XSLT

Quelques bases

Responsable : Patricia Serrano Alvarado

Master INFO 1ère année

Plan

- XSL introduction générale
- XSLT un avant goût
- Le langage de transformation XSLT

Traiter des documents XML

- Trois voies

  - Utiliser feuilles de style CSS, si le but est un habillage HTML document XML

  - Utiliser la programmation Java ou C++, etc. Avec Java, par exemple, on dispose d'API comme DOM, JDOM ou SAX. Pas de limitations sur ce qu'on peut faire

  - Utiliser XSL (eXtensible Stylesheet Language) qui regroupe de langages comme XSLT, spécifiques pour la description de transformations et du rendu de documents XML. En principe, pas de limitation à la nature des traitements réalisables.

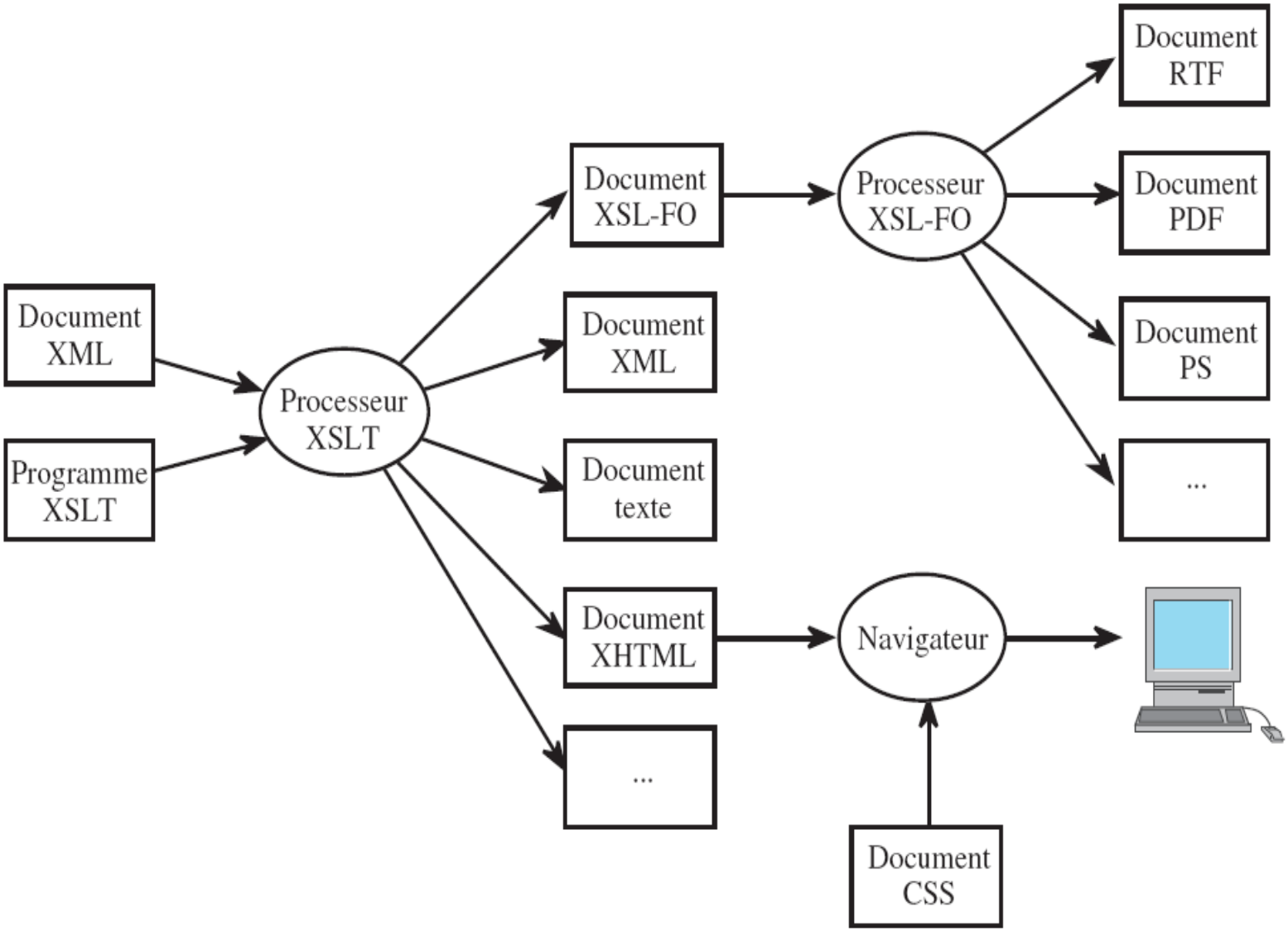
XSL

- Langage pour la définition de feuilles de style dont la spécification est divisée en deux parties

  - XSLT : langage de type XML qui sert à décrire les transformations d'un arbre XML en un autre arbre XML

  - XSLFO (XSL Formating Objects) : langage de type XML utilisé pour la description de page imprimables en haute qualité typographique

XSL : acteurs principaux



XSLT

eXtensible Stylesheet Language Transformation

Un avant goût

Recommandation W3C
Version 1.0 - 16 novembre 1999
http://www.w3.org/TR/xslt
Une version française hébergée par XMLfr
http://xmlfr.org/w3c/TR/xslt/

Recommandation W3C
Version 2.0 - 23 janvier 2007
http://www.w3.org/TR/xslt20/

XSLT

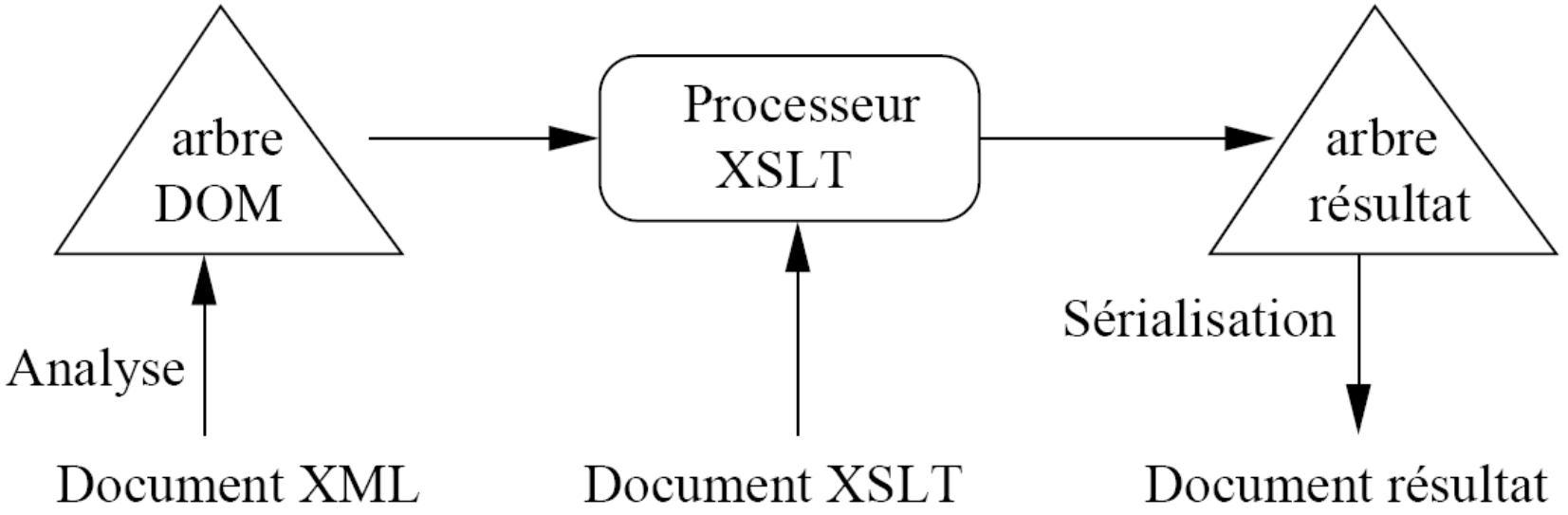
- Langage de transformation à base de règles de documents XML
- Transforme un arbre XML d'un document en un autre arbre
- Modes d'utilisation de XSLT
  - Mode commande
    - Souple
    - N'impose pas de format de sortie particulier (génération de HTML, XML, Latex, etc.)
    - Processeurs : Xt (James Clark), Xalan (Apache), Saxon (Michel Key), MSXSL (Microsoft), etc.
  - Mode navigateur
    - Limité aux navigateurs équipés de d'un processeur XSLT
    - A priori, orienté aux transformations vers HTML
    - Processeurs : Mozilla Firefox, Netscape 6, Internet Explorer 5 et 6, Safari
  - Mode Serveur
    - Un processeur XSLT, chargé en tant que *thread*, peut être invoqué par un API Java (e.g., TrAX)
    - A priori génération de pages HTML ou PDF
    - Processeurs : l'API TrAX (Transformation API pour XML), JAXP (Sun) c'est un API Java pour le traitement XML, Saxon, Xalan

