# UE Ingénierie documentaire
# Master 2ème Année

Fabrice Lefèvre

2021

AVIGNON UNIVERSITÉ

# XML

# Partie 3 - XSLT

Fabrice Lefèvre
fabrice.lefevre@univ-avignon.fr
2021

# XSL, XSLT, XSL-FO

XSL (e*Xtensible Stylesheet Language*) est un langage de **feuilles de style** général pour les documents XML

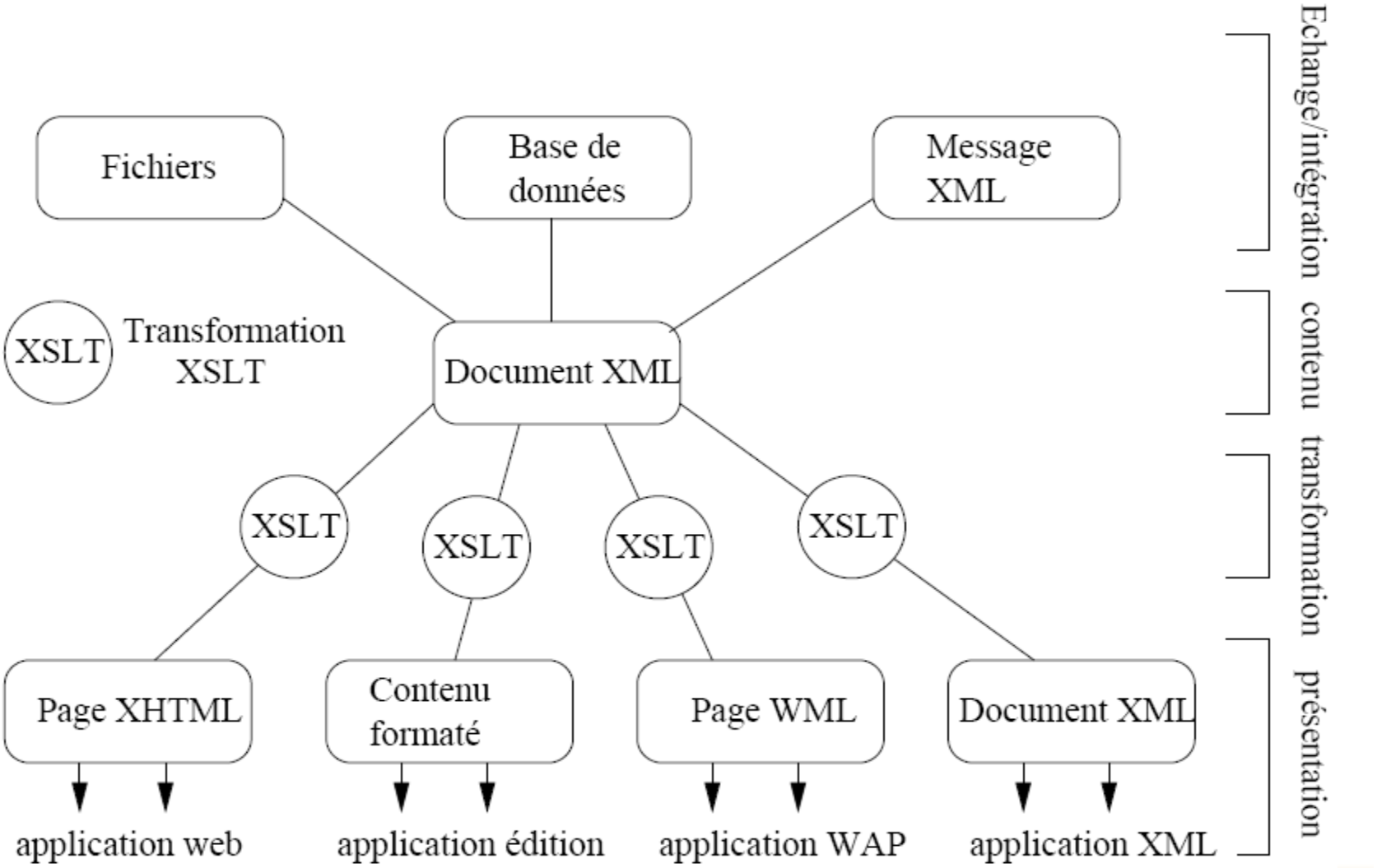
- `comparable à CSS pour HTML`

XSL est lui-même basé sur XML et se compose d'éléments et d'attributs

XSL comprend deux composants importants :

