XSLT

Dan VODISLAV

CY Cergy Paris Université

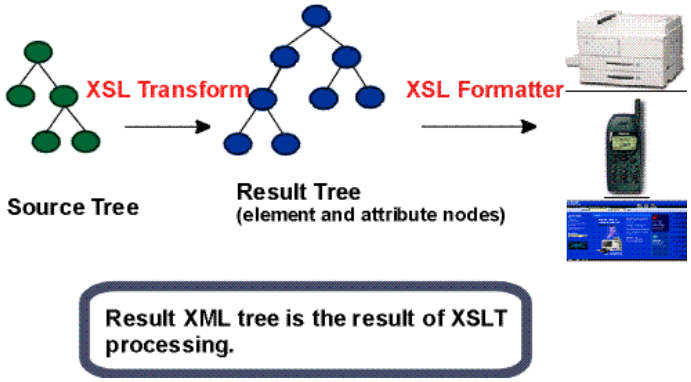
Licence Informatique L3

Plan

- Principes
- Règles XSLT
- Désignation de fragments XML
- Appel de règles
- Exemple d'exécution

XSLT

- XSLT = *eXtensible Stylesheet Language Transformations*
  - Langage de transformation de documents XML



(source: site W3C)

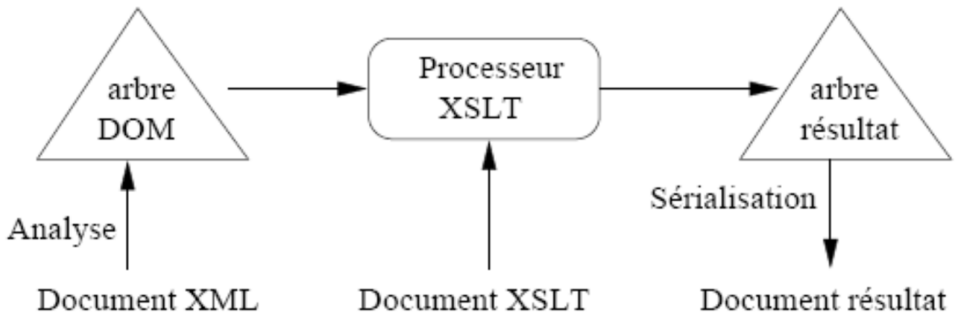
Exemple

```
<FILM>
  <TITRE>Vertigo</TITRE>
  <AUTEUR>Hitchcock</AUTEUR>
  <ANNEE>1958</ANNEE>
  <RESUME>Scotty … </RESUME>
</FILM>
```

- La fiche du film peut être publiée
  - en HTML pour *Firefox/IE*
  - en WML pour des portables *WAP*
  - en SMIL pour *Realplayer*
  - …
- Même contenu, formes différentes
  - Une feuille de style pour chaque forme

Transformation XSLT

- XSLT → transformation d'un document XML en:
  - XML
  - HTML
  - texte simple
  - …

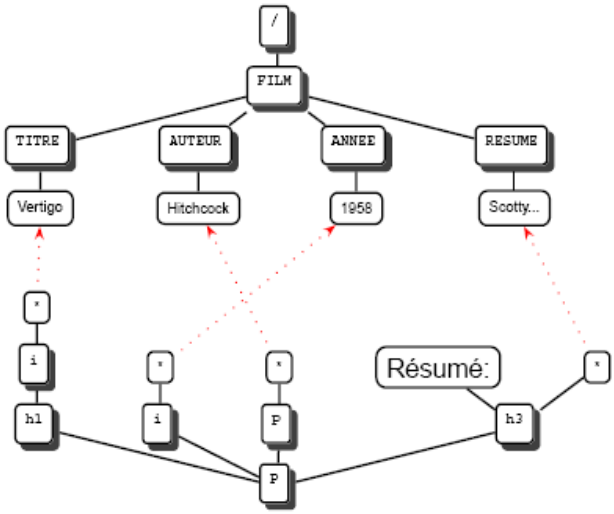


Un programme XSLT

- Programme (« feuille de style ») : ensemble de *règles*
- Une règle:
  - Appliquée à un nœud dans le document d'entrée
  - Produit un résultat dans le document de sortie
  - Opérations qu'une règle peut effectuer
    - extraction de données
    - génération de texte
    - suppression, déplacement, duplication de contenu (nœuds)
    - tri
    - appel d'autres règles
- Choix des nœuds pour appliquer les règles
  - Au début: la racine
  - Opérations réalisées sur des nœuds spécifiés par des chemins dans l'arbre à partir du nœud courant
    - XPath pour spécifier les chemins

Exemple de règle de transformation

```
<xsl:template match="FILM">
 <p>
  <h1>
   <i>
    <xsl:value-of select="TITRE"/>
   </i>
  </h1>
  <i>
   <xsl:value-of select="ANNEE"/>
  </i>
  <p>
   <xsl:value-of select="AUTEUR"/>
  </p>
  <h3>Résumé:
   <xsl:value-of select="RESUME"/>
  </h3>
 </p>
</xsl:template>
```



