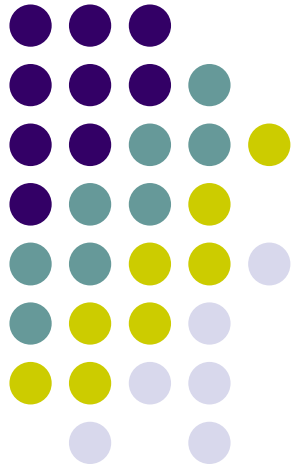
XSLT

eXtensible Stylesheet Language Transformations

Pr. Sidi Mohammed Benslimane
École Supérieure en Informatique
08 Mai 1945 – Sidi Bel Abbes –

s.benslimane@esi-sba.dz

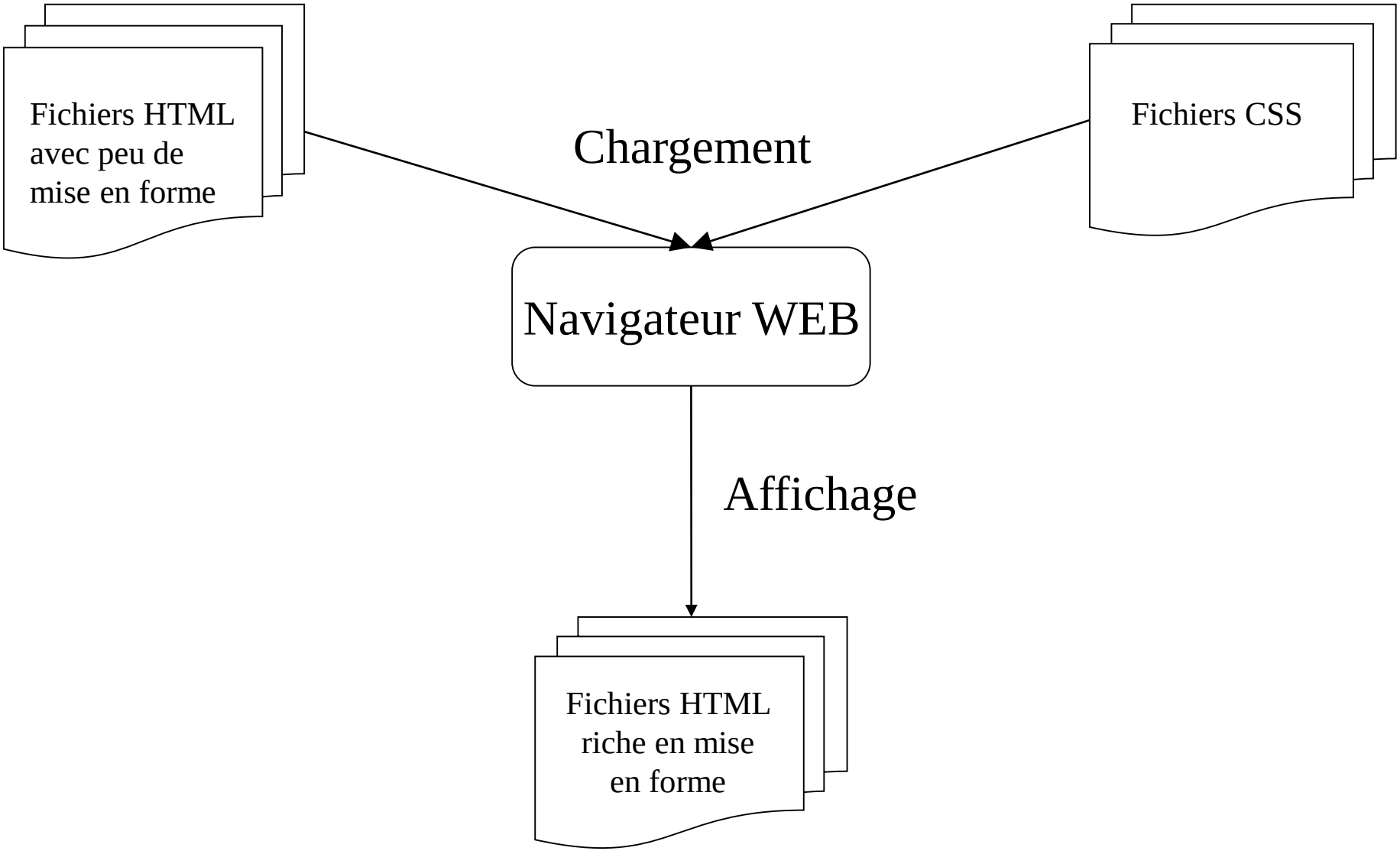


Feuilles de styles (CSS)

Cascading Style Sheets

- Principalement associée à HTML
- Permet de spécifier dans les pages web:
  - Les polices de caractères, leurs attributs d'affichage (gras, italique, souligné, taille, couleur)
  - Les couleur ou l'image de fond
  - Les formats de puces
  - Les alignements par défaut
  - etc.
- Permet d'afficher vers des périphériques particuliers:
  - Imprimantes
  - Téléphones mobile
  - PDA (Personal Digital Assistant)
  - etc.

Processus pour afficher des pages HTML pourvues de feuilles de styles



Feuilles de styles (CSS)

- Les différentes versions officielles de ce standard sont les suivantes:
  - CSS1 (level 1): recommandation depuis le 17 décembre 1996; version de base
  - CSS2 (level 2) recommandation depuis le 12 mai 1998
    - Affichage graphique
    - Gestion du son
    - Impression en braille pour les non-voyants
  - CSS3 (level 3) : 2007
    - Traitement de langues: Arabes et chinoises
  - CSS (level 4): Novembre 2018
    - Wrap-flow : habiller du texte
    - Shape-outside : donner à un élément une forme géométrique complexe
    - Display:grid : disposer les éléments en grille

Limites pour afficher un document XML avec CSS

- CSS a été prévue pour HTML qui possède déjà d'importantes facultés de mise en page.
- CSS propose donc uniquement des fonctions de mise en forme assez simples (on ne peut pas insérer de retour chariot en CSS puisque ceci est normalement à la charge de HTML).
- Pas de possibilité de filtres et de tri.
- Les attributs XML ne peuvent pas être affichés, seulement le contenu des éléments peut l'être.