- un composant pour la **transformation** de données XML en autres données XML
  - `XSLT`
- un composant pour le **formatage** de données XML
  - *`XSL-FO`* `- "XSL Formatting Objects"`

# Gestion de l'information avec XML



# Fonctions d'une feuille de style XSLT

Fonction de base : langage de **règles de transformation** de documents/arbres XML

Transformations disponibles :

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

# Exemple de document XSL

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
          <head><title>Contact List</title></head>
          <body style="text-align: center; background-image: url(newspaper.jpg);>
                <xsl:apply-templates/>
          </body>
    </html>
</xsl:template>

<xsl:template match="headline">
    <div style="width:450px; border-bottom:5px double black; text-align:left; color:black; font-
    family:Verdana, Arial; font-size:26pt”> <xsl:value-of select="."/> </div>
</xsl:template>

<xsl:template match="p">
    <div style="width:450px; text-align: left; margin-bottom:8px; color:black; font-
    family:Verdana, Arial; font-size:10pt"> <xsl:apply-templates/> </div>
</xsl:template>

<xsl:template match="url">
    <span style="font-weight:bold"> <xsl:value-of select="."/> </span>
</xsl:template>
</xsl:stylesheet>
```

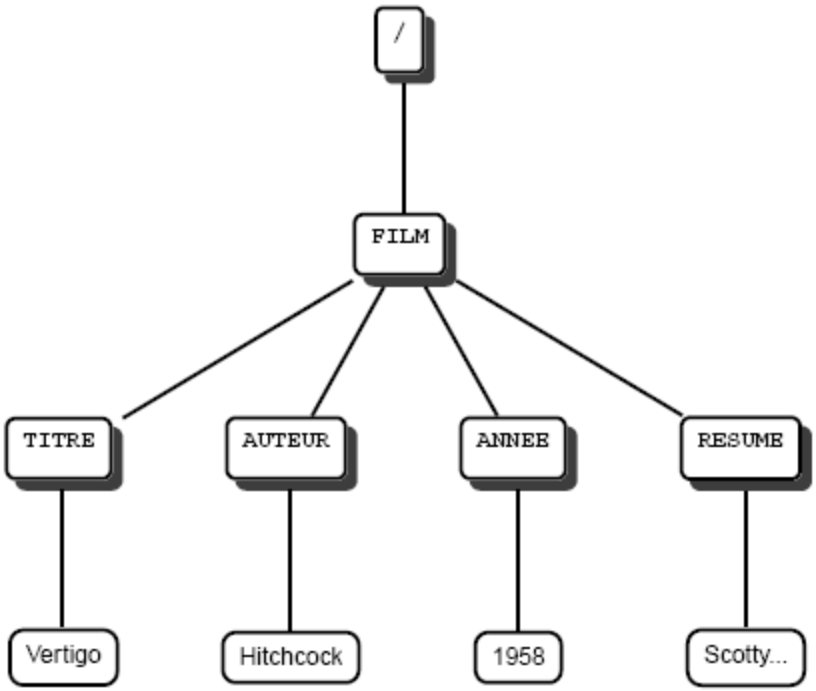
# Exécution d'un programme XSLT

L'exécution d'un programme XSLT consiste à **instancier** des **règles du document XSL** de manière récursive en parcourant le document :

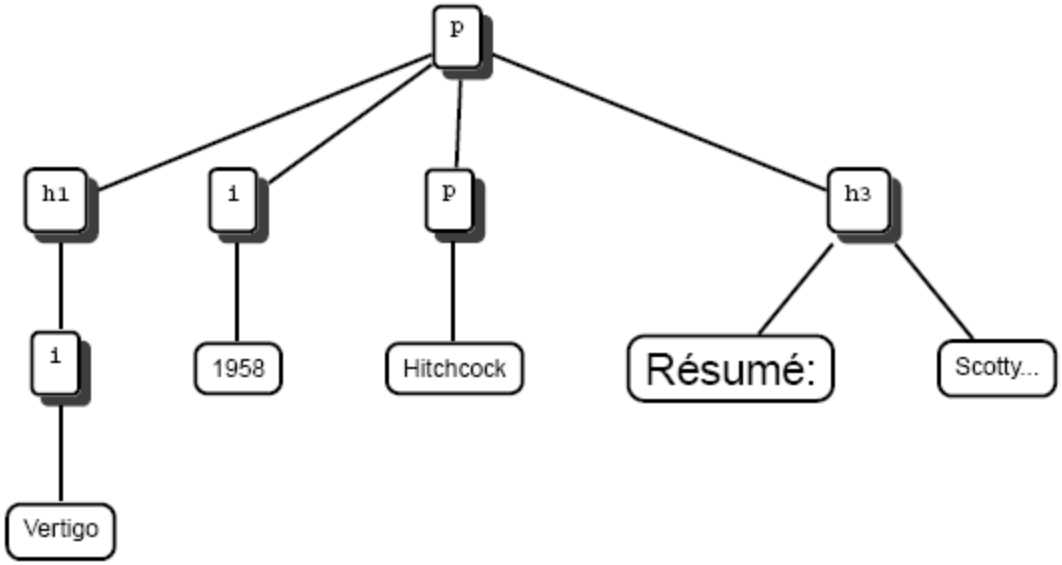
1. le sous –arbre constituant le **corps de la règle s'appliquant au nœud en cours** est inséré dans l'arbre représentant le document résultat
