XML - Semaine 4

Pierre Nerzic

février-mars 2022

Le cours de cette semaine présente :

- L'utilisation de feuilles de style CSS,
- XSLT, un outil pour transformer un document XML.

Le but initial était de fournir une visualisation esthétique pour des documents XML, mais ça a évolué vers la transformation XML vers d'autres formats.

Pierre Nerzic

Feuilles de styles CSS

Pierre Nerzic

Feuille CSS pour un document XML

Normalement, un document XML ne peut pas être affiché dans un navigateur internet car ce n'est pas du HTML. Mais on peut lui associer une feuille de style CSS pour spécifier l'affichage des éléments, ce qui permet de les voir correctement.

Il faut ajouter une balise spéciale qui spécifie la feuille de style CSS à utiliser. Celle-ci définit l'apparence de chacun des éléments du document :

```
ELEMENT {
    ATTRIBUT DE STYLE: VALEUR;
    ...
}
```

NB: cette solution n'est pas recommandée, car incapable d'afficher des documents un peu complexes qu'il faut remanier.

Exemple de document XML

Voici le document `albums.xml` utilisé pour l'exemple : 

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="albums.css"?>
<albums>
  <album numero="1" serie="Tintin">
    <titre>Tintin au pays des Soviets</titre>
    <date>
      <mois>septembre</mois><annee>1930</annee>
    </date>
  </album>
  ...
</albums>
```

La balise `<?xml-stylesheet ...?>` indique la feuille CSS.

Exemple de feuille de style CSS

Voici un extrait de la feuille de style `albums.css` (très simpliste)

```
album {
    display: block;
}
album titre {
    display: inline-block;
    width: 500px;
    font-size: 16pt;
}
album mois, album annee {
    display: inline-block;
    width: 120px;
}
```

Le problème principal est que ça oblige à garder la structure du document XML. On ne peut pas réorganiser les éléments.

XSLT

Pierre Nerzic

Présentation

XSL est une norme pour définir des feuilles de styles en XML. XSLT est un langage permettant de transformer un document XML à l'aide d'expressions XPath écrites dans une feuille XSL. XSLT est indispensable quand la structuration XML ne correspond pas à celle de HTML.

Le premier but de XSLT est de permettre l'affichage d'un document XML dans un navigateur. Dans ce cas, la transformation consiste à générer du code HTML en fonction de ce qu'on trouve dans le document XML.

Par exemple, on souhaite afficher la liste des albums de Tintin dans un tableau HTML, en regroupant le mois et l'année dans une seule chaîne. Ces deux informations sont dans des éléments séparés. Ce n'est donc pas possible avec des styles CSS.

Pierre Nerzic

Exemple de feuille de style

Voici comment s'écrit la transformation en XSLT. On mélange des balises HTML avec des balises XSL :

```
<xsl:template match="/albums">
  <html><body>
  <h2>Albums de Tintin</h2>
  <table border="1">
    <xsl:for-each select="album[@serie='Tintin']">
    <tr>
      <td><xsl:value-of select="titre"/></td>
      <td><xsl:value-of select="date/annee"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body></html>
</xsl:template>
```


Pierre Nerzic

Entête d'une feuille XSLT

Une feuille XSLT doit commencer par ces lignes :

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

  ...TEMPLATES, voir plus loin...

</xsl:stylesheet>
```

`<xsl:output>` indique le type du document en sortie : `xml`, `html` ou `text`. On peut rajouter les attributs `version="1.0"`, `encoding="UTF-8"` et `indent="yes"`.

Transformation d'un document

Pour attribuer une feuille de style XSLT à un document XML, il faut mettre ceci avant la racine du document :

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

Transformation du document XML :

- dans le navigateur, si `<xsl:output method="html"/>`
- en ligne de commande :

```
xsltproc feuille.xsl document.xml
```

Ça affiche le document transformé par la feuille. On peut rediriger la sortie vers un fichier.

Cette commande est capable d'utiliser les valeurs par défaut des attributs définis dans la DTD s'il y en a une.

