultat non littéral <xsl:attribute>

- Arbre en entrée

```
<a href="fic.txt"/>
```

- Template

```
<xsl:template match="a">
   <b><xsl:attribute name="id">
         <xsl:value-of select="@href"/>
      </xsl:attribute></b>
</xsl:template>
```

- En sortie

```
<b id="fic.txt"/>
```


Parcours itératifs

Élement <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>
```


Deux styles de programmation

- Récursive

```
<xsl:apply-templates>
```

- Itérative

```
<xsl:for-each>
```

- Attribut `select` donne l'ensemble de nœuds vers lequel on se déplace

Élément <xsl:comment>

- Sortir les commentaires à l'identique

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

Élément <xsl:processing-instruction>

- Sortir les instructions de traîtement à l'identique

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


Conflits de Règles

- Règle implicite de priorité
  - La règle la plus sélective gagne
  - Parmi 2 templates de même sélectivité, le dernier dans la feuille de style gagne
- Exemple
  - `nom` est plus sélectif que `/|*`
  - `note[clé]` est plus sélectif que `note`
  - `ville[@codepostal='35000']` est plus sélectif que `ville[@codepostal]`

Les modes

- Permet de déclarer plusieurs règles pour un même élément
- Chaque règle traite l'élément différemment

```
<xsl:template match="h1" mode="normal">

<xsl:template match="h1" mode="table-index">
```


Attributs mode

- Dans un élément `apply-templates`

```
<xsl:apply-templates mode="passe1"/>
```

- Dans un élément `template`

```
<xsl:template match="carteDeVisite"
   mode="passe1">
   ...
</xsl:template>
```

- Attention un `apply-templates` n'hérite pas du mode du `template` englobant

Autres outils

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
- En sortie

```
Bekkers, Bartold, Letertre, Apolon
```

Élément <xsl:choose>

- Conditionnelle à choix multiple

```
<xsl:choose>
   <xsl:when test="start-with('35',@codep)">
      <!-- cas 1 -->
   </xsl:when>
   <xsl:when test="start-with('44',@codep)">
      <!-- cas 2 -->
   </xsl:when>
   <xsl:otherwise>
      <!-- autres cas -->
   </xsl:otherwise>
</xsl:choose>
```


<xsl:variable>

- Déclaration de variable 1

```
<xsl:variable name="blackcolor"
  select="'#FFFFCC'" />
```

- Déclaration de variable 2

```
<xsl:variable
  name="blackcolor">#FFFFCC</xsl:variable>
```

- Référence à une variable

```
<BODY BGCOLOR='{$blackcolor}'>
```


<xsl:variable>

- XSL est un langage à assignation unique
- Les « variables » sont des constantes à la manière des constantes `#define` de C
- Une variable ne peut être réaffectée
- La visibilité d'une variable est son élément père
- Une variable peut en cacher une autre

Initialisation conditionnelle

- Exemple

Java

```
if (niveau > 20)
   code = 3;
else
   code = 5;
```

XSLT

```
<xsl:variable name="code">
  <xsl:choose>
    <xsl:when test="$niveau &gt; 20">
      <xsl:text>3</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>5</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```


Les espaces

- Les espaces non significatifs dans l'arbre xsl ne sont pas produits

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

et

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

ont le même effet

Élément <xsl:text>

- Caractères espaces, TAB, CR, LF en sortie

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

Attribut `disable-output-escaping`

- Pour sortir des caractères spéciaux tels quel (sans être sous forme d'entité)
- Valeurs possible : `yes` ou `no` (par défaut)
- où
  - Dans un élément `xsl:text`
  - Dans un élément `xsl:value-of`
- Attention : cela peut produire des documents qui ne sont pas bien formés
- Utiles pour produire des pages ASP ou JSP

Générer une section CDATA - 1

- Vous voulez générer ceci

```
<elem><![CDATA[bla<bla]]></elem>
```

- Solution 1

```
<elem>
<xsl:text disable-output-escaping="yes">
<![CDATA[<![CDATA]]></xsl:text>
[bla<bla<xsl:text>]</xsl:text>
<xsl:text disable-output-
  escaping="yes">]></xsl:text></elem>
```


Générer une section CDATA - 2

- Vous voulez générer ceci

```
<elem><![CDATA[bla<bla]]></elem>
```

- Solution 2
  - Mettre une déclaration
  - `<xsl:output indent="yes" method="xml" cdata-section-elements="elem"/>`
  - Et vous obtenez le résultat voulu !