Présentation du langage XSLT

- XSLT (eXtensible Stylesheet Language Transformations) est un langage de feuilles de styles associé à XML.
- XSLT 1.0 est une recommandation W3C depuis novembre 1999. (XSLT 2.0 Janvier 2007, XSLT 3.0 Juin 2017, XSLT 4.0 Novembre 2020).
- XSLT est (bien sûr) écrit en XML.
- Utilise une technique de filtrage à base de motifs (patterns) et de modèles (template) décrits dans des règles (template rules) pour transformer des arbres.
- XSLT permet la génération d'autres contenus à partir d'un fichier XML, par exemple:
  - du HTML (bien formé)
  - du XML plus compliqué (tables de matière + règles de formatage)
  - du Latex, format Microsoft RTF, PDF, du code Java,
  - …

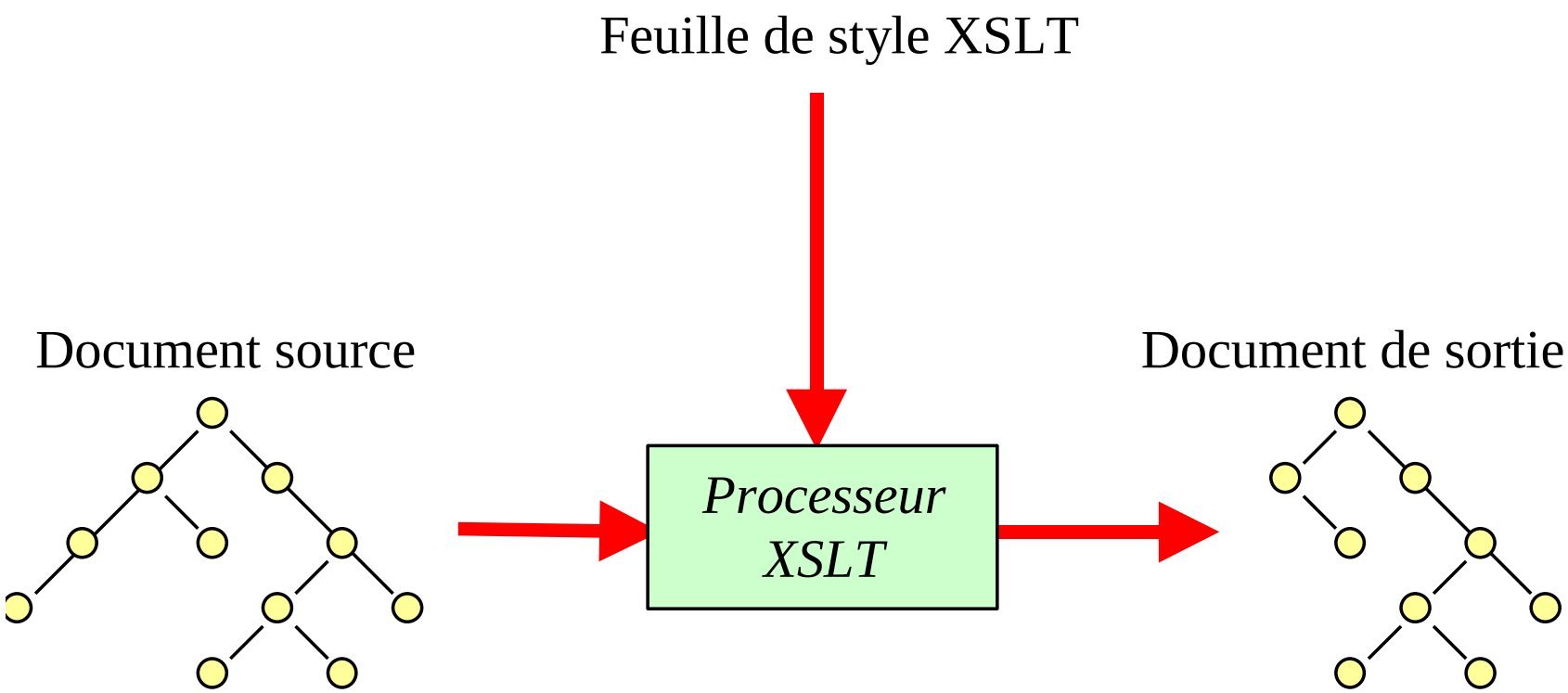
Présentation du langage XSLT

- Une feuille de style XSLT est un document séparé qui contient des règles de transformation XSLT
- Une règle de transformation comporte deux parties:
  - Un modèle de chemin exprimé en termes du langage XPath,
  - Une transformation sous la forme d'une suite d'instructions.
- On peut associer une feuille de style XSLT à un (ou plusieurs) documents XML

