XPath et XSLT

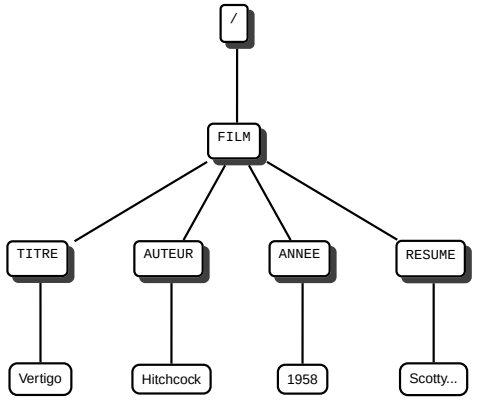
Bernd Amann et Philippe Rigaux

January 19, 2004

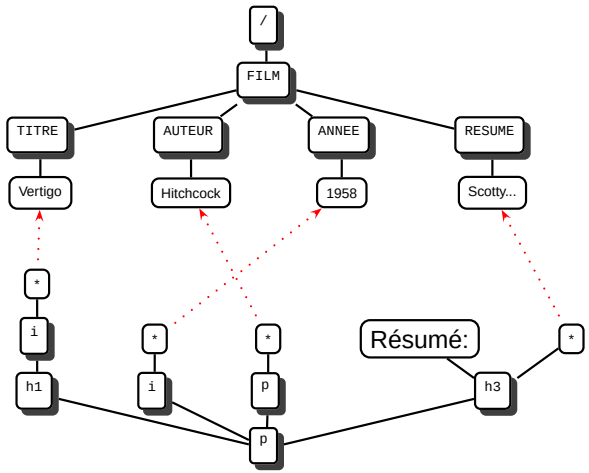
Sélectionner des Fragments XML avec XPath

Exemple: Document XML

Document XML:

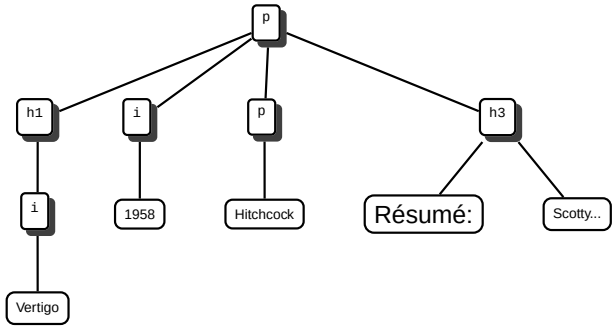


Exemple: Transformation d'un document XML



Exemple: Résultat HTML

Fragment HTML:



XPATH

Il faut pouvoir extraire des fragments (nœuds) XML d'un document ⇒ XPath.

XPath est utilisé par

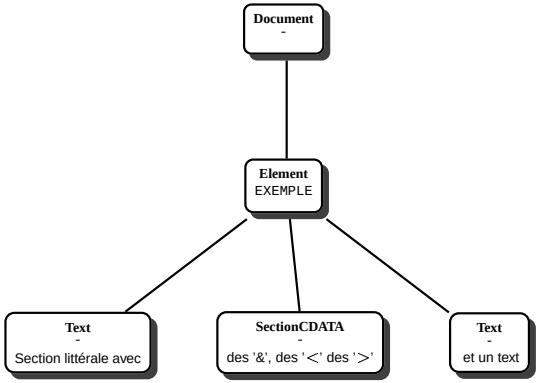
- XSLT pour sélectionner des fragments XML à transformer
- XQuery pour l'interrogation de documents XML
- XML Schéma pour créer des clés et références
- XLink pour créer des liens entre documents/fragments XML

XPath: Le modèle

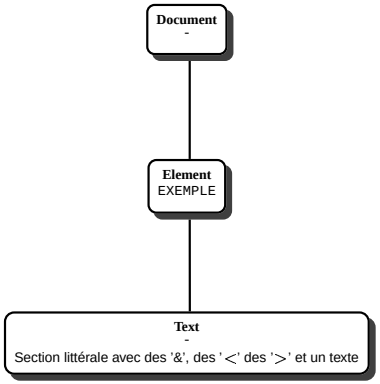
Le langage permet de désigner un ou plusieurs nœuds dans un document XML, à l'aide **d'expressions d'arbres**.

- XPath est fondé sur une représentation arborescente (DOM) du document XML
- Objectif : référencer nœuds (éléments, attributs, commentaires, ...) dans un document XML
- Un typage simplifié par rapport à celui de DOM ⇒ pas d'entité, pas de section littérale

Exemple : l'arbre DOM...



Puis l'arbre XPath...



Expressions XPath : Exemples

- /
- /child::FILM
- /FILM
- /descendant::ACTEUR
- /child::FILM[child::TITRE='Vertigo']
- /FILM[TITRE='Vertigo']
- /child::FILM[child::TITRE='Vertigo']/descendant::ACTEUR
- /child::FILM[child::TITRE='Vertigo']/following::FILM

Expressions XPath : Sémantique

Une expression XPath :

- s'évalue en fonction d'un **nœud contexte**
- désigne un ou plusieurs chemins dans l'arbre à partir du nœud contexte
- a pour résultat
  - un ensemble de nœuds
  - ou une valeur, numérique, booléenne ou alphanumérique

Expressions XPath : Syntaxe

Un chemin XPath est une suite **d'étapes** :

[/]étape$_1$/étape$_2$/.../étape$_n$

Deux variantes : Un chemin peut être

- **absolu** : `/child::FILM/child::RESUME`
  Le nœud contexte est alors la racine du document.
- ou **relatif** : `child::RESUME/child::text()`
  Le nœud contexte est un nœud dans le document (pas forcément la racine).

Étapes XPath

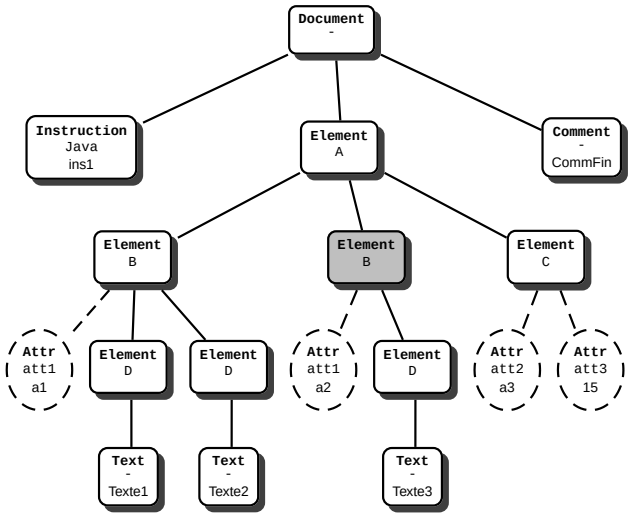
Une étape : trois composants

[axe::]filtre[[prédicat1]][[prédicat2]]...