2. les instructions XSLT contenues dans le corps de la règle **sont exécutées** à leur tour
   - `des instructions permettent de relancer l'exécution sur de nouveaux nœuds → récursivité des règles`
3. le résultat d'une instruction produit un sous-arbre qui vient **remplacer cette instruction** dans l'arbre résultat

**Attention** : le programme s'applique sur le nœud « document » (« / »).

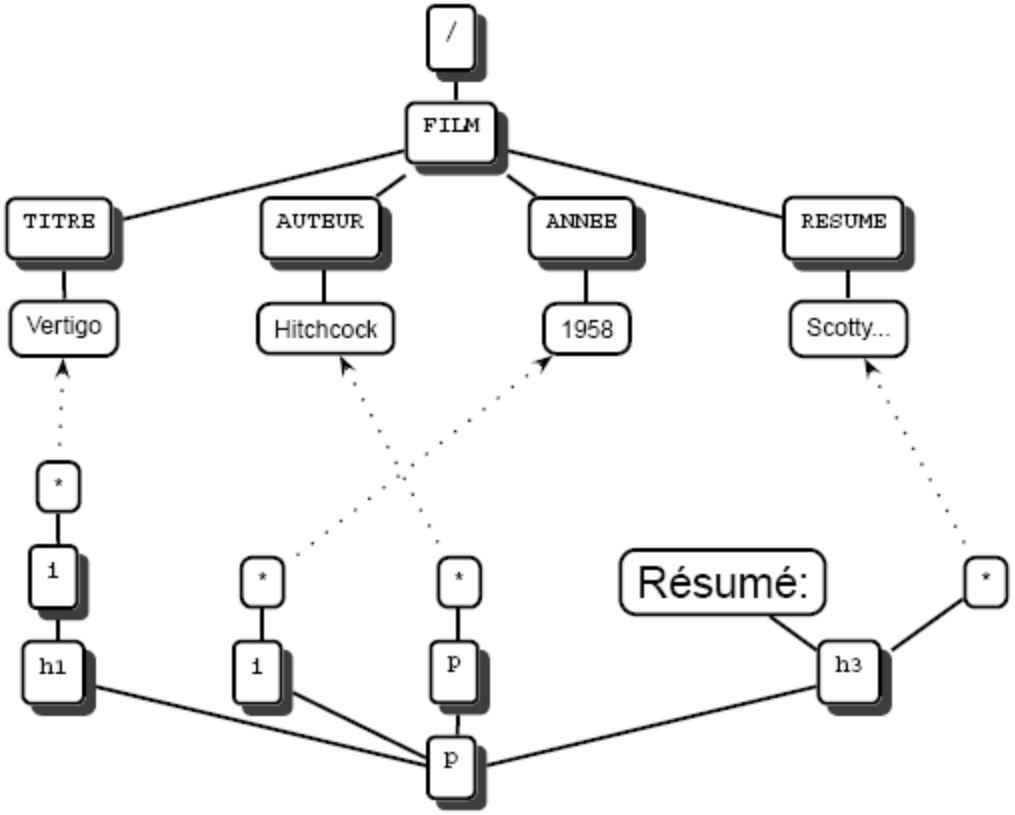
# Exemple, arbre XML



# Exemple, résultat de la transformation



# Exemple, transformation HTML



# Structure de base : les règles

Règle (*template)* : structure de base du document XSL 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 global

Document XSL = **ensemble de règles** pour construire un résultat

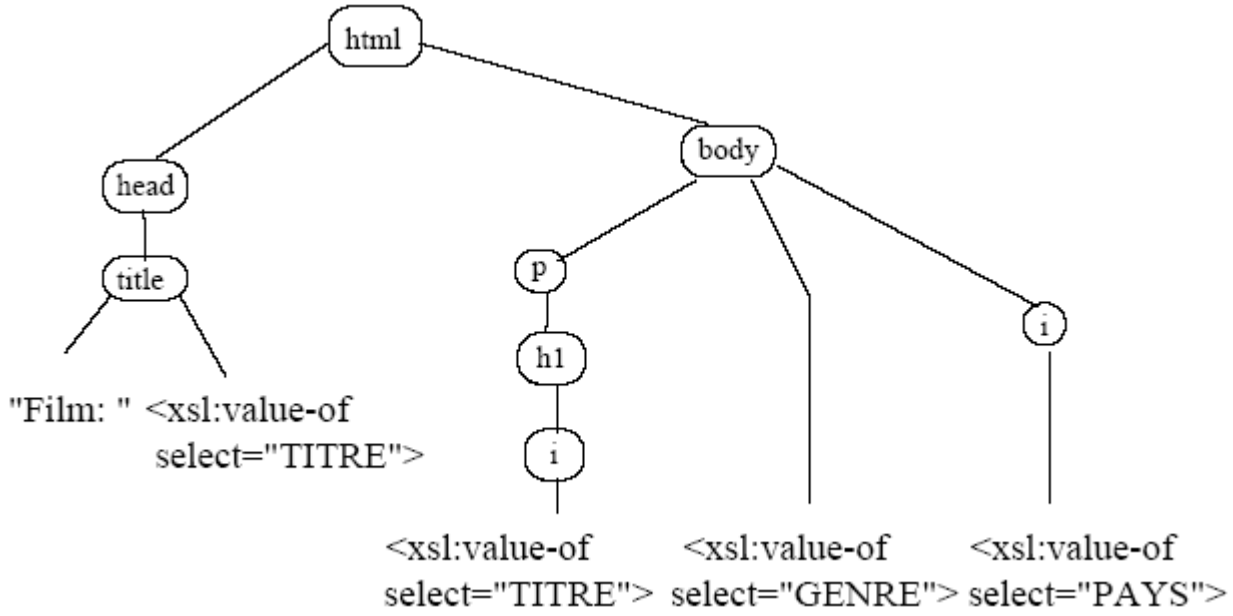
# Exemple de règle