Structure de base : les règles

- Règle = template : élément de base pour produire le résultat
  - Une règle s'applique dans le contexte d'un noeud de l'arbre
  - L'application de la règle produit un fragment du résultat.
- Programme XSLT = ensemble de règles pour construire un résultat (appelé également feuille de style)

Exemple

- L'exemple montré dans la suite introduit un document XML qui sera transformé en HTML
- L'exemple introduit un programme XSLT qui sera utilisée par le processeur XSLT
- Le processeur XSLT utilisé est celui d'un navigateur



Exemple (1) - Un document XML

Fichier Personnes.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
     href="personnes.xslt"?>


<personnes>
    <personne email="dbossard">
        <nom>Bossard</nom>
        <prenom>David</prenom>
        <age>29</age>
    </personne>

    <personne email="jmartin">
        <nom>Martin</nom>
        <prenom>Jacques</prenom>
        <age>32</age>
     </personne>
</personnes>
```

Exemple (2) - Une feuille de transformation

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

<xsl:template match="/">
    <html>
        <head>
            <title>personnes</title>
        </head>
        <body>
            <xsl:apply-templates />
        </body>
    </html>
</xsl:template>

<xsl:template match="personnes">
    <table>
        <thead>
          <tr>
            <th>nom</th>
            <th>prenom</th>
            <th>age</th>
          </tr>
        </thead>
        <xsl:apply-templates />
    </table>
</xsl:template>
```

Fichier personnes.xslt

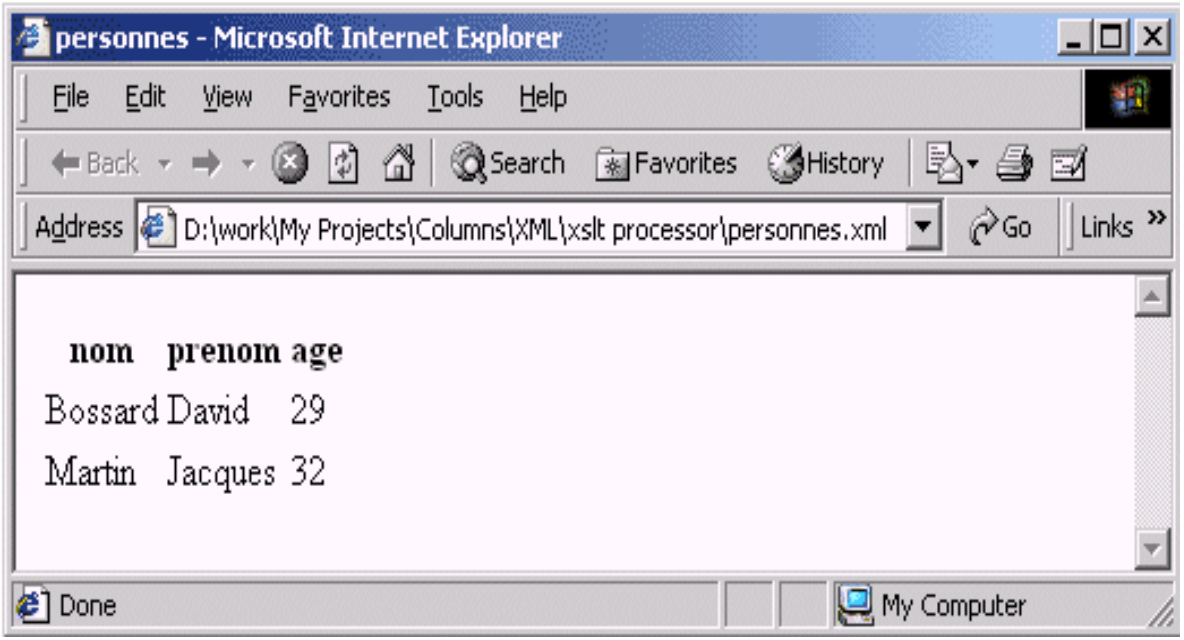
```
<xsl:template match="personne">
    <tr>
        <xsl:apply-templates />
    </tr>
</xsl:template>

<xsl:template match="nom">
    <td><xsl:value-of select="."/></td>
</xsl:template>

<xsl:template match="prenom">
    <td><xsl:value-of select="."/></td>
</xsl:template>

<xsl:template match="age">
    <td><xsl:value-of select="."/></td>
</xsl:template>

</xsl:transform>
```

Exemple (3) – le résultat

```
<html>
    <head>
        <title>personnes</title>
    </head>
    <body>
        <table>
          <thead>
            <th>nom</th>
            <th>prenom</th>
            <th>age</th>
          </thead>
          <tr>
            <td>
                Bossard
            </td>
            <td>
                David
            </td>
            <td>
                29
            </td>
          </tr>
          <tr>
            <td>
                Martin
            </td>
            <td>
                Jacques
            </td>
            <td>
                32
            </td>
          </tr>
        </table>
    <body>
</html>
```

Exemple (4)

- Safari, Internet Explorer 5, Mozilla Firefox ou Netscape 6 incluent un processeur XSLT
- Voici le résultat :



Explication des instructions de l'exemple

- xsl:template
  - Définit une règle XSLT.
  - L'attribut match associe un pattern à la règle. La règle est déclenchée pour tous les nœuds qui satisfont le pattern
  - Le corps de la règle définit le texte produit à chaque fois que la règle s'applique
  - L'exemple contient 6 règles
- xsl:apply-templates
  - Désigne les nœud à traiter au sein du document source. Le processeur détermine pour chaque nœud d'un ensemble de nœuds la règle à appliquer.
  - Cette instruction peut contenir un attribut select qui donne l'ensemble de nœuds. Par défaut, les nœuds sélectionnés sont les fils du nœud courant
  - Dans l'exemple, cette instruction est utilisée 3 fois (éléments : /, personnes, personne)
- xsl:value-of select="."
  - Évalue une expression, puis convertit le résultat en chaîne de caractères
  - L'attribut select détermine le contexte d'exécution
  - Dans l'exemple "." détermine le nœud courant

La navigation des documents XML

- L'une des activités dans XSLT est la sélection dans le document XML des données à transformer

- Le langage XPath est utilisé comme langage de navigation pour permettre de référencer des ensembles d'éléments, d'attributs et de textes dans un document XML

XSLT

eXtensible Stylesheet Language Transformation

XSLT status actuel

- XSL Transformations (XSLT) Version 1.0
- W3C Recommendation 16 November 1999
- http://www.w3.org/TR/xslt

- XSL Transformations (XSLT) Version 2.0
- W3C Recommendation 23 January 2007
- http://www.w3.org/TR/xslt20/

Introduction

- Un programme XSLT est un document XML

- Tous les aspects lexicaux sont du ressort de XML : la syntaxe pour les noms d'éléments, d'attributs, les caractères interdits dans les valeurs des attributs, etc.