Fonctionnalités XSLT

- Extraction de données

```
<xsl:template match="FILM">
  <xsl:value-of select="TITRE"/>
</xsl:template>
```

- Génération de texte

```
<xsl:template match="FILM">
  Texte produit par l'application de cette règle
</xsl:template>
```

- Génération arbre XML

```
<xsl:template match="FILM">
  <body>
    <p>Un paragraphe</p>
  </body>
</xsl:template>
```

- Génération arbre avec extraction

```
<xsl:template match="FILM">
  <body>
    <p>Titre:
      <xsl:value-of select="TITRE"/>
    </p>
  </body>
</xsl:template>
```

Les règles

- Règle (*template*) : élément de base des programmes XSLT
  - une règle s'applique dans le contexte d'un noeud de l'arbre
  - l'application de la règle produit un fragment du résultat
  - la règle peut appeler d'autres règles
- Programme XSLT = ensemble de règles pour construire un résultat
- Exécution
  - La règle pour la racine produit un arbre résultat qui contient d'autres appels de règles
  - L'appel de ces règles produit d'autres fragments dans l'arbre résultat
  - On s'arrête quand l'arbre résultat ne contient plus d'appel de règle

Exemple: document XML

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

<?xml-stylesheet href="Salle.xsl" type="text/xsl"?>
<SALLE NO='2' PLACES='320'>
  <FILM>
    <TITRE>Alien</TITRE>
    <AUTEUR>Ridley Scott</AUTEUR>
    <ANNEE>1979</ANNEE>
    <GENRE>Science-fiction</GENRE>
    <PAYS>Etats Unis</PAYS>
    <RESUME>Près d'un vaisseau spatial échoué sur une lointaine
      planète, des Terriens en mission découvrent de bien étranges
      "oeufs". Ils en ramènent un à bord, ignorant qu'ils viennent
      d'introduire parmi eux un huitième passager particulièrement
      féroce et meurtrier.
    </RESUME>
  </FILM>
  <REMARQUE>Réservation conseillée</REMARQUE>
  <SEANCES>
    <SEANCE>15:00</SEANCE>
    <SEANCE>18:00</SEANCE>
    <SEANCE>21:00</SEANCE>
  </SEANCES>
</SALLE>
```

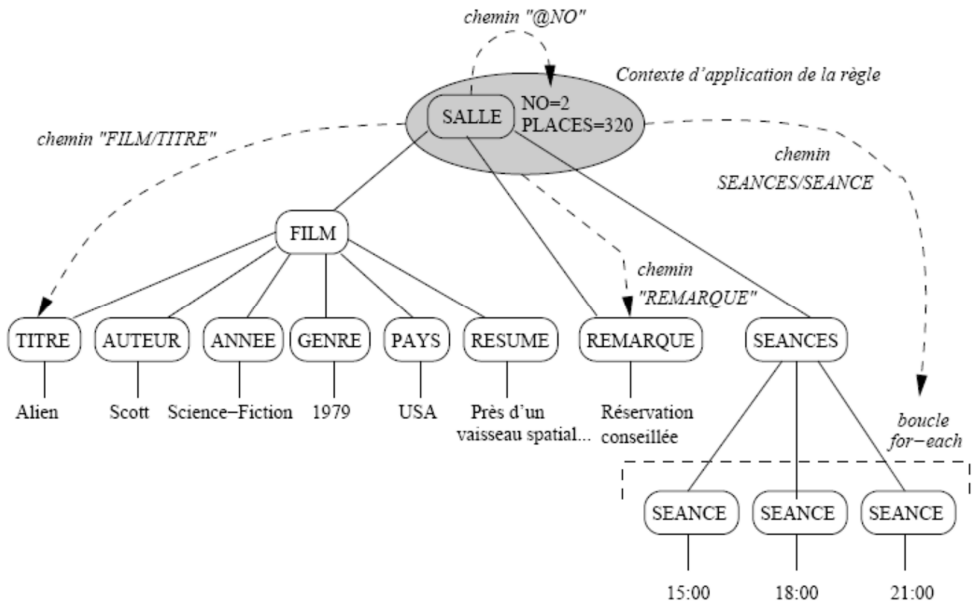
Exemple: règle avec boucle

```
<xsl:template match="SALLE">
  <h2>Salle No <xsl:value-of select="@NO"/></h2>
    Film: <xsl:value-of select="FILM/TITRE"/>
    de <xsl:value-of select="FILM/AUTEUR"/>
  <ol>
    <xsl:for-each select="SEANCES/SEANCE">
      <li><xsl:value-of select="."/></li>
    </xsl:for-each>
  </ol>
</xsl:template>
```

- Résultat : fragment HTML à intégrer dans le document résultat complet

