

XSL (Extensible stylesheet language)

Un document XSL est une feuille de style écrite en XML permettant de transformer un document XML en ...

1 Introduction

XSL est le langage de description de feuilles de style pour les documents xml. Un document xsl permettra donc de décrire comment doivent être transformés (affichés au sens large) des documents xml respectant une même description (dtd ou schéma).

XSL est spécifié en trois modules :

- **XSLT**: langage de transformation de documents xml
- **XPATH** : langage de navigation dans un document xml
- **XSL-FO** : langage de formatage de documents xml

Quelques exemples

A partir du document xml catalogue.xml de la dernière séance,

1. Obtenons un document xml qui se limite à reprendre les catégories et les rayons.

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" />
<xsl:template match="/catalogue" >
<!-- limite le traitement de catalogue aux catégories-->
  <categories>
    <xsl:apply-templates select="categories/categorie" />
  </categories>
</xsl:template>

<xsl:template match="categorie">
  <categorie>
    <libelle><xsl:value-of select="libellecat"/></libelle>
    <rayons>
      <xsl:apply-templates select="rayons/rayon" />
    </rayons>
  </categorie>
</xsl:template>

<xsl:template match="rayon">
  <rayon>
    <rayid><xsl:value-of select="@id"/></rayid>
    <libelle><xsl:value-of select="libellerayon"/></libelle>
  </rayon>
</xsl:template>

</xsl:stylesheet>
```

Utilisons Altova pour procéder à la transformation: `AltovaXML -xslt2 prem.xslt -in catalogue.xml -out rayons.xml` où `prem.xslt` est le document xslt décrivant la transformation (ci-dessus), `catalogue.xml` le document xml des données et `rayons.xml` le document obtenu.

Le document obtenu respecte le dtd suivant:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT categories (categorie)*>
<!ELEMENT categorie (libelle,rayons)>
<!ELEMENT rayons (rayon)*>
<!ELEMENT rayon (rayid,libelle)>
<!ELEMENT libelle (#PCDATA)>
<!ELEMENT rayid (#PCDATA)>
```

2. Obtenons une page html présentant les produits en promotion

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns="http://www.w3.org/1999/xhtml">
<xsl:output method="xml" doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" doctype-
public="-//W3C//DTD XHTML 1.0 Strict//EN" indent="yes" encoding="UTF-8"/>
<xsl:template match="/catalogue">
<html >
<head>
<meta name="author" content="XSLT" />
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Produits en promotion</title>
</head>
<body>
<h2>Les produits en promotion cette semaine</h2>
<table border="1">
<tr>
<th align="left">Image</th>
<th align="left">Rayon</th>
<th align="left">Libellé</th>
<th align="left">Marque</th>
<th align="left">Prix Normal</th>
<th align="left">Promotion</th>
</tr>
<xsl:apply-templates select="produits/produit[@class='promotion']" />
</table>
</body>
</html>
</xsl:template>

