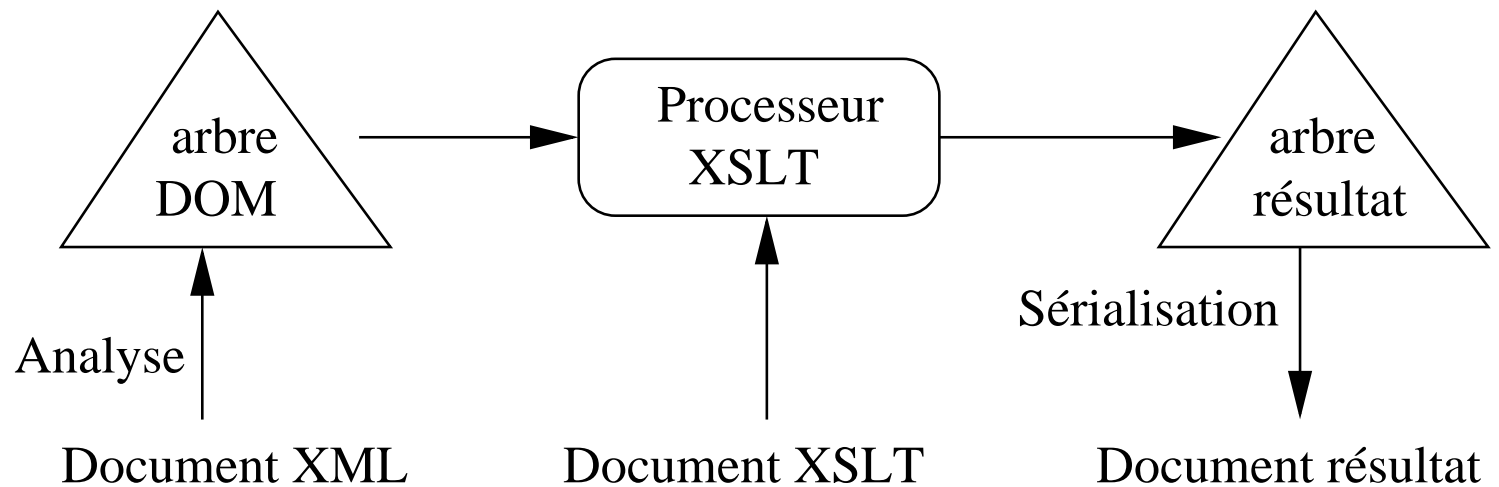


Règles XSLT

Règles XSLT

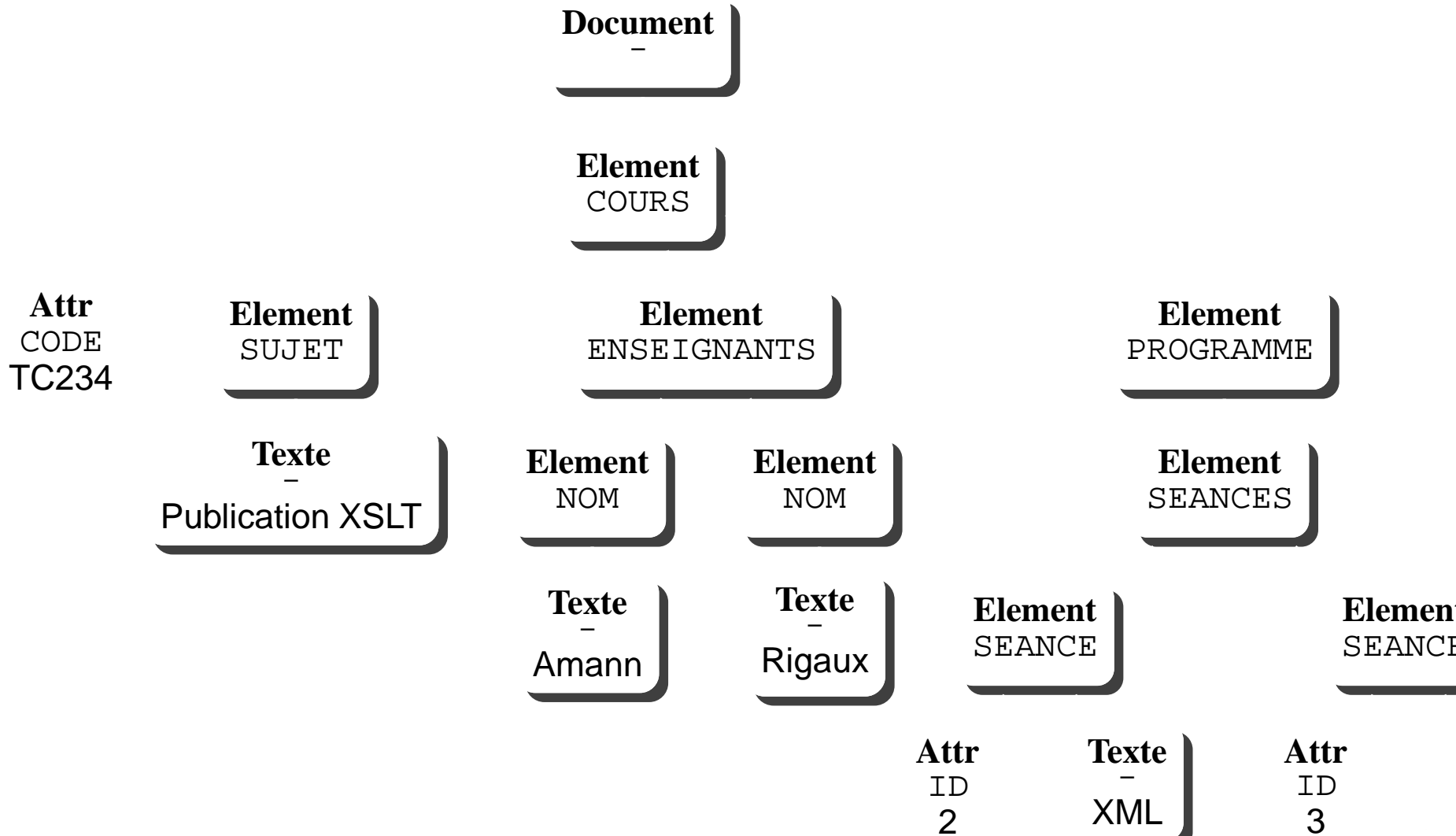
XSLT = production d'un **document résultat** à partir d'un **document source**.



