Transformations XML : XPath et XSL-XSLT

Ingénierie Documentaire
http://doc.crzt.fr

Stéphane Crozat

14 septembre 2016

Table des matières

Introduction à XSL-XSLT

I

1. Un langage pour publier les documents XML

XML lorsqu'il est utilisé pour définir des formats documentaire métier est un format de *représentation* de l'information, et non un format de *publication* (de présentation) de cette information : donc un tel fichier XML *n'est pas utilisable tel que par un lecteur*.

XML ne peut donc être utilisé pour des langages abstrait que si l'on est capable de transformer les documents sources en document publiés lisibles grâce à un format de présentation : HTML par exemple dans le cas de publication Web, ou PDF pour l'impression.

2. Définition de XSL-XSLT

Définition : XSL-XSLT

XSL-XSLT est une partie du standard W3C XSL qui a trait à la transformation des documents XML (l'autre partie étant XSL-FO).

XSL-XSLT est un langage de programmation déclaratif écrit en XML (un programme XSL-XSLT est un document XML).

- XSL-XSLT est langage de manipulation de document XML (fondé sur XPath et sur le modèle arborescent de représentation des documents XML)
- XSl-XSLT est utilisé pour transformer un document XML source dans un autre format, typiquement HTML, mais aussi tout autre format codé sur des caractères dont la syntaxe est connue.
- XSL-XSLT est aussi utilisé pour faire des changements de schéma XML (export d'un XML vers un autre XML structuré différemment) par exemple pour échanger des données selon un standard.

Remarque : XSL-XSLT, XSL-FO, XSLT, XSL, FO

On parle souvent (par simplification) de XSL ou de XSLT pour désigner XSL-XSLT et de FO pour désigner XSL-FO.

XSL utilisé seul désigne donc par convention XSL-XST (et non XSL-FO).

3. Principe de XSL-XSLT

XSL-XSLT fonctionne selon le principe suivant :

1. Il prend en entrée un fichier XML bien formé
2. Il livre en sortie un fichier texte (XML, HTML ou texte sans balise)

Algorithme

L'algorithme général de XSL-XSLT est :

1. Il sélectionne (*match*) les éléments XML du fichier source.
2. Pour chaque élément reconnu il génère une sortie sur le fichier cible.

Notion de règle

Un programme XSL-XSLT est composé d'une succession de règles.

Chaque règle est indépendante des autres et à en charge de sélectionner un élément dans la source et d'effectuer une écriture dans la cible.

Exemple : Application d'une règle XSL-XSLT

Sources :

```
1 <a>
2   <b/><b/><c/>
3 </a>
```

Règle XSL-XSLT :

```
1 <xsl:template match="/a/b">
2     BONJOUR
3 </xsl:template>
```

Résultat :

```
1 BONJOUR BONJOUR
```

Programmation XSL-XSLT

II

1. L'arbre du document XML

Il est possible de représenter un document XML sous forme d'arbre, tel que :

- L'arbre possède une racine / qui a comme fils l'élément racine
- l'élément racine est l'élément du document XML qui contient tous les autres
- chaque nœud a comme fils les éléments et le texte qu'il contient, ainsi que ses attributs.

Exemple : Fichier XML

```
<?xml version="1.0" encoding="UTF-8"?>
<document modele="ULCoursGeneral" code="BP-Incendie3_S1_E2_UL1">
  <entete>
    <identification>
      <titre>L'assurance de la responsabilité de voisinage</titre>
      <date>21/02/01</date>
      <auteur>AEA</auteur>
      <version>1.00</version>
    </identification>
  </entete>
  <corps>
    <contenu>
      <paragraphe>Cette garantie est appelée : recours des voisins et des tiers.</paragraphe>
      <remarque>
        <paragraphe>L'image suivante <ressource URIsrc="img07.jpg" titre="Recours des voisins et des tiers" type="image"/> montre la garantie.</paragraphe>
      </remarque>
    </contenu>
  </corps>
</document>
```

Exemple : Arbre XML

```
/
  |document
      |@modele = "ULCoursGeneral"
      |@code = "BP-Incendie3_S1_E2_UL1"
      |entete
          |identification
              |titre
                  |text() = "L'assurance de ..."
```

```
9                   |date
10                      |text() = "21/02/01"
11                  |auteur
12                      |text() = "AEA"
13                  |version
14                      |text() = "1.00"
15              |corps
16                  |contenu
17                      |paragraphe
18                          |text() = "Cette garantie ..."
19                      |remarque
20                      |paragraphe
21                          |text() = "L'image suivante"
22                          |ressource
23                              |@URIsrc = "img07.jpg"
24                              |@titre = "Recours des voisins..."
25                              |@type = "image"
26                          |text() = "montre la garantie."
```

Remarque : Ordre des nœuds

L'ensemble des nœuds de l'arbre d'un document est muni d'un ordre, qui est celui de l'ordre dans le document XML sérialisé.

2. Introduction à XPath

Définition : Expression XPath

XPath est un langage d'expressions permettant de pointer sur n'importe quel élément d'un arbre XML depuis n'importe quel autre élément de l'arbre.

- Une expression XPath peut-être *absolue* (sa résolution est *indépendante* d'un contexte ou nœud courant : elle commence dans ce cas par `/`.
- Une expression XPath peut-être *relative* (sa résolution est *dépendante* d'un contexte ou nœud courant : elle ne commence dans ce cas pas par `/`, elle peut commencer par `./` (syntaxe développée).

Fondamental

Une expression XPath renvoie

- un *node-set*, ou ensemble de nœuds, c'est à dire un sous-arbre de l'arbre du document
- une chaîne de caractères
- un booléen
- un réel

Exemple : Exemples d'expressions XPath

```
1 /document/entete/identification/titre
2 /document/@modele
3 corps//contenu
4 contenu/*
```

```
5 contenu/remarque[1]
6 ../paragraphe
7 @type
```

Complément : Types de nœuds XPath

- root nodes
- element nodes
- text nodes
- attribute nodes
- namespace nodes
- processing instruction nodes
- comment nodes

http://www.w3.org/TR/xpath/#data-model

Complément

Pour une introduction à XPath : Brillant07 * pp.123-129

Complément : Voir aussi

L'arbre du document XML (cf. p.6)

Syntaxe XPath (cf. p.8)

3. Syntaxe XPath

Définition : Pas de localisation

Une expression XPath est composée de plusieurs *pas de localisation* successifs séparés par des `/`.

Un pas de localisation est caractérisé par :

- un axe,
- un test de nœud,
- un prédicat (éventuellement aucun ou plusieurs).