```
<h2>Salle No 2</h2>
Film: Alien
de Ridley Scott
<ol>
  <li> 15:00</li>
  <li> 18:00</li>
  <li> 21:00</li>
</ol>
```


Exemple: illustration sur l'arbre



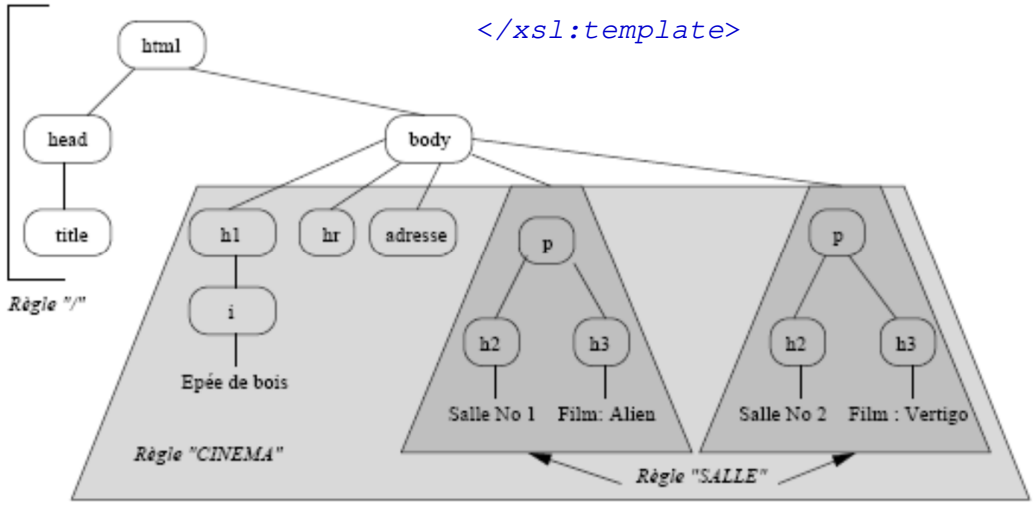
Appel de règles

- Exemple pour HTML
  - On considère un document de racine CINEMA et sous-éléménts NOM, ADRESSE et plusieurs SALLE (avec la structure de l'exemple précédent)
  - Règle `'/'` (→ cadre HTML) , qui appelle la règle `CINEMA`

```
<xsl:template match="/">
  <html>
    <head><title>Programme de
            <xsl:value-of select="CINEMA/NOM"/>
          </title>
    </head>
    <body>
      <xsl:apply-templates select="CINEMA"/>
    </body>
  </html>
</xsl:template>
```


Exemple: règle CINEMA

```
<xsl:template match="CINEMA">
  <h1><i>
    <xsl:value-of select="NOM"/>
  </i></h1>
  <hr/> <xsl:value-of select="ADRESSE"/>
  <xsl:apply-templates select="SALLE"/>
</xsl:template>
```



Programmation XSLT

- Programme (feuille) XSLT
  - Document XML, utilisant des balises spécifiques ayant l'espace de noms `xsl`
  - Élément *racine* du programme: `<xsl:stylesheet>`
- Contenu
  - Déclarations
    - Importation/inclusion d'un programme XSLT : `<xsl:import>, <xsl:include>`
    - Description du format de sortie : `<xsl:output>`
    - Définition de paramètres, de variables : `<xsl:param>, <xsl:variable>`
  - Règles: `<xsl:template>`
    - Instructions de traitement: dans le corps des règles

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="ISO-8859-1" />
  <xsl:template match="/">
    <bonjour>à tout le monde</bonjour>
  </xsl:template>
</xsl:stylesheet>
```


Appel d'un programme XSLT

- Par un outil d'exécution XSLT (ex. en ligne de commande)
- Par une API de programmation (ex. JAPX)
- Sur le web:
  - Côté client: par le navigateur web, au chargement d'un fichier XML qui fait référence à une feuille de style
  - Côté serveur: par CGI, PHP, ASP, JSP, etc.

Déclenchement de règles

- Deux possibilités:
- `xsl:apply-templates`
  - Sélectionne un ensemble de nœuds auxquels il faut appliquer des règles
  - Les règles sont identifiées à l'aide de leur « *pattern* » (l'attribut `match`)
  - Ex: `xsl:template match='FILM'`
  - Déclenchement à partir du corps d'une autre règle par `xsl:apply-templates select='...'`
    - `select`: XPath → choix des nœuds cibles
    - Si `select` est absent, on considère *tous les descendants directs!*
- `xsl:call-template`
  - Applique une règle (identifiée par un *nom* – attribut `name`) au nœud courant
  - Ex: `xsl:template name='TDM'`
  - Déclenchement à partir du corps d'une autre règle par `xsl:call-template name='...'`

Sélection des règles par leur *pattern*

- Problème : étant donné un nœud N sélectionné lors de `xsl:apply-templates`, comment savoir si une règle s'applique?
  - Soit N le nœud et soit une règle R de *pattern* P
  - S'il existe un nœud contexte à partir duquel l'expression de chemin P mène à N → **la règle R s'applique à N**
- Restrictions sur les patterns pour simplifier la vérification
  - Axes autorisés: *child*, *attribute* et //
  - Pas de restriction sur les filtres et les prédicats

  → Conséquence: il suffit de tester les nœuds contexte ancêtres de N
- Question en suspens: si *plusieurs* règles s'appliquent à un nœud, laquelle choisir ?
  - Notion de priorité