Principe général

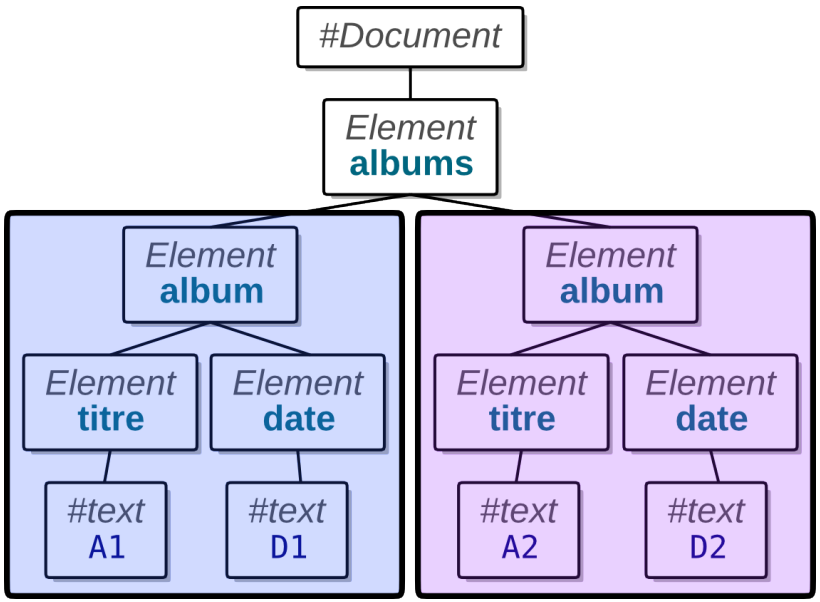
XSLT prend un fichier XML en entrée, ainsi qu'une feuille XSL qui décrit les traitements à appliquer. En sortie, on récupère un document XML, HTML ou texte.

La feuille XSL contient des **patrons** (*templates*). Ce sont des sortes de couples (expression, contenu) : les parties du document XML qui correspondent à l'expression sont remplacées par le contenu.

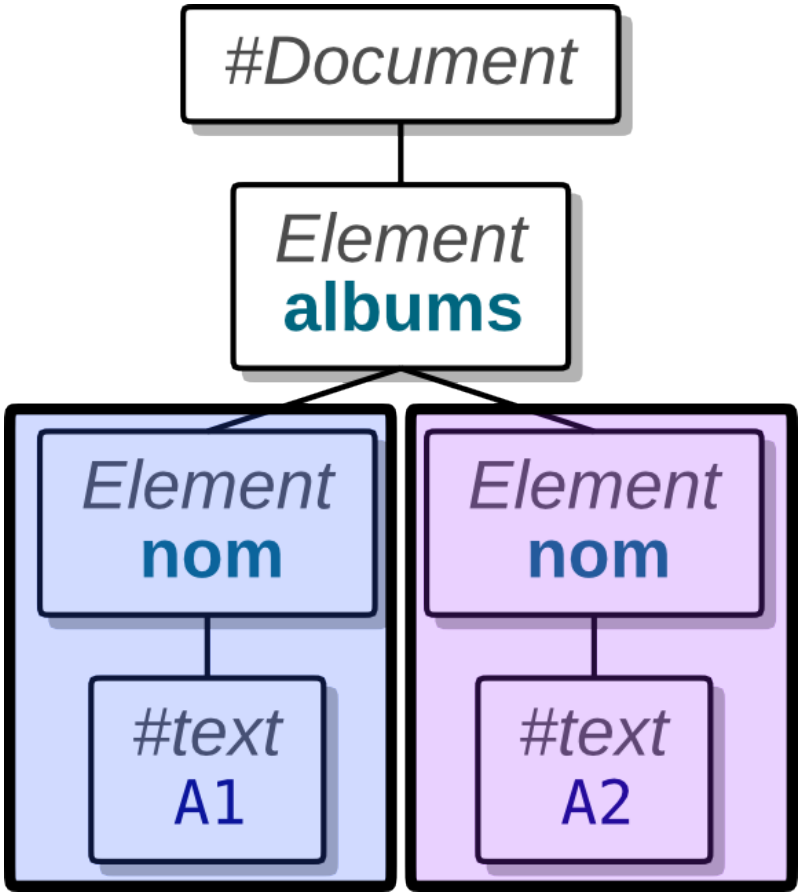
- L'expression est écrite en XPath et porte sur l'arbre XML d'entrée. Elle indique quels sont les nœuds (élément, attribut, document entier...) à remplacer.
- Le contenu sont des éléments et/ou des textes qui sont mis à la place des nœuds sélectionnés.
- Les nœuds XML non sélectionnés restent inchangés.

Exemple de traitement

Voici le document d'origine. On lui applique un patron dont l'expression est `/albums/album`.



Les éléments sélectionnés ont été remplacés.



