Transformer des données XML XSLT

- Objectifs
- Description d'une feuille de style
  - Règles
  - Processus de transformation
  - Exemples

- XSLT : fonctions étendues

- Version 1.0 W3C Recommendation 16 November 1999
- Version 2.0 Recommendation 23 January 2007

Bibliographie

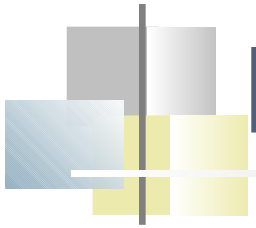
XSLT : XSL Transformation
eXtensible Stylesheet Langage Transformations
http://www.w3.org/TR/xslt

Comprendre XSLT
Bernd Amann et Philippe Rigaux, paru aux Editions O'Reilly

Différents tutoriaux (dont ceux d’Emmanuel Bruno, de Jacques Le Maitre)

XSLT fonctions étendues

Itération et tri
Structure conditionnelle
Variables
Modes
Et… divers



Programmer en XSLT

- Combinaison de deux modes de programmation
  - Déclaratif : on donne les règles, et le processeur fait le reste
  - Impératif : on utilise les structures habituelles d’un langage de programmation (tests, boucles, variables)

L'instruction *xsl:apply-templates*

- Sans attributs
- Avec attributs `select priority` **`mode`**

- **Mode**
  - Parcourir plusieurs fois le même nœud
  - Choisir explicitement une des règles parmi celles qui sont candidates
  - → Un nœud peut être traité plusieurs fois pour générer un résultat différent à chaque fois
  - → Produire plusieurs résultats à partir d'un nœud

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
                version = "1.0" >
    <xsl:output method = "xml" indent = "yes" />
    <xsl:template match = "/" >
            <AAA >
                <xsl:apply-templates select = "//BBB" />
                <xsl:apply-templates select = "//BBB" mode = "xxx" />
                <xsl:apply-templates select = "//BBB" mode = "yyy" />
            </AAA>
    </xsl:template>
    <xsl:template match = "BBB" >
            <OOO >
                <xsl:value-of select = "." />
            </OOO>
    </xsl:template>
    <xsl:template match = "BBB" mode = "xxx" >
            <XXX >
                <xsl:value-of select = "." />
            </XXX>
    </xsl:template>
    <xsl:template match = "BBB" mode = "yyy" >
            <YYY >
                <xsl:value-of select = "." />
            </YYY>
    </xsl:template>
</xsl:stylesheet>
```

exemple

```
<AAA >
    <BBB>10</BBB>
    <BBB>5</BBB>
    <BBB>7</BBB>
</AAA>
```

```
<?xml version="1.0" encoding="utf-8"?>
<AAA xmlns:zvon="http://www.zvon.org/">
    <OOO>10</OOO>
    <OOO>5</OOO>
    <OOO>7</OOO>
    <XXX>10</XXX>
    <XXX>5</XXX>
    <XXX>7</XXX>
    <YYY>10</YYY>
    <YYY>5</YYY>
    <YYY>7</YYY>
</AAA>
```

http://www.zvon.org/xxl/XSLTreference/Output/xslt_template_mode.html

Exemple

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

    <xsl:template match="/">
        <xsl:apply-templates select="//titre" mode="red"/>
        <xsl:apply-templates select="//titre" mode="blue"/>
        <xsl:apply-templates select="//titre"/>
    </xsl:template>

    <xsl:template match="titre" mode="red">
        <div style="color:red">
            <xsl:value-of select="name()"/>
            <xsl:text> id=</xsl:text>
            <xsl:value-of select="@id"/>
        </div>
    </xsl:template>

    <xsl:template match="titre" mode="blue">
        <div style="color:blue">
            <xsl:value-of select="name()"/>
            <xsl:text> id=</xsl:text>
            <xsl:value-of select="@id"/>
        </div>
    </xsl:template>

    <xsl:template match="titre">
        <div style="color:purple">
            <xsl:value-of select="name()"/>
            <xsl:text> id=</xsl:text>
            <xsl:value-of select="@id"/>
        </div>
    </xsl:template>
</xsl:stylesheet>
```