Syntaxe : Expression XPath

```
1 pas-de-localisation-1/.../pas-de-localisation-N
```

Syntaxe : Pas de localisation

```
1 axe::test-de-nœud[prédicat]
```

Syntaxe : Axes et tests de nœuds

- `/` : sélectionne la racine (expression absolue)
- `.` : sélectionne le nœud courant (expression relative)
- `child::x` ou `x` ou `./x` : sélectionne les éléments fils "x"

- `child::x/child::y` ou `./x/y` ou `x/y` : sélectionne les éléments y fils de l'élément fils x (petits fils)
- `attribute::a` ou `./@a` ou `@a` : sélectionne l'attribut "a" du nœud courant
- `child::*` ou `./*` ou `*` : sélectionne tous les éléments fils
- `attribute::*` ou `./*@` ou `*@`: sélectionne tous les attributs
- `child::text()` ou `./text()` ou `text()` : sélectionne les nœuds de type texte
- `parent::x` ou `../x` : sélectionne le père "x" (`ancestor::x` sélectionne les ancêtres "x")
- `descendant::x` ou `.//x` : sélectionne tous les descendants "x" (enfants, petits enfants, etc.)
- `preceding::x`, `following::x` : sélectionne les nœuds précédents ou suivants dans le document (ordre), *à l'exclusion des ancêtres* (et des attributs)
- `preceding-sibling::x`, `following-sibling::x` : similaire à `preceding` et `following`, mais pour les nœuds de même niveau uniquement ("fratrie")

⚠ *Attention : * , @*, text()*

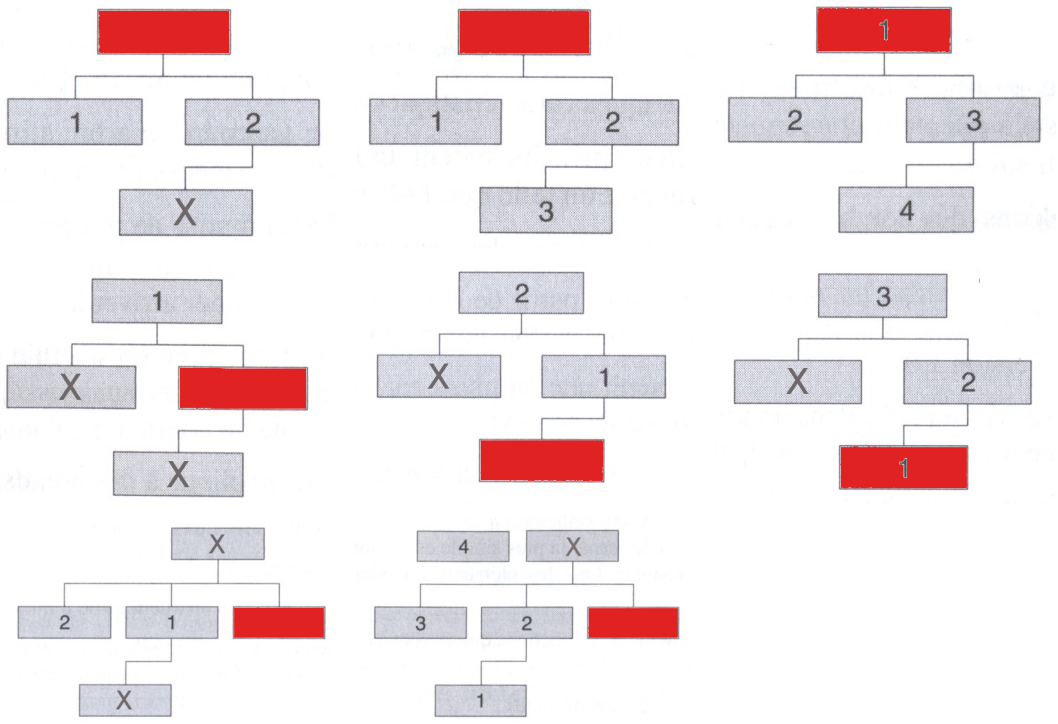
`*` sélectionne les nœuds de type élément, mais pas les attributs (sélectionnés par `@*`), ni les nœuds de type texte sélectionnés par `text()`.

Complément : Axes

Pour bien visualiser le fonctionnement des axes, voir XML : Cours et exercices *, p.124 (Figures 5-1 et 5-2).

- Carré rouge : Le point de départ
- Carré gris numéroté : Nœud sélectionné (avec son ordre de sélection)
- Carré gris non numéroté ; Nœud non sélectionné

Illustration des axes XPath (Brillant 07)



Exemple : Prédicats

- `x[1]`, `x[2]`, etc. : sélectionne le premier x, le second x, etc.
- `x[last()]` : sélectionne le dernier x
- `x[@a='valeur']` : sélectionne les x tels que leur attribut a est égal à "valeur"
- `x[y=1]` : sélectionne les x tels qu'ils ont un élément fils y égal à 1
- `x[@a='v1' and @b='v2']` : sélectionne tous les x tels que leurs attributs a et b sont respectivement égaux à v1 et v2
- `x[y or z]` : sélectionne tous les x tels qu'ils possède un élément fils y ou z

Exemple : Union

Il est possible d'unifier deux expressions XPath en utilisant `|` :

- `x | y` : sélectionne les éléments x *et* les éléments y
- `x[y] | x[z]` : sélectionne les éléments x tels qu'ils contiennent un y ou un z (équivalent ici à `x[y or z]`)

Complément : current()

La fonction `current()` permet de renvoyer le nœud courant dans le contexte d'une exécution XSLT.

Cela permet de différencier :

- `elem1[@att1=@att2]` : Les elem1 qui ont deux attributs att1 et att2 égaux
- et `elem1[@att1=current()/@att2]` : Les elem1 qui ont un attribut att1 égal à l'attribut att2 du nœud courant

Complément : Liste des fonctions XPath

http://fr.selfhtml.org/xml/representation/fonctionsxpath.htm

4. Syntaxe générale XSL-XSLT

Syntaxe : Structure générale d'un programme XSl-XSLT

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3     <xsl:output method="html" indent="yes" encoding="iso-8859-1"/>
4     <xsl:template match="XPATH">
5         ...
6     </xsl:template>
7     <xsl:template match="XPATH">
8         ...
9     </xsl:template>
10    ...
11 </xsl:stylesheet>
```

Syntaxe : Règle (template)