Générateur d'identificateur

- `generate-id(expr)`

génère automatiquement un identificateur XML spécifique au premier nœud de l'ensemble de nœuds donné par l'expression

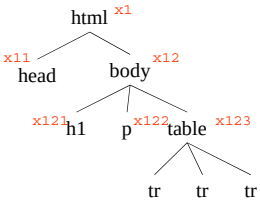
- `generate-id()`

génère automatiquement un identificateur XML spécifique au nœud courant

Fonction `generate-id()`

- Propriétés
  - L'ordre alphabétique des identificateurs est le même que l'ordre des nœuds dans le document
  - Si `generate-id(A)` est un préfixe de `generate-id(B)`, alors A est un ancêtre de B
  - L'identificateur est unique au sein de tous les documents ouverts durant l'exécution.

Une identification par nœud



generate-id(/html/body/table) = x123

Seconde partie

- Modularité
- Tri
- Procédures (Templates nommés)
  - Appel récursif et passage de paramètres
- Indexation par clé

Modularité

Modularité des documents sources

- Document composé de plusieurs documents

```
<livre>
    <chapitre>chap1.xml</chapitre>
    <chapitre>chap2.xml</chapitre>
    <chapitre>chap3.xml</chapitre>
</livre>
```

- Utiliser la fonction `document()`

```
<xsl:template match="chapitre">
   <xsl:apply-templates
        select="document(.)/*"/>
</xsl:template>
```


Fonction `document`

- Fonction `xslt:document(xpath1[,xpath2])`
  - Fixe le document de sortie vers celui identifié par l'URI. L'expression `xpath1` est évaluée et son résultat est convertit en une chaîne interprétée comme une URI
  - La syntaxe des URI acceptables n'est pas fixée par la norme. Il faut consulter la documentation du concepteur du processeur XSLT utilisé pour la connaître
- Base d'adressage
  - Si une l'expression `xpath2` est présente, le répertoire du document source dans lequel a été lu le nœud référencé par `xpath2` sert de base à l'URI.
  - Si une l'expression `xpath2` est absente, la base de l'URI est le répertoire dans lequel a été lue la feuille de style

Base d'adressage

- Si le document source n'a pas été lu sur un fichier mais construit en mémoire par un programme, on dispose éventuellement d'un moyen de fixer la base des URI par programme.
  - Il faut un processeur xslt qui le permette
  - Par exemple avec Java 1.5 qui implémente la norme JAXP 1.3 et Saxon qui la respecte aussi on peut créer un document source en interne et fixer sa base comme suit

```
DocumentSource source = new DocumentSource(inputDoc);
source.setSystemId(rootPath);
```


Elément `<xsl:import>`

- Modularité des feuilles de style

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="xml" version="1.0"
        encoding="ISO-8859-1" indent="yes"/>
        <xsl:import href="feuille1.xsl"/>
...
</xsl:stylesheet>
```

Elément <xsl:document>

- Modularité des documents en sortie

```
<xsl:template match="preface">
  <xsl:document href="{$dir}\preface.html">
    <html><body>
      <xsl:apply-templates/>
    </body></html>
  </xsl:document>
  <a href="{$dir}\preface.html">Preface</a>
</xsl:template>
```

- Version 1.1 de XSLT
  - Seul Saxon l'implémente actuellement

Trier

Élément <xsl:sort>

- Permet de trier l'ensemble des nœuds sélectionnés par les instructions avant de les traîter

```
<xsl:apply-templates>
<xsl:for-each>
```


Exemple

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

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

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

```
<XXX >
    <ccc/>
    <zzz/>
    <ddd/>
    <ooo/>
</XXX>
```

XXX ccc zzz ddd ooo

Exemple (suite)

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

        <xsl:template match = "/" >
            <xsl:apply-templates select = "//*" >
                <xsl:sort select = "name()" />
            </xsl:apply-templates>
        </xsl:template>

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

```
<XXX >
    <ccc/>
    <zzz/>
    <ddd/>
    <ooo/>
</XXX>
```

ccc ddd ooo XXX zzz

Tri sur plusieurs critères

- Trier d'abord par noms puis par prénoms

```
<sxl:templates match="carnetDAdresse">
   <xsl:apply-templates
  select="carteDeVisite">
      <xsl:sort select="nom"/>
      <xsl:sort select="prénom"/>
   </xsl:apply-templates>
</xsl:template>
```

Définition de fonctions

Fonction = templates nommés

- Déclaration de fonction