Pierre Nerzic

Patrons

Voici la forme générale d'un patron :

```
<xsl:template match="EXPR XPATH">
    CONTENU
</xsl:template>
```

- `EXPR XPATH` est une expression XPath, par exemple `/` pour désigner le document entier.
- `CONTENU` est un mélange d'éléments XSL et d'autres choses (éléments et textes) qui doivent respecter la syntaxe XML.

Normalement, l'expression doit être absolue, c'est à dire commencer par un `/`. Inversement, le contenu, s'il contient des expressions XPATH doit être relatif à l'expression.

Contenu des patrons

- Quand on veut produire du texte tel quel :

```
<xsl:text>texte...</xsl:text>
```

On peut aussi mettre ce texte sans la balise `<xsl:text>`, mais il sera formatté par XSLT.

- Quand on veut mettre la valeur d'une expression XPath :

```
<xsl:value-of select="EXPR XPATH"/>
```

Cette expression doit en principe être relative à celle du template et sa valeur sera mise dans le document en sortie.

Pierre Nerzic

Exemple de patron

Le patron de cette feuille remplace les éléments <date> du document albums.xml par <sortie>mois année</sortie> : 📥

```
<xsl:template match="/albums/album/date">
  <sortie>
    <xsl:value-of select="mois"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="annee"/>
  </sortie>
</xsl:template>
```

Notez l'espace dans l'élément <xsl:text>. Sans cette syntaxe, l'espace serait ignoré et absent de la sortie.

NB: ce patron ne peut pas être utilisé seul car il ne génère pas de racine en sortie : les autres éléments XML sortent aussi.

Créer des éléments et des attributs

Rajouter un élément est très facile. Il suffit de l'écrire tel quel, mais on peut aussi utiliser la balise `<xsl:element name="nom">` . Voici comment lui rajouter un attribut :

```
<xsl:template match="album">
  <xsl:element name="ouvrage">
    <xsl:attribute name="annee">
      <xsl:value-of select="date/annee"/>
    </xsl:attribute>
    <xsl:value-of select="titre"/>
  </xsl:element>
</xsl:template>
```

Ce exemple génère `<ouvrage annee="ANNEE">TITRE</ouvrage>` à partir des éléments `<album>` du document d'entrée.

value-of et copy-of

Ces directives permettent d'insérer une partie du contenu :

- `<xsl:value-of select="XPATH"/>` insère la valeur (texte) de l'expression XPATH
- `<xsl:copy-of select="XPATH"/>` insère les nœuds sélectionnés par l'expression XPATH

Exemple :

```
<xsl:template match="//album">
  <BD>
    <xsl:attribute name="nom">
      <xsl:value-of select="titre"/>
    </xsl:attribute>
    <xsl:copy-of select="date/annee"/>
  </BD>
</xsl:template>
```


Pierre Nerzic

Fonctions utiles

L'expression XPATH dans `<xsl:value-of select="XPATH"/>` peut employer des fonctions comme :

- `count(`*expression*`)` retourne le nombre de nœuds sélectionnés
- `sum(`*expression*`)` retourne la somme des nombres sélectionnés
- `string-length(`*chaîne*`)` : longueur de la chaîne
- `substring(`*chaîne*`,`*pos*`,`*lng*`)` : extrait lng caractères à partir de pos (≥ 1)
- `starts-with(`*chaîne*`,`*début*`)` vrai si la chaîne commence par début
- `end-with(`*chaîne*`,`*fin*`)` vrai si la chaîne finit par début
- `contains(`*chaîne*`,`*partie*`)` vrai si partie est dans la chaîne
- `concat(`*chaîne1*`,`*chaîne2*`...)` retourne les chaînes groupées

La liste complète est sur cette page.

Structures de contrôle XSLT

Pierre Nerzic

Variables

Bien que ça ne soit pas trop la philosophie du langage XSLT, il est possible de stocker des données dans des variables puis les utiliser plus tard. Les données sont n'importe quelle expression XPath et lors de l'utilisation, on peut s'appuyer dessus pour construire une expression plus complexe.

```
<xsl:variable name="NOM" select="VALEUR"/>
...
<xsl:value-of select="$NOM"/>
```

Exemple :

```
<xsl:variable name="MonAlbumPrefere"
        select="//album[date/annee=1949]"/>
...
<xsl:value-of select="$MonAlbumPrefere/titre"/>
```


Pierre Nerzic

Conditionnelles

Il est possible de générer du contenu conditionnel à l'aide de la structure XSL suivante :

```
<xsl:if test="CONDITION">
    CONTENU
</xsl:if>
```

Les conditions sont écrites à la manière XPath, voir le cours 3.

