XSL-T
eXtended Stylesheet Language
Cours ≪ Document et Web Sémantique ≫

Nicolas Malandain, Nicolas Delestre

Plan. . .

eXtended Stylesheet Language

XSL est divisé en deux parties :

XSL Transformations (XSLT)

Application XML spécifiant des règles de transformation d'un document en un autre. Une feuille de style XSLT compare les éléments d'un document XML aux modèles de la feuille de style. Pour chaque modèle correspondant son contenu est placé en sortie du nouveau document.

XSL Formatting Objects (XSL-FO)

Application XML pour décrire la mise en page précise de texte.

Squelette d'une feuille XSLT

C'est un document XML

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

L'élément racine du document est :

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

</xsl:stylesheet>
```

Document de démonstration : demoA.xml

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

<personnes>
  <personne naissance="1912" mort="1954">
    <identite>
      <prenom>Alan</prenom>
      <nom>Turing</nom>
    </identite>
    <profession>informaticien</profession>
    <profession>mathématicien</profession>
    <profession>cryptographe</profession>
  </personne>
  <personne naissance="1969" mort="?">
    <identite>
      <prenom>Linus</prenom>
      <nom>Torvalds</nom>
    </identite>
    <profession>informaticien</profession>
    <loisir>construire un OS</loisir>
  </personne>
</personnes>
```


Contenu possible de l'élément xsl:stylesheet

Liste non exhaustive :

- Instructions d'importation ou d'inclusion d'autres feuilles XSLT :
  `<xsl:import href="..."/>`
  `<xsl:include href="..."/>`
- Spéficie si les éléments *texte* contenant uniquement les caractères d'espacement (espace, tabulation, retour chariot, etc.) doivent être conservés :
  `<xsl:strip-space elements="nœuds"/>`
  `<xsl:preserve-space elements="nœuds"/>`
- Spécifie le type de document en sortie (`method` = xml, html, ...), si la sortie doit être indentée ou non, ...
  `<xsl:output method="..." indent="yes/no" .../>`
- Définition de variables et de paramètres
  `<xsl:variable name="..."> ... </xsl:variable>`
  `<xsl:param name="..."> ... </xsl:param>`
- Déclaration de règles (modèles) de transformation
  `<xsl:template match="..."> ... </xsl:template>`
  `<xsl:template name="..."> ... </xsl:template>`

Modèles

Les modèles (règles) sont les éléments principaux d'une feuille XSLT

Syntaxe

```xml
<xsl:template match="pattern xpath">
  instructions XSLT et/ou données littérales
</xsl:template>
```

Fonctionnement

Le processeur XSLT lit (parcours en profondeur) un document source XML et applique une règle chaque fois que son motif (pattern) correspond au nœud courant.

Feuille XSLT demo1.xsl

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/
    XSL/Transform">
  <xsl:output encoding="ISO-8859-1" />
</xsl:stylesheet>
```

Résultat :

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


    Alan
    Turing

  informaticien
  mathématicien
  cryptographe



    Linus
    Torvalds

  informaticien
  construire un OS
```

Commentaires demo1.xsl

Deux règles de transformation cablées

- Si aucun modèle ne correspond aux nœuds du source alors le parcours récursif descend jusqu'aux terminaux
- Si un élément contient du texte (#PCDATA) alors celui ci est recopié par défaut dans le nœud courant de l'arbre produit

Feuille XSLT demo2.xsl

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
2 <xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/
      XSL/Transform">
3   <xsl:output encoding="ISO-8859-1" />
4   <xsl:strip-space elements="*" />
5   <xsl:template match="personne">
6     <p>Quelqu'un</p>
7 ␣␣</xsl:template>
8
9 </xsl:stylesheet>
```

Résultat :