Exemple de référence

```
<?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>2003</ANNEE>
  </PROGRAMME>
</COURS>
```

Le document source



Structure d'un programme XSLT

Programme XSLT = document XML

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

L'élément `xsl:stylesheet`

Élément racine d'un programme :

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

⇒ tous les éléments XSLT doivent être qualifiés par l'espace de nom `xsl:`

Deux types d'éléments

On distingue :

- **Les éléments de premier niveau**, fils de `<xsl:stylesheet>`.
Il s'agit essentiellement des **règles** (*template*)
L'ordre des éléments n'a pas d'importance
- **Les instructions** : on les trouve dans le **corps des règles**.

Principaux éléments de premier niveau

Type d'élément	Description
<code>xsl:import</code>	Import d'un programme XSLT
<code>xsl:include</code>	Inclusion d'un programme XSLT
<code>xsl:output</code>	Indique le format de sortie
<code>xsl:param</code>	Définit un paramètre
<code>xsl:template</code>	Définit une règle XSLT
<code>xsl:variable</code>	Définit une variable XSLT

TAB. 1 –

xsl:import et *xsl:include*

Pour inclure des règles d'un programme dans un autre.
Différence: la gestion des conflits

- avec `xsl:import` les règles importées ont une préséance moindre que celles du programme importateur
- avec `xsl:include` il n'y a pas de notion de préséance

`xsl:import` doit être le premier élément de premier niveau du programme.

Document à importer

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

<xsl:template
  match="SUJET | ENSEIGNANTS | PROGRAMME" />
```

Programme principal

On reprend la règle pour la racine du document, et on « surcharge » les autres.

```
<xsl:import href="Import.xsl" />
<xsl:template match="COURS/ENSEIGNANTS" >
  <ol><xsl:apply-templates select="NOM" /></ol>
</xsl:template>

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

Les règles XSLT

Exécution d'un programme XSLT

L'exécution d'un programme XSLT consiste à **instancier des règles**

- le **corps de la règle** est inséré dans le document résultat
- les instructions XSLT contenues dans le corps de la règle sont exécutées à leur tour
- le résultat d'une instruction vient remplacer cette instruction dans le résultat

Corps de règle : exemple

Des éléments littéraux, du texte et des éléments XSLT

```
<xsl:template match='FILM' >  
  Voici le titre : <b>  
    <xsl:value-of select='TITRE' />  
  </b>  
  <xsl:apply-templates select='ROLES' />  
</xsl:template>
```