```
<xsl:template match="XPATH">
    Instruction XSLT et/ou génération de texte sur la sortie
</xsl:template>
```

5. Principales instructions XSLT

Fondamental : Les deux instructions de base

- `<xsl:apply-templates select="XPATH"/>` : Relance les règles du programme sur le sous-arbre pointé par le `select` (fondement de la récursivité)
- `<xsl:value-of select="XPATH"/>` : Génère le *texte* contenu dans le nœud ou attribut pointé par le `select`
  `{XPATH}` : Génère le texte contenu dans le nœud ou attribut pointé par le XPath entre accolades (alternative à `xsl:value-of` à utiliser dans la génération d'attributs exclusivement, par exemple : `<img src="{@URIsrc}">`

Autres instructions courantes

- `<xsl:copy-of select="XPATH"/>` : Génère le *sous-arbre* pointé par le `select`
- `<xsl:if test="XPATH">...</xsl:if>` : Ne s'exécute que si `test` est vrai
- `<xsl:for-each select="XPATH">...</xsl:for-each>` : Exécute pour chaque sous-arbre renvoyé par le `select`

Complément : Extension XPath "document()"

`document(chemin-accès)` où chemin d'accès permet d'accéder à un fichier sur le disque.

Par exemple :

- `document("c:\monfichier.xml")//x` : Tous les éléments `x` présents dans le fichier `monfichier.xml`.
- `document(child::source)/*` : La racine du document XML pointé par l'expression XPath `child::source`.

Complément : Déclaration explicite sur la cible

- `<xsl:element name="">contenu</xsl:element>`
- `<xsl:attribute name="">valeur</xsl:attribute>`
- `<xsl:text>chaîne de caractère</xsl:text`

Complément : Référence

http://fr.selfhtml.org/xml/representation/elementsxslt.htm

Complément : Références synthétiques

https://developer.mozilla.org/fr/XSLT/Éléments

http://www.daniel-lemire.com/inf6450/mod3/xsltenbref.xhtml

http://personnel.univ-reunion.fr/fred/Enseignement/XML/intro-xslt.html

6. Fonctionnement des programmes XSL-XSLT

La récursivité

```
<xsl:template match="XPATH-1">
      ...
      <xsl:apply-templates select="XPATH-2"/>
       ...
</xsl:template>
<xsl:template match="XPATH-2">
      ...
      <xsl:apply-templates select="XPATH-3"/>
       ...
</xsl:template>
<xsl:template match="XPATH-3">
      ...
      <xsl:value-of select="XPATH-4"/>
       ...
</xsl:template>
```

Règles par défaut

XSLT comporte trois règles par défaut qui sont appliquées quand aucune règle du programme n'est sélectionnée.

1. La première règle s'applique à la racine et à tous les éléments et déclenche un appel récursif sur tous les fils du nœud courant :
   `<xsl:template match="*|/"> <xsl:apply-templates/> </xsl:template>`
2. La deuxième règle s'applique aux nœuds texte et aux attributs et insère le résultat textuel de ces nœuds dans le document résultat :
   `<xsl:template match="text()|@*"> <xsl:value-of select="."/> </xsl:template>`
3. La troisième règle s'applique aux commentaires et aux instructions de traitement et les ignore :
   `<xsl:template match="processing-instruction() | comment()" />`

Exemple : Le programme XSLT minimal

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

Attention

Il faut donc prendre garde aux règles par défaut, sources d'effets de bord.

Conseil

Veiller à bien traiter tous les cas.

Méthode

Court-circuiter les règles par défaut, en générant des messages d'erreurs sur la sortie.

Complément : Mode, priorité, règles nommées

- `priority` : permet de surcharger les règles de priorité par défaut
- `mode` : permet de définir plusieurs règles pour une même prémisse et de choisir explicitement celle que l'on veut au moment de l'appel
- `name` : permet de définir des *templates* nommés, qui sont appelés explicitement (comme des fonctions) par l'instruction `<call-template>` et non plus seulement par le moteur de récursivité.

Complément : Priorité entre les règles

- Si un attribut `priority` est défini sur la règle, la règle est prioritaire, l'attribut le plus élevé est prioritaire sur le moins élevé.
- Sinon, c'est celle dont le XPath est le plus *spécifique* d'abord (expression d'un axe, d'un prédicat)
  Exemple : *http://www.zvon.org/xxl/XSLTutorial/Output_fre/example69_ch2.html*

7. Exemple : Un programme XSLT pour générer du HTML

Exemple : Fichier XML source

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <document titre="XSLT">
3
4 <!--Première division-->
5
6 <div titre="XSLT : Un besoin">
7 <paragraphe>XML est un format de <important>représentation</important> de l'information.
  </paragraphe>
8 <paragraphe>XML n'est pas un format de présentation.</paragraphe>
9 </div>
10
11 <!--Seconde division-->
12
13 <div titre="XSLT : Un langage">
14 <paragraphe>XSLT est un langage de <important>manipulation</important> de documents XML.
  </paragraphe>
15 <paragraphe>XSLT est utilisé pour exporter une source XML sous un autre format, par exemple HTML.
  </paragraphe>
16 </div>
17 </document>
```

Exemple : Fichier HTML cible souhaité

```
<HTML>

<!--Head-->

<HEAD
<TITLE>XSLT</TITLE>
<META content="text/html" charset="iso-8859-1"/>
</HEAD>

<!--Body-->

<BODY>
<H1>XSLT : Un besoin</H1>
<P>XML est un format de <B>représentation</B> de l'information.</P>
<P>XML n'est pas un format de présentation.</P>
<H1>XSLT : Un langage</H1>
<P>XSLT est un langage de <B>manipulation</B> de documents XML.</P>
<P>XSLT est utilisé pour exporter une source XML sous un autre format, par exemple HTML</P>
</BODY>
</HTML>
```

Exemple : Programme XSLT permettant la transformation

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

<!--1ère règle-->

  <xsl:template match="document">
    <HTML>
      <HEAD
        <TITLE><xsl:value-of select="@titre"/></TITLE>
        <META content="text/html" charset="iso-8859-1"/>
      </HEAD>
      <BODY>
        <xsl:apply-templates/>
      </BODY>
    </HTML>
  </xsl:template>

<!--2nde règle-->

  <xsl:template match="div">
    <H1><xsl:value-of select="@titre"/></H1>
    <xsl:apply-templates/>
  </xsl:template>

<!--3ème règle-->

  <xsl:template match="paragraphe">
    <P><xsl:apply-templates/></P>
  </xsl:template>

```

```
<!--4ème règle-->

  <xsl:template match="important">
    <B><xsl:value-of select="."/></B>
  </xsl:template>
</xsl:stylesheet>
```

8. XSLT et namespaces

Exemple : Le problème

Soit le fichier XML et un programme XSLT associé. La transformation ne fonctionne pas comme prévue, car la règle essaye de matcher `document`, alors que le nom développé est `document.fr:document`.