```
<xsl:template name="fun">
   ...
</xsl:template>
```

- Appel de fonction

```
<xsl:call-template name="fun"/>
```


Passage de paramètres

- Déclaration (*paramètre formel*)

Valeur par défaut

```
<xsl:template name="fun">
   <xsl:param name="p" select="0"/>
   ... utilisation de p ...
   <xsl:call-template name="fun">
     <xsl:with-param name="p"
           select="$p+1"/>
   </xsl:call-template>
</xsl:template>
```

- Obtenir la valeur d'un paramètre `select="$p"`

Calcul de la profondeur d'un élément

- Nom de la fonction `max`
- Paramètre `$x` un ensemble de nœuds

```
<xsl:template name="max">
   <xsl:param name="x"/>
   <xsl:choose>
      <xsl:when test="$x">
         <!-- cas où $x est non vide -->
 … (c.f. page suivante)
      </xsl:when>
      <xsl:otherwise>0</xsl:otherwise>
   </xsl:choose>
</xsl:template>
```


Cas où $x est non vide

```
<xsl:variable name="prof1">
   <xsl:call-template name="max">
      <xsl:with-param name="x" select="$x/*[1]"/>
   </xsl:call-template>
</xsl:variable>
<xsl:variable name="prof2">
   <xsl:call-template name="max">
      <xsl:with-param name="x" select="$x/*[position() != 1]"/>
   </xsl:call-template>
</xsl:variable>
<xsl:choose>
   <xsl:when test="$prof1 > $prof2">
      <xsl:value-of select="1 + $prof1"/>
   </xsl:when>
   <xsl:otherwise>
      <xsl:value-of select="1 + $prof2"/>
   </xsl:otherwise>
</xsl:choose>
```

Profondeur max du premier

Profondeur max des suivants

Choix du maximum

Mettre sous forme d'arbre

En entrée

```
<a>
  <h1>titre 1</h1>
  <par>bla bla 1</par>
  <par>bla bla 2</par>
  <par>bla bla 3</par>
  <h1>titre 2</h1>
  <par>bla bla 4</par>
  <par>bla bla 5</par>
  <par>bla bla 6</par>
</a>
```

En sortie

```
<a>
  <h1>
    <titre>titre 1</titre>
    <p>bla bla 1</p>
    <p>bla bla 2</p>
    <p>bla bla 3</p>
  </h1>
  <h1>
    <titre>titre 2</titre>
    <p>bla bla 4</p>
    <p>bla bla 5</p>
    <p>bla bla 6</p>
  </h1>
</a>
```

Mettre sous forme d'arbre (bis)

- Itération sur tous les éléments `<h1>`

```
<xsl:template match="/">
 <a>
  <xsl:for-each select="//h1">
   <h1>
    <titre><xsl:value-of select="."/></titre>
    <xsl:call-template name="frere">
     <xsl:with-param name="nds" select="following-sibling::*"/>
    </xsl:call-template>
   </h1>
  </xsl:for-each>
 </a>
</xsl:template>
```

Sélection des frères qui suivent

Mettre sous forme d'arbre (ter)

```
<xsl:template name="frere">
  <xsl:param name="nds"/>
  <xsl:choose>
   <xsl:when test="$nds[position()=1 and name()='par']">
    <p><xsl:value-of select="$nds[1]"/></p>
    <xsl:call-template name="frere">
     <xsl:with-param name="nds"
      select="$nds[position()!=1]"/>
    </xsl:call-template>
   </xsl:when>
   <xsl:otherwise/>
  </xsl:choose>
</xsl:template>
```

Traitement du premier frère si c'est un élément par

Appel récursif sur les autres frères après traitement du premier

Arrêt de la récursivité si le premier frère est un h1

Paramètrer une feuille de style

- Les paramètres descendants directs d'un élément `<xsl:stylesheet>` sont autorisés

```
<xsl:stylesheet>
   <xsl:param name="dir" select="'mondir'"/>
   ...
</xsl:stylesheet>
```

- On peut passer une valeur dans la ligne de commande

```
java ... dir=monDir
```


Exemple

```
<xsl:stylesheet version="1.0" …>
<xsl:param name="jour"/>
<xsl:param name="mois"/>