- XSLT est un langage hybride car il combine deux modes de programmation :
  - **Déclaratif** : on donne les règles mais l'ordre d'exécution n'est pas défini.
  - **Impératif** : on utilise les structures habituelles d'un langage de programmation (tests, boucles, variables)

Le processeur XSLT

- Une transformation XSLT est une suite d'opérations qui transforme un arbre d'un document XML (arbre source) en un arbre XML (arbre résultat)

- Processus général
  - A partir d'un document XML, construction d'un arbre source XML correspondant
  - L'arbre source est parcouru et selon la transformation XSLT un arbre résultat est construit
  - L'arbre résultat est « sérialisé » en un document XML

Structure d'un programme XSLT

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

 <xsl:template match="COURS">
    <html>
          <head><title>Fiche du cours</title></head>
          <body bgcolor="white">
            <p>
              <h1>
                 <i>
                       <xsl:value-of select="SUJET"/>
                 </i>
              </h1>
              <hr>
                       <xsl:apply-templates/>
              </hr>
          </body>
    </html>
 </xsl:template>
</xsl:stylesheet>
```

Élément racine

- défini l'espace de nom de XSLT

Élément du premier niveau

- Essentiellement de règles (*template*)
- L'ordre des éléments n'a pas d'importance

Instructions

- Elles se trouvent dans le corps des règles

Principaux éléments de premier niveau

Type d'élément	Description
xsl:import	Import d'un programme XSLT
xsl:include	Inclusion d'un programme XSLT
xsl:output	Indique le format de sortie
xsl:param	Définit un paramètre
xsl:template	Définit une règle XSLT
xsl:variable	Définit une variable XSLT
xsl:strip-space	Les fils de type text constitués de blancs sont supprimés
xsl:preserve-space	Les fils de type text constitués de blancs sont préservés

Les règles XSLT

- Un programme XSLT se compose essentiellement de règles.
- Une règle est constituée de
  - Un motif (*pattern*) ou cible exprimé en XPath
  - Un nom (optionnel) qui permet d'appeler la règle par son nom avec xsl:call-template
  - Un modèle de transformation (*template*)

```
<xsl:template name ="RootRule" match="…pattern…">
        <!-- modèle de transformation -->
        …
        <!-- fin du modèle de transformation -->
</xsl:template>
```

Template

- Le corps du template dit par quoi remplacer l'élément courant.
  - Ça peut être du texte simple
  - Du texte et des éléments XML
  - Une expression XPath quelconque
  - De balises HTML ou une autre variante de XML pourvu que le balisage respecte la structure d'un document bien formé imposée par le standard XML
  - Etc.

Sélection des règles

- Étant donné un noeud, trouver la règle qui s'applique

  - Soit N le noeud
  - Soit P le pattern de la règle
  - S'il existe quelque part un noeud C tel que l'évaluation de P à partir de C contient N alors la règle s'applique

Déclenchement de règles

- On prend un noeud du document comme noeud contexte
  - Au départ N c'est la racine du document
- On cherche la règle qui s'applique à ce noeud
- On insère le corps de la règle dans le document résultat
- Les instructions xsl:apply-templates et xsl:for-each permettent de sélectionner de nouveaux nœuds contexte

Priorité d'exécution des règles

Lorsque deux règles peuvent être appliquées à un même nœud

- Implicitement, la règle la plus spécifique a la priorité la plus forte
  - Dans cet exemple, la règle heure a la priorité la plus forte

```
<xsl:template name="Heure" match ="Heure">
    …
</xsl:template>
<xsl:template name="All" match ="*">
    …
</xsl:template>
```

- Expressément, on peut forcer la priorité à une règle
  - L'attribut "priority" permet d'affecter une priorité à une règle
  - Dans cet exemple, la règle HeureTeatre s'exécute en premier

```
<xsl:template name="HeureTeatre" match ="Teatre/Heure" priority="2">
    …
</xsl:template>
<xsl:template name="Heure" match ="Heure" priority="1">
    …
</xsl:template>
```

- Si le choix est impossible, le processeur s'arrête !

Règles par défaut (1)

- Lors du traitement d'un nœud, il est possible qu'aucune règle fourni dans le programme XSLT ne soit applicable (car aucun pattern ne concorde pas avec le nœud courant)

- Dans ce cas, une règle par défaut est appliquée

- Il existe des règles pour
  - La racine du document
  - Les nœuds de type text ou attribute
  - Les nœuds de type comment ou processing-instruction

Règles par défaut (1)

- Les éléménts et la racine du document

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

  - L'application des règles pour les fils du nœud courant est demandée

- Type text ou attribute

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

  - La valeur du nœud text ou attribut est insérée dans le document résultat

- Type comment ou processing-instruction

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

  - Il ne se passe rien

Comportement inattendu

- La présence des règles par défaut peut générer de comportements inattendus lorsque, par exemple, le programme XSLT comporte une erreur

- Pour savoir exactement ce qui se passe on peut ajouter au programme XSLT une règle « attrape tout »

```
<xsl:template match="*">
    erreur : élément non prévu : tag {<xsl:value-of
    select="local-name (.)/>"}
</xsl:template>
```

- Le processeur applique cette règle de préférence à la règle par défaut car celle-ci est choisie quand aucune règle du programme ne convient.

Plusieurs règles sur un même nœud

- Il est possible d'exécuter plusieurs règles différentes sur un même nœud
- Ceci est possible grâce à l'attribut mode de l'instruction template qui fait la différence entre les règles

```
<xsl:template match='toto' mode="mode1"
    …
</xsl:template>
<xsl:template match='toto' mode="mode2"
    …
</xsl:template>
```

- Ensuite l'instruction qui déclenche ces règles peut préciser laquelle doit être sélectionnée

