V. Transformation de documents XML avec XSLT

Plan de cette partie

I. Principe de XSLT

II. Règles XSLT

III. Construction de contenu

IV. Les éléments de structure

V. Variables et paramètres XSLT

VI. Clés XSLT

VII. Extensions apportées à XPath

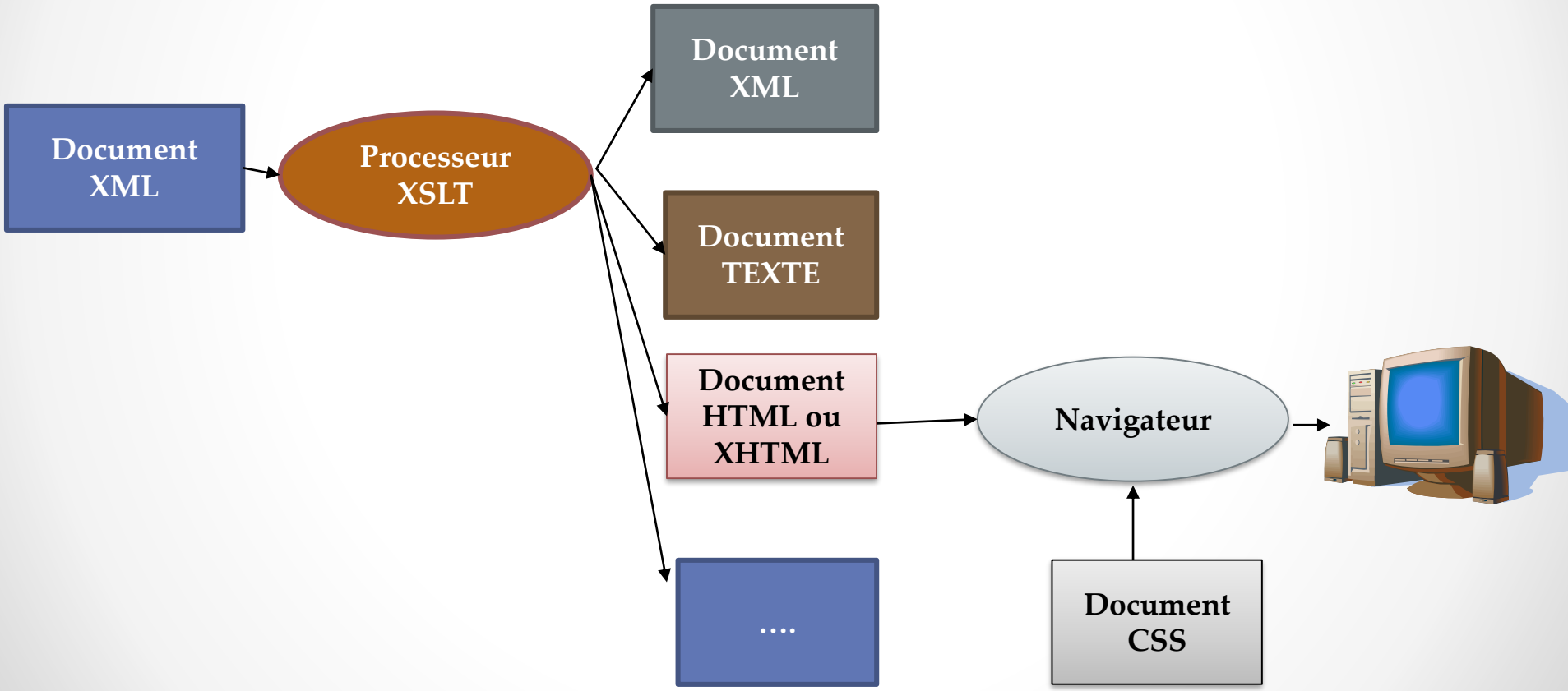
XSL (eXtensible Stylesheet Language)

- **XSL** (*eXtensible Stylesheet Language*) est une famille de spécifications comprenant:
  - **XSLT** (pour *XSL Transformations*, langage de transformations)
  - **XSL-FO** (*XSL Formating Objects*, langage de présentation).