<xsl:template match="/">
<html>
  <body>
  …
  <p><Dernière mise à jour le,
  <xsl:value-of select="$jour"/>
  <xsl:value-of select="$mois"/>.</p>
…
```

déclaration

Utilisation

Mise au point

- Élément `<xsl:message>`

```
<xsl:message>
    code = <xsl:value-of select="$code"/>
</xsl:message>
```

- Trace en sortie dans la fenêtre de commande

```
code = 25
```


Les clés

- Un mécanisme d'indexation des documents

Les clés

- Un mécanisme d'indexation dynamique
  - Permet d'indexer des nœuds de l'arbre d'entrée sans placer d'attribut ID
  - Les déclarations apparaissent dans la feuille de style et non dans le document à transformer (pas d'attribut ID)
- Associe des paires *<nomDeClé,Valeur>* à des nœuds

Déclaration de clé `<xsl:key>`

- Toute déclaration doit être au premier niveau de la feuille de style

```
<xsl:key name="id" match="XPath motif"
    use="XPath value"/>
```

- Les trois attributs sont obligatoires
  - Le nom de la clé est ***id***.
  - L'ensemble de nœuds à décorer (les *nœuds filtrés*) résulte de l'évaluation de l'expression ***XPath motif***,
  - Chaque nœud filtré est décoré par un ensemble de paires (clé,valeur) où les valeurs résultent de l'évaluation de ***XPath value*** (voir ci-après)

Valeur(s) des clés

- Pour chaque nœud filtré l'ensemble des valeurs à lui associer est donné par l'attribut `use="`***`XPath value`***`"`
  - ***XPath value*** est évalué localement à chaque nœud filtré
  - Si ***XPath value*** retourne un ensemble de nœuds, chaque nœud résultat est converti en une chaîne qui donne une valeur (le nœud filtré est indexé par plusieurs paires *<clé,valeur>* !)
  - Tout autre résultat est converti en une chaîne qui donne une valeur unique

Exemple de déclaration

```
<xsl:keys name="carteAvecNote"
    match="//carteDeVisite[note]"
    use="nom"/>
```

1. Déclare une clé appelée `'carteAvecNote'`
2. Y associe tous les éléments `<carteDeVisite>` ayant un fils `<note>`
3. Associe à chaque élément filtré un indexe qui est une paire *<'carteAvecNote',nom de la personne>*
4. Si l'éléments `<carteDeVisite>` comportait plusieurs noms, chaque nom donnerait une paire

Utilisation – la fonction `key()`

- Appel `key(`*nom*`,`*expr*`)`
  - L'argument *nom* est un nom de clé défini par un élément `<xsl:key name="carteAvecNote">`
  - L'argument *expr* est une expression XPath
    - Si *expr* retourne un ensemble de nœuds, chaque nœud résultat est converti en une chaîne qui donne une valeur de clé (la recherche peut porter sur plusieurs clés)
    - Tout autre résultat est converti en une chaîne qui donne une valeur de clé
  - Exemple `key('carteAvecNote','bekkers')`
- Retourne un ensemble de nœuds appartenant au document courant
  - Il s'agit des éléments qui sont indexés par la paire *<'nom',valeur>*

Propriétés des clés

- Les valeurs de clé ne sont pas uniques
  - La même paire peut identifier plusieurs nœuds
- Chaque nœud filtré peut avoir zéro, une ou plusieurs paire qui l'indexe
- La fonction `key()` retourne un ensemble de nœuds qui sont tous localisés dans le même document XML

Exemple1

- Soit le document XML

```
<personne>
  <homme name="Bob" épouse="Alice"/>
  <femme name="Alice"/>
</personne>
```

- Problème : "*Retrouver les époux pour les femmes*"
  - remarque : ils ne sont pas codés directement
- Solution
  - Déclaration d'une référence

```
<xsl:key name="man-by-wifes-name" match="homme"
  use="@épouse"/>
```

  - Utilisation de la référence

```
<xsl:template match="femme">
  <xsl:apply-templates
     select="key('man-by-wifes-name', @name)"/>
```


Exemple2 – explorer une pièce de théâtre

Soit une pièce de théâtre structurée comme suit

```
PLAY
    PERSONAE
        PERSONA
        PERSONA
    ACT
        SCENE
            SPEECH/SPEAKER
            SPEECH/SPEAKER
        SCENE
            SPEECH/SPEAKER
    ACT
        SCENE
            SPEECH/SPEAKER
            SPEECH/SPEAKER
    …
```


Extrait du début d'un acte

liste de personnages de la pièce