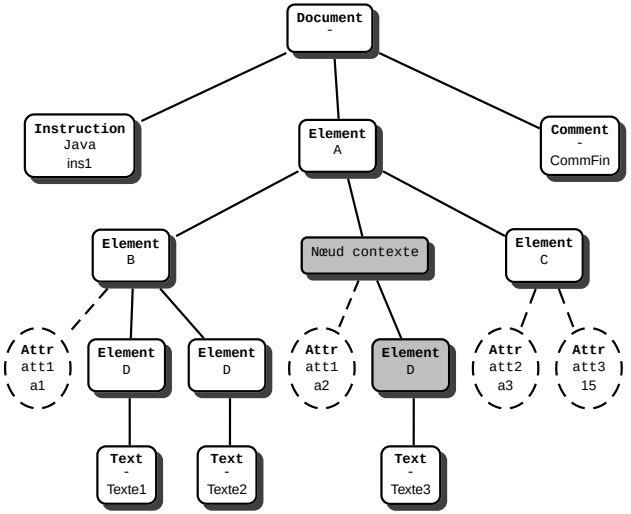
- **l'axe** : sens de parcours des nœuds (par défaut: *child*).
- **le filtre** : type/noms des nœuds qui seront retenus
- **le(s) prédicat(s)** que doivent satisfaire les nœuds retenus

Etape par défaut : descendant-or-self::node()

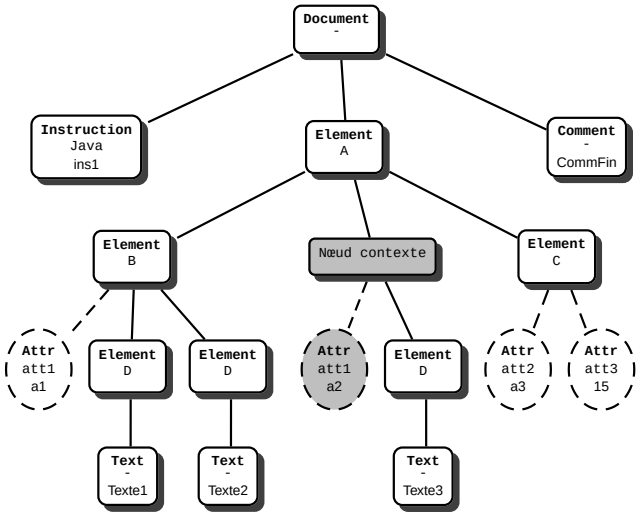
Document XML avec un nœud contexte



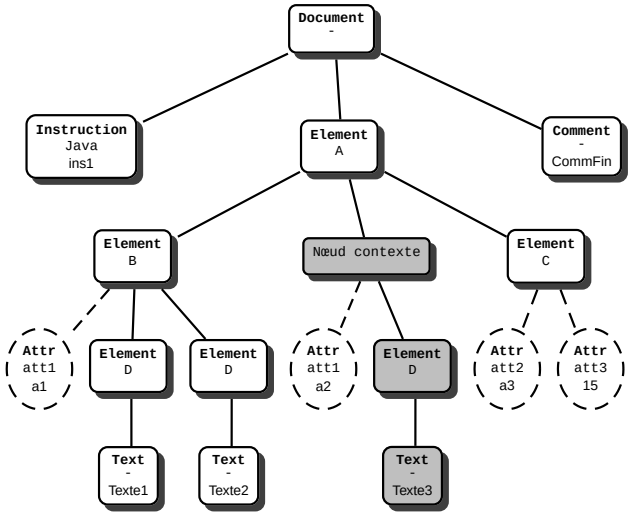
`child::D` ou `D`



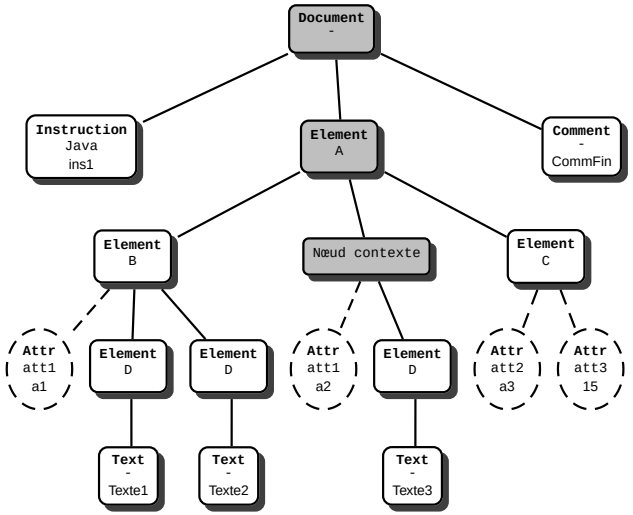
`attribute::att1` ou `@att1`



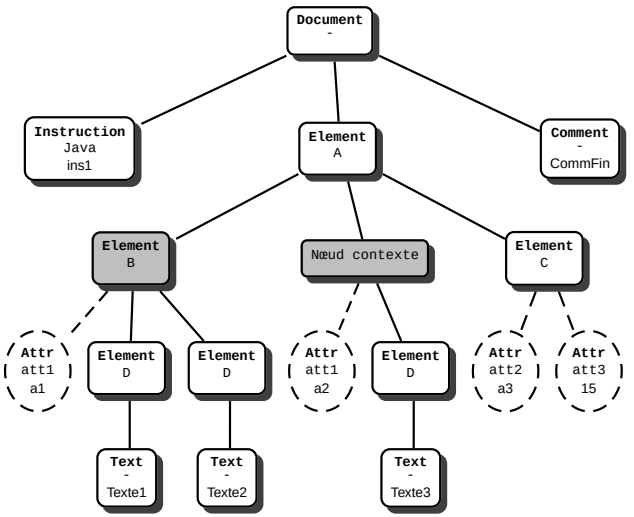
descendant::node()