<xsl:template match="produit">
<tr>
<td>
<img alt="non disponible" >
<!-- création de l'attribut 'src' de la balise img reprenant la référence de l'image à afficher ,
attention nous supposons ici que les images sont dans le répertoire eVentesImages du répertoire courant -->
<xsl:attribute name="src" >
<xsl:value-of select="concat('eVentesImages/',substring(@id,2),'jpg')"/>
</xsl:attribute>
</img>
</td>
<!-- variable 'rayon' reprenant l'id du rayon référencé -->
<xsl:variable name="rayon"> <xsl:value-of select="@rayon"/> </xsl:variable>
<!-- affichage du libellé du rayon référencé -->
<td><xsl:value-of select="/catalogue/categories/categorie/rayons/rayon[@id=$rayon]/libellerayon"/></td>
<td><xsl:value-of select="libelleproduit"/></td>
<!-- variable mq reprenant l'id de la marque référencée -->
<xsl:variable name="mq"> <xsl:value-of select="@marque"/> </xsl:variable>
<!-- affichage du libellé de la marque référencée -->
<td><xsl:value-of select="/catalogue/marques/marque[@id=$mq]/libellemarque"/></td>
<td><xsl:value-of select="prix"/></td>
<td><xsl:value-of select="promotion"/></td>
</tr>
</xsl:template>
</xsl:stylesheet>
```

Supposons avoir sauver le document ci-dessus en deux.xslt, pour tester cet exemple, nous avons 2 possibilités:

ajouter la ligne : `<?xml-stylesheet type="text/xsl" href="deux.xslt"?>` au document xml à transformer. Dans ce cas, la transformation sera à charge du browser (ceci peut nuire à la performance)

exécuter la commande Altova : `AltovaXML -xslt2 deux.xslt -in catalogue.xml -out catalogue.html` qui produira le document html 'catalogue.html' en appliquant au document 'catalogue.xml' la transformation décrite dans 'deux.xslt' .

2 XPATH

Xpath est un langage de navigation dans la structure d'un document xml. Il va nous permettre de référencer des éléments, des attributs, des ensembles ...

Prenons prem.xslt de l'introduction. Nous y retrouvons plusieurs expressions XPATH reprises en gras ci-dessous:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" />
<xsl:template match="/catalogue" >
<!-- limite le traitement de catalogue à catégories-->
  <categories>
    <xsl:apply-templates select="categories/categorie" />
  </categories>
</xsl:template>

<xsl:template match="categorie">
  <categorie>
    <libelle><xsl:value-of select="libellecat" /></libelle>
    <rayons>
      <xsl:apply-templates select="rayons/rayon" />
    </rayons>
  </categorie>
</xsl:template>

<xsl:template match="rayon">
  <rayon>
    <rayid><xsl:value-of select="@id" /></rayid>
    <libelle><xsl:value-of select="libellerayon" /></libelle>
  </rayon>
</xsl:template>

</xsl:stylesheet>
```

- */catalogue*: spécifie l'élément racine 'catalogue'
- *categories/categorie* : spécifie l'ensemble des noeuds enfants des noeuds 'categorie' enfants de noeuds 'categories'.
- *../libellecat*: spécifie le noeud 'libellecat' du grand père du noeud courant.
- *@id*: spécifie la valeur de l'attribut 'id' du noeud courant.
- Sélection de noeuds: cf. http://www.w3schools.com/xpath/xpath_syntax.asp
- Xpath axes : cf. http://www.w3schools.com/xpath/xpath_axes.asp

Exemple-exercice:

Modifier prem.xslt pour ne fournir que les rayons de la catégorie de id 'c4'. Le résultat doit satisfaire le même dtd que le résultat précédent.

Référence: <http://www.w3schools.com/xpath/default.asp>

3 XSLT

Un document xslt, qui est un document xml, doit présenter comme première balise :

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

`<xsl:output .../>` permet de définir le format du document de sortie.

`<xsl:stylesheet ...>` et `<xsl:transform ...>` sont synonymes et définissent l'élément racine du document.

Les templates

Par défaut, tout noeud est transformé par sa valeur. Pour le voir, il nous suffit de transformer notre document xml avec les xslt suivant:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

</xsl:stylesheet>
```

Il est évident que le document produit n'est pas valide au sens xml. Par contre, le suivant bien:

```
<?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="/">
  <contenu>
    <xsl:apply-templates select="catalogue" />
  </contenu>
</xsl:template>
</xsl:stylesheet>
```

Nous pouvons fournir des templates de transformation de noeud. Si nous ajoutons:

```
<xsl:template match="libelleproduit" >
  <lib>
    Ceci est un libellé: <xsl:value-of select="." />
  </lib>
</xsl:template>
```

Chaque noeud libelle produit', où qu'il se trouve, sera transformé (Testez).

Si nous désirons ne pas afficher les marques, nous pourrions ajouter:

```
<xsl:template match="marques" />
```

ou si nous voulons nous contenter des produits, il sera plus efficace de modifier le template de la racine:

```
<xsl:template match="/">
  <contenu>
    <xsl:apply-templates select="catalogue/produits" />
  </contenu>
</xsl:template>
```

Exemple-exercice

Modifiez prem.xslt pour ne fournir que les rayons (id, libelle, et libelle de catégorie).

Dtd du résultat:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ELEMENT rayons (rayon)*>
<ELEMENT rayon (rayid,libelle, libellecat)>
<ELEMENT libellecat (#PCDATA)>
<ELEMENT libelle (#PCDATA)>
<ELEMENT rayid (#PCDATA)>
```

Modifiez prem.xslt pour ne fournir que les rayons (id, libelle, et l'id de la catégorie comme attribut 'idcat' du rayon)

Dtd du résultat:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ELEMENT rayons (rayon)*>
<ELEMENT rayon (rayid,libelle)>
<ELEMENT libelle (#PCDATA)>
<ELEMENT rayid (#PCDATA)>
<ATTLIST rayon idcat CDATA #REQUIRED >
```

Les contrôles de séquence

Il est possible de sortir de la notion d'application de templates au travers des éléments:

```
<xsl:for-each ...> , <xsl:sort ...> , <xsl:if ...> ,
<xsl:choose ...>, <xsl:call-template ...>
```

Les fonctions

De nombreuses fonctions sont disponibles: cf.

http://www.w3schools.com/xpath/xpath_functions.asp

Les variables

Dans un document xslt, il est possible de définir des variables locales ou globales. Une variable locale est déclarée à l'intérieur d'un template, une globale doit être déclarée au premier niveau du document xslt.

```
<xsl:variable name="Section" select="'Gestion'" />
```

ou

```
<xsl:variable name="presentation" />
  <p class='enTeteListeproduits'>Produits en promotion</p>
  <p>Profitez-en</p>
</xsl:variable>
```

Nous accéderons à la valeur de la variable par `<xsl:copy-of select='$Section' />` ou par `$Section` dans une expression XPATH.

Exemple: deux.xslt reprend des exemples de variables.

Les paramètres

Un paramètre local est déclaré à l'intérieur d'un template, un global doit être déclarée au premier niveau du document xslt.

```
<xsl:param name="Section" select="'Gestion'" />
```

L'attribut select est optionnel et s'il est présent reprend la valeur par défaut du paramètre.

Si vous désirez utiliser un paramètre avec Altova, il suffira d'ajouter

`-param nomParam='valuePARAM'` à la fin de la commande de transformation.

Exemples-exercices:

Écrivez un document xsl fournissant un document XHTML valide qui présente la liste des produits ventilés par catégorie et rayon. Veillez à ce que les produits indisponibles n'apparaissent pas. Veillez à spécifier un attribut class='promo' pour les produits en promotion.

Prévoyez un paramètre reprenant l'id d'une catégorie pour pouvoir limiter la présentation des produits à une catégorie.

Que devriez-vous prévoir pour que le document XHTML fasse référence à un css externe?

4 XSL-FO (Extensible Stylesheet Language Formatting Objects)

Nous ne nous étendrons pas sur ce sujet mais Xslfo permet de formater très précisément des documents.

5 Conclusion

Xml et Xsl sont rapidement devenus des standards de représentation, stockage (ceci dans une moindre mesure), transfert et manipulation de l'information.

De nombreux environnements permettent de les mettre en oeuvre : JavaScript (cf. http://www.w3schools.com/ajax/ajax_responsexml.asp), java, .net, ... Le prochain standard SQL (SQL4) définira des moyens de stocker du xml dans les BD relationnelles et d'extraire des données relationnelles sous forme Xml à partir de SQL.

Même si nous n'avons fait qu'aborder superficiellement¹ les technologies xml et xsl et que nous ne les avons pas mises en oeuvre dans des cas concrets, nous pouvons percevoir pourquoi elles s'imposent actuellement comme un must notamment dans l'environnement des applications distribuées: des systèmes hétérogènes peuvent communiquer des données structurées et validables. De plus, grâce à xsl, les différents systèmes peuvent aisément traiter/convertir ces données.

¹ Nous n'avons pas parcouru l'ensemble des spécifications et nous n'avons pas présenté de cas concrets d'utilisation (e.a. Les WebSeervices).