```
<xsl:template match="/">
<html>
<head>
<title>
   Film : <xsl:value-of select="FILM/TITRE" />
</title>
</head>
<body>
<h1>
     Film : <xsl:value-of select="FILM/TITRE" />
</h1>
<p>Le genre du film, c'est
     <xsl:value-of select="FILM/GENRE" /></p>
<p>Son pays d'origine est
     <i><xsl:value-of select="FILM/PAYS" /></i></p>
</body>
</html>
</xsl:template>
```

Motif de sélection :
match="/"

Corps de la règle :
fragment d'arbre à produire

# Exemple, arbre de transformation



# Le résultat en HTML

```
<html>
<head>
<title>Film: Vertigo</title>
</head>
<body>
<h1>Film: Vertigo</h1>
<p>Le genre du film, c'est Suspense</p>
<p>Son pays d'origine est <i>USA</i></p>
</body>
</html>
```

# Structure d'un document XSL

```
<?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">
   <h1><i><xsl:value-of select="SUJET"/></i></h1>
   <hr />
   <xsl:apply-templates/>
</body>
</html>
</xsl:template>
... autres templates ...
</xsl:stylesheet>
```

# Déclaration de la feuille de style

Élement racine d'un document :

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

- Tous les éléments XSLT présents dans la feuille de style doivent être qualifiés par l'espace de nom **xsl:**

# Deux niveaux

Parmi les éléments XSLT, on distingue :

- Les **éléments de premier niveau**, fils directs de <xsl:stylesheet>.
  - Il s'agit principalement des **règles** (*template*)
  - L'ordre des éléments n'a pas d'importance
- Les **instructions de programmation** : elles composent les **corps des règles**.

# Eléments de premier niveau

Types d'élément et leur description :

- xsl:template : définit une règle XSLT
- 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:variable : définit une variable XSLT

# Définition et déclenchement des règles

Une règle est **définie** par l'élément xsl:template.

Deux possibilités :

- L'attribut match est un *pattern* XPath définissant les « cibles » de la règle

```
xsl:template match='FILM'
```

  - déclenchement par xsl:apply-templates
- L'attribut name donne un *nom* à la règle

```
xsl:template name='TDM'
```

  - déclenchement par xsl:call-template

# Chemin d'accès XPath

- *Exemple* : recherche du titre pour le noeud FILM :

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

- Pour l'accès aux informations de l'arbre source, on donne un chemin d'accès **XPath** à un noeud à partir du noeud courant (*noeud contexte*)
- Dans notre exemple, on accède aux fils d'un noeud. En fait on peut :
  - `Accéder à tous les descendants`
  - `Accéder aux parents, aux frères, aux neveux...`
  - `Accéder aux attributs`
  - `Effectuer des boucles`
  - `...`
- XPath est présenté dans le cours suivant : on retiendra qu'il s'agit d'un langage permettant de **désigner des noeuds particuliers** (ou groupe de noeuds) au sein d'un arbre

# Enchaînement des règles

L'application des règles sur le document/arbre source suit un algorithme :

- Un nœud du document est le **nœud contexte**
  - au départ c'est le noeud document !
- On cherche une règle qui s'applique à ce nœud
- On insère le corps de la règle dans le document résultat et les instructions XSLT sont exécutées

- L'instruction `xsl:apply-templates` permet de sélectionner de nouveaux nœuds contexte en déclenchant à nouveau l'algorithme de sélection des règles sur les **fils** du nœud courant

# Appels de règles

En général, on produit un résultat en combinant plusieurs règles :

- La règle initiale s'applique à l'élément racine du document traité ('/')
- On produit alors le "cadre" du document XML de sortie
  - `Niveau haut de l'arbre résultat, avec notamment l'élément racine`
- Cette première règle active d'autres règles avec ses instructions internes permettant de compléter la création du résultat et en parcourant la structure du document XML d'entrée à volonté

# Exemple de combinaison de règles

```
...
<xsl:template match='/'>
<racine>
   <xsl:apply-templates />
</racine>
</xsl:template>
<xsl:template match=`N`>
...
```

```
...
<xsl:template match='/'>
<racine>
   <xsl:apply-templates select='N/N1'/>
   <blabla>
   <xsl:apply-templates select='N/N2'/>
</racine>
</xsl:template>
...
```

# Génération de texte

- Produire une phrase quand on rencontre un noeud 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 noeud 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 noeud FILM :

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

# Composants calculés

- Générer du contenu à partir d'une expression :

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

- Copie de l'élément courant :

```
<xsl:copy>
ou <xsl:copy-of select="SALLE">
```

- Appliquer toutes les règles possibles correspondant à une définition de nœuds :

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

- Appel à une fonction de script :

```
<xsl:eval>Fonction ()</xsl:eval>
```

# Structures de contrôle

Alternatives :

- **`xsl:if`**
- **`xsl:choose, xsl:when, xsl:otherwise`**

Boucles :

- **`xsl:for-each`**

Tris :

- **`xsl:sort`** (placé dans **`xsl:apply-templates`** ou **`xsl:for-each`**)
  - `attributs data-types et order`

→ seront vues en TP

# 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:apply-templaces select="SEANCES/SEANCE">
   </ol>
</xsl:template>

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

# Exemple, traduction de salle, variante