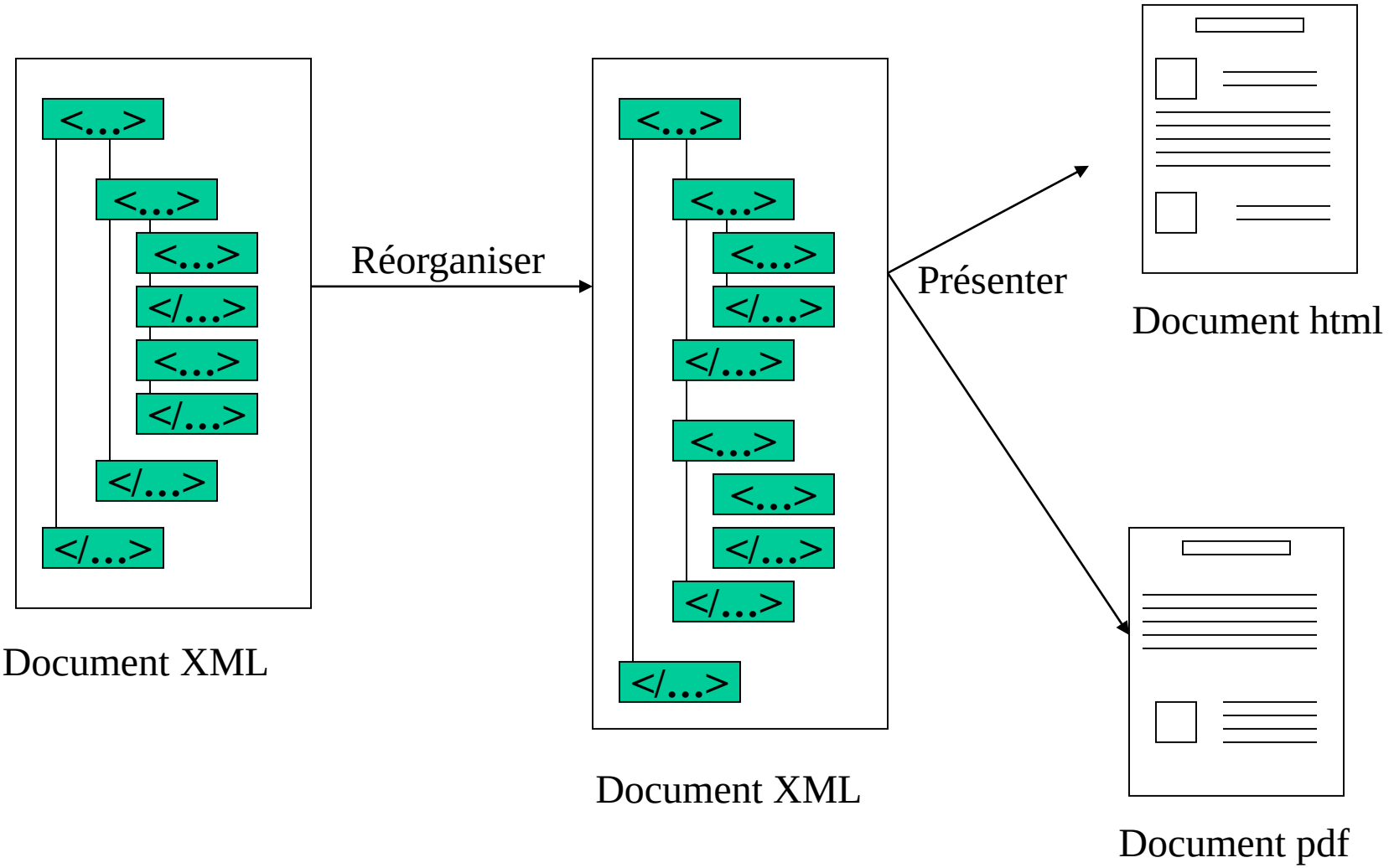
Objectifs du langage XSLT

- Fonctions de base (transformations) offertes par une feuille de style XSLT :
  - Sélection et extraction de données
  - Génération de contenu
  - Suppression de contenu
  - Déplacement de contenu
  - Duplication de contenu
  - Filtre de données
  - Tri de données
  - Etc.

XSLT = Transformation d'arbre



XSLT = Transformation d'arbre



Association d'un fichier XSL à un fichier XML

Une feuille de style XSL (enregistrée dans un fichier dont l'extension est *.xsl*) peut être liée à un document XML en insérant la balise suivante au début du document XML :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="project.xsl" ?>
<racine_du_document>
<-- corps du document XML -->
</racine_du_document>
```

- La valeur de **type** indique le type de la feuille de style: XSL ou CSS
- La valeur de **href** donne le nom du document lié qui contient le programme de transformation.

Structure de document XSLT

- Les instructions sont des éléments XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
 <!-- déclaration de version et de namspace-->
   version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <!-- Format de sortie -->
 <xsl:output method="xml" version="1.0" encoding="UTF-8"/>

 <!-- ... règles XSLT ... -->
  <xsl:template match="condition">
   <!-- ... Actions.. -->
  </xsl:template>
</xsl:stylesheet>
```

Elément <xsl:stylesheet>

Élément racine d'un document XSL.

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

- Attribut `version` : version de langage XSL
- Attribut `xmlns:xsl` : espace de nom XSL.
- Le fait que toutes les balises XSL commencent par xsl: empêche toute confusion.