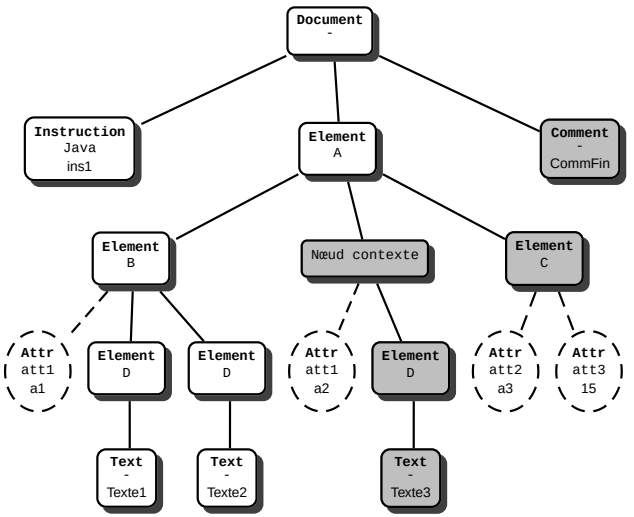
ancestor::node()



preceding-sibling::node()



following::node()



Axes: Résumé

- nœuds enfants et descendants: `child, descendant, descendant-or-self`
- nœud parent et ancêtres: `parent, ancestor`
- frères: `preceding-sibling, following-sibling`
- nœuds précédents et suivants: `preceding, following`
- nœud même: `self`

Filtres

Deux manières de filtrer les nœuds :

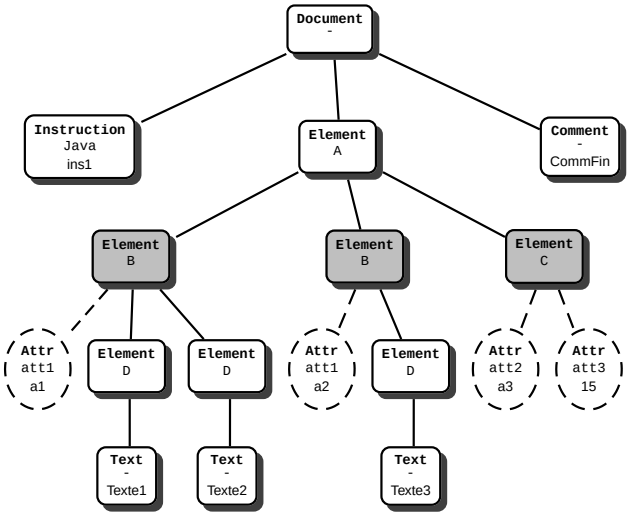
- par leur nom :
  - possible pour les types de nœuds qui ont un nom : **Element**, **ProcessingInstruction** et **Attr**
- par leur type DOM.

Filtrage sur le type de nœud

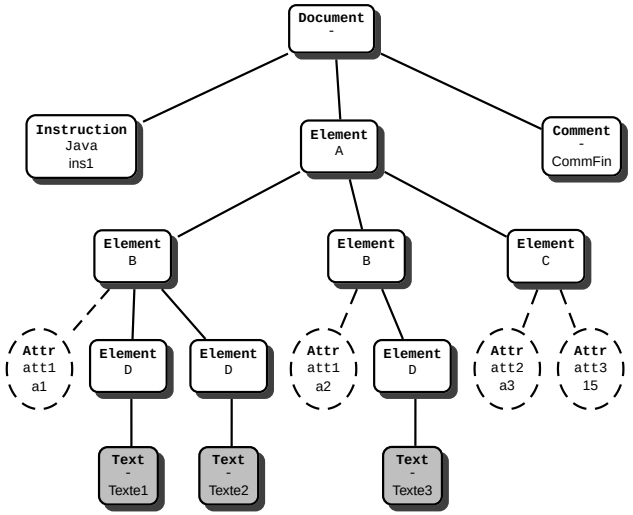
- `*` : Tous les nœuds de type **Element** ou **Attr**
- `text()` : Tous les nœuds de type **Text**
- `comment()` : Tous les œuds de type **Comment**
- `processing-instruction()` : Tous les nœuds de type **ProcessingInstruction**

  Exemple : `/processing-instruction()`, ou `/processing-instruction('java')`
- `node()` : Tous les nœuds

Filtrage par le type : `/A/*`



/A/B//text()



Notation abrégées « . » et « .. »

- L'expression "." désigne l'étape self::node() : le nœud contexte, *quel que soit son type*.
- La notation abrégée "`..`" désigne l'étape parent::node() : le père du nœud contexte, *quel que soit son type*.

Prédicats

- **Prédicat** : expression booléenne constituée d'un ou plusieurs tests, composés avec les connecteurs logiques habituels and et `or`
- **Test** :
  - toute expression XPath, dont le résultat est convertie en booléen;
  - une comparaison, un appel de fonction.

Exemple

Dans l'expression `/A/B[@att1]` :

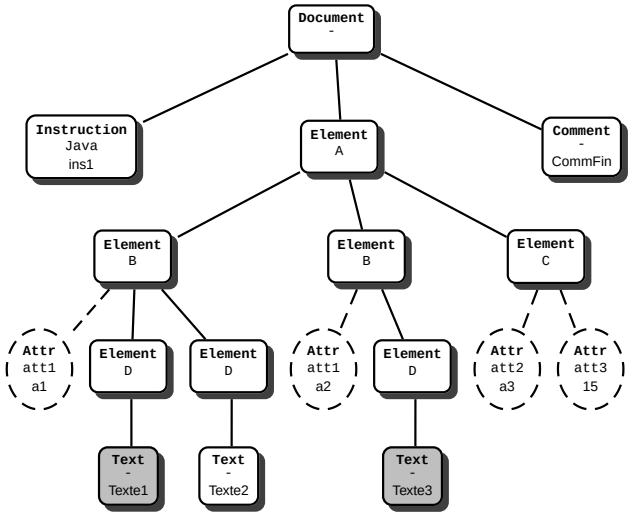
- On s'intéresse aux nœuds de type `B` fils de l'élément racine `A`.
- Parmi ces nœuds on ne prend que ceux pour lesquels le prédicat `[@att1]` s'évalue à `true`
- Cette expression s'évalue avec pour nœud contexte un élément `B`
- `[@att1]` vaut `true` ssi `@att1` renvoie un ensemble de nœuds **non vide**

Prédicats et contexte d'évaluation

Une étape s'évalue en tenant compte d'un **contexte** constitué de

- un nœud contexte, position initiale du chemin ;
- ce nœud fait lui-même partie d'un ensemble obtenu par évaluation de l'étape précédente
  - on connaît la **taille** de cet ensemble (fonction *last*())
  - on connaît la **position** du nœud contexte dans cet ensemble (fonction *position*())

`/A/B/descendant::text()[position()=1]`



Typage avec XPath

On peut effectuer des comparaisons, des opérations. Cela implique un **typage** et des conversions de type.