```
<PLAY>
   <TITLE>The Tragedy of Othello, the Moor of Venice</TITLE>
   <FM>
      …
   </FM>
   <PERSONAE>
        <TITLE>Dramatis Personae</TITLE>
        <PERSONA>DUKE OF VENICE</PERSONA>
        <PERSONA>BRABANTIO, a senator.</PERSONA>
        <PERSONA>Other Senators.</PERSONA>
        <PERSONA>GRATIANO, brother to Brabantio.</PERSONA>
        <PERSONA>LODOVICO, kinsman to Brabantio.</PERSONA>
        <PERSONA>OTHELLO, a noble Moor in the service of the
   Venetian state.</PERSONA>
…
```

Un personnage

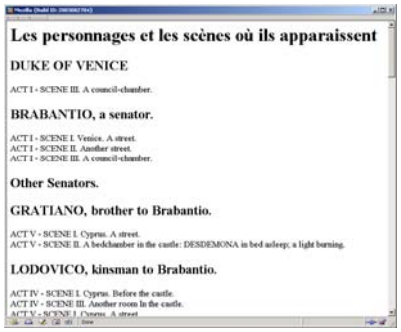
Extrait du début d'un acte

vue d'un discours

```
<ACT>
   <TITLE>ACT I</TITLE>
     <SCENE>
       <TITLE>SCENE I.  Venice. A street.</TITLE>
       <STAGEDIR>Enter RODERIGO and IAGO</STAGEDIR>
       <SPEECH>
         <SPEAKER>RODERIGO</SPEAKER>
         <LINE>Tush! never tell me; I take it …</LINE>
         <LINE>That thou, Iago, who hast had …</LINE>
         <LINE>As if the strings were thine, …</LINE>
       </SPEECH>
       <SPEECH>
         <SPEAKER>IAGO</SPEAKER>
         <LINE>'Sblood, but you will not hear me:</LINE>
```

discours

Présenter les personnages



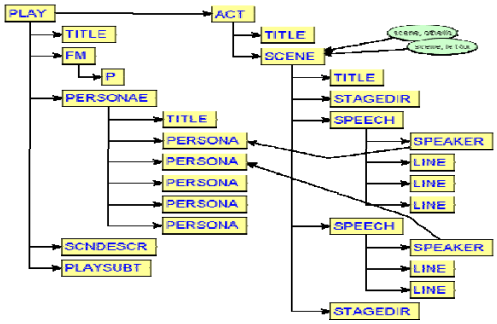
Déclaration

- Déclaration des indexes

```
<xsl:key name="scene" match="/PLAY/ACT/SCENE"
     use="SPEECH/SPEAKER"/>
```

  - Chaque élément `<scene>` est indexé par les noms de personnage qui y apparaissent

Graphe décoré par les indexes



Utilisation

```
<xsl:template match="PERSONA">
<xsl:variable name="nom" select="substring-before(.,',')"/>
      <xsl:variable name="speaker">
        <xsl:choose>
          <xsl:when test="$nom!=''">
            <xsl:value-of select="$nom"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="."/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
    <xsl:for-each select="key('scene',$speaker)">
    <xsl:value-of select="ancestor::ACT/TITLE"/> -
    <xsl:value-of select="TITLE"/>
    <br/>
  </xsl:for-each>
  ...
</xsl:template>
```

Extraction du nom

Est équivalent à