```
<xsl:apply-templates select ='toto' mode="mode2"/>
```

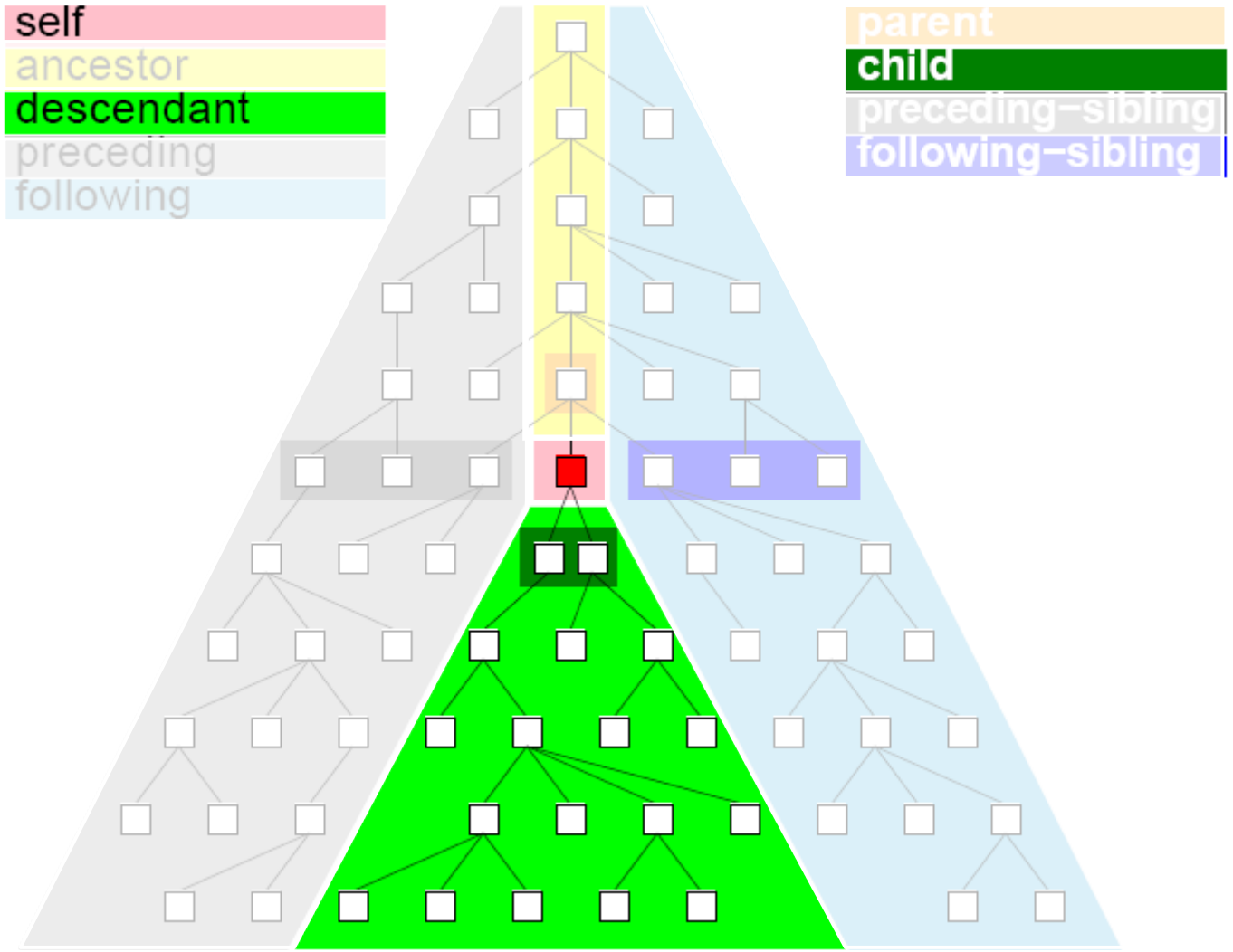
Modèle de traitement d'un processeur XSLT

- Une liste de nœuds source initiale ne contenant que la racine de l'arbre XML à traiter est constituée. Le traitement du document consiste à traiter la liste de nœuds initiale
- La liste de nœuds traite séparément chaque nœud dans l'ordre où il apparaît dans la liste. Ceci produit en résultat autant de fragments de document qu'il y a de nœuds source à traiter
- Le traitement d'un nœud consiste (1) à chercher parmi l'ensemble de règles XSLT définies dans le programme, celle dont le motif correspond au nœud en question, puis (2) à appliquer cette règle
- Si aucune règle n'est trouvée, une règle par défaut est appliqué
- Si plusieurs règles sont trouvées, un algorithme de calcul de priorité sélectionne une des règles.

Pattern

- Le pattern est l'expression de l'attribut match qui doit toujours désigner un ensemble de nœuds
- Un pattern est un ensemble de chemins de localisation séparés par | qui veut dire « ou »
- Seulement les axes de localisation suivants sont autorisés:
  - Les fils d'un élément : child
  - Les attributs d'un élément : attribute
  - L'abréviation // de descendant-or-self::node()/
- Ceci est dû au fait qu'on peut savoir si une règle doit être déclenchée uniquement en regardant les descendants du noeud contexte
- Une chemin de localisation peut avoir de prédicats dans lesquels aucun type d'axe de localisation n'est interdit

Axes de localisation à utiliser dans les pattern



Exemples de patterns

Forme longue	Forme courte
child::chapitre/child::section/child::paragraphe	chapitre/section/paragraphe
child::chapitre/descendant-or-self::node()/ child::paragraphe	chapitre//paragraphe
child::chapitre/child::section/ ou child/annexe	chapitre/section | annexe
child::paragraphe/attribute::alignement	paragraphe/@alignement
child::paragraphe/child::processing-instruction	paragraphe/processing-instruction
child::chapitre/child::section/ child::paragraphe[attribut::alignement = "centré"]	chapitre/section/ paragraphe[@alignement = "centré"]
child::chapitre[following-sibling::annexe]/descendant-or-self::node()/child::paragraphe	chapitre[following-sibling::annexe]//paragraphe

Programmation impérative avec XSLT

- Jusqu'ici on a vu la programmation déclarative de XSLT.
  - Les règles sont données par le programme XSLT et le processeur les exécute dans un ordre non préétabli
  - Le résultat est déterministe : on arrive toujours au même résultat

- La programmation impérative
  - Utilisation de structures déclaratives
    - Boucles (for)
    - Test (if)
    - Variables
  - Pas de variables incrementables

Instruction *for-each* (1)

- Cette instruction crée une liste de nœuds à traiter récursivement grâce à l'attribut select

- A différence de l'instruction apply-templates, avec for-each, la règle à appliquer à chaque nœud de la liste n'est pas recherchée parmi l'ensemble de règles du programme XSLT. Uniquement le modèle de transformation dans le corps du for-each est appliqué à chaque nœud

- Cette instruction permet également la répétition d'une même transformation sur plusieurs éléments