La condition doit impérativement être correcte du point de vue XML, c'est à dire que les caractères < " > doivent être remplacés par leurs entités `<` `&quot;` `>` et il faut mettre des espaces entre les valeurs et les opérateurs.

NB: il n'y a pas de *else*, mais on peut écrire deux `<xsl:if>` avec des conditions opposées, et voir `<xsl:choose>` plus loin.

Exemple de contenu conditionnel

Le patron suivant ne génère un élément <date> que si l'année est inférieure à 1940. Remarquez le signe < écrit sous forme d'entité et les espaces :

```
<xsl:template match="/albums/album/date">
  <xsl:if test="annee &lt; 1940">
      <date>
        <xsl:value-of select="annee"/>
      </date>
  </xsl:if>
</xsl:template>
```

NB: ce patron ne gère pas la racine du document, donc tous les éléments autre que <date> sortent aussi.

Comment faire un *else* ?

Il existe une autre structure permettant toutes les combinaisons

```
<xsl:choose>
  <xsl:when test="condition1">
    ...contenu si condition1 est vraie
  </xsl:when>
  <xsl:when test="condition2">
    ...contenu si condition2 est vraie
  </xsl:when>
  ...
  <xsl:otherwise>
    ...contenu si aucune condition n'est vraie
  </xsl:otherwise>
</xsl:choose>
```

Seule la première condition vraie est appliquée.

Pierre Nerzic

Remarques sur les tests

Pour tester si un élément est présent et/ou possède un contenu :

- `<xsl:if test="element">` est vrai si le nœud courant a un enfant appelé `<element>`, qu'il soit vide ou non
- `<xsl:if test="not(element)">` est vrai si le nœud courant n'a pas d'enfant appelé `<element>`
- `<xsl:if test="element != ''">` est vrai si le nœud courant a un enfant non-vide appelé `<element>`
- `<xsl:if test="element = ''">` est vrai si le nœud courant a un enfant vide appelé `<element/>`

Boucle sur les nœuds enfant

Lorsqu'on veut traiter tous les enfants d'un élément, on emploie l'élément `xsl:for-each` :

```
<xsl:for-each select="XPATH">
    CONTENU
</xsl:for-each>
```

Le document de sortie contiendra autant d'exemplaires du `CONTENU` que de nœuds sélectionnés par le chemin `XPATH`. Ce contenu est paramétré par le nœud courant de la boucle : `current()`, mais on fait généralement directement référence aux enfants du nœud désigné par le select avec des expressions relatives.

Exemple de patron de type *boucle*

Le patron de cette feuille remanie le document `albums.xml` en `<liste><nom>titre</nom>...</liste>` :

```
<xsl:template match="/albums">
  <liste>
    <xsl:for-each select="album">
      <nom><xsl:value-of select="titre"/></nom>
    </xsl:for-each>
  </liste>
</xsl:template>
```

Ce patron sélectionne l'élément `<albums>` et le remplace par `<liste>`, contenant des éléments `<nom>`, un par `<album>` du document d'entrée ; chaque élément `<nom>` contient le titre de l'album. Le `titre` à l'intérieur de la boucle est relatif à l'`album` courant.

Exemple de patron (suite)

Le même exemple peut être écrit autrement car il n'y a qu'un seul élément `<titre>` dans un album :

```
<xsl:template match="/">
  <liste>
    <xsl:for-each select="albums/album/titre">
      <nom><xsl:value-of select="current()"/></nom>
    </xsl:for-each>
  </liste>
</xsl:template>
```

Notez l'usage de la fonction `current()` pour désigner l'élément `<titre>`. Elle retourne le même résultat que `.` mais permet d'être employée dans une expression plus complexe.

Pierre Nerzic

Tri des itérations

La structure `xsl:for-each` itère sur une liste de nœuds du document. Cette liste est dans l'ordre du document, mais peut être triée selon un autre critère, voir cette page :

```
<xsl:template match="/">
    <xsl:for-each select="albums/album">
      <xsl:sort select="date/annee" data-type="number"/>
      <nom><xsl:value-of select="titre"/></nom>
    </xsl:for-each>
</xsl:template>
```

- type des données : `data-type="text"` ou `data-type="number"`
- sens du tri : `order="ascending"` ou `order="descending"`

Pierre Nerzic

Traitement d'un document complexe

Pour traiter un document complet, il est fréquent de faire appel à plusieurs patrons : un pour le document entier / ou /racine et des patrons pour ses éléments. Dans ce cas, il faut explicitement signaler au patron racine d'appeler les patrons des éléments :

```
<xsl:template match="XPATH1">
    <xsl:apply-templates select="XPATH2"/>
</xsl:template>
```

Le patron qui traite XPATH1 lancera le patron qui traite XPATH2 pour les éléments qui sont sélectionnés par cette expression.