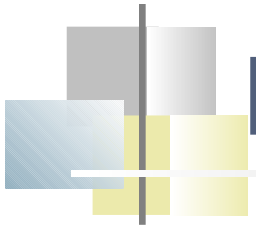
Construction d’éléments et d’attributs

- Créer des éléments ou des attributs à l'aide de valeurs extraites du document source

- Une solution pour introduire des attributs dans un élément quand :
  - on ne connaît pas la valeur de l’attribut
  - on ne connaît ni la valeur ni le nom
- Même remarque avec les éléments

Construction d’éléments et d’attributs

- Deux modes de construction
  - Instanciation explicite
  - Appel de fonction



Exemples

- Instanciation explicite

```
<title num='1'>
    <xsl:value-of select='@title'>
</title>
```

- Appel de fonction

```
<xsl:element name='{@title}'>
   <xsl:attribute name='size'>
      <xsl:value-of select='{count(Section)}'/>
   </xsl:attribute>
</xsl:element>
```

→ *Les { } indique qu'il s'agit d'un chemin XPath qui doit être remplacé par sa valeur*

Construction d’éléments et d’attributs

- Deux modes de construction
  - Instanciation explicite
  - Appel de fonction

  → **Méthode mixte**

```
<title name='{@title}'/>
```

Répétition

- Structure de répétition `xsl:for-each`
  - parcourir un ensemble de nœuds sélectionnés avec `select`
  - Prend les nœuds un par un comme nœud contexte
  - Le contenu de `xsl:for-each` - Les instructions sont appliquées successivement à chaque nœud sélectionné

Pas de variable, donc pas d'incrémentation

Exemple

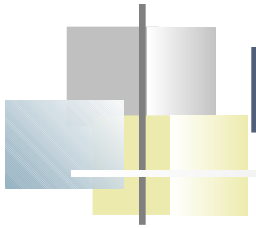
```
<?xml version="1.0" encoding="utf-8"?>
<Texte>
    <Chapitre>
     <Titre>titre1</Titre>
     <Section>
       <Titre>titresect1</Titre>
     </Section>
    </Chapitre>
    <Chapitre>
     <Titre>titre2 </Titre>
     <Section>
       <Titre>titresect2</Titre>
     </Section>
    </Chapitre>
</Texte>
```

```
<?xml version="1.0" encoding="utf-8"?>
<R>
  <H1>titre1</H1>
  <H1>titresect1</H1>
  <H1>titre2</H1>
  <H1>titresect2</H1>
</R>
```

```
<xsl:template match="/">
   <R>
      <xsl:for-each select=".//Titre">
         <H1>
            <xsl:apply-templates/>
         </H1>
      </xsl:for-each>
   </R>
</xsl:template>
```

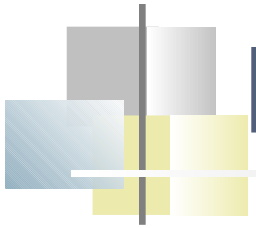
Tri

- `xsl:sort`
  - Tri des nœuds traités par `xsl:for-each` ou `xsl:apply-templates`
  - Un critère de tri est une expression XPath
  - A placer après la balise ouvrante de `xsl:for-each` ou `xsl:apply-templates`
  - Par défaut, croissant (attribut `order`, valeur `ascending` ou `descending`)
- Il peut avoir plusieurs critères qui définissent les différents niveaux de tri
- Par défaut les nœuds sont triés sur leur position



Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<Texte>
    <Chapitre>
     <Titre>titre1</Titre>
     <Section>
       <Titre>titresect1</Titre>
     </Section>
    </Chapitre>
    <Chapitre>
     <Titre>titre2 </Titre>
     <Section>
       <Titre>titresect2</Titre>
     </Section>
    </Chapitre>
</Texte>
```



Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<Texte>
    <Chapitre>
     <Titre>titre1</Titre>
     <Section>
       <Titre>titresect1</Titre>
     </Section>
    </Chapitre>
    <Chapitre>
     <Titre>titre2 </Titre>
     <Section>
       <Titre>titresect2</Titre>
     </Section>
    </Chapitre>
</Texte>
```

```
<xsl:template match="/">
   <R>
    <xsl:for-each select=".//Titre">
     <xsl:sort select='.'/>
     <H1><xsl:apply-templates/></H1>
    </xsl:for-each>
   </R>
 </xsl:template>
```

