1 Transformer des données avec XSLT

Motivation

1. motivation à l'origine : associer un style à un document XML
   - XSL = XML Stylesheet Language
   - Les CSS que l'on utilise pour XHTML ne permettent pas d'afficher les valeurs des attributs, de transformer la structure du document, ni de créer de nouvelles données.
   - De plus CSS pour XML est un peu lourd car il n'y a pas de style par défaut comme en XHTML
   - XSL = XSL-FO (pour l'aspect formatage) + XSL-T (pour l'aspect transformation)
2. transformation d'un document XML en un autre document XML
   - Transformation d'un arbre source en un arbre cible
   - Une transformation est donnée par un ensemble de règles
   - XSLT utilise XPath 2.0

Le modèle de données de XPath 2.0

Le résultat d'une requête XPath est une séquence de nœuds ou de valeurs. Qui dit séquence dit

- ordonné
- doublons possibles

Définir une feuille XSLT

- XSLT est un dialecte XML, défini par un schéma de l'espace de noms `http://www.w3.org/1999/XSL/Transform`
- version 1.0 qui date de novembre 99 et version 2.0 de janvier 2007.
- La racine d'un document XSLT est un élément `xsl:stylesheet`, ou bien `xsl:transform` (synonyme).

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/xhtml1/strict">
   ...
</xsl:stylesheet>
```

Appliquer une feuille XSLT à un document

1. Attacher la feuille de style au document, comme on le fait pour une feuille CSS.

   ```
   <?xml-stylesheet type="text/xsl" href="transfo1.xsl" ?>
   ```

2. Utiliser un programme qui applique la transformation au document pour produire un autre document

Règles de transformation

- une règle est définie avec un élément `xsl:template`.
- on y trouve :
  - un critère de sélection de nœuds dans le document source (attribut `match`). C'est une requête XPath 2.0 restreinte aux axes verticaux descendants.
  - parfois des paramètres
  - Un constructeur de la séquence résultat. Il est évalué pour produire une séquence d'items qui sont écrits dans l'arbre résultat.
- L'élément `xsl:apply-templates` permet d'appliquer explicitement une règle sur une séquence de nœuds définie par une sélection XPath.
- L'attribut `match` du template est nécessaire, sauf si le template a un attribut `name`. Dans ce cas, l'appel au template se fait par son nom : `<xsl:call-template name="..."/>`

Exemple

```
<xsl:template match="/">
  <html>
    <head><title>biblio</title></head>
    <body>
      <h1>Les Livres</h1>
      <xsl:apply-templates select = "library/book"/>
      <h1>Les Auteurs</h1>
      <xsl:apply-templates select = "library/author"/>
    </body>
  </html>
</xsl:template>
```

Application des règles

- L'attribut `select` de l'élément `apply-templates` permet de définir la séquence de nœuds sur lesquels il faut appliquer une règle. Sa valeur est une requête XPath, qu'on évalue à partir du nœud courant. Le résultat de cette requête XPath doit être une séquence de nœuds.
- En l'absence de `select`, l'instruction `apply-templates` traite tous les nœuds enfants du nœud courant, y compris les nœuds textes.
- L'idée est de traiter les nœuds de façon descendante, mais si le chemin XPath valeur de l'attribut select permet de remonter dans le document, il est possible de créer une transformation qui ne s'arrête pas (boucle).

Exemple de transformation qui boucle

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <a>
    <xsl:apply-templates select=".//book"/>
    </a>
  </xsl:template>

  <xsl:template match="book">
     <b>
     <xsl:apply-templates select="/"/>
     </b>
  </xsl:template>
</xsl:stylesheet>
```

En sortie : <a><b><a><b>...

Constructeur de séquence

- Séquence de nœuds frères dans la feuille de style (donc en particulier dans un template).
- Chacun d'eux est, au choix,
  1. une instruction XSLT
  2. un littéral élément
  3. un nœud texte
  4. une instruction d'extension

Constructeur de séquence

- Le constructeur de séquence est évalué pour chaque item de la séquence *s* des nœuds à traiter (dans un template, *s* est la séquence renvoyée par le `select`, ou bien la séquence des nœuds fils)
- Quand un constructeur de séquence est évalué, le processeur conserve donc une trace (appelée *focus*) de la séquence *s* des items en cours de traitement :
  - L'item contexte (souvent nœud contexte),
  - La position de l'item contexte dans la séquence *s*
  - La taille de la séquence *s*

Résolution des conflits

- Il est possible que plusieurs règles puissent s'appliquer sur l'item courant.
- Pour deux règles qui sont applicables :
  1. On compare les attributs `priority` des règles, s'ils existent
  2. Sinon, on calcule une priorité en fonction de la "sélectivité" du pattern de la règle.
  3. Si malgré tout, il subsiste plusieurs règles, alors le processeur peut déclencher une erreur, ou bien choisir la dernière règle dans l'ordre du document XSLT.