⇒ la règle s'instancie dans le contexte d'un nœud particulier (de type `FILM`)

Déclenchement de règles

- On prend un nœud du document comme **nœud contexte**
⇒ au départ c'est la racine du document
- On cherche la règle qui s'applique à ce nœud
- On insère le corps de la règle dans le document résultat
- l'instruction `xsl:apply-templates` permet de sélectionner de nouveaux nœuds contexte

Exemple typique

Combinaison d'un `xsl:template` et de un ou plusieurs

`xsl:apply-templates`

```
<xsl:template match='/'>
```

```
<racine>
```

```
  <xsl:apply-templates select='N1' />
```

```
  <xsl:apply-templates select='N2' />
```

```
</racine>
```

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

Règles : définition et déclenchement

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

- L'attribut `match` est un *pattern* XPath définissant les « cibles » de la règle
Ex: `xsl:template match='FILM'`
déclenchement par `xsl:apply-templates`
- L'attribut `name` donne un nom à la règle
Ex: `xsl:template name='TDM'`
déclenchement par `xsl:call-template`

Les patterns

On ne peut pas mettre n'importe quelle expression XPath dans l'attribut `match`.

- ❌ L'expression doit *toujours* désigner un ensemble de nœuds

Pas bon: `<xsl:template match="1">`

- ❌ De plus certaines expressions seraient trop complexes à évaluer.

Interdit: `<xsl:template match="preceding::node()[5]">`

Exemple : des appels de règles

Trois appels `xsl:apply-templates` : on sélectionne des nœuds, en demandant l'application de règles.

```
<xsl:template match="/">
<DOC>
  <xsl:apply-templates select="//NOM" />
  <xsl:apply-templates select="//SEANCE/@ID" />
  <xsl:apply-templates select="//SEANCE" />
</DOC>
</xsl:template>
```

Exemple : des patterns

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

<xsl:template match="@ID" >
  <IDSEANCE>
    <xsl:value-of select="." />
  </IDSEANCE>
</xsl:template>

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

Sélection des règles

Problème : étant donné un nœud, comment trouver la règle qui s'applique ?

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

Exemple simple : la première règle

Au départ du processus :

- Le nœud-contexte N est la racine du document
- Il existe une règle R dont le *pattern* est « / »
- \Rightarrow en prenant n'importe quel nœud, l'évaluation de « / » est N , donc la règle s'applique.

Il est donc bon (mais pas indispensable) d'avoir une règle avec *pattern* « / ».

Avec `select='//NOM'`

Les nœuds sont ceux de type `NOM`. La règle suivante s'applique :

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

Car : en prenant pour nœud `ENSEIGNANTS`, et en évaluant l'expression `NOM`, on obtient le nœud contexte.

NB : la règle est déclenchée pour tous les nœuds de type `NOM`, quel que soit le père.

Avec `select=' //SEANCE/@ID'`

Les nœuds sont des attributs de nom `ID`. La règle suivante s'applique :

```
<xsl:template match="@ID" >
```

Car : en prenant pour nœud le père et en évaluant l'expression, on obtient le nœud contexte.

Même remarque : cette règle est déclenchée pour tous les attributs `ID`, quel que soit leur élément-père.

Avec `select=' //SEANCE '`

Les nœuds sont les éléments SEANCE. La règle suivante s'applique :

```
<xsl:template match="PROGRAMME/SEANCE" >
```

Car : en prenant pour nœud l'élément COURS et en évaluant l'expression, on obtient le nœud contexte.

Cette fois la règle ne s'applique qu'aux éléments SEANCE fils d'un élément PROGRAMME

Patterns : ce qui est autorisé

Seulement les axes suivants :

- Les fils d'un élément : `child`
- Les attributs d'un élément : `attribute`
- L'abréviation `// de descendant-or-self::node()`

Pourquoi ? **Parce qu'on peut savoir si une règle doit être déclenchée uniquement en regardant les ancêtres du nœud contexte**

Autres exemples de patterns

- `/COURS/ENSEIGNANTS` : nœuds `ENSEIGNANTS` fils d'un élément racine `COURS`
- `//SEANCE[@ID=2]` tout nœud de type `SEANCE` ayant un attribut `ID` valant `2`
- `NOM[position()=2]` tout nœud qui est le deuxième fils `NOM` de son père
- `/COURS/@CODE[.="TC234"]` l'attribut de nom `CODE`, fils de l'élément racine `<COURS>`, et de valeur `TC234`

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 nœud courant.

Conséquence

On peut se contenter de définir une règle pour l'élément racine, et ignorer la racine du 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 COURS.