Exemple

```
<xsl:template match="/">
  <R>
   <xsl:for-each select=".//Titre">
    <xsl:sort select='.'/>
    <H1><xsl:apply-templates/></H1>
   </xsl:for-each>
  </R>
</xsl:template>
```

```
<?xml version="1.0" encoding="utf-8"?>
<R>
 <H1>titre1</H1>
 <H1>titre2</H1>
 <H1>titresect1</H1>
 <H1>titresect2</H1>
</R>
```

Traitement conditionnel

- `xsl:if` permet le traitement conditionnel d'un nœud

```
<list>
   <entry type='c'>item1</entry>
   <entry type='b'>item2</entry>
   <entry type='a'>item3</entry>
<list>
```



```
item1,item2,item3
```

Traitement conditionnel

- `xsl:if` permet le traitement conditionnel d'un nœud

```
<xsl:template match="list">
  <xsl:for-each select="entry">
    <xsl:value-of select="."/>
    <xsl:if test="not(position()=last())">
      <xsl:text>,  </xsl:text>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Pas de else

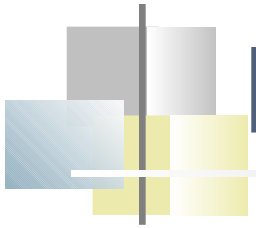
```
<list>
   <entry type='c'>item1</entry>
   <entry type='b'>item2</entry>
   <entry type='a'>item3</entry>
<list>
```

```
item1,item2,item3
```

Structure de test - xsl:choose

- Test à choix multiples (équivalent au switch/case du C)

```
<xsl:choose>
   <xsl:when test=’expression1’>
       Corps de règle 1
   </xsl:when>
   <xsl:when test=’expression2’>
       Corps de règle 2
   </xsl:when>
   ...
   <xsl:otherwise>
       Corps de règle par défaut
   </xsl:otherwise>
</xsl:choose>
```



Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<Livres>
    <Livre>
     <Titre>titre1</Titre>
     <Annee>1945</Annee>
    </Livre>
    <Livre>
     <Titre>titre2</Titre>
    </Livre>
    <Livre>
     <Titre>titre3</Titre>
     <Annee>2000</Annee>
    </Livre>
</Livres>
```

Exemple

```
<xsl:template match="Livre">
   <xsl:choose>
     <xsl:when test="Annee &lt; 1960">
         "<xsl:value-of select="TITRE"/>" est ancien
     </xsl:when>
     <xsl:when test="Annee &gt;= 1960">
        "<xsl:value-of select="TITRE"/>" est récent
     </xsl:when>
     <xsl:otherwise>
        De quand date "<xsl:value-of select="TITRE"/>" ?
     </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Copie de nœud

- Copie du nœud courant

```
<xsl:copy select='.'/>
```

- Copie de l'arbre

```
<xsl:copy-of select='.'/>
```

Variables

- XSLT permet de définir des variables (nom + valeur)

```
<xsl:variable name='nom'>valeur</xsl:variable>
```

- Les variables sont visibles dans toute la sous arborescence

Exemples

```
<xsl:variable name='v1'>
  ceci est un <mot-cle>contenu</mot-cle> de variable
</xsl:variable>
```

→ Le contenu de l'élément `xsl:variable` est la valeur de la variable

```
<xsl:variable name='v1' select='12'/>
```

→ Attribut `select` avec la valeur (une constante)

```
<xsl:variable name='v1' select='/COURS/ENSEIGNANTS'/>
```

→ Attribut select avec une expression XPath

→ Arbre temporaire auquel on peut appliquer une expression XPath

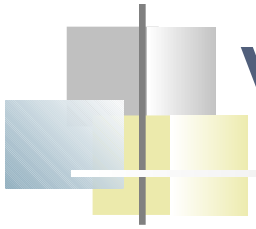
Exemple – variable globale