```
<?xml-stylesheet href="doc.xsl" type="text/xsl"?>
<document xmlns="document.fr">
  <texte>Mon texte</texte>
</document>
```

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:template match="document">
<html>
  <body><p><xsl:value-of select="texte"/></p></body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Syntaxe : XSL 1.0

En XSLT 1.0, il faut matcher le nom des éléments en utilisant explicitement les *namespaces* :

1. Il faut déclarer l'espace de nom dans la feuille XSLT
   Exemple : `<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:toto="toto.fr">`
2. Il faut que les noms d'éléments soient préfixés
   Exemple : `<xsl:template match="toto:e1/toto:e2">`

Exemple : Solution en XSLT 1.0

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
    xmlns:d="document.fr">
<xsl:template match="d:document">
<html>
  <body><p><xsl:value-of select="d:texte"/></p></body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Exemple : Solution en XSLT 1.0 avec namespace cible

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0"
   xmlns:d="document.fr">
<xsl:template match="d:document">
<html xmlns="http://www.w3.org/1999/xhtml">
  <body><p><xsl:value-of select="d:texte"/></p></body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Syntaxe : XSLT 2.0

En XSLT 2.0, l'attribut `xpath-default-namespace` permet de spécifier le *namespace* par défaut d'une XSLT et évite ainsi de préfixer tous les éléments :

`<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xpath-default-namespace="toto.fr">`.

Exemple : Solution en XSLT 2.0 avec namespace cible

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="2.0"
   xpath-default-namespace="document.fr"
>
<xsl:template match="document">
<html xmlns="http://www.w3.org/1999/xhtml">
  <body><p><xsl:value-of select="texte"/></p></body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Rappel

Namespace (cf. p.)

9. Transformation identité

Définition : Identité

Une transformation identité est une transformation qui délivre en sortie le même fichier XML qu'entrée.

Exemple

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

```
  version="1.0"
  >
<xsl:template match="node()|@*">
  <xsl:copy>
    <xsl:apply-templates select="node()|@*"/>
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

Méthode

L'avantage de la transformation identité est qu'elle peut servir de base pour des modifications partielles de document XML. Il est en effet possible de lui ajouter des règles pour traiter spécifiquement certains éléments, sans modifier les autres.

Complément : node()

La fonction XPath `node()` doit en principe renvoyer tous les nœuds d'un élément selon le standard. Mais il est fréquent que les parseurs ne renvoient pas les attributs. D'où la nécessité de matcher `node()|@*` pour avoir tous les nœuds.

10. Transformation substitution de namespace

Définition : Substitution de namespace

Une substitution de namespace est une transformation qui substitue un namespace par un autre.

Exemple

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

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

    <!-- Namespace substitution for source-target elements -->
    <xsl:template match="source:*">
        <xsl:element namespace="target-namespace" name="{local-name()}">
            <xsl:apply-templates select="node()|@*"/>
        </xsl:element>
    </xsl:template>

</xsl:stylesheet>

```

Exemple : Version avec XSLT 2.0 et préfixe de namespace par défaut

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

    <xsl:output method="xml" indent="yes"/>

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

    <!-- Namespace substitution for hdoc elements -->
    <xsl:template match="*">
        <xsl:element name="{local-name()}">
            <xsl:apply-templates select="node()|@*"/>
        </xsl:element>
    </xsl:template>

</xsl:stylesheet>

```

Complément : local-name()

La fonction XPath `local-name()` renvoie le nom d'un élément privé de son *namespace*.

11. Approfondissement

Complément : Standards

http://www.w3.org/TR/xslt

http://www.w3.org/TR/xpath/

Complément : Cours détaillés

[cf. W3C Xpath][cf. W3C XSL-XSLT]

Complément : Références rapides

http://www.mulberrytech.com/quickref/XSLT_1quickref-v2.pdf

XML précis & concis *

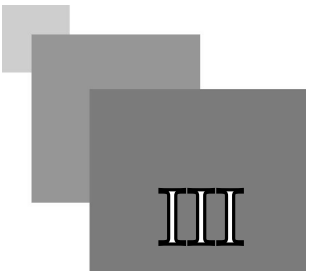
Complément : Ouvrage détaillé

XML et les bases de données *, chapitre 8.

Complément : Tutoriel

http://www.zvon.org/xxl/XSLTutorial/Output_fre/contents.html

Exercices



1. Exercices XSLT

1.1. Exercice : CV

Un schéma de CV

Soit le schéma XML suivant (formalisme DTD) :

```
<!ELEMENT cv (nom, prenom, age?, rubrique+)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT rubrique (titre, contenu)>
<!ELEMENT contenu  (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
```

Question 1

[solution n°1 p.33]

Écrire un document XML valide par rapport à ce schéma avec au moins un élément age et deux rubriques.

Indice :

```
<cv>
    <nom>...</nom>
    ...
    <rubrique>
        ...
    </rubrique>
    <rubrique>
        ...
    </rubrique>
</cv>
```

Question 2

[solution n°2 p.33]

Écrire un document HTML cible d'une transformation XSLT de ce document.

Les nom, prénom et age seront en italique, les titres de rubriques seront en gras.

Indice :

```
<html>
    <head>
        <title>...</title>
    </head>
    <body>
        <p><i>...</i></p>
        <p><i>...</i></p>
        <p><i>...</i></p>
        <p><b>...</b></p>
        <p>...</p>
        ...
    </body>
</html>
```

Question 3

[solution n°3 p.33]

Écrire le programme de transformation XSLT d'un fichier XML en fichier HTML.

Indice :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="cv">
    <html>
        <head>
            <title>CV de <xsl:value-of select="..."/ <xsl:value-of select="..."/> </title>
        </head>
        <body>
            <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>
<xsl:template match="nom">
    <p><i>...</i></p>
</xsl:template>
...
<xsl:template match="rubrique">
    ...
</xsl:template>
<xsl:template match="titre">
    ...
</xsl:template>
...
</xsl:stylesheet>
```

1.2. Exercice : Poème

Soit l'extrait de poème suivant écrit en XML :

```
<poeme titre="The Stone Troll" auteur="JRR Tolkien">
<strophe>
  <vers>Troll sat alone on his seat of stone,</vers>
  <vers>And munched and mumbled a bare old bone;</vers>
  <vers>For many a year he had gnawed it near,</vers>
  <vers>For meat was hard to come by.</vers>
  <vers>Done by! Gum by!</vers>
  <vers>In a cave in the hills he dwelt alone,</vers>
  <vers>And meat was hard to come by.</vers>
</strophe>
<strophe>
  <vers>Up came Tom with his big boots on.</vers>
  <vers>Said he to Troll: 'Pray, what is yon?</vers>
  <vers>For it looks like the shin o' my nuncle Tim.</vers>
  <vers>As should be a-lyin' in the graveyard.</vers>
  <vers>Caveyard! Paveyard!</vers>
  <vers>This many a year has Tim been gone,</vers>
  <vers>And I thought he were lyin' in the graveyard.</vers>
</strophe>
</poeme>
```

Question

Écrire un programme XSL-XSLT permettant de le transformer selon le format HTML suivant :