Pour le texte et les attributs

Par défaut, on insère dans le document résultat la valeur du nœud **Text**, ou de l'attribut.

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

Cela suppose (surtout pour les attributs) d'avoir utilisé un `xsl:apply-templates` sélectionnant ces nœuds.

Conséquence

Si on se contente des règles par défaut, on obtient la concaténation de nœuds de type **Text**.

Programme minimal :

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

NB : ne prend pas les attributs (pourquoi ?)

Résultat du programme minimal

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

```
Publication XSLT
```

```
Amann  
Rigaux
```

```
Introduction  
Documents XML  
Programmation XSLT
```

Pour les autres nœuds

Pour les instructions de traitement et les commentaires : on ne fait rien.

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

⇒ si on ne les sélectionne pas explicitement, en définissant une règle pour les traiter, il ne se passe rien.

L'instruction xsl:apply-templates

Attributs : `select`, `mode` et `priority`.

- `select` doit sélectionner un **ensemble de nœuds**.
Ils constituent le contexte d'évaluation
⇒ pour chaque nœud on va chercher la règle à instancier.
- `mode` permet de choisir explicitement une des règles parmi celles qui sont candidates
- `priority` permet de définir une priorité pour que le processeur choisisse.

xsl:apply-templates

Appliqué à notre document : un déclenchement de règle sur le nœud ENSEIGNANTS

```
<xsl:template match="ENSEIGNANTS" >
  <xsl:comment>
    Application de la règle ENSEIGNANTS
  </xsl:comment>
  <xsl:apply-templates/>
</xsl:template>
```

NB : la valeur par défaut de `select` est `child::node()`.

Les règles

```
<xsl:template match="NOM" >
  <xsl:value-of select="position()" /> :
  Noeud NOM
</xsl:template>

<xsl:template match="text()" >
  <xsl:value-of select="position()" /> :
  Noeud de texte
</xsl:template>

<xsl:template match="comment()" >
  <xsl:value-of select="position()" /> :
  Noeud de commentaire
</xsl:template>
```

Le résultat

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
      Application de la règle ENSEIGNANTS
-->
1 : Noeud de texte
2 : Noeud de commentaire
3 : Noeud de texte
4 : Noeud NOM
5 : Noeud de texte
6 : Noeud NOM
7 : Noeud de texte
```

On voit que les nœuds sont de types différents, mais sont issus du même contexte (attribut `select` du `xsl:apply-templates`)

Sélection d'une règle

Que faire quand 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.

Exemple : filtre de certains nœuds

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FILM>
<TITRE>Vertigo</TITRE>
<ANNEE>1958</ANNEE><GENRE>Drame</GENRE>
<MES>Alfred Hitchcock</MES>
<RESUME>Scottie Ferguson, ancien inspecteur
police, est sujet au vertige depuis qu'il a
</RESUME>
</FILM>
<FILM>
<TITRE>Alien</TITRE>
<ANNEE>1979</ANNEE><GENRE>Science-fiction</G
<MES>Ridley Scott</MES>
<RESUME>Près d'un vaisseau spatial échoué su
une lointaine planète, des Terriens en missi
</RESUME>
</FILM>
```


Le programme XSLT

Un règle pour <RESUME> qui ne fait rien...

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

<xsl:template match="RESUME"/>

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

... moins prioritaire que les autres.

Priorités implicites

Intuition : plus c'est « spécifique », 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)

Les modes

Raison d'être : un même nœud peut être traité plusieurs fois. Exemple :

- On parcourt tous les chapitres et sections pour produire une table des matières.
- On les parcourt à nouveau pour publier le contenu.

Donc il faut des règles différentes s'appliquant aux mêmes nœuds : on les distingue par le mode.

Exemple : création de liens HTML

On peut créer des ancres « internes » à un document.

```
<a name='Alien' />
```

On peut ensuite créer un lien vers cette ancre

```
<a href='#Alien'>Lien vers le film  
Alien</A>
```

Objectif : une règle pour créer les liens, une autre pour créer les ancres.

Les deux règles

```
<xsl:template match="FILM" mode="Ancres" >
  <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>
```

L'appel des règles Résultat

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

Résumé : sélection d'une règle

Soit un `xsl:apply-templates`, et N un des nœud sélectionnés.

- On ne prend que les règles avec le même mode que `xsl:apply-templates`
- On teste le *pattern* pour savoir si le nœud satisfait la règle.
- On prend celle qui a la plus grande priorité.