```
//SCENE[SPEECH/SPEAKER=$speaker]
```


Les sources de l'exemple

- La dtd
  - ~general/xml/XSL/keys/play.dtd
- Othello en XML
  - ~general/xml/XSL/keys/othello.xml
- La feuille de style
  - ~general/xml/XSL/keys/keys.xslt
- Le résultat HTML
  - ~general/xml/XSL/keys/personnages.html

Conclusion sur XSLT

XSLT qu'est-ce que c'est ?

- Un langage déclaratif (Turing complet !)
  - avec une syntaxe ésotérique !
- Utilise une technique de *filtrage* à base de *motifs* (patterns) *et de modèles* (template) décrits dans des *règles* (template rule) pour transformer des arbres
- Bien plus puissant que CSS …

Conclusion

- Un vrai langage de programmation
  - Les programmes XSLT sont plus que des feuilles de style !

Limites de CSS par rapport à XSLT

- CSS ne permet pas de visualiser les attributs
- CSS ne permet pas de réarranger l'information
- CSS n'offre pas de vrai moyen de faire des calculs
- La cible ne peut pas être un autre dialecte XML (CSS est un outil de présentation)

Documentation XSLT

- La documentation de Saxon constitue une excellente documentation pour XSLT
  - Elle est précise, concise …
    http://saxon.sourceforge.net/saxon6.5.3/index.html
  - N'hésitez pas à la consulter, même si vous n'utilisez pas Saxon

Outils

Extensions à la norme

- Saxon
  - *Elements xslt* : éléments sql, ...
  - *Functions xpath* : formatage de date, conversions de valeurs, évaluations xpath, try/catch ...
- EXSLT : une librairie portable en java (conseillée par saxon)
  - Fonctions mathématiques, expressions régulières, random, heures-dates, chaînes, ensembles ...
  - Site de chargement : http://www.exslt.org/

Lancer une transformation XSLT

- Nombreux moyens
  1. En ligne de commande
  2. En Java, en utilisant l'API JAXP 1.1, connu aussi comme TrAX (paquetage javax.xml.transform)
  3. Par la tâche `<style>` du processeur `Ant`
     - Sous Eclipse
     - En ligne de commande
  4. À l'aide XMLSpy
  5. À l'aide MS Internet Explorer

Éditer xslt, xpath

- Eclipse *(nombreux plugins)*
  - WTP, Oxygen, XmlBuddy, NiceXsl, OrangeVoltXsl, ...
- XmlSpy
- StylusStudio
- Vex
- Jedith

En ligne de commande

- Lancer Saxon7 en ligne de commande

```
java  net.sf.saxon.Transform  [options]
    source-document  stylesheet
    [ params...]
```

Utiliser l'API JAXP 1.1 (TrAX)

```
public static void apply(String in, String out, String xslt)
        throws FileNotFoundException, TransformerException {
    // Créer une fabrique de transformer
     TransformerFactory factory = TransformerFactory.newInstance();
    // Utiliser la fabrique pour créer un template
    // contenant la feuille de style
    Templates template = factory.newTemplates(new StreamSource(
                        new FileInputStream(xslt)));
    // Utiliser le template pour créer un transformateur
    Transformer xformer = template.newTransformer();

    // Préparer l'entrée et la sortie
    Source source = new StreamSource(new FileInputStream(in));
    Result result = new StreamResult(new FileOutputStream(out));

    // Transformer
    xformer.transform(source, result);
}
```


Effectuer la transformation par le navigateur

- Mettre l'instruction de traitement suivante en entête du document XML

```
<?xml-stylesheet type="text/xsl"
  href="../style3.xsl" ?>
```

- Faire lire le document XML par le navigateur IExplorer

Tâche `<style>` de `Ant`

```
<property name="prefixe" value="othello"/>
<property name="xslt.dir" value="…"/>
<project name="myProject" default="svg">
  <target name="svg">
    <style in="${prefixe}.xml" out="${prefixe}.svg"
           style="${xslt.dir}/xml2svg.xslt">
      <param name="text.color" expression="blue"/>
      <param name="box.color" expression="#FFFF99"/>
    </style>
  </target>
</project>
```


Ant et XSLT

- Ant utilise par défaut son propre processeur XSLT
- On peut lui changer ce processeur, voici un exemple de lancement de Ant avec Saxon pour processeur XSLT

```
java -cp \
myjar.jar;.;saxon.jar;ant.jar;jaxp.jar \
-Djavax.xml.transform.TransformerFactory=\
com.icl.saxon.TransformerFactoryImpl \
-Djavax.xml.parsers.SAXParserFactory=\
org.apache.crimson.jaxp.SAXParserFactoryImpl \
org.apache.tools.ant.Main %1 %2 %3 %4 %5 %6 %7 %8 %9
```


Conclusion

- Oui
  - XSLT est un vrai langage de programmation
  - XSLT n'a pas son équivalent pour la transformation d'arbre
- Mais
  - La mise au point de programmes XSLT peut s'avérer « délicate »
  - La maintenabilité est discutable

Des références

- *XSLT Programmer's Reference*
  Micheal H. Kay, Wrox Press
  `http://www.wrox.com`
- Saxon : Michael Kay
  `http://saxon.sourceforge.net/`
  très bonne documentation en ligne
- Xalan, Cocoon : projet Apache
  `http://xml.apache.org`