```
<html>
  <head>
    <title>The Stone Troll (JRR Tolkien)</title>
  </head>
  <body>
    <p>Troll sat alone on his seat of stone,</p>
    <p>And munched and mumbled a bare old bone;</p>
    <p>For many a year he had gnawed it near,</p>
    <p>For meat was hard to come by.</p>
    <p>Done by! Gum by!</p>
    <p>In a cave in the hills he dwelt alone,</p>
    <p>And meat was hard to come by.</p>
    <hr/>
    <p>Up came Tom with his big boots on.</p>
    <p>Said he to Troll: 'Pray, what is yon?</p>
    <p>For it looks like the shin o' my nuncle Tim.</p>
    <p>As should be a-lyin' in the graveyard.</p>
    <p>Caveyard! Paveyard!</p>
    <p>This many a year has Tim been gone,</p>
    <p>And I thought he were lyin' in the graveyard.</p>
  </body>
</html>
```

Indices :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="poeme">
  <html>
    <head>
      <title>...</title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
<xsl:template match="strophe">
  <xsl:apply-templates/>
  ...
</xsl:template>
<xsl:template match="vers">
  ...<xsl:value-of select="..."/>...
</xsl:template>
  ...
</xsl:stylesheet>
```

Pour gérer l'absence de `<hr/>` sur la dernière strophe, ajouter une règle qui sélectionne `strophe[last()]`.

1.3. Exercice : Glossaire II

Soit le schéma RelaxNG suivant permettant de représenter un glossaire.

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar
xmlns="http://relaxng.org/ns/structure/1.0"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
<start>
    <element name="glossaire">
        <oneOrMore>
            <ref name="Definition"/>
        </oneOrMore>
    </element>
</start>
<define name="Definition">
    <element name="definition">
        <attribute name="id"><data type="ID"/></attribute>
        <element name="terme"><text/></element>
        <element name="explication"><text/></element>
        <zeroOrMore>
            <element name="voirAussi">
                <attribute name="ref"><data type="IDREF"/></attribute>
            </element>
        </zeroOrMore>
    </element>
</define>
</grammar>
```

Question 1

Expliquer ce qu'exprime les *datatypes* `ID` et `REFID`. Préciser le format que doit respecter un attribut ou un élément de type `ID`.

Question 2

[solution n°4 p.34*]*

Instancier un document de type `glossaire`, sur une thématique au choix.

Question 3

Réaliser un programme XSLT permettant de publier les documents de type `glossaire` en HTML.

Indice :

Utiliser les ancres en HTML pour gérer les "voir aussi" (on se limitera pour le moment à utiliser l'id comme texte de l'ancre).

Question 4

Ajouter un sommaire au début du fichier HTML, permettant de pointer chaque définition.

Proposer deux implémentations une avec `xsl:template mode="..."` et une avec `xsl:for-each`.

Trier les définitions par ordre alphabétique.

Indice :

Pour trier les termes par ordre alphabétique utiliser `xsl:sort`.

```
<xsl:for-each select="...">
    <xsl:sort select="..."/>
    ...
```

```
<xsl:apply-templates select="...">
    <xsl:sort select="..."/>
</xsl:apply-templates>
```

Question 5

Pour les "voir aussi", afficher le terme de la définition plutôt que l'id.

Indice :

Utiliser la fonction XPath `current()`.

1.4. Exercice : Namespaces et FO

Soit le fichier XSLT suivant, permettant de transformer un fichier XML en fichier *Formatting Objects* (standard W3C pour la publication de fichiers imprimables).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
xmlns:fo="http://www.w3.org/1999/XSL/Format"
>
<xsl:template match="doc">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master master-name="A4" page-width="297mm" page-height="210mm" margin-top="1cm" margin-bottom="1cm" margin-left="1cm" margin-right="1cm">
        <fo:region-body margin="3cm"/>
        <fo:region-before extent="2cm"/>
        <fo:region-after extent="2cm"/>
        <fo:region-start extent="2cm"/>
        <fo:region-end extent="2cm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="A4" format="A">
      <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates select="para"/>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
```

```
</xsl:template>
<xsl:template match="para">
  <fo:block><xsl:value-of select="."/></fo:block>
</xsl:template>
<xsl:template match="para[1]">
  <fo:block><fo:inline font-weight="bold"><xsl:value-of select="."/></fo:inline></fo:block>
</xsl:template>
</xsl:stylesheet>
```

Question 1

[solution n°5 p.34]

Indiquer quels sont les *namespaces* et préfixes définis dans ce fichier XSLT, expliquer à quoi ils servent en général et dans ce cas précis, et en particulier pourquoi l'on ne pourrait pas s'en passer ici.

Question 2

[solution n°6 p.35]

Inventer un fichier XML source cohérent avec la transformation XSLT proposée, et produire le fichier FO correspondant au résultat de la transformation.

1.5. Exercice : Bulletins météo

Soit le fichier `f1.smi` suivant (très proche de la syntaxe SMIL) :

```
<?xml version="1.0" encoding="UTF-8"?>
<smil>
    <body>
        <seq>
            <par>
                <text dur="1">Île de France</text>
            </par>
            <par>
                <text dur="1">8 janvier 2010</text>
            </par>
            <par>
                <text dur="3">5 cm de neige en moyenne sur la région</text>
            </par>
            <par>
                <text dur="3">Jusqu'à 10cm de neige sur le sud de la région</text>
            </par>
        </seq>
    </body>
</smil>
```

Pour rappel :

- `seq` décrit une séquence de parties `par`, contenant du contenu affiché pendant une durée `dur`
- les `par` d'une `seq` sont présentes en même temps à l'écran (pour simplifier dans notre cas, les unes en dessous des autres)

Question 1

[solution n°7 p.35]

Ce fichier est-il un fichier XML bien formé (justifier) ?

Question 2

[solution n°8 p.36]

Soit la DTD `smilSuperLight.dtd` suivante :

```
<!ELEMENT smil (meta*, body)>
<!ATTLIST meta
    name (title) #REQUIRED
    content CDATA #REQUIRED>
<!ELEMENT body (seq)>
<!ELEMENT seq (par+)>
<!ELEMENT par (text)>
<!ELEMENT text (#PCDATA)>
<!ATTLIST text
    dur CDATA #REQUIRED>
```

Le fichier `f1.smi` est-il valide par rapport à cette DTD (justifier) ?

Question 3

[solution n°9 p.36]

Si `smilSuperLight.dtd` est un sous-ensemble du schéma SMIL officiel du W3C, le fichier XML `f1.smi` est-il valide par rapport à la DTD SMIL du W3C (expliquer) ?