```
1 <?xml version="1.0" encoding="ISO-8859-1"?><p>Quelqu'un</p><p>
      Quelqu'un</p>
```

Question

Ce document XML est-il bien formé ?

Commentaires demo2.xsl

- `<xsl:strip-space elements="*"/>`
  Lorsqu'un nœud textuel ne contient que des caractères d'espacement alors le nœud est ignoré.
- `<xsl:template match="personne">`
  Pour chaque élément `personne` rencontré, le contenu a été ignoré et remplacé par `<p>Quelqu'un</p>`.
- La feuille de style devant être bien formée, cela implique que le balisage produit doit lui aussi être bien formé.

Obtenir la valeur d'un nœud

En règle général le texte de sortie est lié au texte d'entrée, il faut donc pouvoir récupérer des valeurs.

Syntaxe

```
<xsl:value-of select="pattern␣xpath" />
```

Feuille XSLT demo3.xsl

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
2 <xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/
      XSL/Transform">
3   <xsl:output encoding="ISO-8859-1" />
4
5   <xsl:template match="personne">
6     <p><xsl:value-of select="identite" /></p>
7   </xsl:template>
8
9 </xsl:stylesheet>
```

Résultat :

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2   <p>
3       Alan
4       Turing
5     </p>
6   <p>
7       Linus
8       Torvalds
9     </p>
```


Appliquer des modèles

- Chaque modèle associé à un nœud est activé avant les modèles de ses fils.
- Un modèle peut changer l'ordre de parcours de l'arbre XML source avec l'instruction suivante :

  ```
  <xsl:apply-templates select="pattern␣xpath" />
  ```

- Seuls les modèles correspondant au pattern seront activés, si l'attribut `select` est omis alors seuls les modèles correspondants aux fils du nœud contextuel sont traités.
- Cette instruction permet de choisir l'endroit où seront placés les résultats de l'activation de modèles (demo5.xsl).

Feuille XSLT demo4.xsl

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
2 <xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/
      XSL/Transform">
3   <xsl:output encoding="ISO-8859-1" />
4
5   <xsl:template match="personne">
6     <xsl:apply-templates select="identite" />
7   </xsl:template>
8
9   <xsl:template match="identite">
10    <xsl:value-of select="prenom" />, <xsl:value-of select="nom" />
11  </xsl:template>
12
13 </xsl:stylesheet>
```

Résultat

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2   Alan, Turing
3   Linus, Torvalds
```


Feuille XSLT demo5.xsl 1 / 2

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
2 <xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/
      XSL/Transform">
3   <xsl:output encoding="ISO-8859-1" />
4   <xsl:strip-space elements="*" />
5
6   <xsl:template match="personnes">
7     <html>
8       <head><title>Liste de personnes</title> </head>
9       <body>
10        <xsl:apply-templates/>
11      </body>
12    </html>
13  </xsl:template>
14
15  <xsl:template match="personne">
16    <p><xsl:value-of select=".//prenom" />, <xsl:value-of select="
        .//nom" /></p>
17  </xsl:template>
18
19 </xsl:stylesheet>
```

Feuille XSLT `demo5.xsl` 2 / 2

Résultat

```
1 <html>
2    <head>
3       <meta http-equiv="Content-Type" content="text/html; charset
            =ISO-8859-1">
4       <title>Liste de personnes</title>
5    </head>
6    <body>
7       <p>Alan, Turing</p>
8       <p>Linus, Torvalds</p>
9    </body>
10 </html>
```

Remarque

la déclaration de texte `<?xml ... ?>` a disparu, le processeur XSLT a de lui même détecté qu'il s'agissait d'un document HTML par l'élément racine.

Modèles implicites

Il existe 7 types de nœuds (cf. cours XPath) dans un document XML. XSLT prédéfinit un modèle par défaut pour chaque type de nœuds.