- XSLT est un langage permettant de produire un document XML, HTML, XHTML ou texte à partir d'un autre document en appliquant des règles de transformation.

- *XSL*-FO (Extensible Stylesheet Language Formatting Objects) est un langage qui permet de formatter l'affichage et/ou l'impression d'un document XML.

XSL (eXtensible Stylesheet Language)



- Présentation
- Structure
- Associer un document XSLT à un document XML
- Déclarations
- Principe

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="Livre.xsl"?>

<livre titre="mon livre">
   <auteurs>
    <auteur nom="Martin"  prenom="Bill" />
    <auteur nom="Bob" prenom="Bobby"/>
   </auteurs>
  <sections>
     <section titre="Section1">
              <chapitre titre="un chapitre">
                        <paragraphe>paragraphe 1 </paragraphe>
                        <paragraphe>paragraphe 2 </paragraphe>
              </chapitre>
      </section>
       <section titre="Section2">
               <chapitre titre="autrechapitre">
                        <paragraphe>autreparagraphe1 </paragraphe>
                        <paragraphe>autreparagraphe2 </paragraphe>
              </chapitre>
     </section>
</sections>
</livre>
```

Livre.xml

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"></xsl:output>

<!--Règle 1-->
<xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates select="//section"></xsl:apply-templates>
      </body>
    </html>
</xsl:template>
<!--Règle 2-->
<xsl:template match="section">
     <h2>Titre d'une section: <xsl:value-of select="@titre"/>
     </h2>
  </xsl:template>

</xsl:stylesheet>
```

Titre d'une section: Section1

Titre d'une section: Section2

```
<html>
   <body>
      <h2>Titre d'une section: Section1</h2>
      <h2>Titre d'une section: Section2</h2>
   </body>
</html>
```

Livre.xsl

Comment définir une feuille de style XSL?

- Chaque feuille de style XSL doit commencer par l'élément racine xsl:stylesheet.

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

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

<!—Mettre des règles de transformation-->
</xsl:stylesheet>
```

  - L'attribut version **(obligatoire)** précise la version de la spécification XSL(T) (1.0 ou 2.0 ou 3.0 (recommandation depuis 2017)).
  - Attribut xmlns:xsl : espace de noms XSL
  - La feuille de style est contenue dans l'élément racine xsl:stylesheet.
  - Les fichiers XSLT ont l'extension .xslt ou de préférence .xsl

Comment définir le type de format en sortie?

- Élément **<xsl:output>:** Format de sortie du document résultat

```
<xsl:output method="xml" version="1.0"
      encoding="UTF-8" indent="yes"/>
```

  - Attribut method: type de document en sortie
  - Attribut encoding: codage du document
  - Attribut indent: indentation en sortie

- Différents types de document en sortie:
  - Xml: vérifie que la sortie est bien formée (sortie par défaut)
  - Html: accepte les balises manquantes, génère les entité HTML. (Sortie par défaut si XSL reconnait l'arbre de sortie HTML4)
  - Text: tout autre format textuel
  - XHTML

Comment associer un document XSLT à un document XML?

Inclure dans le document XML, après son prologue, **une instruction de traitement** dont le rôle est de lier le document XML à une feuille de style XSLT.

```
<?xml-stylesheet type="text/xsl" href="./feuilleDeStyle.xslt"?>
```

Comment inclure/importer un autre document XSLT?

- **Inclusion de feuilles XSL**
  - Syntaxe

```
<xsl:include href = "uri-reference"/>
```

  - **Href:** Obligatoire. Référence URI (Uniform Resource Identifier) identifiant le fichier XSLT à inclure.
  - <xsl:include> est enfant de l'élément <xsl:stylesheet>
  - Le contenu du document référencé est interprété exactement comme s'il avait toujours fait partie du document XSLT dans lequel on réalise l'inclusion.

- **Importation de feuilles XSL**

Cette déclaration doit figurer en tête d'une feuille de style.

Les règles importées sont moins prioritaires que les règles définies dans la feuille courante.

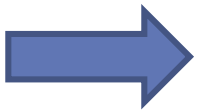
Syntaxe