Question 4

[solution n°10 p.36]

Pourquoi le fichier `f2.smi` n'est-il pas valide par rapport à la DTD `smilSuperLight.dtd` ?

```
<?xml version="1.0" encoding="UTF-8"?>
<smil>
    <body>
        <seq>
            <par>
                <text dur="1">Champagne</text>
                <text dur="1">8 janvier 2010</text>
            </par>
            <par>
                <text dur="3">Brouillard dominant et éclaircies en fin de matinée</text>
            </par>
        </seq>
    </body>
</smil>
```

Question 5

[solution n°11 p.36]

Modifier `smilSuperLight.dtd` afin que `f2.smi` soit à présent valide par rapport à une nouvelle DTD `smilSuperLight2.dtd`

Soit les deux fichiers XML suivants, structurés selon une sémantique métier :

```
<?xml version="1.0"?>
<meteo>
    <region>Picardie</region>
    <date>8 janvier 2010</date>
    <phenomene>Pluie et brouillard sur toute la région</phenomene>
    <temperature>
        <matin>-5</matin>
```

```
        <soir>-3</soir>
    </temperature>
</meteo>
```

```
<?xml version="1.0"?>
<meteo>
    <region>Nord</region>
    <date>8 janvier 2010</date>
    <phenomene>Brouillard et neige sur le nord de la région</phenomene>
    <phenomene>Vents violents sur le bord de mer</phenomene>
    <phenomene>Verglas sur toute la région</phenomene>
</meteo>
```

Question 6

[solution n°12 p.36*]*

Proposer un schéma XML `meteo.dtd` selon le formalisme des DTD de telle façon que ces *deux* fichiers soient valides par rapport à ce schéma.

Question 7

[solution n°13 p.37*]*

Proposer un programme XSLT permettant de transformer les fichiers de type `meteo` en `smilSuperLight2`, en affichant séquentiellement :

- la région avec la date dans une même première partie (1 seconde chacune, donc 2 secondes en tout),
- puis chaque phénomène (les températures ne sont pas affichées) pendant 3 secondes chaque.

Indices :

Cela correspond à l'exemple de `f2.smi`.

Il y a une difficulté particulière en XSLT à regrouper région et date dans une même partie, si vous ne parvenez pas à résoudre cette difficulté, faite l'exercice *sans les regroupez*. Cela revient dans ce cas à faire une transformation en `smilSuperLight`, à l'image de `f1.smi`.

2. Quiz XSLT et XPath

2.1. Exercice

[solution n°14 p.38]

Soit le fichier XML ci-après.

Le nœud courant est un des éléments `terme`, écrivez 4 expressions XPath différentes permettant de renvoyer le titre du document :

1. Sachant que `titre` est unique dans tout le document.
2. Sachant que `titre` est le fils de l'élément racine `papier`.
3. Sachant que `titre` est le fils du père du père du nœud courant.
4. Sachant que `titre` est avant le nœud courant dans l'ordre du document.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<papier type="scientifique">
  <titre>Réinterroger les structures documentaires</titre>
  <sousTitre>De la numérisation à l'informatisation</sousTitre>
  <auteur>Stéphane Crozat</auteur>
  <auteur>Bruno Bachimont</auteur>
  <resume>Nous proposons dans cet article d'aborder ...</resume>
  <abstract>In this paper we define...</abstract>
  <motsCles>
    <terme>Ingénierie des connaissances</terme>
    <terme>XML</terme>
    <terme>Document</terme>
  </motsCles>
  <keywords>
    <word>Knowledge engineering</word>
    <word>XML</word>
    <word>Document</word>
  </keywords>
  <publication date="2004-07-05"/>
  <version maj='1' min='0'/>
  <ressource uriSrc="http://archivesic.ccsd.cnrs.fr/docs/00/06/23/97/PDF/sic_00001015.pdf"/>
</papier>
```

1. //
2. /
3. ..
4. preceding

2.2. Exercice

[solution n°15 p.39]

Écrire le programme XSLT permettant de transformer le fichier `file.xml` en `result.xhtml`.

```
<!--file.xml-->
<doc>
<para>Lorem ipsum dolor sit amet.</para>
<para>Consectetur adipiscing elit.</para>
<para>Nunc eu lectus in diam.</para>
</doc>
```

```
<!--result.xhtml-->
<xhtml>
<body>
<p><i>Lorem ipsum dolor sit amet.</i></p>
<p>Consectetur adipiscing elit.</p>
<p>Nunc eu lectus in diam.</p>
</body>
</xhtml>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="                ">
  <                >
  <                >
  <xsl:apply-templates select="                "/>
  </               >
  </               >
</xsl:template>
<xsl:template match="                ">
  <                ><xsl:value-of select="                "/></               >
</xsl:template>
<xsl:template match="                ">
  <                ><                ><xsl:value-of select="                "/></               ></
               >
</xsl:template>
</xsl:stylesheet>
```

2.3. Exercice

[solution n°16 p.40]

Soit le fichier XML ci-après.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<papier type="scientifique">
  <titre>Réinterroger les structures documentaires</titre>
  <sousTitre>De la numérisation à l'informatisation</sousTitre>
  <auteur>Stéphane Crozat</auteur>
  <auteur>Bruno Bachimont</auteur>
  <resume>Nous proposons dans cet article d'aborder ...</resume>
  <abstract>In this paper we define...</abstract>
  <motsCles>
    <terme>Ingénierie des connaissances</terme>
    <terme>XML</terme>
    <terme>Document</terme>
  </motsCles>
  <keywords>
    <word>Knowledge engineering</word>
    <word>XML</word>
    <word>Document</word>
  </keywords>
  <publication date="2004-07-05"/>
  <version maj='1' min='0'/>
  <ressource uriSrc="http://archivesic.ccsd.cnrs.fr/docs/00/06/23/97/PDF/sic_00001015.pdf"/>
</papier>
```

Compléter le fichier XSLT `transf.xsl` afin de générer, pour chaque élément `terme`, une instruction SQL d'insertion dans une table relationnelle de schéma : `tMotsCles (terme, titre, url)` (où `terme` est le terme sélectionné, `titre` est le titre du document et `url` est l'adresse de la ressource associée).