Types XPath :

- *les numériques*
- *les chaînes de caractères*
- *les booléens* (`true` et `false`)
- enfin *les ensembles de nœuds*

Les conversions vers une **chaîne de caractère** ou vers un **booléen** sont **toujours possibles**.

Numériques

Notation décimale habituelle

- Comparaisons habituelles (<, >, !=)
- Opérations : +, -, *, div, mod
- La fonction *number*() permet de tenter une conversion
- Si la conversion échoue on obtient `NaN` (*Not a Number*). **À éviter...**

Ex : `//node()[number(@att1) mod 2=1]`

Conversions booléennes

- **Pour les numériques** : 0 ou `NaN` sont `false`, tout le reste est `true`
- **Pour les chaînes** : une chaîne vide est `false`, tout le reste est `true`
- **Pour les ensembles de nœuds** : un ensemble vide est `false`, tout le reste est `true`

Comparaisons : des règles de conversion étranges...

Fonctions XPath

Quelques fonctions utiles dans les prédicats :

- *concat(chaîne1, chaîne2, ...)* pour concaténer des chaînes
- *contains(chaîne1, chaîne2)* teste si *chaîne1* contient *chaîne2*
- *count (expression)* renvoie le nombre de nœuds désignés par *expression*
- *name()* renvoie le nom du nœud contexte
- *not(expression)* : négation

Axes d'avancements et axes inverses

La position d'un nœud dépend de l'axe choisi.

Axes « d'avancement » :

- `child::*[3]`: le 3e enfant
- `child::*[position()=3]`: idem
- `child::*[last()]`: le dernier enfant

Axes « inverses » :

- `ancestor::*[1]`: le premier ancêtre du nœud contexte (dernier dans l'ordre du document).
- `preceding-sibling::*[last()]`: le dernier frère qui précède le nœud contexte (premier dans l'ordre du document).

Prédicats: Exemples

Test du nombre d'occurrences :

- `CINEMA[count(SEANCE) > 1]`: cinémas avec au moins 2 séances
- `FILM[count(ACTEUR) = 0]` ou `FILM[not(ACTEUR)]`: films sans acteur

Sélection par valeur:

- `FILM[ACTEUR/NOM='Willis']` ou `FILM[ACTEUR[NOM='Willis']]`: films avec Bruce Willis
- `FILM[not(ACTEUR[NOM='Willis'])]`: films sans Bruce Willis

Prédicats: Exemples

Test sur la structure : chemins imbriqués avec connecteurs logiques (qualifiers)

- `ACTEUR[NOM and DATENAISSANCE]` ou `ACTEUR[NOM][DATENAISSANCE]`: les acteurs avec un nom et une date de naissance
- `FILM[@TITRE = 'Brazil' and ACTEUR/NOM = 'De Niro']`: le film Brazil avec l'acteur De Niro
- `FILM[@TITRE = 'Brazil'][ACTEUR/NOM = 'De Niro']`: idem

La fonction `position()`

Attention: Les deux premières expressions sont equivalentes, mais pas la troisième! Pourquoi?

- `FILM[ACTEUR and position()=1]`
- `FILM[position()=1][ACTEUR]`
- `FILM[ACTEUR][position()=1]`

XPath : Résumé

XPath est un langage pour extraire des noeuds dans un arbre XML :

- On navigue dans l'arbre grâce à des axes de navigation.
- Un chemin de navigation est une séquence d'étapes.
- Dans chaque étape on choisi un axe, un filtre et éventuellement des prédicats.
- Le résultat d'une étape (d'une séquence d'étapes) est un séquence de noeuds.

Publication de données avec XSLT

Publication de documents XML

Objectif: **Séparer** la gestion du contenu de la présentation

- Gestion du contenu => décrire nos informations, avec un vocabulaire XML
- Présentation => mettre en forme nos documents pour une application particulière

Le langage XSLT permet d'écrire des programmes de conversions, très adaptés au traitement de documents XML

Application Cinéma

Nous avons décrit notre cinéma avec notre propre format XML.
XSLT va permettre de traduire ce langage vers d'autres formats :

- HTML pour la présentation Web standard
- WML pour la présentation WAP
- SMIL pour une présentation multimédia
- XSL-FO pour la production de documents papier

Version HTML

HTML revisité :

- Un document HTML **est** un document XML
- Le vocabulaire est fixé, ainsi que la syntaxe
- Chaque balise a une signification bien définie

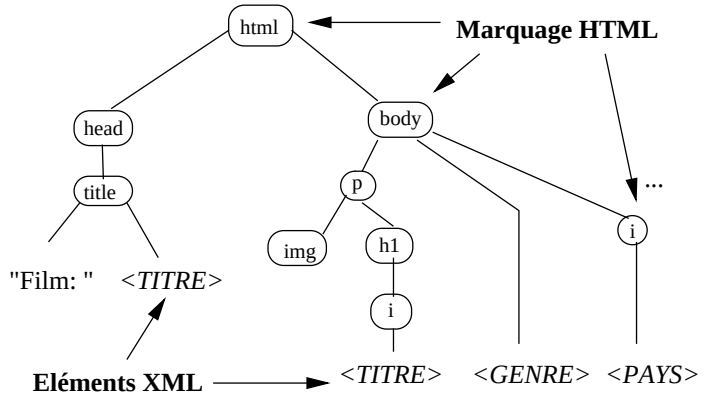
HTML a été normalisé comme « dialecte » XML
=> c'est XHTML

Ce qu'on veut obtenir

(démo)

```
<html>
  <head>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
     <title>Film: Vertigo</title>
   </head>
   <body bgcolor="white">
      <p><img SRC="Vertigo.gif" align="left" height="220">
         <h1><i>Vertigo</i></h1>Drame, <i>Etats Unis</i>, 1958
      </p><p>
         Mis en sc&egrave;ne par  <b>Alfred Hitchcock</b>
        <h3>R&eacute;sum&eacute;</h3>Scottie Ferguson,
            ancien inspecteur de police, est sujet
            au vertige depuis qu'il a vu mourir son
            coll&egrave;gue. Elster, son
            ami, le charge de surveiller sa femme, Madeleine, ayant des
            tendances suicidaires. Amoureux de la jeune femme Scottie ne
            remarque pas le pi&egrave;ge qui se trame autour de lui
            et dont il va &ecirc;tre la victime...
      </p>
   </body>
</html>
```

HTML, sous forme d'arbre



Le rôle de XSLT

XSLT permet :