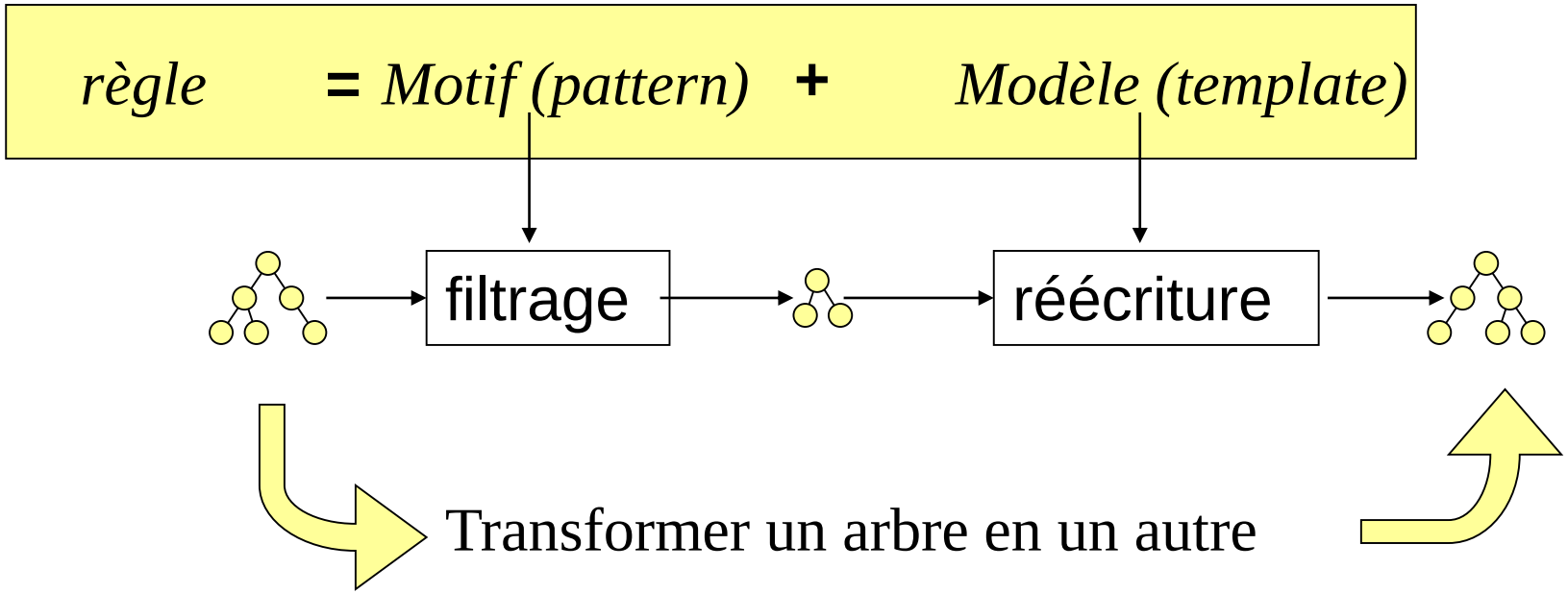
Elément <xsl:output>

- Format de sortie du document résultat

```
<xsl:output method="xml" encoding="UTF-8" …/>
```

  - Attribut `method` : type du document en sortie (XML, HTML, du code Java, format Microsoft RTF, LaTex,…)
  - Attribut `encoding` : codage du document (ISO-8859-1, UTF-8,…)

Règles de réécriture
Template rules



Élément <xsl:template>

- Un programme XSLT est un ensemble de règles
- Une règle est décrite sous forme d'un élément XML :

```
<xsl:template match="condition">
<!-- Actions -->
</xsl:template>
```

- ***Condition***: c'est une expression XPATH
- ***Actions***: c'est des instructions XSLT et/ou données littérales.

Actions de base

Les actions à l'intérieur de **xsl:template** peuvent être:

- Commande de recopie du contenu, sans balises

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

- Commande de recopie de l'élément (contenu + balises)

```
<xsl:copy-of select="Expression-Xpath"/>
```

- Commande de traitement des éléments suivants

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

Élément <xsl:template>

- La transformation XSLT débute par le traitement de la règle racine (c'est la première chose que le processeur XSLT recherche).

```
<xsl:template match="/">
    <!-- actions -->
</xsl:template>
```

- Cette règle s'applique au nœud racine du fichier XML.

Exemple

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

    <xsl:output method="html"/>

    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Welcome</TITLE>
            </HEAD>
            <BODY>
                Welcome!
            </BODY>
        </HTML>
    </xsl:template>

</xsl:stylesheet>
```

Condition :
« *A la racine du document d'entrée !* »

Arbre en sortie

Action :
« *Document html inclus à générer* ! »

Élément <xsl:value-of>

- Insère la valeur du nœud sélectionné sous la forme de texte.

Syntaxe

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

Exemple 1

- Arbre en entrée

```
<carteDeVisite>
  <nom> Mohamed </nom>
</carteDeVisite>
```

- Template

```
<xsl:template match="carteDeVisite">
  <xsl:value-of select="nom"/>
</xsl:template>
```

- Arbre en sortie

```
Mohamed
```

Exemple 2

Arbre en entrée

```
<carnetDAdresse>
    <carteDeVisite> <nom> Ali </nom></carteDeVisite>
    <carteDeVisite><nom> Salima </nom></carteDeVisite>
    <carteDeVisite><nom> Mohmed </nom></carteDeVisite>
</carnetDAdresse>
```

Template

```
<xsl:template match="carteDeVisite">
 <li> Nom : <xsl:value-of select="nom"/> </li>
</xsl:template>
```

Arbre en sortie

```
<li>Nom : Ali</li>
<li>Nom : Salima</li>
<li>Nom : Mohmed</li>
```

- Nom: Ali
- Nom: Salima
- Nom: Mohmed

Élément <xsl:copy-of>

- Recopie de l'élément (contenu + balises).

Syntaxe

```
<xsl:copy-of select="Expression-Xpath"/>
```

Exemple

- Arbre en entrée

```
<carteDeVisite>
  <nom> Mohamed </nom>
</carteDeVisite>
```

- Template

```
<xsl:template match="carteDeVisite">
  <xsl:copy-of select="nom"/>
</xsl:template>
```

- Arbre en sortie

```
  <nom> Mohamed </nom>
```

<xsl:apply-templates>

- Permet d'appliquer un modèle de manière récursive aux fils du nœud courant .

```
<xsl:template match="Expression-Xpath">
    …
  <xsl:apply-templates select="Fils-à-traiter"/>
    …
</xsl:template>
```

- L'attribut `select` permet de spécifier certains éléments enfants auxquels la transformation doit être appliquée.
- En l'absence de l'attribut `select`, l'instruction `<xsl:apply-templates>` traite tous les fils du nœud courant, y compris les nœuds textuels.

Exemple

- Pour un document source contenant 4 cartes de visite

```
<carnetDAdresse>
   <carteDeVisite> <nom> Mohamed </nom> </carteDeVisite>
   <carteDeVisite> <nom> Ali </nom> </carteDeVisite>
   <carteDeVisite> <nom> Amina</nom> </carteDeVisite>
   <carteDeVisite> <nom> Zohra </nom> </carteDeVisite>
</carnetDAdresse>
```

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="carnetDAdresse">
  <html>
     <body>
       <h1>Liste des Noms</h1>
       <xsl:apply-templates/>
     </body>
  </html>
</xsl:template>
<xsl:template match="carteDeVisite">
     <p>Nom : <xsl:value-of select="nom"/>
     </p>
</xsl:template>
```

```
<html>
 <body>
 <h1>Liste des Noms</h1>
       <p>Nom : Mohamed</p>
       <p>Nom : Ali</p>
       <p>Nom : Amina</p>
       <p>Nom : Zohra</p>
 </body>
</html>
```