```
<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>
```

# Exemple, le résultat

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

Appliqué à *Salle1.xml*

# Règles par défaut

Quand aucune règle n'est sélectionnée, XSLT applique des **règles par défaut**

- Première règle pour les éléments et la racine du document :

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

- On demande l'application de règles pour les fils du noeud courant.

# Première règle

- On peut se contenter de définir une règle pour l'élément racine, et ignorer le nœud document :

```
<xsl:template match="COURS">
      corps de la règle
</xsl:template>
```

- le processeur traite la racine du document avec la règle par défaut
- l'instruction `xsl:apply-templates` de la règle par défaut déclenche la règle sur le nœud fils COURS

# Défaut pour le texte et les attributs

- Deuxième règle pour le texte et les attributs :

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

- Par défaut, on insère dans le document résultat la valeur d'un nœud **Text**, ou de l'attribut d'un élément
- Cela suppose (surtout pour les attributs) d'avoir utilisé un `xsl:apply-templates` sélectionnant ces nœuds.

# Sélection de règle

Que se passe-t-il lorsque plusieurs règles sont candidates pour un même nœud ?

- il existe des priorités **implicites** qui permettent au processeur de choisir
- on peut aussi donner **explicitement** une priorité
- si le choix est impossible : le processeur s'arrête !

# Priorités implicites

**Principe** : plus c'est *spécifié*, plus c'est prioritaire

- **Priorité 0** : les *patterns* constitués d'une seule étape XPath, avec un nom d'élément ou d'attribut et sans prédicat
- **Priorité -0.5** : les filtres autres qu'un nom d'élément ou d'attribut ont une priorité égale à -0,5 ( node(), *)
- Tous les autres ont une **priorité de 0.5** (prédicats, plusieurs étapes)

# Exemple, l'Épée de bois

```
<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>
```

« Cadre » HTML, puis appel
de la règle CINEMA

# Exemple, règle CINEMA

```
<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>
```

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

# Règles nommées

Il est possible de nommer des règles et les activer directement par leur nom avec xsl:call-template

- comparable à un appel de fonction.
- contrairement à xsl:apply-templates, l'appel ne change pas le contexte d'évaluation de la règle
- on peut passer des paramètres avec xsl:param

# Exemple, règle nommée

```
<xsl:template name="Afficher">
   <xsl:value-of select="position()"/> :
   <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="NOM">
   <xsl:call-template name="Afficher"/>
</xsl:template>
<xsl:template match="text()">
   <xsl:call-template name="Afficher"/>
</xsl:template>
<xsl:template match="comment()">
   <xsl:call-template name="Afficher"/>
</xsl:template>
```

# Règles avec paramètres

On peut passer des paramètres à `xsl:call-template` et `xsl:apply-templates`

- avec `xsl:param`, on définit dans la règle les paramètres attendus
- on associe un ou plusieurs `xsl:with-param` à `xsl:call-template` ou `xsl:apply-templates`

# Exemple, paramètres

La règle Afficher attend une chaîne :

```
<xsl:template name="Afficher">
  <xsl:param name="texte" select="string('inconnu')"/>
  <xsl:value-of select="concat(position(),' : ', $texte)"/>
</xsl:template>
```

- texte est le nom du paramètre
- select donne la valeur par défaut
- $texte désigne le paramètre

# Exemple, passage de paramètres

```
<xsl:template match="NOM">
  <xsl:call-template name="Afficher">
    <xsl:with-param name="texte" select="."/>
  </xsl:call-template>
</xsl:template>

<xsl:template match="comment()">
  <xsl:call-template name="Afficher">
    <xsl:with-param name="texte" select="string('Commentaire')"/>
  </xsl:call-template>
</xsl:template>
```

# Définition et portée des paramètres

On peut définir des paramètres à deux niveaux :

- **Dans le corps d'une règle** : le paramètre est alors local à la règle, et fourni par l'appel de règle (`call-template` ou `apply-templates`)
- **Élément de premier niveau** (fils de `xsl:stylesheet`) : le paramètre est global, et fourni par le processeur

# Exemple, moteur de recherche (1)

```
<html>
<head>
  <title>Formulaire de Recherche</title>
</head>
<body bgcolor="white">
  <h1>Formulaire de Recherche</h1>
  <form method='get' action='Moteur.xsl' 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>
```

# Exemple, moteur de recherche (2)