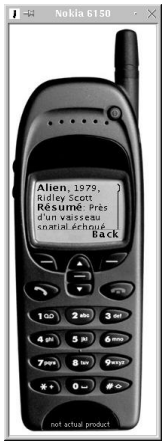
- De prendre en entrée un **document XML source**
- De produire en sortie un autre arbre XML
- D'insérer dans le document en sortie des fragments du document source

Donc bien adapté à une transformation XML -> HTML

WML, autre dialecte de XML

- Document WML : marqué par la balise `<wml>`
- Il est divisé en *cartes*, unité d'affichage sur le mobile (`<card>`)
- Elements principaux :
  - des balises simples de mise en forme (`<b>`, `<i>`)
  - des *ancres* pour passer d'une carte à une autre

Exemple d'une carte WML




```
<?xml version="1.0"
      encoding="iso-8859-1"?>
<wml>
  <card>
    <p>
      <b>
        Alien
      </b>
      , 1979, Ridley Scott
    <br/>
    Près d&apos;un vaisseau spatial
    échoué sur une lointaine
    planète, ..
    </p>
  </card>
</wml>
```

Un site WAP

On envoie un **ensemble** de cartes :

- Dotées d'une **identité** :

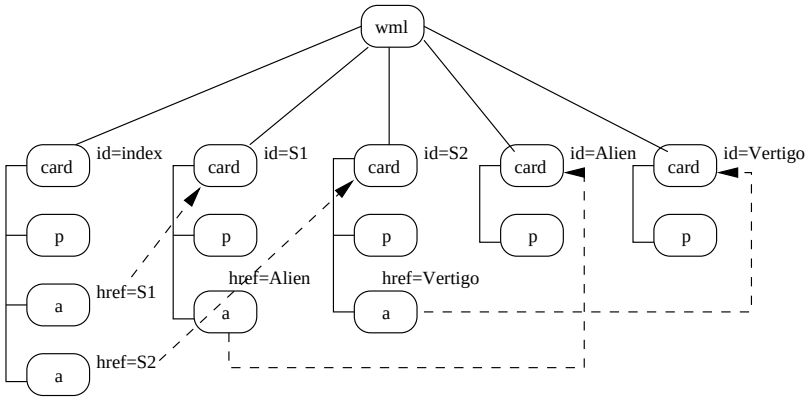
  ```
  <card id="Alien"> ...  suite de la carte
  </card>
  ```

- **Référençant** d'autres cartes :

  ```
  <a href="#Alien">lien vers la carte Alien</a>
  ```

Les cartes sont « compilées » et transmises par le réseau sans fil.

Arbre XML du site WML



Le document

```
<wml>
  <card id="index" title="Programme">
      ...
      <a href="#S1"> Salle 1: </a>
      ...
  </card>
  <card id="S1">
      S&#xE9;ances salle 1  <p>
      <a href="#Alien"> Film : Alien</a>
      ...
  </card>
  <card id="Alien">
      ...
  </card>
</wml>
```


Qu'est-ce qu'on peut faire avec XSLT?

Transformer un document XML en

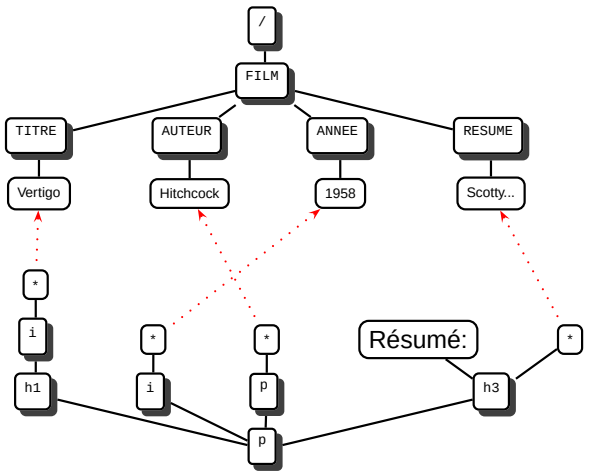
- un ou plusieurs documents XML, HTML, WML, SMIL
- document papier: PDF (XSL-FO), LaTeX
- texte simple

Fonctions d'une Feuille de Style XSLT

Fonction de Base: Langage de règles de transformation de documents/arbres XML:

- extraction de données
- génération de texte
- suppression de contenu (noeuds)
- déplacer contenu (noeuds)
- dupliquer contenu (noeuds)
- trier

Exemple



Exemple: Règle de transformation

```
1  <xsl:template match="FILM">
2    <p>
3      <h1>
4        <i>
5         <xsl:value-of select="TITRE"/>
6        </i>
7      </p>
8      <i>
9        <xsl:value-of select="ANNEE"/>
10     </i>
11     <p>
12       <xsl:value-of select="AUTEUR"/>
13     </p>
14     <h3>Résumé:
15         <xsl:value-of select="RESUME"/>
16     </h3>
17   </p>
18 </xsl:template>
```


Extraction de données

- Exemple : recherche du titre pour le nœud `FILM` :

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

- Plus généralement : on donne un chemin d'accès XPath à un nœud à partir du nœud courant (noeud contexte).

Génération de texte

Produire une phrase quand on rencontre un nœud `FILM` :

```
<xsl:template match="FILM">
 Ceci est le texte produit par
 application de cette règle.
</xsl:template>
```


Génération d'un arbre XML

Produire un document/fragment/arbre XML quand on rencontre un nœud `FILM` :

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


Génération avec extraction

Produire un document/fragment/arbre XML quand on rencontre un nœud `FILM` :

```
1 <xsl:template match="FILM">
2    <body>
3      <p>
4        <xsl:text>Titre: </xsl:text>
5        <xsl:value-of select="TITRE"/>
6      </p>
7    </body>
8 </xsl:template>
```


Structure de base : les règles

Règle = *template* : élément de base pour produire le résultat

- une règle s'applique dans le contexte d'un nœud de l'arbre
- l'application de la règle produit un fragment du résultat.