```
<xsl:import href = "uri-reference"/>
```

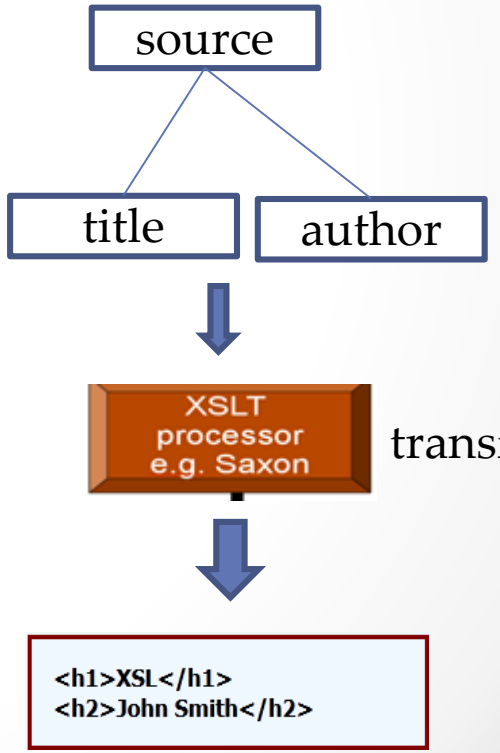
Principe de fonctionnement de XSLT

Lorsqu'un processeur XSLT est invoqué, plusieurs traitements sont effectués :

```
<source>
   <title> XSL </title>
   <author>John Smith </author>
</source>
```



construction



- à partir du document XML source, construction de l'arbre correspondant ;
- Parcours de l'arbre grâce à des expressions XPath
- Application de **règles de transformation** sur l'arbre initial ;
- production du document résultat par
- sérialisation du nouvel arbre.

Transformer un document

- XSLT permet de construire un nouveau document (en XML, XHTML, etc) à partir d'un document XML existant en le transformant

- Extraire des fragments d'un document et les assembler différemment dans une structure nouvelle.

  - **À l'aide des feuilles de style**: un document XML qui contient un ensemble de **règles (template)**

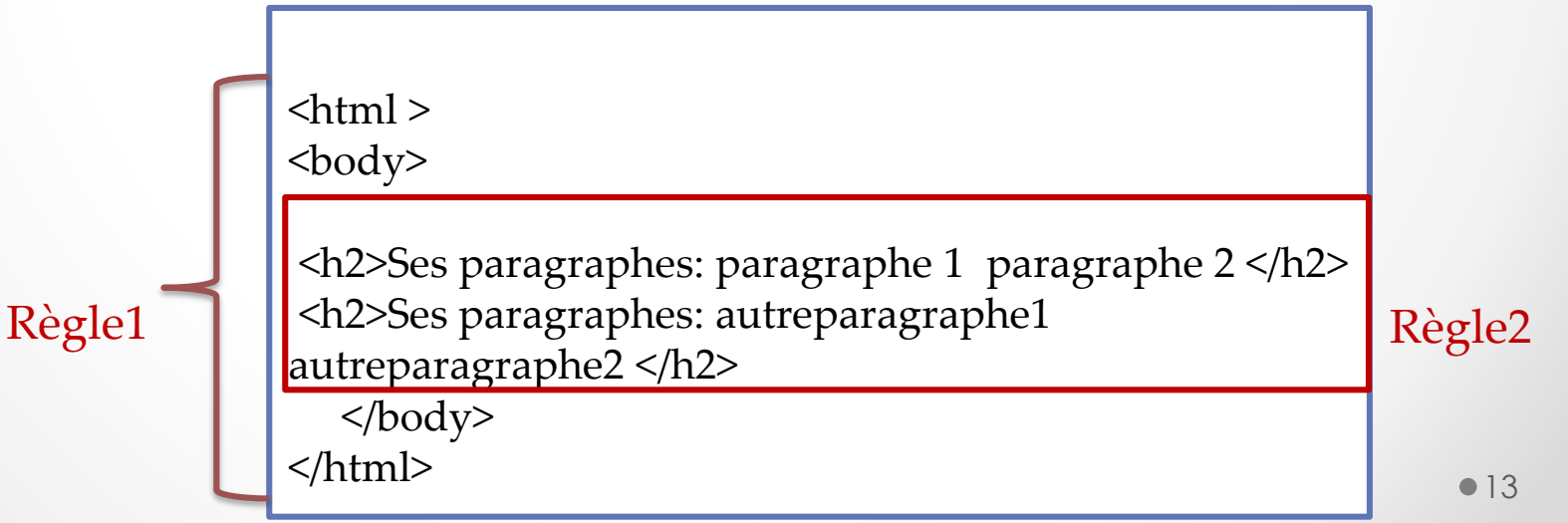
  - **Chaque règle** décrit une transformation à appliquer à certains composants

- XSLT opère sur l'arbre (ordonné) du document source.

Exemple