Règles par défaut

- Appliquées quand aucune règle n'existe pour un noeud
- Règle pour les éléments et la racine du document

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

→ application des règles pour les fils du nœud courant

- Règle pour nœuds texte et attributs

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

→insertion dans le résultat de la valeur du nœud texte ou de l'attribut

- Règle pour instructions de traitement et commentaires: on ne fait rien

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


Règles par défaut: conséquence

- Sans aucune règle: seules les règles par défaut s'appliquent
  - Résultat: concaténation des nœuds de type *Text*
  - Remarque: le programme sans règles n'affiche pas les attributs
- Programme minimal

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

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

L'instruction `xsl:apply-templates`

- Attributs : `select`, `mode`
  - `select` doit sélectionner un **ensemble de nœuds** (contexte d'évaluation)
    - pour chaque nœud on va chercher la règle à instancier
  - `mode` permet de choisir explicitement une des règles parmi les candidates
- Sélection d'une règle
  - Si le pattern et le mode ne permettent pas de choisir une règle → utilisation des priorités
  - Priorités: implicites ou explicites
  - Si le choix est impossible → le programme s'arrête

Exemple de choix de règles

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

<FILMS>
  <FILM>
    <TITRE>Vertigo</TITRE>
    <ANNEE>1958</ANNEE><GENRE>Drame</GENRE>
    <MES>Alfred Hitchcock</MES>
    <RESUME>Scottie Ferguson, ancien inspecteur de
      police, est sujet au vertige depuis qu'il a vu
    </RESUME>
  </FILM>
  <FILM>
    <TITRE>Alien</TITRE>
    <ANNEE>1979</ANNEE><GENRE>Science-fiction</GENRE>
    <MES>Ridley Scott</MES>
    <RESUME>Près d'un vaisseau spatial échoué sur
      une lointaine planète, des Terriens en mission
    </RESUME>
  </FILM>
</FILMS>
```

Exemple: programme XSLT

- Programme qui efface les nœuds de type RESUME

```
<xsl:template match="RESUME"/>

<xsl:template match="@*|node()" priority="-1">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

- Priorités implicites
  - Intuition: priorité aux patterns les plus spécifiques
  - Priorité 0: patterns constitués d'une seule étape XPath, avec un nom d'élément ou d'attribut et sans prédicat
  - Priorité -0.5: filtres autres qu'un nom d'élément ou d'attribut (node(), *)
  - Priorité 0.5: tous les autres (prédicats, plusieurs étapes)

Les modes

- Besoin : traiter un même noeud plusieurs fois
  - Il faut des règles différentes s'appliquant aux même nœuds
  - On les distingue par le mode
- Exemple :
  - On parcourt tous les chapitres et sections pour produire une table des matières
  - On les parcourt à nouveau pour publier le contenu

Exemple d'utilisation des modes

- Création de liens HTML
  - On peut créer des ancres internes à un document

    ```
    <a name='Alien'/>
    ```
  - On peut ensuite créer des liens vers cette ancre

    ```
    <a href='#Alien'>Lien vers le film Alien</a>
    ```
- Programme
  - On crée une table de matières (liens vers les ancres des films), suivie d'une description des films.
  - Une règle pour créer les descriptions de film + ancres, une autre pour créer les liens
  - Une règle qui active successivement les deux règles ci-dessus

Exemple: les deux règles sur FILM

```
<xsl:template match="FILM" mode="Liens">
  <a href="#{TITRE}">
    <xsl:value-of select="TITRE"/>
  </a>
</xsl:template>


<xsl:template match="FILM">
  <a name="{TITRE}"/>
    <h1><xsl:value-of select="TITRE"/></h1>
    <b><xsl:value-of select="TITRE"/>,</b>
    <xsl:value-of select="GENRE"/>
    <br/>
    <b>Réalisateur</b> : <xsl:value-of select="MES"/>
 </xsl:template>
```

Exemple: la règle d'appel

```
<xsl:template match="FILMS">
  <html>
    <head>
      <title>Liste des films</title>
    </head>
    <body>
      <xsl:apply-templates select="FILM" mode="Liens"/>
      <xsl:apply-templates select="FILM"/>
    </body>
  </html>
</xsl:template>
```


Paramètres

- Dans la feuille de style
  - Valeur transmise par une méthode dépendante du contexte d'utilisation
  - Attribut `select` : valeur par défaut

```
<xsl:param name="age" select ="0"/>
```

- Dans une règle appelée avec `apply-templates` ou `call-template`

```
<xsl:template name="appelé">
  <xsl:param name="age" select ="0"/>
  ...
</xsl:template>

<xsl:template match="*">
  <xsl:call-template name="appelé">
    <xsl:with-param name="age" select="20"/>
  </xsl:call-template>
  ...
</xsl:template>
```

Instruction conditionnelle

- xsl:if

```
<xsl:template match ="FILM" >
  <xsl:if test="ANNEE &lt;1970">
    <xsl:copy-of select ="."/>
  </xsl:if>
</xsl:template>
```