- Pour tout nœud de texte ou d'attribut : copie sa valeur dans le document de sortie :

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

- Pour tout nœud d'élément et le nœud racine : applique les règles à leurs nœuds fils. Les nœuds d'attribut et d'espace de noms ne sont pas considérés comme des fils (déshérités) :

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

- Pour tout nœud de commentaire ou d'instruction de traitement : ne produit rien :

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


Modes et modèles

- Un même contenu en entrée doit apparaître plusieurs fois en sortie mais avec un formatage différent
- Déclaration (**l'attribut** `mode`)

```
<xsl:template match="pattern xpath" mode="nomDuMode">
  instructions XSLT et/ou données littérales
</xsl:template>
```

- L'utilisation :

```
<xsl:apply-templates select="pattern xpath" mode=""nomDuMode"/>
```


Feuille XSLT `demo6.xsl` 1 / 2

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
2 <xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:output encoding="ISO-8859-1" />
4
5   <xsl:template match="personnes">
6     <html>
7       <head><title>Liste de personnes</title> </head>
8       <body>
9         <h1>Index</h1>
10        <ul><xsl:apply-templates select="personne" mode="index" /></ul>
11        <h1>Détail</h1>
12        <xsl:apply-templates select="personne" />
13      </body>
14    </html>
15  </xsl:template>
16
17  <xsl:template match="personne" mode="index">
18    <li><xsl:value-of select=".//prenom" />, <xsl:value-of select=".//nom" /></li>
19  </xsl:template>
20
21  <xsl:template match="personne">
22    <p><xsl:value-of select=".//prenom" /><xsl:text> </xsl:text><xsl:value-of select="
          .//nom" />
23    né(e) en <xsl:value-of select="@naissance" /></p>
24  </xsl:template>
25
26
27 </xsl:stylesheet>
```

Feuille XSLT demo6.xsl 2 / 2

Résultat

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>Liste de personnes</title>
   </head>
   <body>
      <h1>Index</h1>
      <ul>
         <li>Alan, Turing</li>
         <li>Linus, Torvalds</li>
      </ul>
      <h1>Détail</h1>
      <p>Alan Turing
         né(e) en 1912
      </p>
      <p>Linus Torvalds
         né(e) en 1969
      </p>
   </body>
</html>
```

Nommage de règles (modèles)

Objectif

Faciliter la réutilisation de règles

Syntaxe

- Déclaration :

```
<xsl:template name="nomDeLaRegle" ...>
    Instructions XSLT et/ou données littérales
</xsl:template>
```

- Utilisation :

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

Modèles de valeur d'attribut

Objectif

Inclure une valeur à l'attribut d'un élément résultat littéral

Exemple

```
<xsl:template match="personne">
  <identite nom="{.//nom}" prenom="{.//prenom}"
            naissance="{@naissance}" />
</xsl:template>
```

Résultat

```
<identite naissance="1912" prenom="Alan" nom="Turing" />
<identite naissance="1969" prenom="Linus" nom="Torvalds" />
```

Paramètre 1 / 2

Déclaration

- Syntaxe :

```
<xsl:param name="nom paramètre" select="pattern xpath" />
```

ou

```
<xsl:param name="nom paramètre">
  instructions XSLT et/ou données littérales
</xsl:param>
```

- La valeur spécifiée est considérée comme valeur par défaut du paramètre
- La déclaration peut se placer à deux endroits :
  - dans une règle `xsl:template` afin de la paramétrer
  - au premier niveau (dans `xsl:stylesheet`), c'est alors au processeur XSLT de déterminer la valeur
    - par la ligne de commande option `-PARAM`
    - par le protocole HTTP (`GET`, `POST`) quand le processeur est intégré au serveur (Cocoon)

Paramètre 2 / 2

Utilisation

- Syntaxes :
  - `<xsl:with-param name="nom paramètre" select="pattern xpath" />`

    le type du paramètre est l'un des quatres types xpath : boolean, number, string ou node-set (idem à la déclaration)
  - `<xsl:with-param name="nom paramètre">`
    `instructions XSLT et/ou données littérales`
    `</xsl:with-param>`

    le type du paramètre est un *result tree fragment* = chaîne de caractère résultant de la sérialisation du fragment calculé, il est alors impossible d'effectuer une sélection sur ce fragment (pour résoudre ce problème, voir XSLT 2.0 ou fonction d'extension `nodeset(...)`) (idem à la déclaration)
- L'utilisation peut se faire dans 3 instructions XSLT : `<xsl:call-template>`, `<xsl:apply-template>`, `<xsl:apply-imports>`