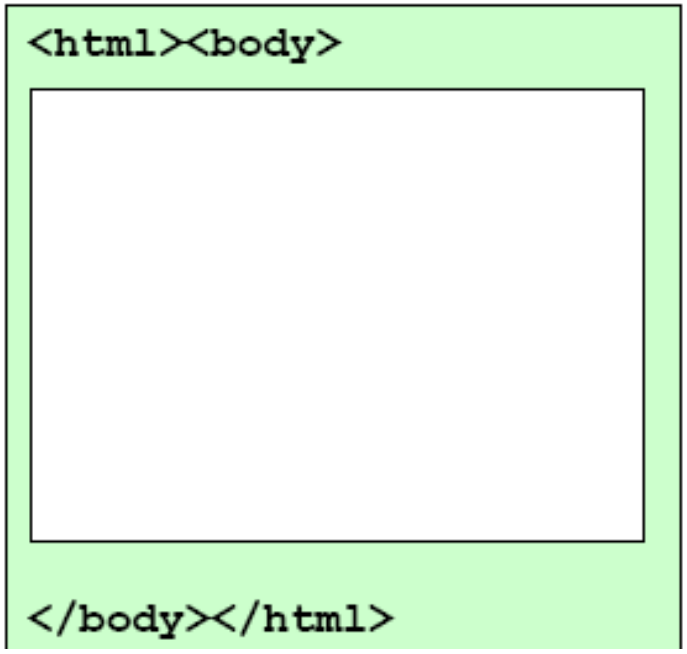
Exemple

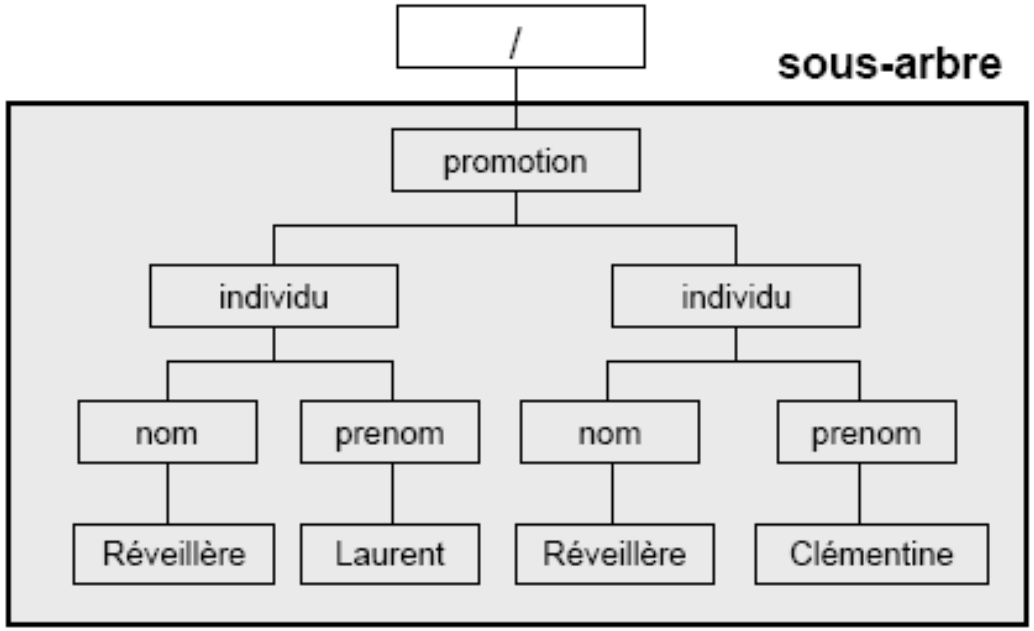
```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<promotion>
   <individu> <nom>Réveillère</nom>  <prenom>Laurent</prenom> </individu>
   <individu> <nom>Réveillère</nom>  <prenom>Clémentine</prenom> </individu>
</promotion>
```

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   <xsl:template match="/">
      <html><body> <xsl:apply-templates/> </body></html>
   </xsl:template>
   <xsl:template match="promotion">
      <table border="1"> <xsl:apply-templates/> </table>
   </xsl:template>
   <xsl:template match="individu">
      <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:stylesheet>
```

Étape 1

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



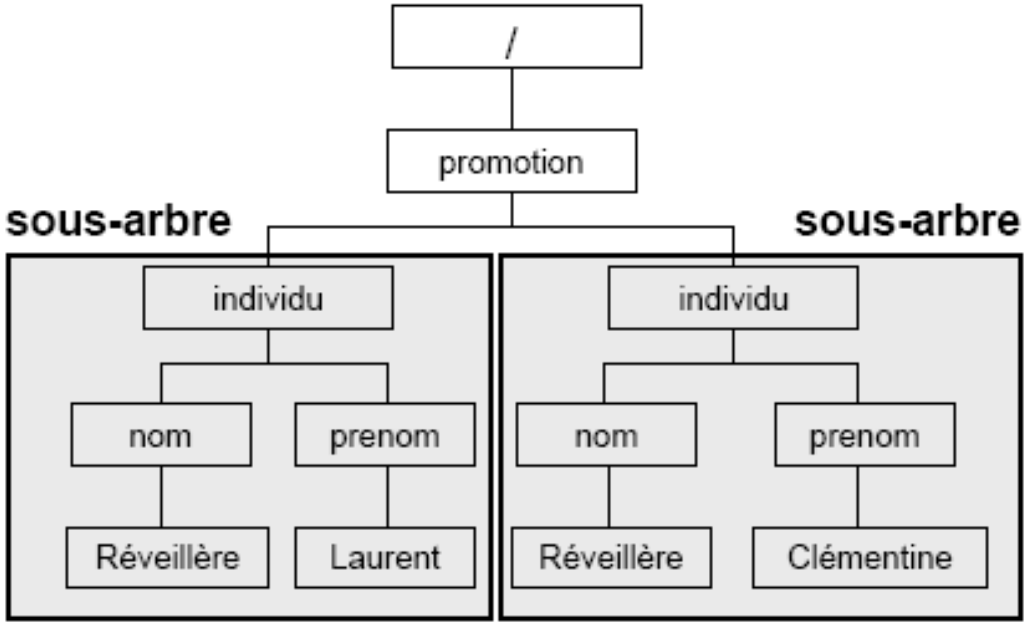


Étape 2