  - *Remarque*: `xsl:copy-of` copie le nœud avec tout le sous-arbre en dessous, tandis que `xsl:copy` copie seulement le nœud
  - Pas de *else* dans XSLT, utiliser `xsl:choose`

- xsl:choose

```
<xsl:choose>
  <xsl:when test="GENRE='Comédie'">C</xsl:when>
  <xsl:when test="GENRE='Drame'">D</xsl:when>
  <xsl:when test="GENRE='Science-fiction'">SF</xsl:when>
  <xsl:otherwise>?</xsl:otherwise>
</xsl:choose>
```


Boucle

- xsl:for-each

```
<xsl:for-each select ="FILM">
  <xsl:sort select ="ANNEE" order ="ascending"
            data-type="number"/>
  <xsl:value-of select ="TITRE" />
  sorti en
  <xsl:value-of select ="ANNEE" />
</xsl:for-each>
```

  - `xsl:sort` (optionnel) trie les nœuds produits par le `select` du `xsl:for-each`
  - `xsl:sort` peut être aussi utilisé dans un `xsl:apply-templates` pour trier les nœuds sur lesquels se fait l'appel

Autres fonctions utiles

- Nœud courant: `current()`
  - Dans une règle: nœud auquel s'applique la règle à l'exécution
  - Dans une boucle: nœud courant de l'itération

```
<xsl:apply-templates select="//seance[film/@ref=current()/@id]" />
```

- Espaces
  - `xsl:strip-space`: pour enlever les nœuds texte ne contenant que des espaces
  - `xsl:preserve-space`: pour spécifier des exceptions

```
<xsl:strip-space elements="*" />
<xsl:preserve-space elements="SEANCES" />
```

- Texte: `xsl:text`
  - Utile pour rajouter du texte qui ne peut pas être écrit directement (ex. espaces)

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

- Création dynamique d'éléments et attributs

```
<xsl:element name="{concat('FILM', GENRE)}">
  <xsl:attribute name="annee">
    <xsl:value-of select ="ANNEE" />
  </xsl:attribute>
</xsl:element>
```

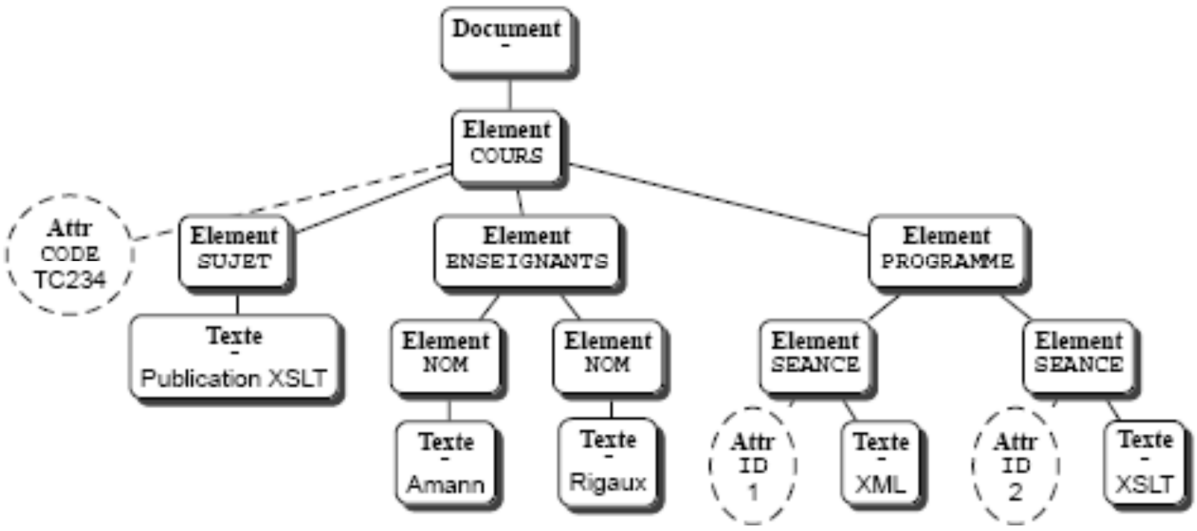
  - L'expression entre { } est évaluée d'abord