```
<!--Règle 1-->
<xsl:template match="/">
  <html><body>
 <xsl:apply-templates/>
     </body></html>
  </xsl:template>


<!--Règle 2-->
  <xsl:template match="chapitre">
      <h2>Ses paragraphes: <xsl:value-of select="paragraphe"/>
      </h2>
   </xsl:template>
```



Plan de cette partie

I. Principe de XSLT

II. Règles XSLT
   I. Définition d'une règle XSLT
   II. Les Patterns
   III. Modèle de transformation
   IV. Priorité entre règles

III. Construction de contenu

IV. Les éléments de structure

V. Variables et paramètres XSLT

VI. Clés XSLT

VII. Extensions apportées à XPath

Définition d'une Règle XSLT

- **<xsl:template>** (enfant de <xsl:stylesheet>): permet de définir une règle et précise par un motif XPath, les nœuds sur lesquels elle s'applique.

Syntaxe:

```
<xsl:template
        match = Pattern
        priority = number
        mode = QName>
 <!-- Modèle de transformation-->

</xsl:template>
```

- **Le Pattern** permet d'atteindre des nœuds cibles de la transformation. S'exprime sous forme d'une expression XPath (*l'attribut match*).

- **Le modèle de transformation** décrit ce par quoi il faut remplacer le sous-arbre que le pattern désigne (ou les sous-arbres si le motif en désigne plusieurs).

Les patterns (motifs)

- **Un pattern** est une expression qui, évaluée par rapport à un certain noeud contexte, désigne un certain ensemble de noeuds de l'arbre XML d'un document.

- On ne peut pas associer n'importe quelle expression XPath à l'attribut match
  - Certaines expressions seraient trop complexes à évaluer
  - L'expression doit toujours désigner un ensemble de nœuds

```
<xsl:template match="1">
<xsl:template match="preceding::node()[5] ">
```

NON

- Les axes possibles:
  - Child→ les nœuds enfants d'un élément
  - Attribute → les attributs d'un élément
  - Dans un prédicat, aucun type d'axe de localisation n'est interdit

Modèle de transformation

- Décrit ce par quoi il faut remplacer le sous-arbre que le motif désigne (ou les sous-arbres si le motif en désigne plusieurs).

  →Construction de l'arbre résultat

- Comprend du texte et différentes *instructions XSLT*
  - *Instruction fondamentale* **<xsl:apply-templates>**
  - *Instructions de construction de noeuds*

Instruction xsl:apply-templates

- Permet d'**appliquer** explicitement une règle sur une séquence de nœuds.
- Syntaxe:

```
<xsl:apply-templates
    select = Expression
    mode = QName >
</xsl:apply-templates>
```

- *Sans attributs*: les règles seront appliquées **à tous les fils du nœud contexte**
- Attribut ***select*****:** contenant une expression Xpath: règles appliquées sur les nœuds sélectionnés par l'attribut select
- Attribut ***mode***
  - Permet de choisir explicitement une des règles parmi celles qui sont candidates

  →Un nœud peut être traité plusieurs fois pour générer un résultat différent à chaque fois

  →Produire plusieurs résultats à partir d'un noeud

Exemple

```
<exemple>
    <titre>ceci est un titre</titre>
    <auteur>ceci est un auteur</auteur>
</exemple>
```

→

```
<html>
    <head>
    <title> transformer le document exemple </titre>
    </head>
    <body>
      <h1>ceci est un titre</h1>
      <h1>ceci est un auteur</h1>
    </body>
</html>
```

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
    <xsl:template match="/">
        <html>
        <head> <title> transformer le document exemple</title> </head>
        <body> <xsl:apply-templates/> </body>
    </html>
    </xsl:template>

    <xsl:template match="titre|auteur">
      <h1> <xsl:value-of select="."></xsl:value-of>
      </h1>
    </xsl:template>

</xsl:stylesheet>
```

Règle 1: s'applique à la racine '/' crée la structure du document HTML.