```
<?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-massilla.fr/CM.xml">
]>
<MOTEUR>
  <CINEMA>
    &EpeeDeBois;
  </CINEMA>
  <CINEMA>
    &CineMarseille;
  </CINEMA>
</MOTEUR>
```

# Exemple, moteur de recherche (3)

```
...
<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>
<xsl:template match="CINEMA">
… Affiche les salles correspondantes
</xsl:template>
…
```

Le serveur web exécute le fichier XSLT.

# XSL-FO

XSL-FO (*extended Style Sheet Language-Formatting Object*) est un langage de description de **documents à imprimer** avec XML

- On indique les paramètres de mise en page permettant d'exprimer le rendu d'un document :
  - `pagination`
  - `notes de bas de page`
  - `marges`
  - `emplacement des différents objets sur la page`
  - `polices des caractères`
  - `affichage de tableaux…`
- Dans un document XSL-FO, le contenu est entre des balises de formatage → un processeur se charge de produire le document

# L'approche XSL-FO

Traitement de texte WYSIWYG classique :

- on indique le contenu **et** la mise en forme
- difficultés :
  - `pas facile d'être expert en contenu et en mise en forme`
  - `pas pratique de penser aux deux simultanément`
- du coup, il est très difficile de faire de beaux documents (et impossible d'intégrer des contenus hétérogènes)

# Quelques principes originaux

Avec XSL-FO :

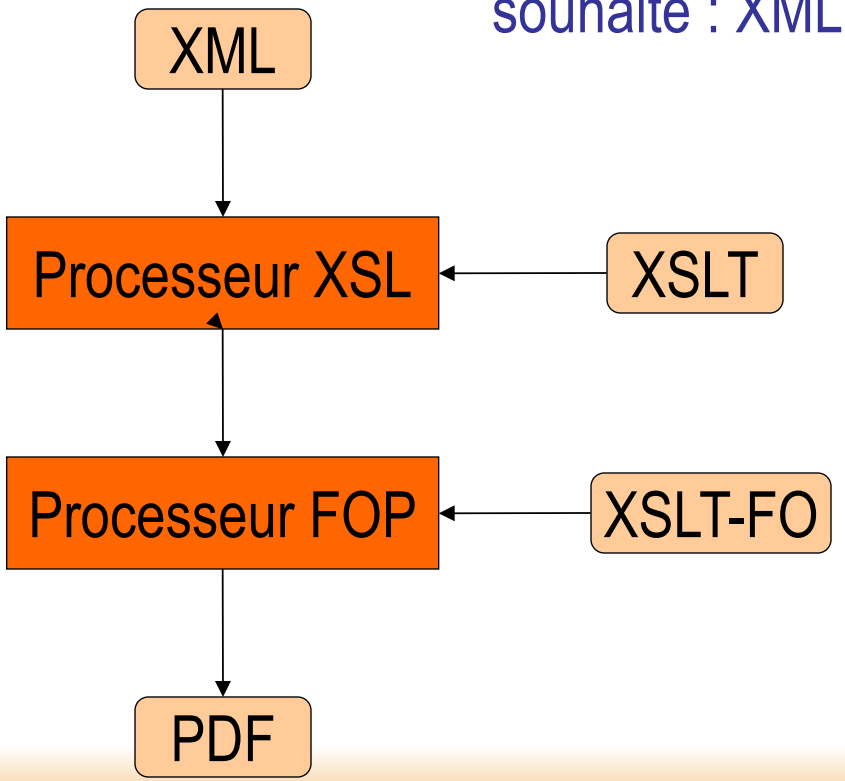