Exemple d'évaluation d'un programme XSLT

- Document source

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

<COURS CODE="TC234">
  <SUJET>Publication XSLT</SUJET>
  <ENSEIGNANTS>
    <NOM>Amann</NOM>
    <NOM>Rigaux</NOM>
  </ENSEIGNANTS>
  <PROGRAMME>
    <SEANCE ID="1">XML</SEANCE>
    <SEANCE ID="2">XSLT</SEANCE>
  </PROGRAMME>
</COURS>
```

Document source en DOM

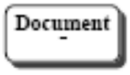


Règles du programme XSLT

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


<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>


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


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

Résultat initial



La règle principale

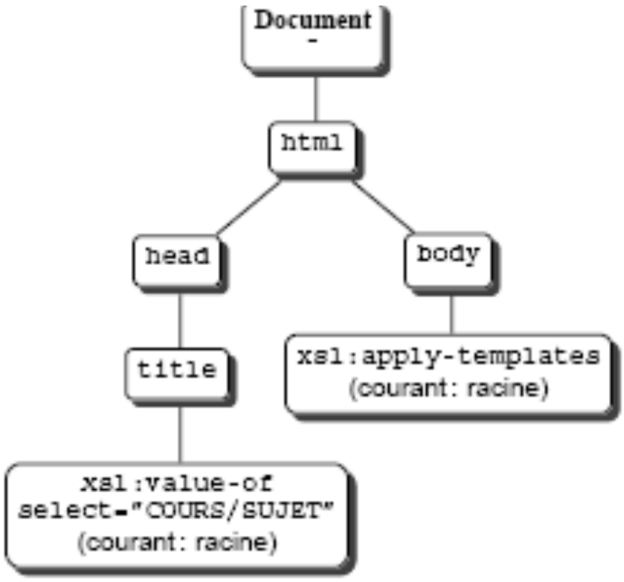
- On démarre avec la racine

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

→ Génération du cadre HTML

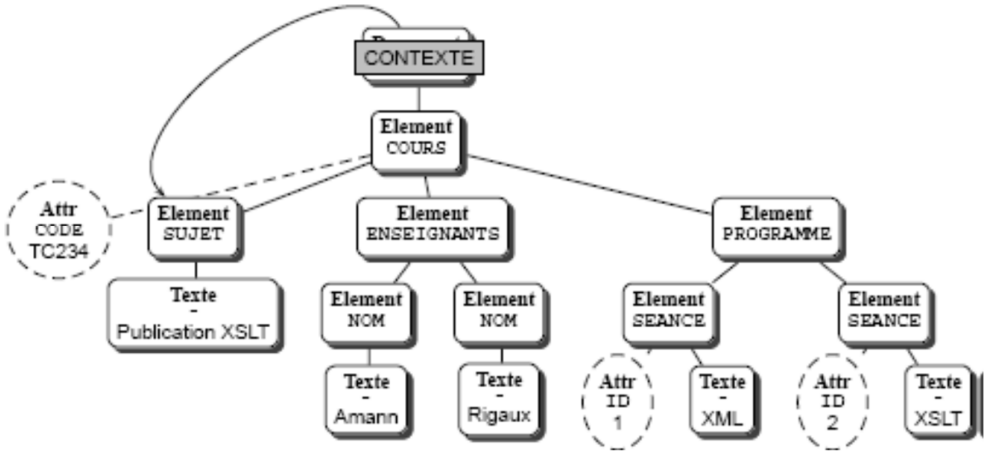
Ajout du corps de la règle principale

- La règle s'applique à la racine du document → contexte pour XPath

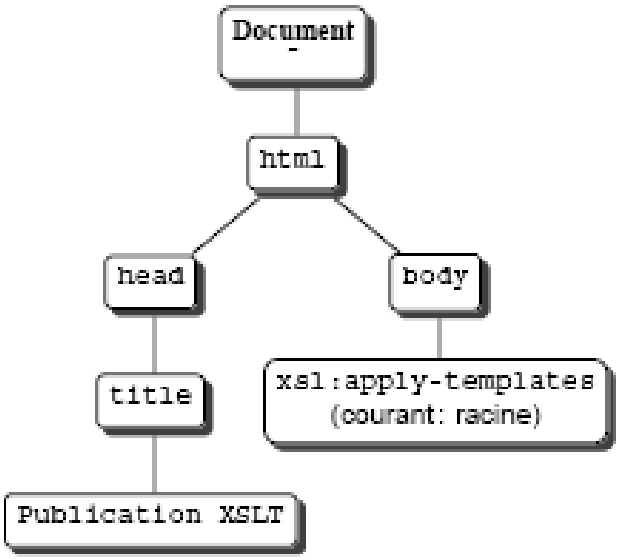


Évaluation de *xsl:value-of*