```
<xsl:for-each select="…">
    <!-- modèle de transformation -->
    …
    <!-- fin du modèle de transformation -->
</xsl:for-each>
```

Instruction *for-each* (2)

- L'instruction for-each, comme apply-templates, ne peut pas apparaître comme une instruction de premier niveau

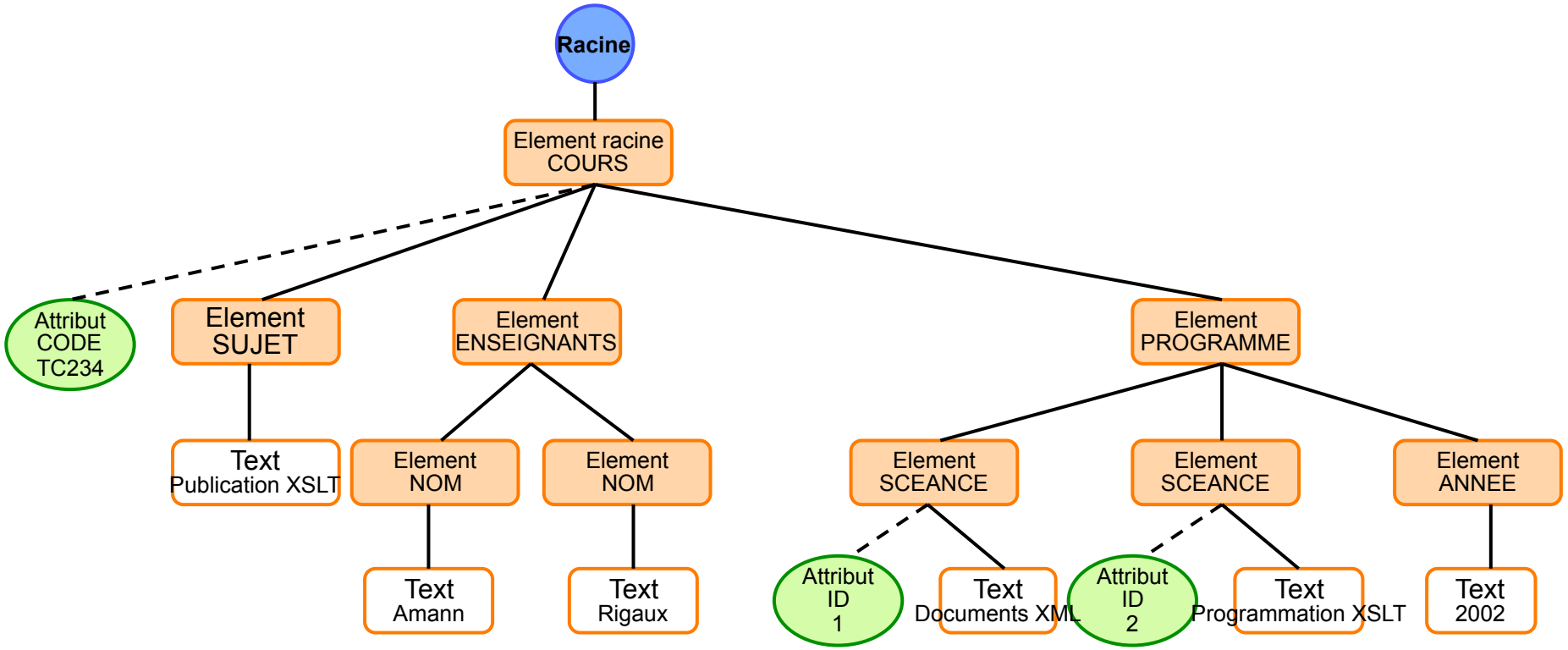
```
<xsl:template match="…motif (pattern)…">
    <!--modèle de transformation englobant -->
    …texte ou instruction XSLT …
    <xsl:for-each select="…">
        <!-- modèle de transformation -->
        …texte ou instruction XSLT
        <!-- fin du modèle de transformation -->
    </xsl:for-each>
…texte ou instruction XSLT …
    <!-- fin du modèle de transformation englobant -->
</xsl:template>
```

Exemple : un document XML

Fichier enseignants.xml

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

<COURS CODE="TC234">
    <SUJET>Publication XSLT</SUJET>
    <ENSEIGNANTS>
        <!-- Enseignant responsable -->
        <NOM>Amann</NOM>
        <NOM>Rigaux</NOM>
    </ENSEIGNANTS>
    <PROGRAMME>
        <SEANCE ID="1">Documents XML</SEANCE>
        <SEANCE ID="2">Programmation XSLT</SEANCE>
        <ANNEE>2002</ANNEE>
    </PROGRAMME>
</COURS>
```

L'arbre du document XML



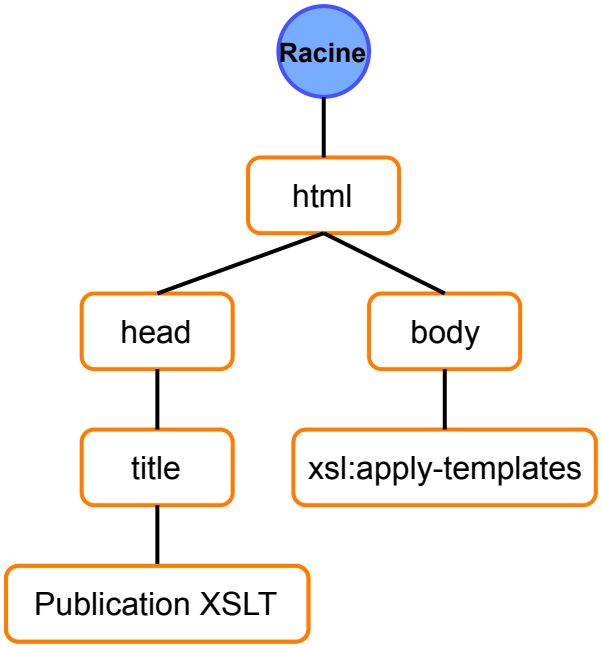
Traitement : liste initial – L'élément racine

Element racine
COURS

- La première règle défini le cadre du document html à générer

Extrait du fichier enseignants.xslt

```
<xsl:template match="/">
    <html>
      <head><title>
          <xsl:value-of select="COURS/SUJET"/>
      </title></head>
      <body bgcolor="white">
          <xsl:apply-templates/>
      </body>
    </html>
</xsl:template>
```



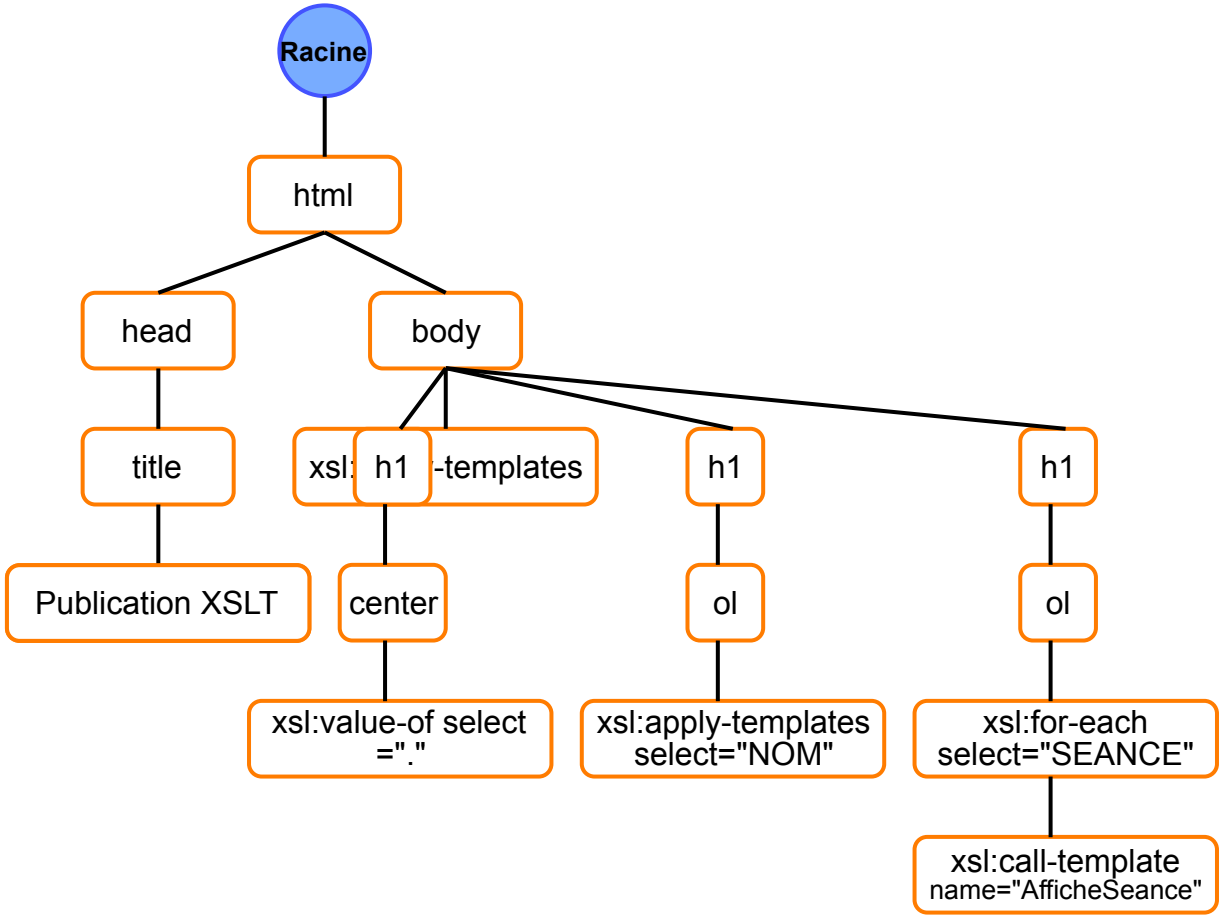
Traitement liste d'enfants de la racine (1)