Pour rappel, la syntaxe d'insertion de données dans une table relationnelle en SQL : `INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>) VALUES (<Liste ordonnée des valeurs à affecter>)`.

```
<!--transf.xsl-->

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

<xsl:output method="text"/>

<xsl:template match="                ">

<xsl:apply-templates select="                "/>

</xsl:template>

<xsl:template match="                ">

                                                (terme, titre, url)                 (

'<xsl:value-of select="                "/>',

'<xsl:value-of select="//                "/>',

'<xsl:value-of select="//                "/>'

);

</xsl:template>
```

```
</xsl:stylesheet>
```

Solutions des exercices

> **Solution** n°1

Exercice p. 21

```
<cv>
    <nom>Brassens</nom>
    <prenom>Georges</prenom>
    <age>33</age>
    <rubrique>
        <titre>Compétences</titre>
        <contenu>Chanteur, auteur, compositeur, guitariste</contenu>
    </rubrique>
    <rubrique>
        <titre>Langues étrangères</titre>
        <contenu>Espagnol, lu, parlé, chanté</contenu>
    </rubrique>
</cv>
```

> **Solution** n°2

Exercice p. 21

```
<html>
    <head>
        <title>CV de Georges Brassens</title>
    </head>
    <body>
        <p><i>Georges</i></p>
        <p><i>Brassens</i></p>
        <p><i>33 ans</i></p>
        <p><b>Compétences</b></p>
        <p>Chanteur, auteur, compositeur, guitariste</p>
        <p><b>Langues étrangères</b></p>
        <p>Espagnol, lu, parlé, chanté</p>
    </body>
</html>
```

> **Solution** n°3

Exercice p. 22

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

```
3
4 <xsl:template match="cv">
5     <html>
6         <head>
7             <title>CV de <xsl:value-of select="nom"/> <xsl:value-of select="prenom"/> </title>
8         </head>
9         <body>
10            <xsl:apply-templates/>
11        </body>
12    </html>
13 </xsl:template>
14
15 <xsl:template match="nom">
16    <p><i><xsl:value-of select="."/></i></p>
17 </xsl:template>
18
19 <xsl:template match="prenom">
20    <p><i><xsl:value-of select="."/></i></p>
21 </xsl:template>
22
23 <xsl:template match="age">
24    <p><i><xsl:value-of select="."/> ans</i></p>
25 </xsl:template>
26
27 <xsl:template match="rubrique">
28    <xsl:apply-templates/>
29 </xsl:template>
30
31 <xsl:template match="titre">
32    <p><b><xsl:value-of select="."/></b></p>
33 </xsl:template>
34
35 <xsl:template match="contenu">
36    <p><xsl:value-of select="."/></p>
37 </xsl:template>
38
39 </xsl:stylesheet>
```

> **Solution** n°4

Exercice p. 24

Exemple

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <glossaire>
3     <definition id="xml">
4         <terme>XML</terme>
5         <explication>Méta-langage...</explication>
6     </definition>
7     <definition id="sgml">
8         <terme>SGML</terme>
9         <explication>Méta-langage...</explication>
10        <voirAussi ref="xml"/>
11    </definition>
12 </glossaire>
```

> **Solution** n°5

Exercice p. 26

Les deux *namespaces* sont :

- http://www.w3.org/1999/XSL/Transform
- http://www.w3.org/1999/XSL/Format

Les préfixes correspondant sont respectivement :

- xsl
- fo

Les *namespaces* permettent d'assurer l'unicité des noms des éléments XML lorsque plusieurs schémas sont utilisés. Dans ce cas, ils servent à différencier les balises du schéma de transformation XSLT et les balises du schéma de publication cible *Formatting Objects*. Sans cette mécanique, si les schémas de XSLT et de FO définissaient deux éléments différents, mais avec le même nom, le processeur XSLT ne saurait pas les différencier et engendrerait des erreurs.

> **Solution** n°6

Exercice p. 26

```
<!--source.xml-->
<doc>
<para>Lorem ipsum dolor sit amet.</para>
<para>Consectetur adipiscing elit.</para>
<para>Nunc eu lectus in diam.</para>
</doc>
```

```
<!--result.fo-->
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="A4" page-width="297mm" page-height="210mm" margin-top="1cm" margin-bottom="1cm" margin-left="1cm" margin-right="1cm">
      <fo:region-body margin="3cm"/>
      <fo:region-before extent="2cm"/>
      <fo:region-after extent="2cm"/>
      <fo:region-start extent="2cm"/>
      <fo:region-end extent="2cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="A4" format="A">
    <fo:flow flow-name="xsl-region-body">
      <fo:block><fo:inline font-weight="bold">Lorem ipsum dolor sit amet.</fo:inline></fo:block>
      <fo:block>Consectetur adipiscing elit.</fo:block>
      <fo:block>Nunc eu lectus in diam.</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

> **Solution** n°7

Exercice p. 26

Oui car :

- Il y a un élément racine qui contient tous les autres
- Chaque élément contient totalement ses fils (toutes les balises sont fermées et il n'y a pas de croisement de balises).

On vérifie par ailleurs qu'il n'y a pas de problème de syntaxe (attributs, caractères utilisés pour les noms des balises, etc.).

> **Solution** n°8

Exercice p. 27

Le fichier est valide par rapport à la DTD :

- L'élément racine est bien `smil`
- `smil` contient un `body` (qui est obligatoire) et pas de `head` (qui est optionnel)
- Le `body` contient bien une `seq` obligatoire et unique
- la `seq` contient quatre `par` (elle devait en contenir au moins un)
- Chaque `par` contient bien un unique `text` (obligatoire) avec un attribut `dur` (obligatoire)
- Les `text` contiennent uniquement des caractères (#PCDATA).

> **Solution** n°9

Exercice p. 27

Si `smilSuperLight.dtd` est un sous-ensemble du schéma SMIL officiel cela signifie que le schéma SMIL permet *plus* d'éléments que `smilSuperLight.dtd`, mais que ces éléments sont *optionnels*.

Donc tous les documents XML valides par rapport à `smilSuperLight.dtd` seront valides par rapport au schéma officiel, puisqu'ils en respecteront les règles.

> **Solution** n°10

Exercice p. 27

Parce qu'il y a *deux* éléments `text` dans le premier `par`, qui n'en autorise qu'un et un seul.

> **Solution** n°11

Exercice p. 27

Il suffit de modifier la ligne définissant `par` en ajoutant un `+` après `text`, pour exprimer le fait qu'il peut y en avoir un à plusieurs.

```
1 <!ELEMENT par (text+)>
```

> **Solution** n°12

Exercice p. 28

```
<!ELEMENT meteo (region, date, phenomene+, temperature?)>
<!ELEMENT region (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT phenomene (#PCDATA)>
<!ELEMENT temperature (matin, soir)>
<!ELEMENT matin (#PCDATA)>
<!ELEMENT soir (#PCDATA)>
```

> **Solution** n°13

Exercice p. 28

Solution simplifiée smilSuperLight

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="meteo">
        <smil>
            <body>
                <seq>
                    <xsl:apply-templates select="*"/>
                </seq>
            </body>
        </smil>
    </xsl:template>
    <xsl:template match="region">
        <par>
            <text dur="1"><xsl:value-of select="."/></text>
        </par>
    </xsl:template>
    <xsl:template match="date">
        <par>
            <text dur="1">xsl:value-of select="."/></text>
        </par>
    </xsl:template>
    <xsl:template match="phenomene">
        <par>
            <text dur="3"><xsl:value-of select="."/></text>
        </par>
    </xsl:template>
    <xsl:template match="temperature"/>
</xsl:stylesheet>
```

Solution complète smilSuperLight2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="meteo">
        <smil>
            <body>
                <seq>
```

```
                    <par>
                        <text dur="1"><xsl:value-of select="region"/></text>
                        <text dur="1"><xsl:value-of select="date"/></text>
                    </par>
                    <xsl:apply-templates select="phenomene"/>
                </seq>
            </body>
        </smil>
    </xsl:template>
    <xsl:template match="phenomene">
        <par>
            <text dur="3"><xsl:value-of select="."/></text>
        </par>
    </xsl:template>
    <xsl:template match="temperature"/>
</xsl:stylesheet>
```

> **Solution** n°14

Exercice p. 29

Soit le fichier XML ci-après.