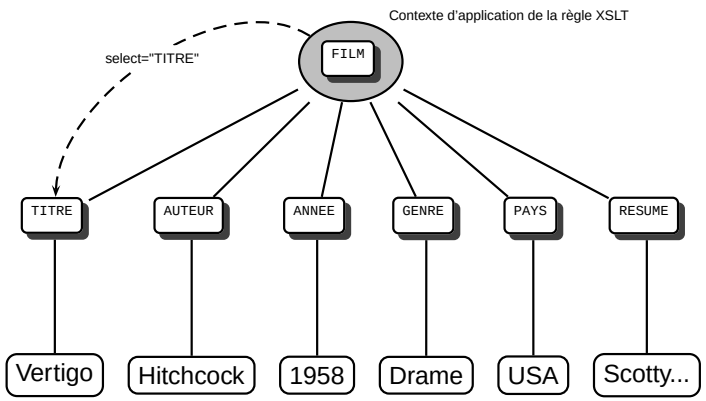
Programme XSLT = ensemble de règles pour construire un résultat

Un exemple

```
<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>
```

- **Motif de sélection** : `match="FILM"`
- **Corps de la règle** = fragment d'arbre à produire

Illustration



Une règle complète

```
<xsl:template match="FILM">
 <html>
   <head>
    <title>Film:
      <xsl:value-of select="TITRE"/>
    </title>
   </head>
   <body>
     Le genre du film, c'est
     <b><xsl:value-of select="GENRE"/></b>
   </body>
 </html>
</xsl:template>
```


Le résultat

On obtient :

```
    <html>
      <head>
        <title>Film:
            Vertigo
        </title>
      </head>
      <body>
        Le genre du film, c'est
          <b>Suspense</b>
      </body>
    </html>
```

Chemins complexes

Dans l'exemple précédent, on accédait aux fils d'un nœud.

En fait on peut :

- Accéder à tous les descendants
- Accéder aux parents, aux frères, aux neveux...
- Accéder aux attributs
- Effectuer des boucles

Le langage d'expression de chemins : XPath.

Exemple: Document XML

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

<?xml-stylesheet href="Salle.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
<SALLE NO='1' 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: Traduction de Salle

```
<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>
```


Le résultat

Appliqué à *Salle1.xml* :

```
 <h2>Salle No 1 </h2>
 Film:  Alien
 de     Ridley Scott
 <ol>
        <li>  15:00</li>
        <li>  18:00</li>
        <li>  21:00</li>
 </ol>
```

NB : c'est un **fragment HTML**, à intégrer dans un document complet.

Chemins complexes

- **Descendants** : le titre du film est désigné par `select="FILM/TITRE"` ;
- **Attributs** : le numéro de la salle est désigné par `select="@NO"` ;
- **Boucles** : avec `xsl:for-each` sur l'ensemble des séances désignées par `select="SEANCES/SEANCE"` ;
- **L'élément contexte** : désigné par le symbole '.', comme dans `select="."`.

Appels de règles

En général on produit un résultat en combinant plusieurs règles :

- La règle initiale s'applique à la racine du document traité ('/')
- On produit alors le **cadre** du document HTML
- On appelle d'autres règles pour compléter la création du résultat

Exemple : L'Épée de bois

« Cadre » HTML, puis appel de la règle `CINEMA`

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

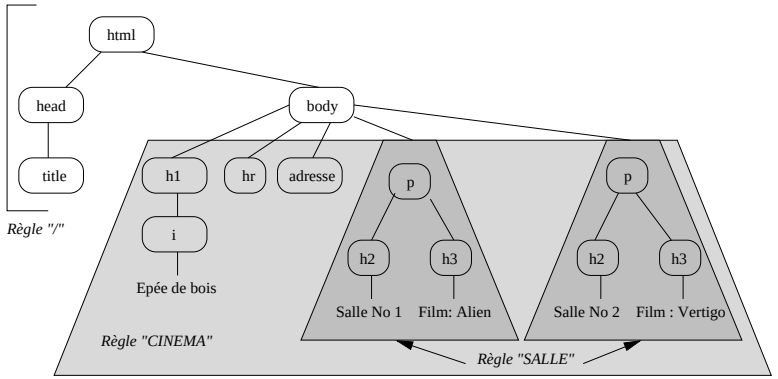

Règle `CINEMA`

Exploitation de l'élément `CINEMA`, puis appel à la règle `SALLE`

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

 </xsl:template>
```

Vue d'ensemble



La même chose, en WML

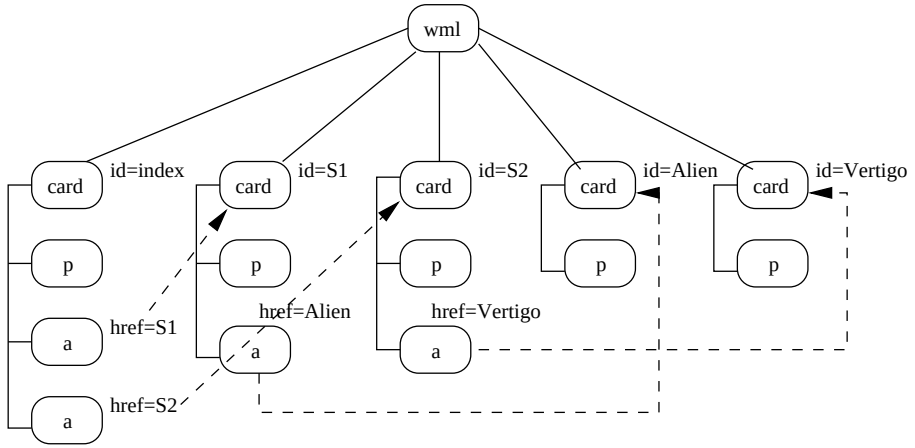
On génère les cartes du site

```
<xsl:template match="/">
 <wml>
   <!-- Carte d'accueil -->
   <xsl:apply-templates select="CINEMA"/>

   <!-- Cartes pour les salles et séances -->
   <xsl:apply-templates select="CINEMA/SALLE"/>

   <!-- création des cartes pour les films -->
   <xsl:apply-templates select=".//FILM"/>
 </wml>
</xsl:template>
```


Arbre XML du site



Règle `CINEMA`

```
<xsl:template match="CINEMA">
  <card id="index" title="Programme">
    <p align="center">
     <xsl:value-of select="NOM"/>
    </p>
    <xsl:for-each select="SALLE">
      <p> <a href="#S{@NO}">
            Salle <xsl:value-of select="@NO"/>:
          </a>
            <xsl:value-of select="FILM/TITRE"/>
      </p>
    </xsl:for-each>
  </card>
</xsl:template>
```

Résultat

```
<card id="index" title="Programme">
  <p align="center">
    Ep&#xE9;e de bois
  </p>
  <p>
    <a href="#S1"> Salle 1: </a>
    Alien
  </p>
  <p>
    <a href="#S2"> Salle 2: </a>
    Vertigo
  </p>
</card>
```


Règle SALLE