```
<xsl:value-of select="COURS/SUJET"/>
```

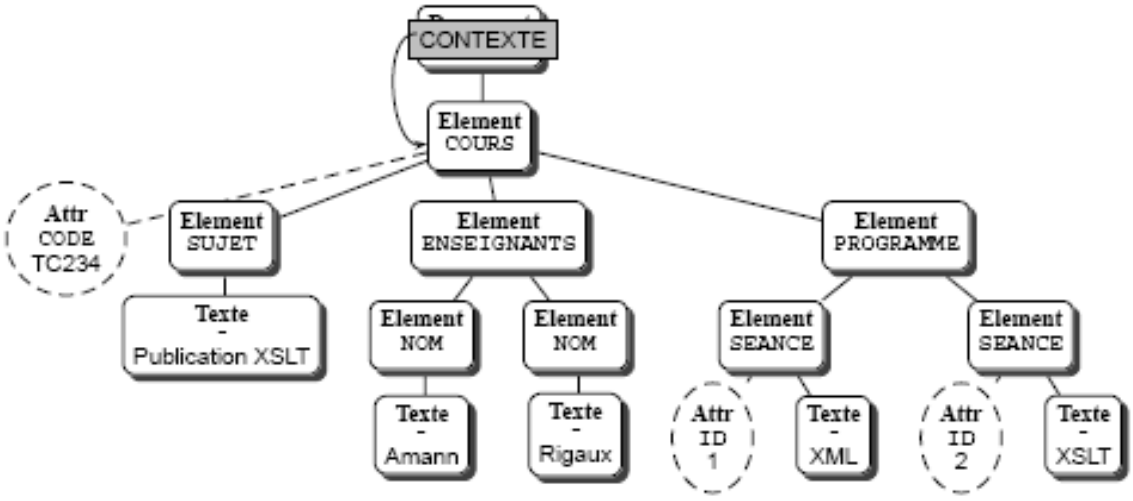


Résultat après évaluation de *xsl:value-of*



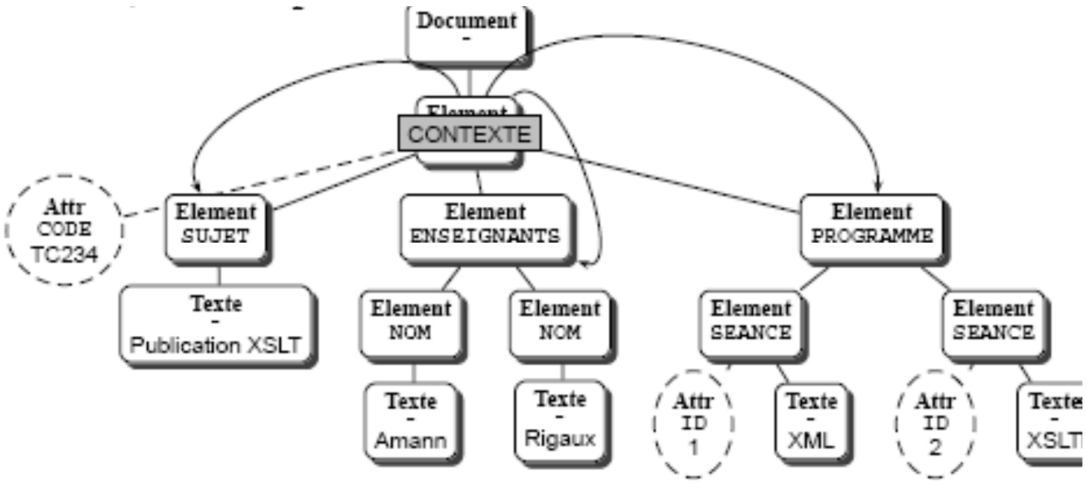
Évaluation de *xsl:apply-templates*

```
<xsl:apply-templates select="child::node()"/>
```



Règle par défaut

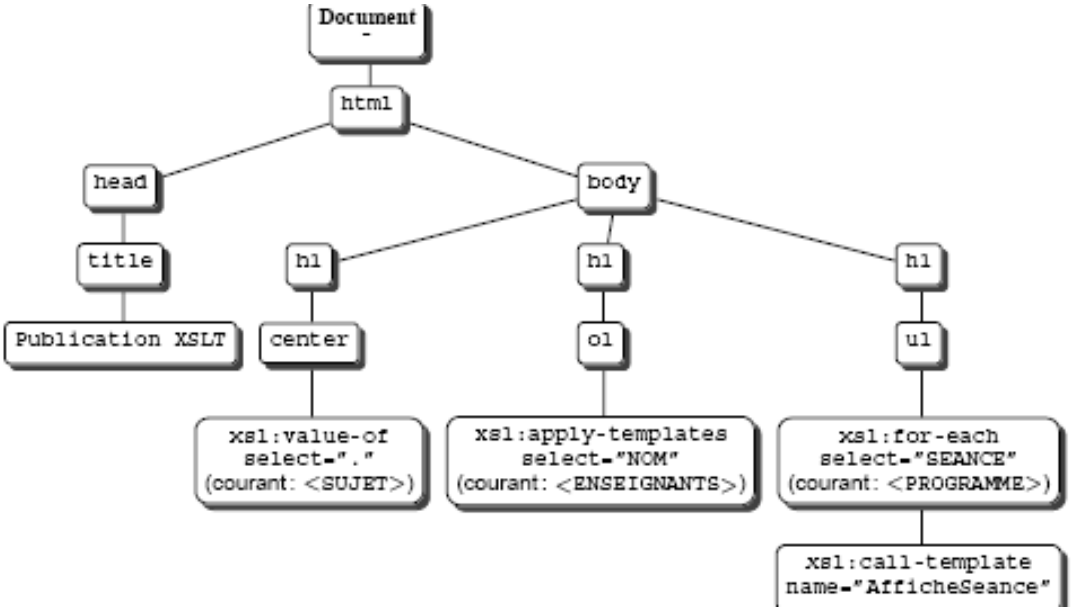
```
<xsl:template match="*|/">
  <xsl:apply-templates />
</xsl:template>
```



Règles pour SUJET, ENSEIGNANTS, PROGRAMME