- L'instruction apply-templates dans le corps de la règle génère une liste de nœuds avec les enfants directs de l'élément contexte
  - COURS
- Le père de l'attribut CODE est COURS mais CODE n'est pas fils de COURS
- Pas de règle pour COURS alors la règle par défaut s'exécute et la nouvelle liste de nœuds est :
  - SUJET
  - ENSEIGNANTS
  - PROGRAMME

Extrait du fichier enseignants.xslt

```
<xsl:template match="SUJET">
        <h1><center>
          <xsl:value-of select="."/>
        </center></h1>
</xsl:template>

<xsl:template match="ENSEIGNANTS">
        <h1>Enseignants</h1>
          <ol>
              <xsl:apply-templates select="NOM"/>
          </ol>
</xsl:template>

<xsl:template match="PROGRAMME">
        <h1>Programme</h1>
        <ul>
          <xsl:for-each select="SEANCE">
              <xsl:call-template
    name="AfficheSeance"/>
          </xsl:for-each>
        </ul>
</xsl:template>
```

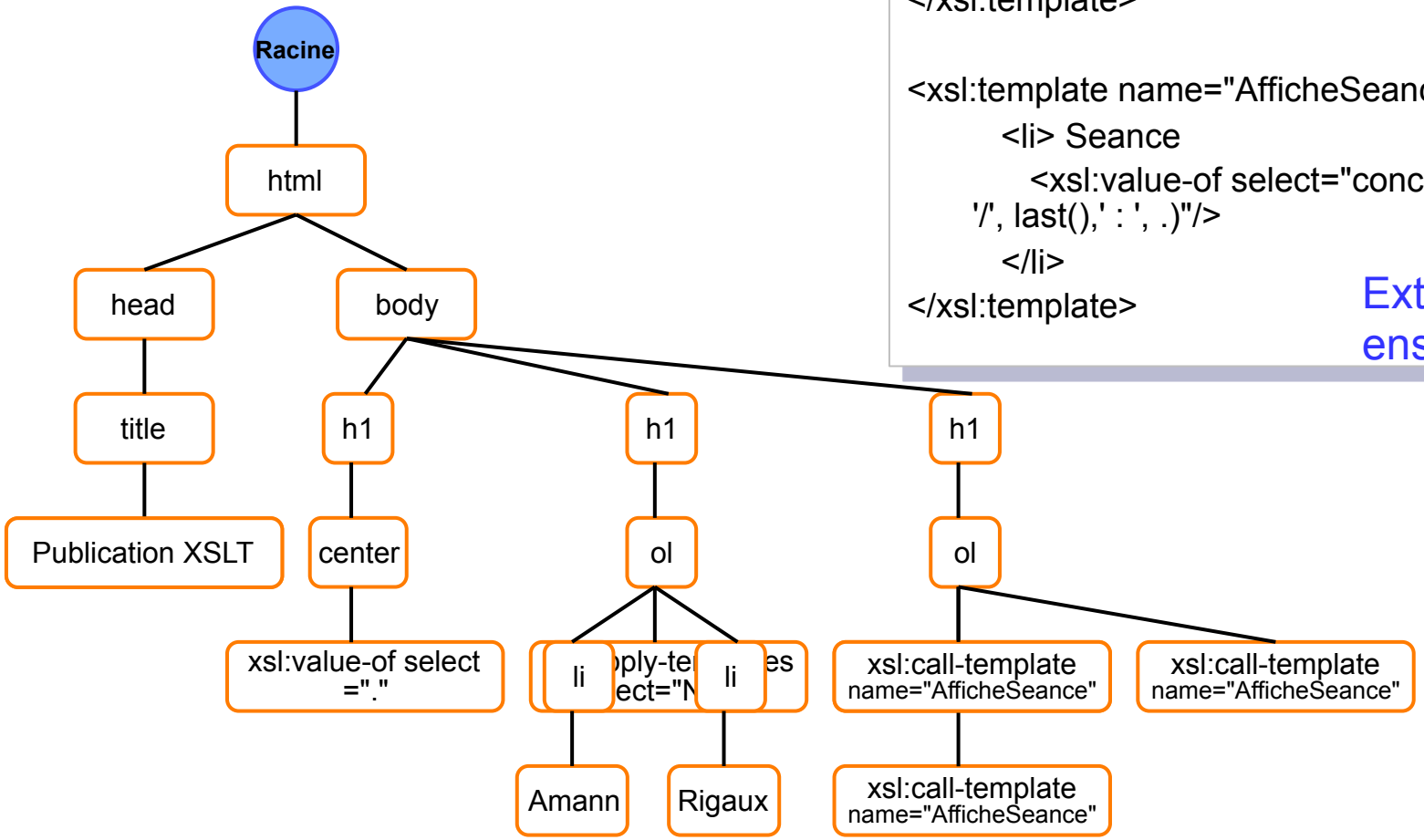
Traitement liste d'enfants de la racine (2)



Les dernières règles

```
<xsl:template match="NOM">
        <li><xsl:value-of select="." /></li>
</xsl:template>

<xsl:template name="AfficheSeance">
        <li> Seance
          <xsl:value-of select="concat (position(),
    '/', last(),' : ', .)"/>
        </li>
</xsl:template>
```