Examine tous les noeuds enfants dans l'ordre

Règle 2: s'applique à l'élément titre

Règle 3: s'applique à l'élément auteur

- Quelle règle choisir dans le cas de plusieurs règles éligibles?
- La priorité peut être spécifiée explicitement avec l'attribut **priority**.
- Sinon c'est la règle la plus spécifique qui est choisie

```
<?xml version="1.0" encoding="UTF-8"?>
<table> <description>personne1 etage4 </description>
        <personne><nom>Bond</nom></personne>
        <personne><nom>Lupin</nom></personne>
        <personne><nom>Templar</nom><bureau>U3</bureau></personne>
</table>….
```

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

<xsl:template match="personne[bureau]">
    <autreregle> <xsl:apply-templates"/></autreregle>
  </xsl:template>
```

Pour <personne>…<personne>→la première règle s'applique
Pour <personne>…<bureau>…</bureau><personne> → seule la seconde s'applique

Exemple

- Appliquer la feuille de style livre.xsl sur votre document livre.xml
- Quel résultat obtenez vous?
- **Maintenant ajouter dans votre feuille de style :**

```
<!—Règle 3-->
<xsl:template match="chapitre">
    <h2>Ses paragraphes: <xsl:value-of select="paragraphe"/>
    </h2>
  </xsl:template>
```

Quel résultat obtenez vous? Modifier l'exemple pour obtenir un résultat

Plan de cette partie

I. Principe de XSLT
II. Règles XSLT

III. Construction de contenu
   1. Nœud textuel par XPath
   2. Texte brut
   3. Nœud élément
   4. Nœud attribut
   5. Groupe d'attributs
   6. Copie de nœud
   7. Création de commentaire
   8. Création d'une instruction de traitement

IV. Les éléments de structure
V. Variables et paramètres XSLT
VI. Clés XSLT
VII. Extensions apportées à XPath

Construction de contenu

- Chaque application de règle de la feuille de style produit un fragment du résultat.
- Ce fragment est construit à partir du contenu de l'élément <xsl:template> et d'autres éléments permettant d'insérer d'autres nœuds calculés.
- Construction de contenu
  - Noeud textuel par Xpath
  - Texte brut
  - Nœuds élément et attribut
  - Liste d'attributs
  - Copie de nœuds
  - Commentaire et instruction de traitement
  - etc

1. Nœud textuel par XPath

- Produire un noeud textuel identifié par une expression XPATH en utilisant l'instruction **<xsl:value-of>**
- Syntaxe:

```
<xsl:value-of
    select = Expression
    disable-output-escaping = "yes" | "no" >
</xsl:value-of>
```

- *<xsl:value-of select="..." />* est remplacée lors de l'instanciation du modèle par la valeur textuelle de ce qui est désigné par **l'attribut select** (obligatoire).

→ Extraction du contenu de l'arbre en entrée

- **disable-output-escaping** (optionnel): pour le traitement des caractères spéciaux. Par exemple ">" sera affiché ">" si cette propriété est à "yes"

Exemple

- En entrée:

```
<carteDeVisite>
   <nom> Martin </nom>
</ carteDeVisite >
```

- Règle:

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

- En sortie:

```
<p>Nom:Martin </p>
```

Exemple

- En entrée:

```
<note> enseigne <clé>XML</clé> au Master </note>
```

- Règle:

```
<xsl:template match=" note" >
<xsl:value-of select=" text()"/>
</ xsl:template >
```

- En sortie:

```
Enseigne au Master
```

Seul le premier élément sélectionné est produit

2. Texte brut

- L'élément **<xsl:text>** utilise son contenu pour créer un nœud textuel dans le document résultat.

- Syntaxe:

```
<xsl:text disable-output-escaping = "yes" | "no">
  ....Texte....
</xsl:text>
```

2. Texte brut

- Exemple:

```
<root>
    <!-- document quelconque -->
</root>
```

```
<xsl:template match="/">
    <xsl:text>
    un texte
       avec des espaces &amp; des sauts de ligne
    </xsl:text>
    <xsl:text  disable-output-escaping="yes">
       un autre texte avec des espaces &amp; des sauts de ligne
    </xsl:text>
</xsl:template>
```

```
un texte
avec des espaces &amp;

des sauts de ligne

un autre texte
avec des espaces &

des sauts de ligne
```

3. Nœud élément

- **<xsl:element>** Crée un élément de sortie et lui donne le nom spécifié (nom calculé dynamiquement).

  Syntaxe:

```
<xsl:element
        name = "element-name"
        namespace = "uri-reference"
        use-attribute-sets = QName>
</xsl:element>
```

  - **Name (attribut obligatoire):** nom de l'élément à créer
  - **Namespace (optionnel):** URI d'espace de noms de l'élément créé.
  - **Use-attribute-sets:** Liste d'ensembles d'attributs, séparés par des espaces

Exemple

```
<xsl:template match="part">
    <xsl:element name="partie">
        <xsl:value-of select="title"/>
    </xsl:element>
</xsl:template>
```

<partie> Le titre </partie>

```
<?xml version="1.0"?>

<part>
  <title> Le titre </title>
</part>
```

```
<xsl:template match="part">
    <partie>
        <xsl:value-of select="title"/>
    </partie>
</xsl:template>
```

<partie> Le titre </partie>

Exemple

```
<chapitre titre="Premier_chapitre " >
            <paragraphe>premier</paragraphe>
            <texte>corps</texte>
</chapitre>
```

Résultat

```
<description>
            <premier/>
            <corps/>
</description>
```

```
<xsl:template match="chapitre">
    <xsl:element name="description">
        <xsl:element name="{paragraphe}"></xsl:element>
        <xsl:element name="{texte}"></xsl:element>
     </xsl:element>
</xsl:template>
```

Le nom (ou le contenu) est calculé dynamiquement et communiqué dans l'attribut name sous la forme d'une expression entre accolades.

4. Nœud attribut

- **<xsl:attribute>** Crée un nœud d'attribut et le joint à un élément de sortie.

Syntaxe:

```
<xsl:attribute
  name = "attribute-name"
  namespace = "uri-reference">
</xsl:attribute>
```

  - Name (attribut obligatoire): nom de l'attribut à créer
  - Namespace (optionnel): URI d'espace de noms de l'attribut créé.

Exemple

```
<personne>
   <nom> Martin </nom>
   <prenom> Jacques </prenom>
</personne>
```

```
<xsl:template match = "personne" >
    <xsl:element name = "presentateur" >
        <xsl:attribute name = "{name()}">
            <xsl:value-of select="nom"/>
            <xsl:text>  </xsl:text>
            <xsl:value-of select="prenom"/>
        </xsl:attribute>
    </xsl:element>
</xsl:template>
```

```
<presentateur personne="Martin Jacques"/>
```

Résultat

5. Groupe d'attributs

- **<xsl:attribute-set>:** Définit un ensemble nommé d'attributs.

→Regrouper les définitions d'attributs pour les réutiliser associées à plusieurs éléments (tableaux, paragraphes, images, etc.)

- Syntaxe:

```
<xsl:attribute-set
  name = QName
  use-attribute-sets = QNames>
</xsl:attribute-set>
```

  - Name (attribut obligatoire): nom de l'ensemble d'attributs
  - Use-attribute-sets: Liste d'ensembles d'attributs, séparés par des espaces

- Les listes sont définies en dehors des règles.

Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:attribute-set name="MonStyle">
        <xsl:attribute name="bgcolor">white</xsl:attribute>
        <xsl:attribute name="font-name">Helvetica</xsl:attribute>
        <xsl:attribute name="font-size">18pt</xsl:attribute>
    </xsl:attribute-set>
    <xsl:template match="/">
        <html>
        <head><title>Ficher personne</title></head>
        <body xsl:use-attribute-sets="MonStyle">
        Exemple de use-attribut-sets
        </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Résultat

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Ficher personne</title>
</head>
<body bgcolor="white" font-name="Helvetica" font-size="18pt">
   Exemple de use-attribut-sets
</body>
</html>
```

6. Copie de nœud

- **Copie du nœud courant (sans attribut select)** dans le document résultat:

```
<xsl:copy
  use-attribute-sets = QNames>
</xsl:copy>
```

- **Copie de nœuds:**

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

est instanciée comme une copie conforme des éléments sélectionnés

- permet de copier des nœuds sélectionnés ainsi que tout son sous arbre (nœuds d'attributs, espaces de noms et les enfants du nœud d'élément) dans le document résultat.