Le nœud courant est un des éléments `terme`, écrivez 4 expressions XPath différentes permettant de renvoyer le titre du document :

1. Sachant que `titre` est unique dans tout le document.
2. Sachant que `titre` est le fils de l'élément racine `papier`.
3. Sachant que `titre` est le fils du père du père du nœud courant.
4. Sachant que `titre` est avant le nœud courant dans l'ordre du document.

```
<?xml version="1.0" encoding="UTF-8"?>
<papier type="scientifique">
  <titre>Réinterroger les structures documentaires</titre>
  <sousTitre>De la numérisation à l'informatisation</sousTitre>
  <auteur>Stéphane Crozat</auteur>
  <auteur>Bruno Bachimont</auteur>
  <resume>Nous proposons dans cet article d'aborder ...</resume>
  <abstract>In this paper we define...</abstract>
  <motsCles>
    <terme>Ingénierie des connaissances</terme>
    <terme>XML</terme>
    <terme>Document</terme>
  </motsCles>
  <keywords>
    <word>Knowledge engineering</word>
    <word>XML</word>
    <word>Document</word>
  </keywords>
  <publication date="2004-07-05"/>
  <version maj='1' min='0'/>
  <ressource uriSrc="http://archivesic.ccsd.cnrs.fr/docs/00/06/23/97/PDF/sic_00001015.pdf"/>
</papier>
```

1. `//titre`
2. `/papier/titre`
3. `../../titre`

4. preceding`::titre`

> **Solution** n°15

Exercice p. 30

Écrire le programme XSLT permettant de transformer le fichier `file.xml` en `result.xhtml`.

```
<!--file.xml-->
<doc>
<para>Lorem ipsum dolor sit amet.</para>
<para>Consectetur adipiscing elit.</para>
<para>Nunc eu lectus in diam.</para>
</doc>
```

```
<!--result.xhtml-->
<xhtml>
<body>
<p><i>Lorem ipsum dolor sit amet.</i></p>
<p>Consectetur adipiscing elit.</p>
<p>Nunc eu lectus in diam.</p>
</body>
</xhtml>
```

<?xml version="1.0" encoding="UTF-8"?>

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

<xsl:template match="`doc`">

<`xhtml`>

<`body`>

<xsl:apply-templates select="`para`"/>

</`body`>

</`xhtml`>

</xsl:template>

<xsl:template match="`para`">

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

</xsl:template>

<xsl:template match="`para[1]`">

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

</xsl:template>

</xsl:stylesheet>

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="doc">
  <xhtml>
  <body>
  <xsl:apply-templates select="para"/>
  </body>
  </xhtml>
```

```
</xsl:template>
<xsl:template match="para">
  <p><xsl:value-of select="."/></p>
</xsl:template>
<xsl:template match="para[1]">
  <p><i><xsl:value-of select="."/></i></p>
</xsl:template>
</xsl:stylesheet>
```

> **Solution** n°16

Exercice p. 31

Soit le fichier XML ci-après.

```
<?xml version="1.0" encoding="UTF-8"?>
<papier type="scientifique">
  <titre>Réinterroger les structures documentaires</titre>
  <sousTitre>De la numérisation à l'informatisation</sousTitre>
  <auteur>Stéphane Crozat</auteur>
  <auteur>Bruno Bachimont</auteur>
  <resume>Nous proposons dans cet article d'aborder ...</resume>
  <abstract>In this paper we define...</abstract>
  <motsCles>
    <terme>Ingénierie des connaissances</terme>
    <terme>XML</terme>
    <terme>Document</terme>
  </motsCles>
  <keywords>
    <word>Knowledge engineering</word>
    <word>XML</word>
    <word>Document</word>
  </keywords>
  <publication date="2004-07-05"/>
  <version maj='1' min='0'/>
  <ressource uriSrc="http://archivesic.ccsd.cnrs.fr/docs/00/06/23/97/PDF/sic_00001015.pdf"/>
</papier>
```

Compléter le fichier XSLT `transf.xsl` afin de générer, pour chaque élément `terme`, une instruction SQL d'insertion dans une table relationnelle de schéma : `tMotsCles (terme, titre, url)` (où `terme` est le terme sélectionné, `titre` est le titre du document et `url` est l'adresse de la ressource associée).

Pour rappel, la syntaxe d'insertion de données dans une table relationnelle en SQL : `INSERT INTO <Nom de la relation> (<Liste ordonnée des propriétés à valoriser>) VALUES (<Liste ordonnée des valeurs à affecter>)`.

<!--transf.xsl-->

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

<xsl:output method="text"/>

<xsl:template match="`papier`">

<xsl:apply-templates select="`./motsCles/terme`"/>

</xsl:template>

<xsl:template match="`terme`">

INSERT INTO tMotsCles (terme, titre, url) VALUES (

'<xsl:value-of select="."/>',

'<xsl:value-of select="//titre"/>',

'<xsl:value-of select="//ressource/@uriSrc"/>'

);

</xsl:template>

</xsl:stylesheet>

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--transf.xsl-->
3 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
4 <xsl:output method="text"/>
5 <xsl:template match="papier">
6 <xsl:apply-templates select="./motsCles/terme"/>
7 </xsl:template>
8 <xsl:template match="terme">
9 INSERT INTO tMotsCles (terme, titre, url) VALUES (
10 '<xsl:value-of select="."/>',
11 '<xsl:value-of select="//titre"/>',
12 '<xsl:value-of select="//ressource/@uriSrc"/>'
13 );
14 </xsl:template>
15 </xsl:stylesheet>
```

Bibliographie

Alexandre Brillant, *XML : Cours et exercices*, Eyrolles, 2007 [ISBN 978-2212126914]

Robert Eckstein, Michel Casabianca, *XML précis & concis*, O'Reilly, 2000.

Kevin Williams, Michael Brundage, Patrick Dengler, Jeff Gabriel, Andy Hoskinson, Michael Kay, Thomas Maxwell, Marcello Ochoa, Johnny Papa, Mohan Vanmane, *XML et les bases de données*, Eyrolles, 2001.