Document de démonstration : `demoB.xml`

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

<personnes>
  <personne naissance="1912" mort="1954">
    <identite>
      <prenom>Alan</prenom>
      <nom>Turing</nom>
    </identite>
    <liste puce="Lettre">
      <profession>informaticien</profession>
      <profession>mathématicien</profession>
      <profession>cryptographe</profession>
    </liste>
  </personne>
  <personne naissance="1969" mort="?">
    <identite>
      <prenom>Linus</prenom>
      <nom>Torvalds</nom>
    </identite>
    <liste puce="chiffre">
      <profession>informaticien</profession>
      <loisir>construire un OS</loisir>
    </liste>
  </personne>
</personnes>
```


demo8.xsl 1 / 3

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

<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:xalan="http://xml.apache.org/xalan" exclude-result-prefixes="xalan">
  <xsl:output method="xml" indent="yes"/>
  <xsl:output encoding="ISO-8859-1" />

  <xsl:template match="//personne">
    <xsl:value-of select="identite/prenom" /><xsl:text> </xsl:text> <xsl:value-of select
         ="identite/nom" />
    <xsl:text>
    </xsl:text>
    <xsl:apply-templates select="liste" />
  </xsl:template>

  <xsl:template match="liste[@puce='chiffre']">
    <xsl:call-template name="constructionliste">
      <xsl:with-param name="formatpuce" select="'1. '" />
    </xsl:call-template>
  </xsl:template>

  <xsl:template match="liste[@puce='Lettre']">
    <xsl:call-template name="constructionliste">
      <xsl:with-param name="formatpuce">A. </xsl:with-param>
    </xsl:call-template>
  </xsl:template>

  <xsl:template match="liste">
    <xsl:call-template name="constructionliste" />
  </xsl:template>
```


demo8.xsl 2 / 3

```
  <xsl:template name="constructionliste">
    <xsl:param name="formatpuce" select="'I. '" />
    <xsl:for-each select="profession|loisir">
      <p><xsl:number value="position()" format="{$formatpuce}" /><xsl:value-of select="."
           " /></p>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

demo8.xsl 3 / 3

Résultat

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  Alan Turing
    <p>A. informaticien</p>
<p>B. mathématicien</p>
<p>C. cryptographe</p>
  Linus Torvalds
    <p>1. informaticien</p>
<p>2. construire un OS</p>
```

Déclaration et utilisation de variables

Syntaxe

```
<xsl:variable name="nom variable" select="pattern xpath/>
```

où

```
<xsl:variable name="nom variable">
  instructions XSLT et/ou données littérales
</xsl:variable>
```

Remarques

- Les types des variables sont les mêmes que pour les paramètres
- La déclaration peut se placer à deux endroits :
  - au premier niveau (dans `xsl:stylesheet`) : variable globale visible dans toute la feuille XSLT
  - dans une règle : variable locale visible dans tous les frères droits de la déclaration
- Les variables sont utilisables n'importe où dans le document, via $nomvariable

Autres instructions XSLT

Il existe bien d'autres instructions XSLT permettant :

- la numérotation et le tri des éléments (`xsl:sort`)
- le traitement conditionnel (`xsl:if` et `xsl:choose`)
- l'itération (`xsl:for-each`)
- la création dynamique d'éléments et d'attributs (`xsl:element` et `xsl:attribute`)
- la copie de nœuds (`xsl:copy` et `xsl:copy-of`)

Association d'un document XML et d'une feuille de style XSLT

Déclaration interne au document XML

Il suffit d'ajouter une instruction de traitement du type :

```
<?xml-stylesheet type="text/xsl" href="URI de la feuille" ?>
```

Dans Cocoon

Cocoon fonctionne par un système de pipelines se déclenchant sur des patterns de requêtes

Exemple de configuration :

pour toute requête à un fichier `*.html`, il associe un fichier `{1}.xml` et une feuille de style `{1}2html.xsl`