```
<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>
```

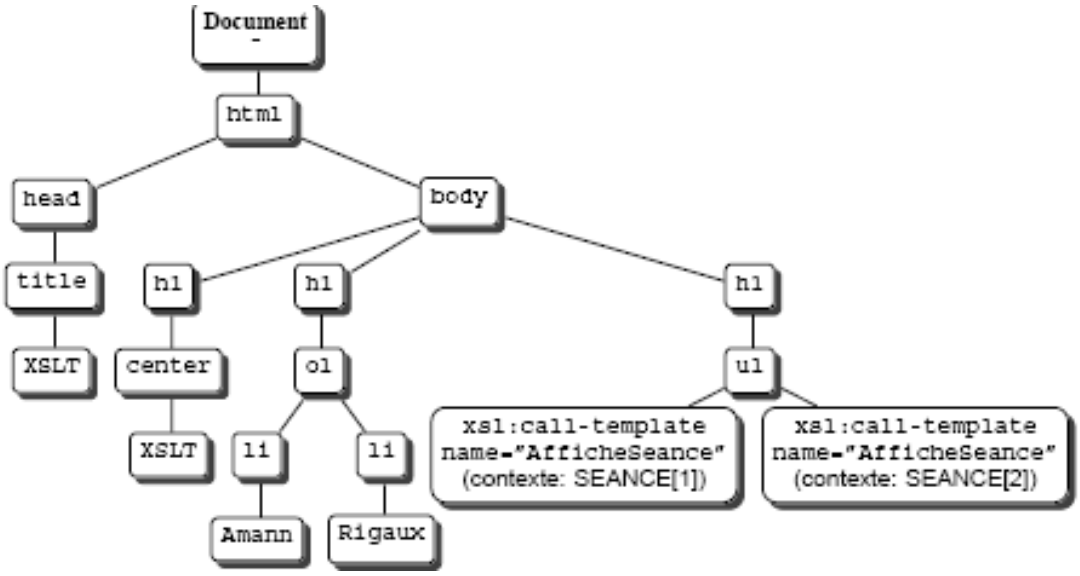
Application des règles



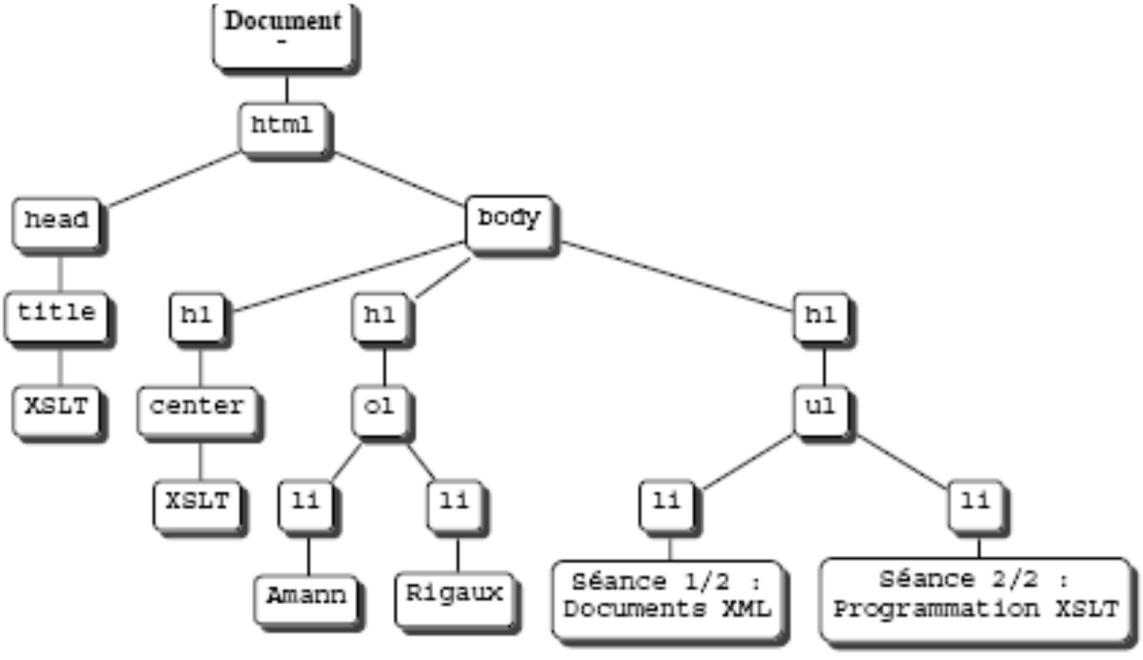
Les règles pour les noms et les séances

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


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

Application aux noms



Résultat final



Conclusions XSLT

- Langage totalement adapté au traitement de documents XML
  - Parcours d'un document, vu comme un arbre
  - Déclenchement de règles sur certains nœuds
  - Structure récursive → programmes récursifs
  - Association de plusieurs programmes à un même document

Bibliographie spécifique

- XSLT sur le site W3C
  `http://www.w3.org/TR/xslt`
- B. Amann, P. Rigaux, *Comprendre XSLT*, O'Reilly