```
<xsl:template match="SALLE">
 <card id="S{@NO}">
   <p>Séances salle
       <xsl:value-of select="@NO"/></p>
   <p>
    <a href="#{FILM/TITRE}">
      Film :
       <xsl:value-of select="FILM/TITRE"/>
      </a>
    </p>
  <xsl:apply-templates select="SEANCES"/>
 </card>
</xsl:template>
```


Résultat

```
<card id="S2">
  <p>
    S&#xE9;ances salle 2
  </p>
  <p>
    <a href="#Vertigo">
        Film : Vertigo</a>
  </p>
  <p>
    S&#xE9;ance 22:00
  </p>
</card>
```


XSLT avec paramètres

```
<html>
<head>
  <title>Formulaire de Recherche</title>
</head>
<body bgcolor="white">
 <h1>Formulaire de Recherche</h1>
 <form method='get' action='Moteur.xml'
       name='Form'>
 Film: <input type='text' name='titre'> <br>
 Séance: <input type='text' NAME='seance' >
        (hh:mm)<br>
 Ville: <input type='text' name='ville'><br>
 <input type='submit' name='chercher'
       value="Chercher"/>
 </form>
</body>
</html>
```

Le document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE MOTEUR [
 <!ENTITY EpeeDeBois
     SYSTEM "http://epee-de-bois.fr/EDB.xml">
 <!ENTITY CineMarseille
     SYSTEM "http://cine-marseille.fr/CM.xml">
]>
<MOTEUR>
  <CINEMA>
    &EpeeDeBois;
   </CINEMA>
  <CINEMA>
    &CineMarseille;
   </CINEMA>
</MOTEUR>
```


Traitement des paramètres

```
<xsl:param name="titre"/>
<xsl:param name="seance"/>
<xsl:param name="ville"/>

<xsl:template match="MOTEUR">
 <xsl:for-each select="CINEMA">
   <xsl:if test="
           CINEMA//TITRE = $titre) and
           CINEMA//HEURE &gt;= $seance) and
           CINEMA/VILLE = $ville)">
     <xsl:apply-templates select="." /><p/>
   </xsl:if>
 </xsl:for-each>
</xsl:template>
```


Conclusion sur XSLT

Un langage totalement adapté au traitement de documents XML

- Parcours d'un document, vu comme un arbre
- Déclenchement de règles sur certains nœuds
- Association de plusieurs programmes à un même document

Bibliographie sur XSLT

1. Recommendation XSLT sur le site du W3C
2. B. Amann et P. Rigaux, Comprendre XSLT, O'Reilly
   `http://cortes.cnam.fr:8080/XSLT`
3. P. Wadler, A formal semantics of patterns in XSLT

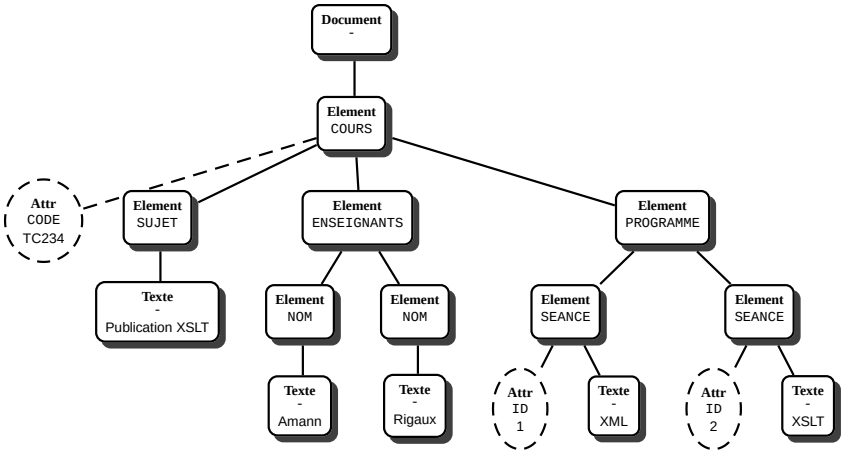
Evaluation d’un programme XSLT

Exemple de référence

```
<?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>
```


Le document source



Document résultat initial

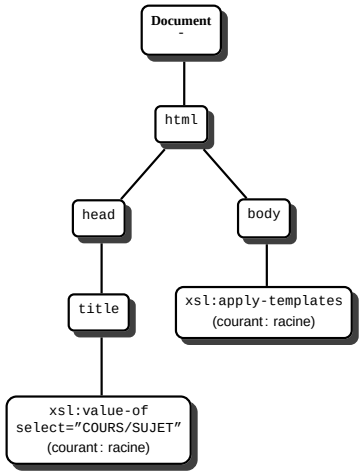
La règle principale (racine)

Rappel: On démarre avec la racine.

Génération du cadre HTML :

```
1  <xsl:template match="/">
2    <html>
3      <head><title>
4           <xsl:value-of select="COURS/SUJET"/>
5      </title></head>
6      <body bgcolor="white">
7
8       <xsl:apply-templates/>
9
10     </body>
11   </html>
12 </xsl:template>
```

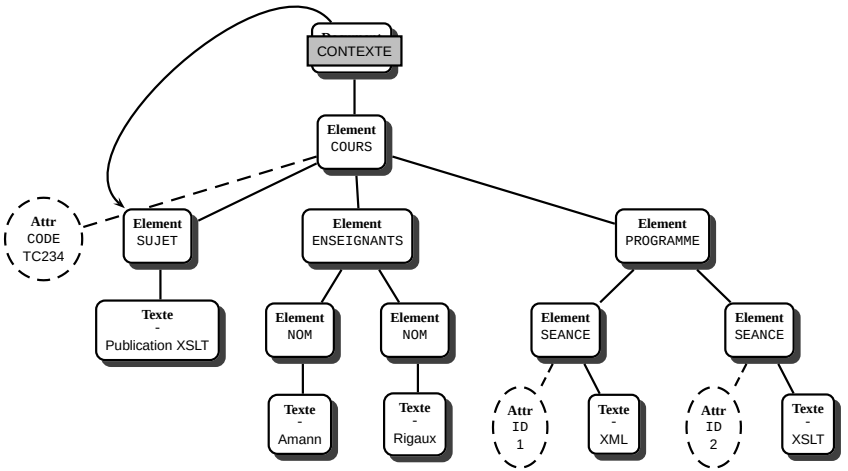

Ajout du corps de la règle principale



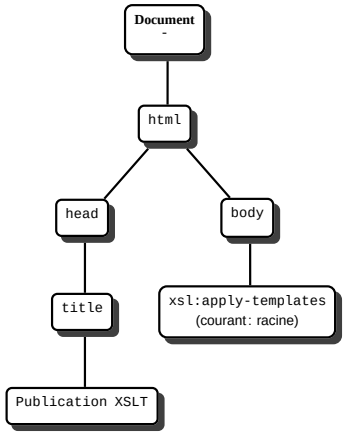
Rappel: la règle s'applique à la racine du document qui devient le contexte pour les expressions XPath.