- un responsable pour le contenu (XML)
  - `provenant de n'importe où (BD, sites, ...)`
- un responsable pour la mise en forme (XSL-FO)
  - `décide de la présentation`
- un processeur pour produire le résultat

mais le langage est malgré tout assez complexe a apprendre pour un non spécialiste ...

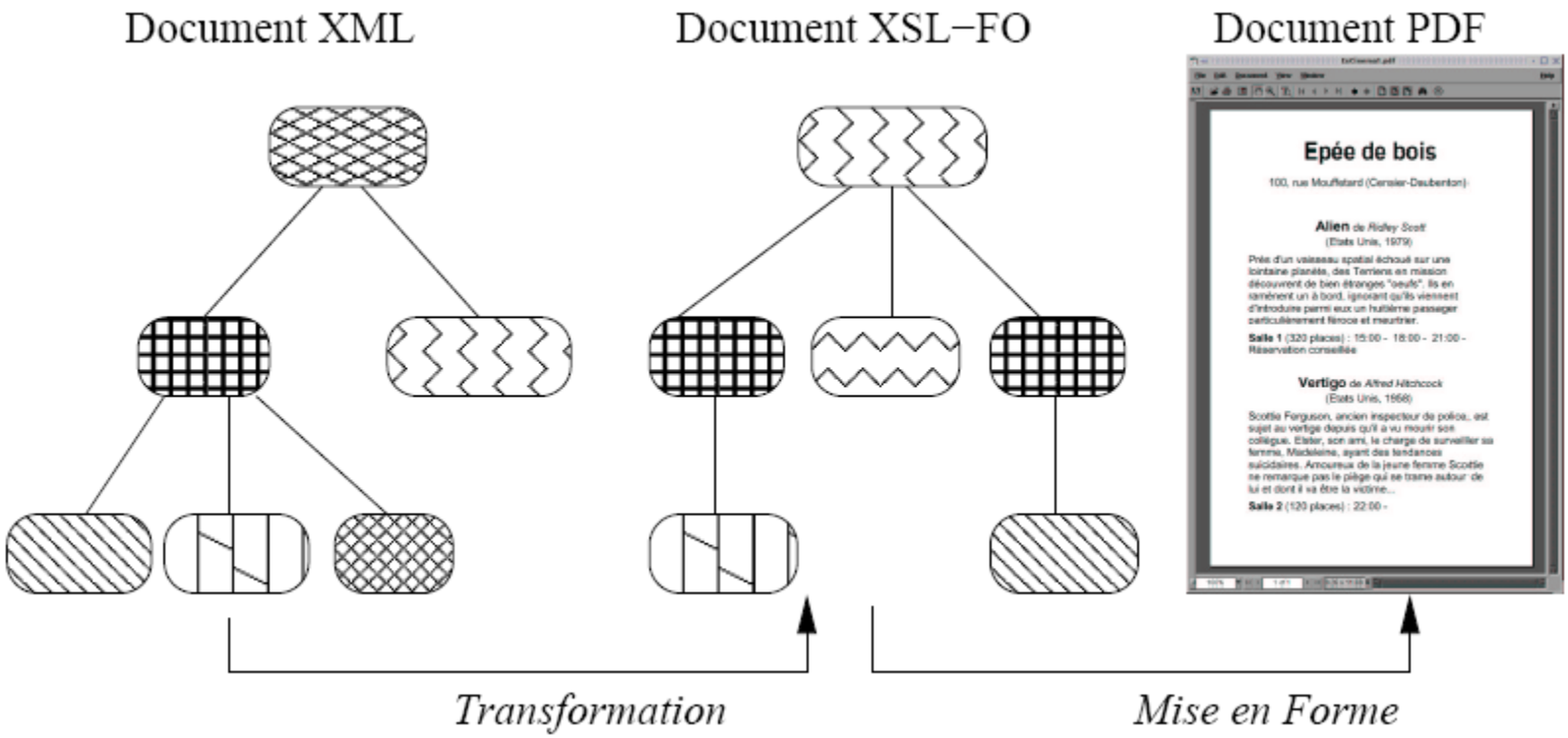
# Processeur XSL-FO

FOP (*Formatting Object Processor*) une application java qui utilise XSL-FO

- FOP lit un arbre de formatage d'objet (FO) et renvoie une page suivant le format de sortie souhaité : XML, PDF, PS, SVG, TXT…



# Transformation et mise en forme



# Exemples d'un document XSL-FO

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fo:root>
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page"
  page-height="29.7cm" page-width="21cm"/>
  </fo:layout-master-set>
  <fo:page-sequence master-name='simple'>
    <fo:flow font-size="20pt">
      <fo:block>
        Ceci est le premier paragraphe,
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

# Conclusion sur XSL

XSL est un langage totalement adapté au traitement général de documents XML

- Parcours d'un document, vu à travers sa forme arborescente
- Déclenchement de règles sur certains nœuds, désignés pas des expressions XPath
- Association de plusieurs programmes à un même document en fonction du contexte, des besoins