Pierre Nerzic

Exemple de patrons imbriqués

Ces patrons extraient les albums de Tintin sous la forme `<tintin><nom>titre</nom>...</tintin>` :

```
<xsl:template match="/">
  <tintin>
    <xsl:apply-templates
            select="/albums/album[@serie='Tintin']"/>
  </tintin>
</xsl:template>

<xsl:template match="album">
  <nom><xsl:value-of select="titre"/></nom>
</xsl:template>
```

C'est un peu plus simple qu'une boucle.

Pierre Nerzic

XSL 1.0 vs XSL 2.0

Pierre Nerzic

Remarque sur les boucles

En XSL 2.0, il est possible d'itérer facilement sur des choses assez complexes. Par exemple :

```
<xsl:template match="/">
  <calendrier>
    <xsl:for-each select="distinct-values(//mois)">
      <mois>
        <xsl:value-of select="current()"/>
        <xsl:text> : </xsl:text>
        <xsl:value-of
            select="count(//album[date/mois=current()])"/>
        <xsl:text> albums</xsl:text>
      </mois>
    </xsl:for-each>
  </calendrier>
</xsl:template>
```

XSL 2.0

Hélas, XSL 2.0 utilise XPath 2.0 et n'est pas implémenté par xsltproc, ni de nombreux outils dont les navigateurs.

On doit donc trouver des rustines. Pour l'exemple précédent, on fait appel à une astuce proposée par Steve Muench. Elle consiste à créer un index, c'est à dire des couples (valeur, liste des éléments ayant cette valeur). Par exemple, les couples (mois, liste des albums sortis ce mois).

Index XSL

Un index, en XSL, est un tableau à deux colonnes : clé, valeurs. La clé est une chaîne de caractère tirée du fichier XML et les valeurs peuvent être des nœuds quelconques (éléments, attributs, textes), tous associés à la même valeur.

Cette directive demande à XLST de mémoriser un index appelé *NOM* et contenant les valeurs associées à la clé :

```
<xsl:key name="NOM" match="VALEURS" use="CLÉ" />
```

La clé est une partie de la valeur. Par exemple, la valeur est un élément `<mois>` et la clé est le nom du mois :

```
<xsl:key name="index" match="//mois" use="text()" />
```


Pierre Nerzic

Index en XSL, suite

L'instruction `<xsl:key ...>` précédente construit un tableau :

clé	éléments
janvier	`<mois de l'album numéro="20">,`
"	`<mois de l'album numéro="21">,...`
mars	`<mois de l'album numéro="31">`
avril	`<mois de l'album numéro="15">,...`

Il faut bien comprendre que le même nom de mois est associé à plusieurs éléments distincts dans le document. Certains mois ne sont pas dans l'index car aucun album ne leur correspond.

La fonction `key('index', clé)` retourne la liste des élements `<mois>` associés à la clé.

Identifiants internes d'éléments

Une autre partie de l'astuce repose sur le fait que chaque élément possède un identifiant interne qui le distingue des autres. La fonction `generate-id()` (mal nommée, car l'identifiant n'est pas généré, il existe de toutes façons) permet de connaître cet identifiant pour l'élément courant.

La fonction `generate-id(elements)` permet d'obtenir l'identifiant des éléments fournis en paramètre.

L'astuce repose sur le fait qu'on va demander l'identifiant du premier élément indexé par le mois. On ne fera une boucle que pour cet élément, et pas pour les autres. Ainsi, il n'y aura qu'un seul exemplaire de chaque mois en sortie.

Hélas, on ne peut pas boucler sur les clés de l'index.

Groupement de Steve Muench

Ensuite, on fait une boucle sur les clés comme ceci :

```
<xsl:template match="/">
  <calendrier>
    <xsl:for-each
        select="//mois[
            generate-id()=generate-id(key('index',text())[1])]">
      <mois>
        <xsl:value-of select="current()"/>
        <xsl:text> : </xsl:text>
        <xsl:value-of
            select="count(//album[date/mois=current()])"/>
        <xsl:text> albums</xsl:text>
      </mois>
    </xsl:for-each>
  </calendrier>
</xsl:template>
```


Pierre Nerzic