Les modes

- Les modes permettent de traiter plusieurs fois le même nœud du document source, en produisant des résultats différents.
- Grâce à l'attribut `mode` de `template` et `apply-template`, on peut définir pour quel mode une règle s'applique
- L'instruction `apply-templates` dispose aussi d'un attribut `mode`, qui permet de dire dans quel mode on veut appliquer une règle.
- Il existe un mode par défaut, qui n'a pas de nom, utilisé quand aucun nom de mode n'est donné explicitement

Règles prédéfinies

- S'il n'existe pas de règle qui s'applique sur un nœud sélectionné par un `apply-templates`, on évalue une règle prédéfinie (quelque soit le mode)
- Pour un nœud élément, on traite les nœuds fils, c'est donc pareil que d'évaluer `<apply-templates />`
- Pour un nœud texte ou attribut, on construit un nœud texte qui contient la valeur textuelle du nœud contexte.

Règles prédéfinies (2)

La feuille suivante (vide !) permet d'extraire tout le texte (pas les attributs) d'un document :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

Règles prédéfinies (2')

Elle est équivalente à :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="*">
    <!-- on selectionne les noeuds fils, (pas les attributs) -->
    <xsl:apply-templates select="child::node()"/>
  </xsl:template>
  <xsl:template match="text()">
    <xsl:value-of select="string(.)"/>
  </xsl:template>
</xsl:stylesheet>
```

Création de nœuds et de séquences

Quels sont les différents éléments XSLT qui permettent de créer des nœuds (éléments, attributs, textes) et des séquences de nœuds dans l'arbre résultat?

Création d'éléments

- On peut utiliser un littéral élément, en écrivant directement les balises que l'on veut en sortie
- On peut utiliser un élément `xsl:element`, ça permet de construire un élément dont le nom est calculé dynamiquement.

Création d'attributs

- On peut écrire l'attribut "en dur" dans un littéral élément
- On peut aussi utiliser des accolades autour de l'expression définissant la valeur de l'attribut ; $\{exp\}$ représente la valeur de l'évaluation de l'expression *exp*. Par exemple :

```
<xsl:template match="photograph">
<img src="/images/{href}" width="{size/@width}"/>
</xsl:template>
```

Création d'attributs

évalué sur :

```
<photograph>
  <href>headquarters.jpg</href>
  <size width="300"/>
</photograph>
```

a pour résultat :

```
<img src="/images/headquarters.jpg" width="300"/>
```

Création d'attributs

- On peut aussi utiliser un élément `xsl:attribute` pour définir un nœud attribut. Par exemple :

```
<xsl:attribute
  name="colors" select="'red', 'green', 'blue'"/>
```

a pour résultat : `colors = "red green blue"`

Création d'un nœud texte

- On peut écrire directement du texte dans le constructeur de séquence
- On peut utiliser l'élément `xsl:text`

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

- On peut générer un nœud texte dont le contenu n'est pas statique, grâce à l'élément `xsl:value-of`

```
<xsl:value-of select="./ville"/>
```

Création d'un nœud texte

Autre exemple, qui permet d'extraire tous les (contenus textuels des nœuds) titres d'un document :

```
<xsl:template match="/">
  <les-titres>
    <xsl:value-of select=".//titre"/>
    <!-- les titres sélectionnés sont
          séparés par des espaces -->
  </les-titres>
</xsl:template/>
```

Recopie d'un nœud (*shallow copy*)

- l'élément `xsl:copy` permet de copier l'item contexte.
- Si l'item contexte est une valeur atomique, l'instruction `xsl:copy` retourne cette valeur, et ne tient pas compte du constructeur de séquence.
- Si l'item contexte est un nœud élément ou un nœud document, l'instruction xsl:copy retourne un nœud de même type que le nœud contexte, et qui contient le résultat de l'évaluation du constructeur de séquence contenu dans l'élément xsl:copy. Donc, les attributs et le contenu du nœud contexte n'est pas recopié.
- Si l'item contexte est un autre type de nœud (nœud attribut, un nœud texte, ...), l'instruction xsl:copy retourne un nœud de même type que le nœud contexte, et qui contient la même valeur texte que le nœud contexte.

Copie récursive (*deep copy*)

- instruction `xsl:copy-of` avec un attribut `select` obligatoire.
- Les items de la séquence résultat du `select` sont traités de la manière suivante :
  1. Si l'item est un nœud élément ou un nœud document, alors on ajoute au résultat un nouveau nœud du même type et de même contenu (attributs, texte, sous-éléments ...) que l'item source
  2. Si l'item est un nœud d'un autre type (nœud attribut, un nœud texte, ...), alors la copie est la même qu'avec l'instruction `xsl:copy`
  3. Si l'item est de valeur atomique, sa valeur est ajoutée à la séquence résultat.

Exemple : transformation identité

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

équivalent à :

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

Répétition

- L'instruction `xsl:for-each` traite chaque item d'une séquence d'items et pour chacun évalue le constructeur de séquence
- Le résultat d'un `xsl:for-each` est la séquence concaténation des séquences obtenues pour chaque item.

Exemple