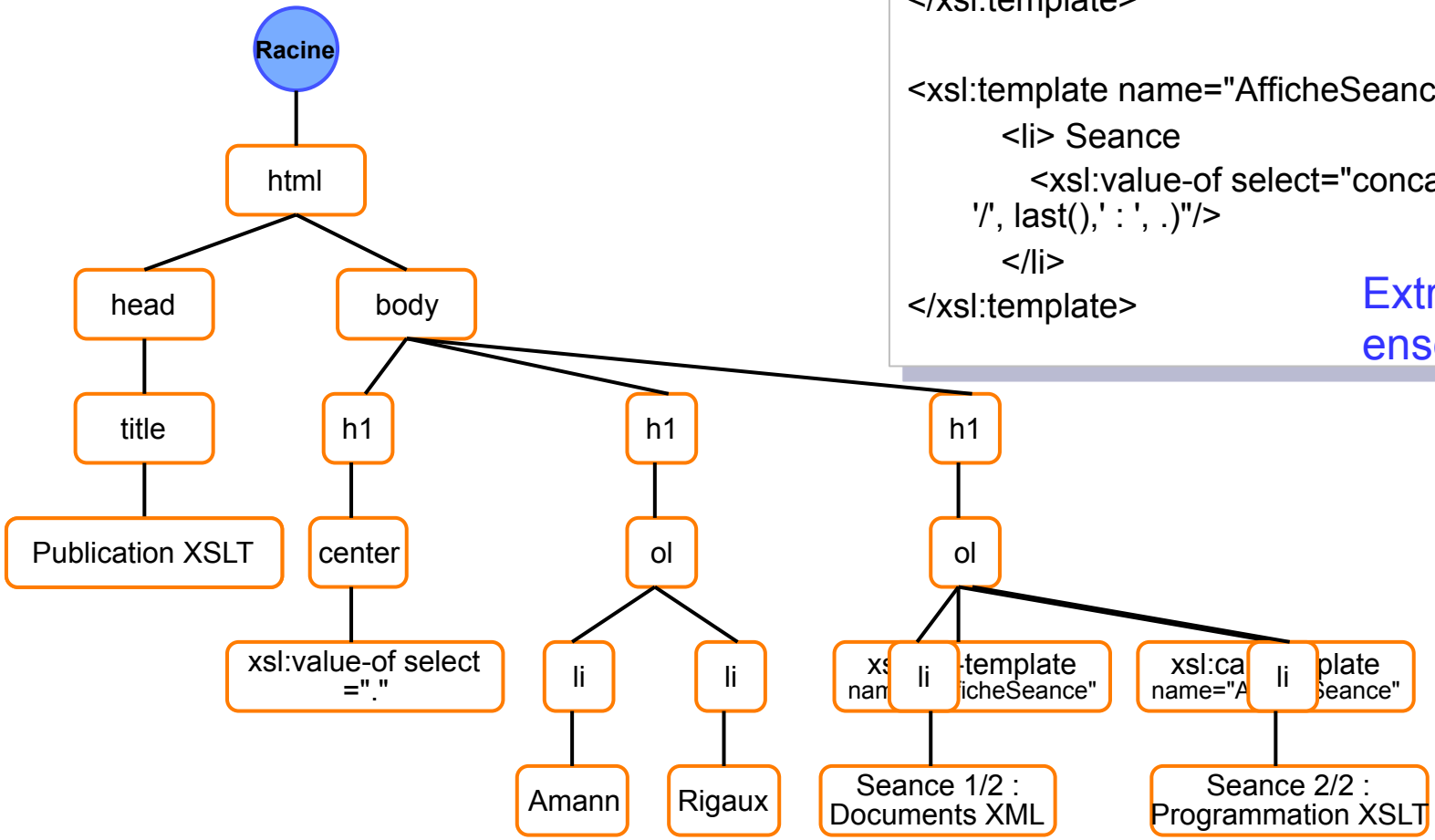
Extrait du fichier enseignants.xml



Les dernières règles

```
<xsl:template match= "NOM">
        <li><xsl:value-of select="." /></li>
</xsl:template>

<xsl:template name="AfficheSeance">
        <li> Seance
          <xsl:value-of select="concat (position(),
    '/', last(),' : ', .)"/>
        </li>
</xsl:template>
```

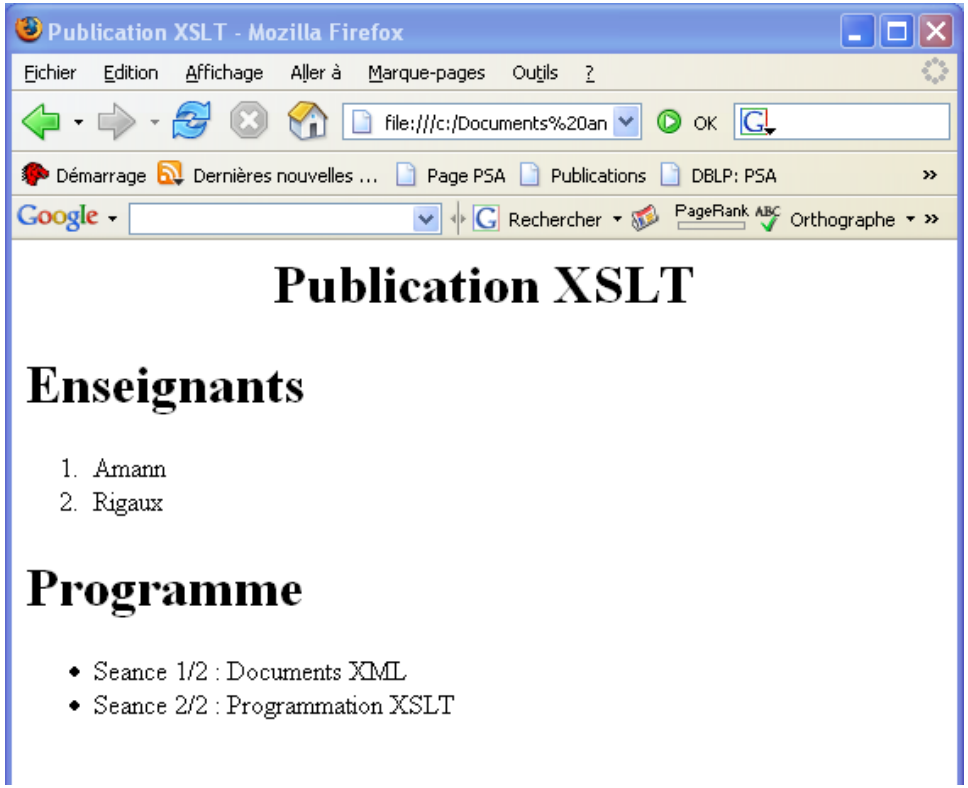
Extrait du fichier enseignants.xml



Exécution de l'exemple

- Cet exemple peut être exécuté en utilisant Xalan de Apache
- Voici la ligne de commande à utiliser :

  java org.apache.xalan.xslt.Process -in enseignants.xml -xsl enseignants.xslt -out enseignants.html



Instruction *if*

- Cette instruction instance son contenu si l'expression de l'attribut test s'évalue à « true »
- Il n'y a pas de else ! Mais possibilité de l'utiliser avec l'instruction choose
- Cette instruction ne doit pas apparaître comme instruction de premier niveau

```
<xsl:template match="…motif (pattern)…">
    <xsl:if test="…expression XPath…">
        <!-- modèle de transformation -->
        …texte ou instruction XSLT
        <!-- fin du modèle de transformation -->
    </xsl:if>
</xsl:template>
```

Instruction *choose* (1)

- Cette instruction est associée aux instructions when et otherwise pour créer l'équivalent des structures de type if-then-else ou switch
- Son contenu est une liste d'éléments when, chacun étant associé à un test et chacun ayant un contenu sous forme de corps de règle
- Le premier élément when pour lequel le test s'évalue à true voit son corps de règle instancié. Les éléments qui suivent sont ignorés
- Si aucun élément when ne s'évalue à true, le contenu de l'élément otherwise, s'il existe, est instancié

Instruction *choose* (2)

- Cette instruction ne doit pas apparaître comme instruction de premier niveau

```
<xsl:template match="…motif (pattern)…">
    <xsl:choose>

        <xsl:when test="…expression XPath…">
            <!-- modèle de transformation -->
            …texte ou instruction XSLT
            <!-- fin du modèle de transformation -->
        </xsl:when>

        <xsl:otherwise test="…expression XPath…">
            <!-- modèle de transformation -->
            …texte ou instruction XSLT
            <!-- fin du modèle de transformation -->
        </xsl:otherwise>

    </xsl:choose>
</xsl:template>
```

Instruction *variable* (1)

- Cette instruction permet de déclarer une variable XSLT
- Les variables ne peuvent pas changer de valeur
- L'affectation d'une valeur à la variable se passe uniquement lors de sa déclaration
- Une même variable ne peux pas être déclarée plusieurs fois
- Deux types de variables :
  - **Variables globales** (éléments de premier niveau). Elles sont visibles dans tout le programme XSLT.
  - **Variables locales** (dans les corps de règle). Visibles dans les following-sibling et leurs descendants
- Chevauchement des portées interdit, sauf pour une variable globale et une variable locale.