```
<xsl:template match="promotion">
   <table border="1"> <xsl:apply-templates/> </table>
</xsl:template>
```

```
<html><body>
<table border="1">
```

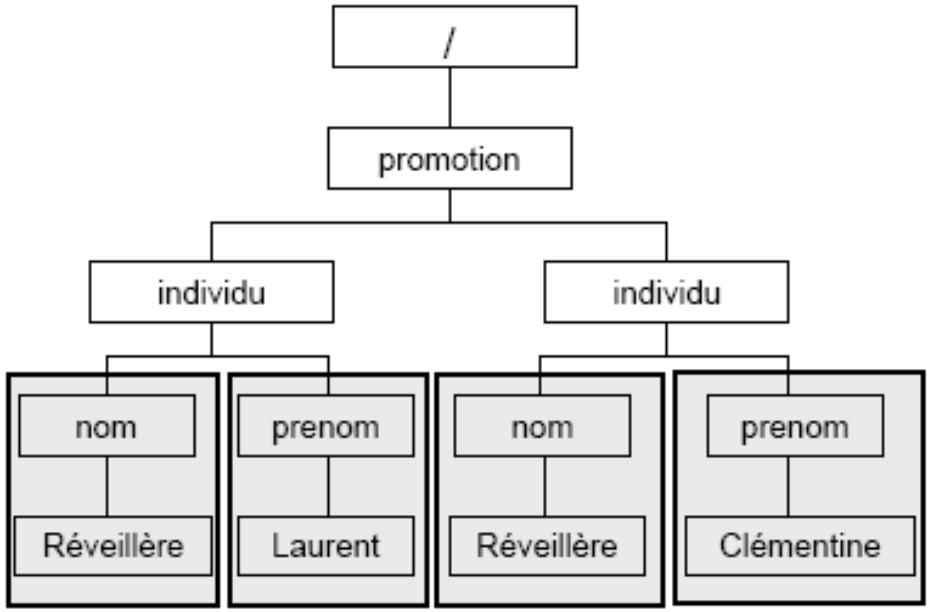
```
</table>
</body></html>
```



Étape 3

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

```
<html><body>
<table border="1">
  <tr>
    
  </tr>
  <tr>
    
  </tr>
</table>
</body></html>
```

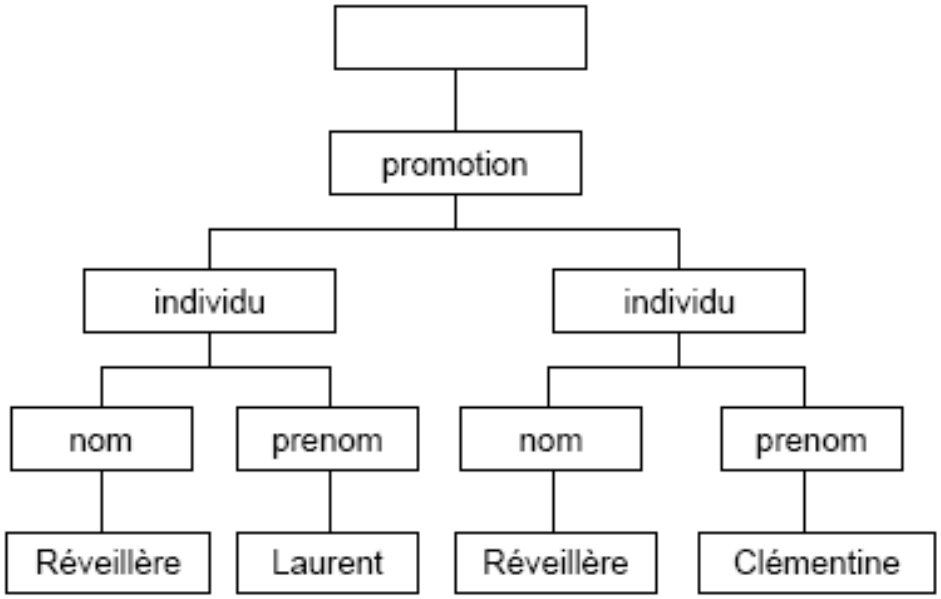


Étape 4

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

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

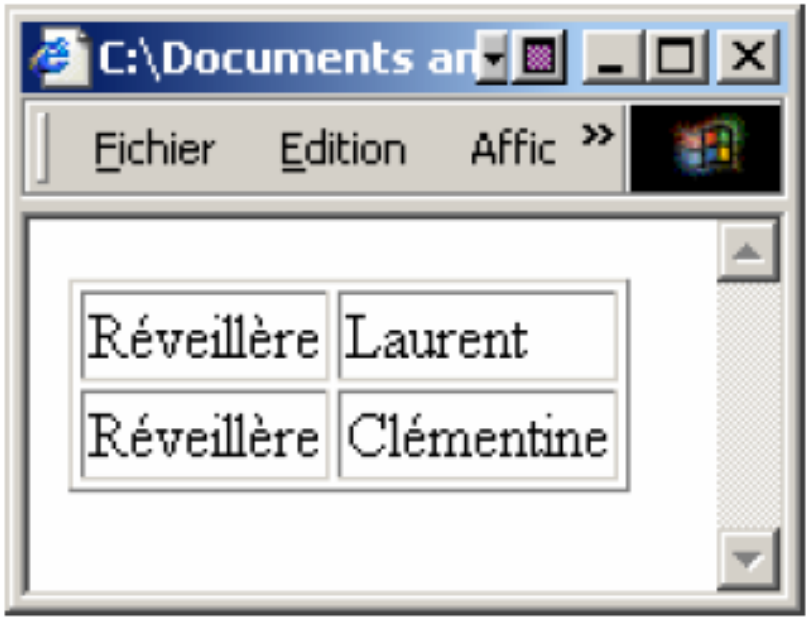
```
<html><body>
<table border="1">
  <tr>
    <td>Réveillère</td>
    <td>Laurent</td>
  </tr>
  <tr>
    <td>Réveillère</td>
    <td>Clémentine</td>
  </tr>
</table>
</body></html>
```



Résultat

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<promotion>
   <individu> <nom>Réveillère</nom>  <prenom>Laurent</prenom> </individu>
   <individu> <nom>Réveillère</nom>  <prenom>Clémentine</prenom> </individu>
</promotion>
```

```
<html><body>
<table border="1">
  <tr>
    <td>Réveillère</td>
    <td>Laurent</td>
  </tr>
  <tr>
    <td>Réveillère</td>
    <td>Clémentine</td>
  </tr>
</table>
</body></html>
```