```
<xsl:template  match="/">>
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Clients</title></head>
    <body>
      <table>
        <tbody>
        <xsl:for-each select="clients/client">
          <tr>
            <th><xsl:apply-templates select="nom"/></th>
            <xsl:for-each select="cmde">
              <td><xsl:apply-templates/></td>
             </xsl:for-each>
           </tr>
          </xsl:for-each>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>
```

Transformation équivalente(1)

```
<xsl:template match="/">
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title>Clients</title>
    </head>
    <body>
      <table>
        <tbody>
          <xsl:apply-templates select="clients/client"/>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>
```

Transformation équivalente(2)

```
<xsl:template match="client">
  <tr>
    <th>
      <xsl:apply-templates select="nom"/>
    </th>
    <xsl:apply-templates select="cmde"/>
  </tr>
</xsl:template>

<xsl:template match="cmde">
  <td>
    <xsl:apply-templates/>
  </td>
</xsl:template>
```

Instructions conditionnelles

- Instruction `xsl:if` avec un attribut `test`.
  - Le test est une condition XPath.
  - Si le test est évalué à vrai, alors le constructeur de séquence à l'intérieur de l'instruction `if` est évalué ; sinon, la séquence vide est retournée.
- Instruction `xsl:choose`

```
<xsl:choose>
  <xsl:when test = "...">...</xsl:when>
  ...
  <xsl:when test = "...">...</xsl:when>
  <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>
```

Définition de variable

- élément `xsl:variable`, avec un attribut `name` obligatoire, et des attributs `select` et `as` optionnels.
- La valeur de la variable est
  - soit la valeur de l'évaluation de l'expression du `select`,
  - soit la valeur de l'évaluation du constructeur de séquence de l'élément `xsl:variable`.
- Si l'attribut `select` est présent, le constructeur de séquence doit être vide.
- L'attribut `as` donne le type de la variable.
  - S'il est absent, alors la variable prend le type de sa valeur.
  - S'il est présent alors la valeur de la variable est convertie en une valeur de ce type.
- Une variable sans attribut `as` et sans attribut `select` dont le contenu est non vide, est évaluée en un nœud document qui a pour fils l'évaluation du constructeur de séquence (cf exemples)

Exemples

`<xsl:variable name="i" as="xs:integer*" select="1 to 3"/>` a pour valeur la séquence d'entiers (1 2 3)

`<xsl:variable name="i" as="xs:integer" select="@size"/>` a pour valeur l'entier valeur de l'attribut `size`

`<xsl:variable name="doc"><c/></xsl:variable>` a pour valeur un nœud document qui a pour fils un élément vide `c`

Utilisation des variables

- Il suffit de préfixer le nom de la variable par $.
- Par exemple :

```
<xsl:variable name="n" select="2"/>
...
<xsl:value-of select="td[$n]"/>
```

Règles paramétrées

- l'élément `xsl:param` est utilisé pour définir un paramètre d'un template.
- La définition d'un paramètre ressemble fort à la définition d'une variable. L'attribut `select` ou le constructeur de séquence servant à donner une valeur par défaut.
- On a aussi un attribut `required`, faux par défaut, qui indique si le paramètre est obligatoire.
- Quand un paramètre est obligatoire, il ne peut pas avoir de valeur par défaut.
- Dans un `apply-templates` ou un `call-templates`, on donne une valeur au paramètre grâce à l'élément `with-param`.

Exemple (on reprend l'exemple des clients)

```
<xsl:template match="client">
  <!-- le parametre indique le nombre minimum de commandes obligatoire
  <xsl:param name="mini" required="yes"/>
  <!-- la variable qui contient le nombre de commandes du client couran
  <xsl:variable name="nb-cmdes" select="count(cmde)"/>
  <xsl:if test="$mini &lt;= $nb-cmdes">
  <tr>
    <th>
      <xsl:apply-templates select="nom"/>
    </th>
    <xsl:apply-templates select="cmde"/>
  </tr>
  </xsl:if>
</xsl:template>

<xsl:template match="cmde">
  <td><xsl:apply-templates/></td>
</xsl:template>
```

Exemple (2)

On ne veut conserver que les clients qui ont au moins 2 commandes

```
<xsl:template match="/">
  <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title>Clients</title>
    </head>
    <body>
      <table>
        <tbody>
          <xsl:apply-templates select="clients/client">
            <xsl:with-param name="mini" select="2"/>
          </xsl:apply-templates>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>
```

Conclusion

- Langage de transformation d'arbres
- Langage fonctionnel, récursif
- Utilisation de XPath
- Ce langage contient encore plein de fonctionnalités non présentées ici !
- gestion des espaces de noms
- modularité et importation de feuilles
- définition de fonctions
- paramétrage de la feuille de style (pas seulement des règles)
- génération de plusieurs arbres résultats, génération de résultats intermédiaires
- ...