Instruction *variable* (2)

- L'attribut select d'une instruction variable est optionnel, il permet l'initialisation de la variable

- Trois possibilités de déclaration

  - L'attribut select contient une valeur constante

    ```
    <xsl:variable name='var1' select='12'/>
    ```

  - L'attribut select contient une expression XPath

    ```
    <xsl:variable name='var2' select='/COURS/ENSEIGNANTS'/>
    ```

  - Pas d'attribut select. Le contenu de l'instruction est la valeur de la variable (chaîne de caractères)

    ```
    <xsl:variable name='var3'>
        Ceci est un <mot-cle>contenu</mot-cle> de variable
    </xsl:variable>
    ```

Exemple de variable globale (1)

Fichier myFilms.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FILMS>
    <FILM>
        <TITRE>Vertigo</TITRE>
        <ANNEE>1958</ANNEE>
        <GENRE>Drame</GENRE>
        <PAYS></PAYS>
        <MES idref="3"></MES>
        <ROLES>
            <ROLE idref="15">John Ferguson</ROLE>
            <ROLE idref="16">Madeleine Elster</ROLE>
        </ROLES>
        <RESUME>…</RESUME>
    </FILM>
    <FILM>
        <TITRE>Alien</TITRE>
        <ANNEE>1979</ANNEE>
        <GENRE>Science-fiction</GENRE>
        <PAYS></PAYS>
        <MES idref="4"></MES>
        <ROLES>
            <ROLE idref="5">Ripley</ROLE>
        </ROLES>
        <RESUME>...</RESUME>
    </FILM>
</FILMS>
```

Exemple de variable globale (2)

Fichier myFilms.xslt

```
<xsl:output method='text' encoding='UTF-8'/>
<xsl:variable name="annee" select="1970"/>
<xsl:template match="FILM">

    <xsl:choose>
      <xsl:when test="ANNEE &lt; $annee">
            "<xsl:value-of select="TITRE"/>" est ancien
      </xsl:when>

      <xsl:when test="ANNEE &gt;= $annee">
            "<xsl:value-of select="TITRE"/>" est récent
      </xsl:when>
    </xsl:choose>

</xsl:template>
```

Fichier myFilms.txt

```
"Vertigo" est ancien
"Alien" est recent
```

java org.apache.xalan.xslt.Process -in myFilms.xml -xsl myFilms1.xslt -out myFilms.txt

Exemple de variable locale (1)

Fichier enseignants.xml

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

<COURS CODE="TC234">
    <SUJET>Publication XSLT</SUJET>
    <ENSEIGNANTS>
        <!-- Enseignant responsable -->
        <NOM>Amann</NOM>
        <NOM>Rigaux</NOM>
    </ENSEIGNANTS>
    <PROGRAMME>
        <SEANCE ID="1">Documents XML</SEANCE>
        <SEANCE ID="2">Programmation XSLT</SEANCE>
        <ANNEE>2002</ANNEE>
    </PROGRAMME>
</COURS>
```

Exemple de variable locale (2)

```
<xsl:template match="PROGRAMME">
  <SEANCES>
    <xsl:variable name="phrase">
          (valable pour l'an <xsl:value-of select="ANNEE"/>)
    </xsl:variable>
    <xsl:for-each select="SEANCE">
          <xsl:value-of select="concat(., $phrase)"/>
    </xsl:for-each>
  </SEANCES>
</xsl:template>
```

Fichier enseignants3.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
    Publication XSLT
          Amann
          Rigaux
    <SEANCES>
          Documents XML (valable pour l'an 2002)
          Programmation XSLT (valable pour l'an 2002)
    </SEANCES>
```

Fichier enseignants3.html

```
java org.apache.xalan.xslt.Process -in enseignants.xml -xsl enseignants3.xslt -out enseignants3.html
```

Conclusion (1)

- XSLT est un langage de programmation déclaratif à 99%.

- Il est Turing-complet. Une machine de Turing peut calculer tout ce qui est calculable.

  Avec XSLT on peut programmer tout ce qu'une machine de Turing peut calculer.

- C'est langage idéal uniquement pour la transformation de documents XML.

Conclusion (2)

- Avec XSLT il est possible de
  - Extraire de l'information à partir de documents XML
  - Créer de documents de la famille XML (HTML, XHTML, WML, etc.)
    - Homogénéisation
    - Reformatage
    - Habillage
    - Etc.
  - Partitionner un document XML en plusieurs documents (à l'aide de la fonction `write` de l'extension `redirect` proposée par Xalan)
  - Joindre plusieurs documents XML en un document XML (à l'aide de la fonction `document()`)

Références

- XSLT fondamental avec 20 design patterns prêts à l'emploi. Philipe Drix. Ed Eyrolles, 2002

- Comprendre XSLT. Philippe Rigaux, Bernd Amann. Ed O'Reilly, 2002
  http://cortes.cnam.fr:8080/XBOOK/SITE/index.html

- Le processeur XSLT de Microsoft

  http://www.microsoft.com/france/msdn/support/colones/info/info.asp?mar=/france/msdn/support/colones/info/xslt_processor.html

- XSLT tutorial
  http://www.w3schools.com/xsl/

- Processeur XSLT Xalan-Java
  http://xml.apache.org/xalan-j/

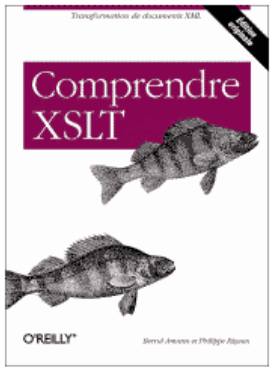
Comprendre XSLT (B. Amann et P. Rigaux) - Mozilla Firefox

Fichier Edition Affichage Aller à Marque-pages Outils ?

http://cortes.cnam.fr:8080/XBOOK/SITE/index.html

Démarrage Dernières nouvelles ... Page PSA Publications DBLP: PSA 20 Minutes Nantes EL UNIVERSAL La Jornada Le Conjugueur Dictionary univ LINA BiblioSTIC laser_1erhp_RV

Google Rechercher PageRank Orthographe Options



Comprendre XSLT

A propos de ce site

Ce site est associé au livre **Comprendre XSLT** de Bernd Amann et Philippe Rigaux, paru aux Editions O'Reilly. Vous y trouverez bien entendu les exemples du livre, ainsi que diverses documentations complémentaires, et notamment le matériel pédagogique que nous utilisons au Cnam et à l'Université d'Orsay pour nos cours autour de XML, XSLT et les techniques d'échange et d'intégration de données sur l'Internet. Les différentes rubriques du site sont les suivantes :

- les sources de tous les exemples du livre;
- quelques études de cas avec leurs sources;
- de la documentation compléméntaire comprenant des transparents de cours et des instructions d'installation pour quelques environnements représentatifs;
- et des liens vers d'autres serveurs.

Si vous avez des commentaires sur le livre, le site, des suggestions ou même des éléments complémentaires à nous fournir, vous pouvez nous contacter par email.

Organisation du livre

Le livre suit une double démarche de présentation des aspects les plus simples, puis, progressivement, les plus complexes du langage XSLT, et d'application des mécanismes de ce langage à des cas concrets d'utilisation. Il débute par un chapitre introductif en forme d'étude de cas qui propose, sur une application de type "Officiel des spectacles" adaptée au Web, une déclinaison des différents thèmes couverts. Ce chapitre est librement disponible ci-dessous:

L'introduction à XML/XSLT