```
<xsl:variable name="year" select="1970"/>

<xsl:template match="Livre">
   <xsl:choose>
     <xsl:when test="Annee &lt; $year">
         "<xsl:value-of select="TITRE"/>" est ancien
     </xsl:when>
     <xsl:when test="Annee &gt;= $year">
        "<xsl:value-of select="TITRE"/>" est récent
     </xsl:when>
     <xsl:otherwise>
        De quand date "<xsl:value-of select="TITRE"/>" ?
     </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Exemple – variable locale

```
<xsl:template match="LIVRES">
  <INFO>
    <xsl:variable name="phrase">
      (année de parution
        <xsl:value-of select="ANNEE"/>)
    </xsl:variable>
    <xsl:for-each select="LIVRE">
       <xsl:value-of select="concat(TITRE,':' $phrase)"/>
    </xsl:for-each>
  </INFO>
</xsl:template>
```



Variables

- Un type particulier de variables → les paramètres

```
<xsl:param name='X'/>default</xsl:param>
```

Templates nommés et Fonctions

- XSLT permet de nommer un *template* et de l'appeler explicitement à n'importe quel endroit
  - Cela permet de définir des fonctions

```
<xsl:template name = 'nom'>
   …
</xsl:template>
```

```
<xsl:call-template name = 'nom'>
  …
</xsl:call-template>
```

Exemple

- On nomme une règle `Afficher`

```
<xsl:template name="Afficher">
    <xsl:value-of select="position()" />
    :
    <xsl:value-of select="." />
</xsl:template>
```

- On appelle la règle `Afficher`

```
<xsl:template match="NOM">
     <xsl:call-template name="Afficher"/>
</xsl:template>
```

Templates nommés et Fonctions

- XSL permet de nommer un *template* et de l'appeler explicitement à n'importe quel endroit

- Cela permet de définir des fonctions
  - On peut passer des paramètres avec `xsl:param`

Passage de paramètres

- On peut passer des paramètres à `xsl:call-template` ou `xsl:apply-templates`

- `xsl:param`
  - définition dans la règle des paramètres attendus
- `xsl:with-param`
  - association d'un ou plusieurs paramètres à `xsl:call-template` ou `xsl:apply-templates`

Exemple - paramètres

puce est le nom du paramètre
$puce désigne le paramètre

Avec l'instruction `xsl:param` à placer après `xsl:template`

```
<xsl:template name="ItemApuce">
  <xsl:param name= "puce">- </xsl:param>
  <fo:block>
    <xsl:value-of select="$puce"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="li">
  <xsl:call-template name="ItemApuce">
    <xsl:with-param name="puce">* </xsl:with-param>
  </xsl:call-template>
</xsl:template>

<xsl:template match="dd">
  <xsl:call-template name="ItemApuce">
    <xsl:with-param name="puce"># </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Avec l'instruction `xsl:with-param` à placer après `xsl:call-template`

Autre exemple - paramètre

La règle `Afficher` attend un paramètre

```
<xsl:template match="NOM">
  <xsl:call-template name="Afficher">
    <xsl:with-param name="texte" select="."/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="Afficher">
   <xsl:param name="texte">
   <xsl:value-of
              select="concat(position(),' : ', $texte)"/>
</xsl:template>
```

Définition et déclenchement de règles

- Une règle est donc définie par l'élément `xsl:template`
- Deux possibilités :
  - L'attribut `match` est une expression *XPath* définissant les « cibles » de la règle
    - déclenchement par `xsl:apply-templates`
  - L'attribut `name` donne un nom à la règle
    - déclenchement par `xsl:call-template`

Inclusion/Import de feuille de style

- Dans un document XML
  - exprimée dans le document source
  - avec une instruction de traitement
- Dans un feuille de style
  - inclusion de règles d'une feuille XSLT dans une autre
  - Utilisation de deux éléments de haut niveau

```
<!-- Category: top-level-element -->
  <xsl:include
    href = uri-reference />
  <xsl:import
    href = uri-reference />
```

Utilisation de plusieurs feuilles de style

- Différence: la gestion des conflits

```
<xsl:include href = uri-reference />
```

→ pas de notion de préséance

```
<xsl:import  href = uri-reference />
```

→ les règles importées ont une préséance moindre que celles du programme importateur