Production d'élément

- Avec des littéraux

```
<xsl:template match="CarteDeVisite">
    <BusinessCard>
            <!-- contenu de l'élément -->
    </BusinessCard>
</xsl:template>
```

- Avec xsl:element

```
<xsl:template match="CarteDeVisite">
   <xsl:element name="BusinessCard">
         <!-- contenu de l'élément -->
   </xsl:element>
</xsl:template>
```

- En sortie

```
<BusinessCard> <!-- contenu de l'élément --> </BusinessCard>
```

Production d'attribut

- Avec xsl:attribute

```
<xsl:template match="CarteDeVisite">
   < BusinessCard>
         < xsl:attribute name="date">15/04/2020 < /xsl:attribute>
    </BusinessCard>
</xsl:template>
```

- En sortie

```
<BusinessCard date="15/04/2020" />
```

Production d'attribut

- Arbre en entrée

```
<a href="fic.txt"/>
```

- Template

```
<xsl:template match="a">
   <b><xsl:attribute name="id">
         <xsl:value-of select="@href"/>
      </xsl:attribute>
   </b>
</xsl:template>
```

- En sortie

```
<b id="fic.txt"/>
```

Autres productions

- Création de texte :

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

- Création de commentaires :

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

Élement <xsl:for-each>

- Applique un modèle de manière répétée, c'est-à-dire à chaque nœud d'une collection.
- Traite l'un après l'autre tous les nœuds obtenus par l'attribut select.

```
<xsl:for-each select="Expression-Xpath">
  <!-- actions -->
</xsl:for-each>
```

Exemple

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="foreach.xsl" ?>
<customers>
	<customer>
		<name>John Smith</name>
		<address>123 Oak St.</address>
		<state>WA</state>
		<phone>(206) 123-4567</phone>
	</customer>
	<customer>
		<name>Zack Zwyker</name>
		<address>368 Elm St.</address>
		<state>PA</state>
		<phone>(206) 423-4537</phone>
	</customer>
	<customer>
		<name>Albert Aikens</name>
		<address>369 Elm St.</address>
		<state>PA</state>
		<phone>(206) 423-4538</phone>
	</customer>
</customers>
```

Feuille de style foreach.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
      <HTML>
         <BODY>
           <TABLE>
            <xsl:for-each select="customers/customer">
                <TR>
                  <TD><xsl:value-of select="name" /></TD>
                  <TD><xsl:value-of select="address" /></TD>
                  <TD><xsl:value-of select="phone" /></TD>
                </TR>
             </xsl:for-each>
           </TABLE>
         </BODY>
      </HTML>
 </xsl:template>
</xsl:stylesheet>
```

```
John Smith    123 Oak St  (206) 123-4567
Zack Zwyker   368 Elm St. (206) 423-4537
Albert Aikens 368 Elm St. (206) 423-4538
```

Élément <xsl:sort>

- Permet de trier l'ensemble des nœuds sélectionnés avant de les traiter avec possibilité de choisir l'ordre du tri.

- Exemple : trier les cartes de visite par noms

```
<xsl:for-each select="carnetDAdresse/carteDeVisite">
   <xsl:sort select="nom"
             order="descending"
             case-order="lower-first" />
   <!-- actions -->
</xsl:for-each>
```

Tri sur plusieurs critères

- Trier d'abord par noms puis par prénoms

```
<xsl:template match="carnetDAdresse">
   <xsl:for-each select="carteDeVisite">
      <xsl:sort select="nom"/>
      <xsl:sort select="prénom"/>
   </xsl:for-each>
</xsl:template>
```

Exemple

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="foreach.xsl" ?>
<customers>
	<customer>
		<name>Zack Zwyker</name>
		<address>368 Elm St.</address>
		<state>PA</state>
		<phone>(206) 423-4537</phone>
	</customer>
	<customer>
		<name>Albert Aikens</name>
		<address>369 Elm St.</address>
		<state>PA</state>
		<phone>(206) 423-4538</phone>
	</customer>
	<customer>
		<name>John Smith</name>
		<address>123 Oak St.</address>
		<state>WA</state>
		<phone>(206) 123-4567</phone>
	</customer>
</customers>
```

Feuille de style foreach.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
      <HTML>
         <BODY>
           <TABLE>
            <xsl:for-each select="customers/customer">
              <xsl:sort select="state" order="descending"/>
              <xsl:sort select="name"/>
                <TR>
                  <TD><xsl:value-of select="name" /></TD>
                  <TD><xsl:value-of select="address" /></TD>
                  <TD><xsl:value-of select="phone" /></TD>
                </TR>
              </xsl:for-each>
            </TABLE>
          </BODY>
      </HTML>
 </xsl:template>
</xsl:stylesheet>
```

```
John Smith    123 Oak St  (206) 123-4567
Albert Aikens 368 Elm St. (206) 423-4538
Zack Zwyker   368 Elm St. (206) 423-4537
```