Evaluation de `xsl:value-of`

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

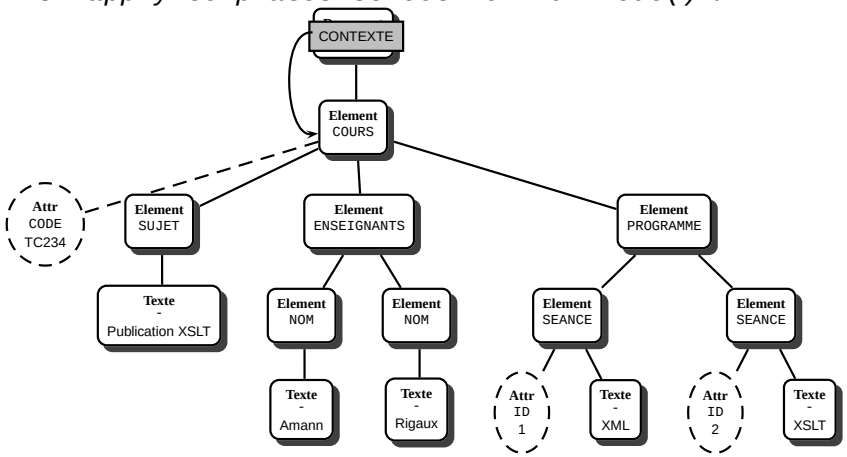


Après évaluation de `xsl:value-of`



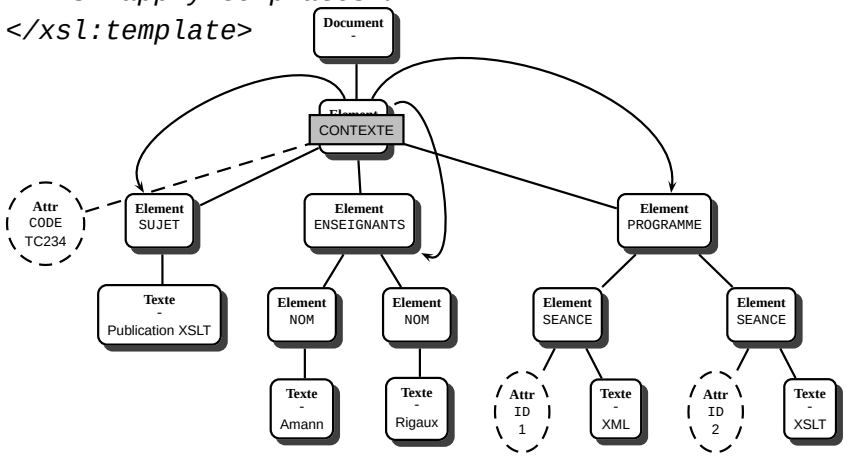
Evaluation de `xsl:apply-templates`

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



Règle par défaut

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



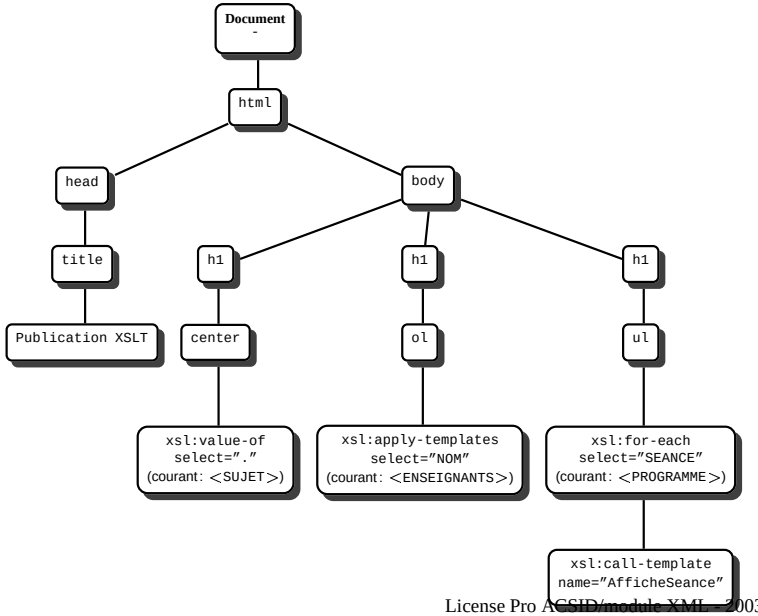
Règles pour SUJET et ENSEIGNANTS

```
1   <xsl:template match="SUJET">
2    <h1><center>
3      <xsl:value-of select="."/>
4     </center></h1>
5   </xsl:template>
6
7   <xsl:template match="ENSEIGNANTS">
8    <h1>Enseignants</h1>
9     <ol>
10      <xsl:apply-templates select="NOM"/>
11    </ol>
12  </xsl:template>
```

La règle pour PROGRAMME

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


Évaluation de `xsl:apply-templates`



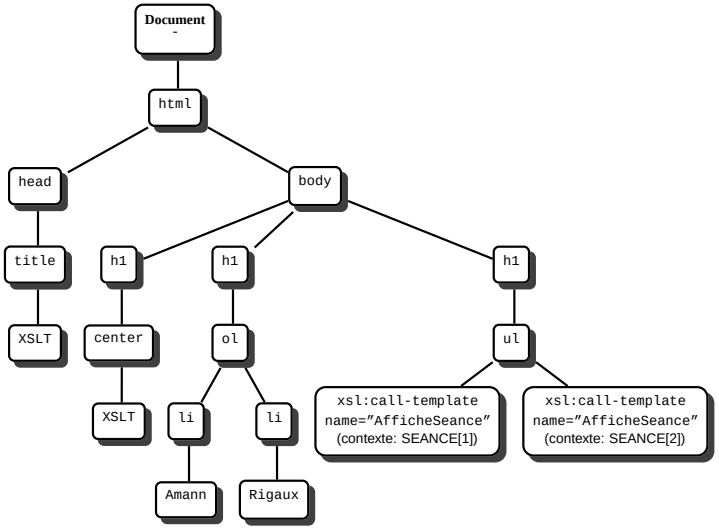
Les dernières règles

```
<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