```
<source>
   <H1>IMPORTING STYLESHEETS</H1>
</source>
```

Exemples

id2.xsl

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

<xsl:variable name="id2">Stylesheet 1(id2.xsl)</xsl:variable>
<xsl:variable name="t">Variable t from id2.xsl</xsl:variable>

</xsl:stylesheet>
```

id3.xsl

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

<xsl:variable name="id3">Stylesheet 2(id3.xsl)</xsl:variable>
<xsl:variable name="t">Variable t from id3.xsl</xsl:variable>

</xsl:stylesheet>
```

Exemples

```
<source>
   <H1>IMPORTING STYLESHEETS</H1>
</source>
```

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

<xsl:import href="id3.xsl"/>
<xsl:include href="id2.xsl"/>
<xsl:template match="/">
     <P>
          <xsl:value-of select="$id2"/>
     </P>
     <P>
          <xsl:value-of select="$id3"/>
     </P>
</xsl:template>


</xsl:stylesheet>
```

```
<P>Stylesheet 1(id2.xsl)</P>
<P>Stylesheet 2(id3.xsl)</P>
```

Exemples

```
<source>
   <H1>IMPORTING STYLESHEETS</H1>
</source>
```

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:import href="id2.xsl"/>
<xsl:import href="id3.xsl"/>
<xsl:template match="/">
     <P>
          <xsl:value-of select="$id2"/>
     </P>
     <P>
          <xsl:value-of select="$id3"/>
     </P>
     <P>
          <xsl:value-of select="$t"/>
     </P>
</xsl:template>
</xsl:stylesheet>
```

```
<P>Stylesheet 1(id2.xsl)</P>
<P>Stylesheet 2(id3.xsl)</P>
<P>Variable t from id3.xsl</P>
```

Exemples

```
<source>
   <H1>IMPORTING STYLESHEETS</H1>
</source>
```

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:import href="id3.xsl"/>
<xsl:import href="id2.xsl"/>
<xsl:template match="/">
     <P>
          <xsl:value-of select="$id2"/>
     </P>
     <P>
          <xsl:value-of select="$id3"/>
     </P>
     <P>
          <xsl:value-of select="$t"/>
     </P>
</xsl:template>
</xsl:stylesheet>
```

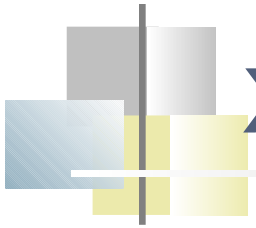
```
<P>Stylesheet 1(id2.xsl)</P>
<P>Stylesheet 2(id3.xsl)</P>
<P>Variable t from id2.xsl</P>
```

Exemples

```
<source>
   <H1>IMPORTING STYLESHEETS</H1>
</source>
```

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:import  href="id3.xsl"/>
<xsl:include href="id2.xsl"/>
<xsl:template match="/">
     <P>
          <xsl:value-of select="$id2"/>
     </P>
     <P>
          <xsl:value-of select="$id3"/>
     </P>
     <P>
          <xsl:value-of select="$t"/>
     </P>
</xsl:template>
</xsl:stylesheet>
```

```
<P>Stylesheet 1(id2.xsl)</P>
<P>Stylesheet 2(id3.xsl)</P>
<P>Variable t from id2.xsl</P>
```



xsl:output

`<xsl:output>`

- placé au début pour indiquer le format

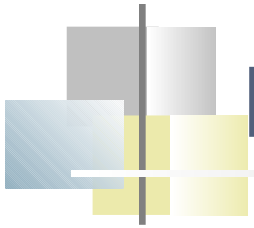
`<output method='format'>`

- format *xml* (le défaut) indique une sortie dans un dialecte XML (par exemple WML)
- Le format *html* indique une sortie HTML
- Le format *text* produit un document sans balise

Résumé

- Les éléments de premier niveau, fils de `<xsl:stylesheet>`

Type d'élément	Description
`xsl:import`	Import d'un programme XSLT
`xsl:include`	Inclusion d'un programme XSLT
`xsl:output`	Indique le format de sortie
`xsl:param`	Définit un paramètre
`xsl:template`	Définit une règle XSLT
`xsl:variable`	Définit une variable XSLT



Bilan

- Langage de règles
  - Transformer pour …
  - Intégration de XPath
  - Instructions …
    - Ce n’est pas un langage de requêtes
    - Une feuille de style pour *un document (ou une classe de document)*