Exemple

```
<carnetDAdresse>
    <carteDeVisite> <nom>Ali</nom></carteDeVisite>
    <carteDeVisite><nom>Salima</nom></carteDeVisite>
    <carteDeVisite><nom>Mohmed</nom></carteDeVisite>
</carnetDAdresse>
```

```
<xsl:for-each select="carteDeVisite">
   <xsl:value-of select="nom"/>
</xsl:for-each>
```

En sortie

AliSalimaMohamed

Élément <xsl:if>

Syntaxe

```
<xsl:if test="expression">
  ...some output if the expression is true...
</xsl:if>
```

```
<carnetDAdresse>
   <carteDeVisite> <nom>Ali</nom></carteDeVisite>
   <carteDeVisite><nom>Salima</nom></carteDeVisite>
   <carteDeVisite><nom>Mohmed</nom></carteDeVisite>
</carnetDAdresse>
```

```
<xsl:for-each select="carteDeVisite">
  <xsl:value-of select="nom"/>
  <xsl:if test="position()!=last()">, </xsl:if>
</xsl:for-each>
```

En sortie Ali, Salima, Mohamed

Élément <xsl:choose>

- Utilisé lors de choix parmi différentes conditions possibles.
- Il contient un ou plusieurs éléments <xsl:when> suivi de l'élément optionnel <xsl:otherwise>.
- **Syntaxe**

```
<xsl:choose>
   <xsl:when test="Condition1">
          <!-- actions -->
   </xsl:when>
   <xsl:when test="ConditionN">
          <!-- actions -->
   </xsl:when>
   <xsl:otherwise>
          <!-- autres cas -->
   </xsl:otherwise>
</xsl:choose>
```

Élément <xsl:choose>

```
<xsl:choose>
    <xsl:when test="starts-with(@codep,'22')">
        <xsl:text>Sidi Bel Abbes</xsl:text>
    </xsl:when>
    <xsl:when test="starts-with(@codep,'16')">
        <xsl:text>Alger</xsl:text>
    </xsl:when>
    <xsl:otherwise>
        <xsl:text>Autres villes</xsl:text>
    </xsl:otherwise>
</xsl:choose>
```

Attention !

S'il y a plusieurs conditions, dès que l'une d'elles est considérée comme vraie, toutes les autres sont ignorées (même si elles sont vraies elles aussi).
Seule l'action contenue dans la première condition vraie sera réalisée.

<xsl:variable>

- Les «variables» sont des constantes à la manière des constantes `#define` de C.
- La portée d'une variable est limitée à l'élément dans lequel elle a été définie.
- Une fois déclarée, la valeur d'une variable ne peut être modifiée par la suite.

Syntaxe de la déclaration :

```
<xsl:variable name="MIN" select="0"/>
<xsl:variable name="MAX">9</xsl:variable>
```

Syntaxe de l'invocation:

Une variable sera appelée par son nom, précédé du signe $

```
<xsl:text name="MAX"> Max : {$MAX} </xsl:text>
<xsl:apply-templates select="personne[$MIN]">
```

Initialisation conditionnelle

- Exemple

XSLT

```
<xsl:variable name="code">
  <xsl:choose>
    <xsl:when test="$niveau gt; 20">
      <xsl:text>3</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>5</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

Java

```
if (niveau > 20)
    code = 3;
else
    code = 5;
```

<xsl:param>

La principale différence entre `variable` et `param` est que `param` peut être passé en paramètre à un template (fonction).

Syntaxe de déclaration de paramètre:

```
<xsl:param name="MIN" select="0"/>
<xsl:param name="MAX">9</xsl:param>
```

Syntaxe de déclaration de template (fonction):

```
<xsl:template name="ma_fonction">
    <xsl:param name="MIN"/>
    <!-- actions -->
</xsl:template>
```

Syntaxe de l'invocation:

```
<xsl:call-template name="ma_fonction">
  <xsl:with-param name="debut" select="$MIN"/>
</xsl:call-template>
```

Déroulement de l’exécution des règles

1. Le "moteur" XSLT cherche d’abord à exécuter la première règle qu’il trouve pour l’élément racine ou le cas échéant, pour le "début du fichier XML" ("/").
2. Cette première règle doit faire appel à d’autres règles (sinon l'exécution va s'arrêter et les autres règles seront ignorées).

Il existe deux alternatives pour "faire appel" à d'autres règles:

A. Implicitement:

<xsl:apply-templates/>

B. En faisant appel à des règles précises:

<xsl:apply-templates select="regle"/>

Conclusion

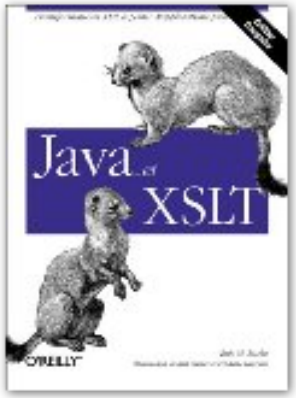
- Oui
  - XSLT est un vrai langage de programmation
  - XSLT n’a pas son équivalent pour la transformation d’arbre
- Mais
  - L’écriture de programmes XSLT peut s’avérer « délicate »
  - La maintenabilité est discutable

Des références

JAVA ET XSLT / BURKE ERIC M

Public | ISBD

Titre :	JAVA ET XSLT
Type de document :	texte imprimé
Auteurs :	BURKE ERIC M, Auteur
Editeur :	O'REILLY
Année de publication :	2002
ISBN/ISSN/EAN :	978-2-84177-205-6
Langues :	Français (*fre*)



XML cours et exercices / ELEXANDRE BRILLANT

Public | ISBD

Titre :	XML cours et exercices : modélisation.schéma et DTD.design patterns.XSLT.DOM .relaxNG.XPath.SOAP.XQuery.XSL-FO.SVG.exist/2e EDITION
Type de document :	texte imprimé
Auteurs :	ELEXANDRE BRILLANT, Auteur
Editeur :	EYROLLES
Année de publication :	2010
ISBN/ISSN/EAN :	978-2-212-12691-4
Langues :	Français (*fre*)