Exemple

```
<xsl:template match="sections">
<xsl:copy-of select="section[1]"/>
</xsl:template>
```

Résultat

```
<section titre="Section1">

<chapitre titre="un chapitre">
   <paragraphe>paragraphe 1 </paragraphe>
    <paragraphe>paragraphe 2 </paragraphe>
 </chapitre>
</section>
```

7. Création de commentaires

- **<xsl:comment>** Génère un commentaire dans la sortie.
- Syntaxe:

```
<xsl:comment>
    ....texte du commentaire..
</xsl:comment>
```

- Exemple

```
<xsl:comment> mon commentaire  </xsl:comment>
```

En sortie

```
<!--mon commentaire-->
```

8. Création d'instruction de traitement

- **<xsl:processing-instruction>** permet de générer une instruction de traitement
- Exemple

```
<xsl:processing-instruction name="xml-stylesheet">
href="book.css" type="text/css"
</xsl:processing-instruction>
```

Génère dans l'arbre final le nœud instruction de traitement suivant:

```
<?xml-stylesheet href="book.css" type="text/css"?>
```

Plan de cette partie

I. Principe de XSLT
II. Règles XSLT
III. Construction de contenu

IV. Les éléments de structure
1. Les sections conditionnelles
2. Le traitement conditionnel multiple
3. Boucle
4. Le tri

V. Variables et paramètres XSLT
VI. Clés XSLT
VII. Extensions apportées à XPath

Les éléments de structure

1. Les sections conditionnelles: *<xsl:if>*

2. Le traitement conditionnel multiple: *<xsl:choose>*, *<xsl:when>*

3. Boucle: <xsl : for-each>

4. Le tri: <xsl:sort>

1.Traitement conditionnel : *<xsl:if>*

Syntaxe :

```
<xsl:if test="condition-booléenne">
                    Instructions...
</xsl:if>
```

Pas de else

Exemple

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

  <xsl:if test="paragraphe">
        Nombre de paragraphes: <xsl:value-of select="count(paragraphe)"/>
</xsl:if>

 </xsl:template>
```

Nombre de paragraphes: 2 Nombre de paragraphes: 2

2. Traitement conditionnel : *<xsl:choose>*

- Permet de choisir une forme à appliquer parmi un éventail de possibilités, chacune identifiée par un test. Chaque alternative de forme est indiquée au sein d'une instruction <xsl:when>.

- <xml:choose> avec : <xsl:when> et <xsl:otherwise>

```
<xsl:choose>
    <xsl:when test="expression XPath“>
            [action]
     </xsl:when>
   <xsl:when test=“autre expression XPath“>
            [action]
     </xsl:when>
…..
     <xsl:otherwise>
       [action]
  </xsl:otherwise>
</xsl:choose>
```

Exemple

- Expliquer ce fragment de feuille de style

```
<xsl:choose>
    <xsl:when test="titre">
            <xsl:value-of select="titre"></xsl:value-of>
     </xsl:when>
<xsl:otherwise>
 <xsl:number level="single" count="section"></xsl:number>
 <xsl:text>titre</xsl:text>
</xsl:otherwise>
</xsl:choose>
```

1titre 2titre

- Le fragment de feuille de style retourne le contenu de l'enfant titre de nœud courant si cet enfant existe ou construit un titre avec un numéro sinon.
- <xsl:number>: permet de générer un numéro calculé et de l'insérer dans l'arbre résultat. Permet ainsi de générer les listes numérotées.

3. Boucle

Structure de répétition **<xsl:for-each>**

- parcourir un ensemble de noeuds sélectionnés avec select
- Les instructions sont appliquées successivement à chaque noeud sélectionné

Pas de variable, donc pas d'incrémentation

Syntaxe:

```
<xsl:for-each  select=Expression>
                    Instructions...
 </ xsl:for-each >
```

Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<Texte>
    <Chapitre>
        <Titre>titre1</Titre>
        <Section>
            <Titre>titresect1</Titre>
        </Section>
    </Chapitre>
    <Chapitre>
        <Titre>titre2 </Titre>
        <Section>
            <Titre>titresect2</Titre>
        </Section>
    </Chapitre>
</Texte>
```

```
<xsl:template match="/">
    <R>
        <xsl:for-each select=".//Titre">
            <H1>
             <xsl:apply-templates/>
            </H1>
        </xsl:for-each>
    </R>
</xsl:template>
```

```
<?xml version="1.0" encoding="utf-8"?>
<R>
  <H1>titre1</H1>
  <H1>titresect1</H1>
  <H1>titre2</H1>
  <H1>titresect2</H1>
</R>
```

Résultat

titre1

titresect1

titre2

titresect2

Exemple

- Modifier livre.xsl pour afficher pour chaque auteur son nom et prénom

```
<xsl:template match="auteurs">
        <xsl:for-each select="auteur">
                <xsl:value-of select="@nom"/>
                <xsl:text>     </xsl:text>
                <xsl:value-of select="@prenom"/>
        </xsl:for-each>
</xsl:template>
```

4. Tri: *xsl:sort*

- Instruction de tri
- Permet de trier de nœuds sélectionnés par les instructions *xsl:apply-templates* ou *xsl:for-each*
- À placer après la balise ouvrante de *xsl:for-each* ou *xsl:apply-templates.*
- Par défaut, l'ordre du tri est croissant (éléments ordonnés suivant l'ordre lexicographique de la valeur textuelle de chaque élément)
- En l'absence d'une instruction *<xsl:sort/>*, *<xsl:for-each>* et *<xsl:apply-templates>* constituent une liste des éléments à traiter, basée sur **l'ordre naturel de lecture** du document XML.

4.Critères de Tri: *xsl:sort*

- Syntaxe

```
<xsl:sort  select=Expression
           order={"ascending"|"descending"}
           case-order={"upper-first"  | "lower-first " }>
</ xsl:sort >
```

- Utilisation des attributs: *select, order, case-order*, *lang, data-type*.
  - **Select:** définit la clé du tri. Prendra comme valeur une expression XPath (valeur par défaut (.)).
  - **Order:** définit l'ordre du tri (ascendant ou descendant). Peut prendre l'une des 2 valeurs *ascending* (valeur par défaut) ou *descending*.
  - **Case-ordre:** définir la relation d'ordre entre les lettres minuscules et majuscules. Peut prendre les valeurs *upper-first* ou *lower-first*. (valeur par défaut dépend de la langue utilisée).

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<personnes>
   <personne>John</personne>
   <personne>Bob</personne>
   <personne>Diana</personne>
   <personne>Jack</personne>


</personnes
```

```
<xsl:template match="personnes">
   <ol>
    <xsl:apply-templates select="personne">
      <xsl:sort select="text()" order="ascending" lang="FR" data-type="text"/>

    </xsl:apply-templates>
   </ol>

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

1. Bob
2. Diana
3. Jack
4. John

Résultat

Plan de cette partie

I. Principe de XSLT

II. Règles XSLT

III. Construction de contenu

IV. Les éléments de structure

V. Variables et paramètres XSLT
   1. Variables
   2. Paramètres et templates
   3. Templates nommés et fonctions

VI. Clés XSLT

VII. Extensions apportées à XPath

Variables et paramètres

- **Variables:**
  - Les variables servent à stocker des valeurs (atomique, un nœud ou une suite de ces valeurs).
  - Elles peuvent être utilisées dans les expressions XPath.
  - Élément: **< xsl:variable>**

- **Paramètres:**
  - servent à transmettre des valeurs aux règles (l'élément *xsl:param*) .
  - L'élément **<xsl:with-param>** permet d'instancier un paramètre lors de l'appel à une règle.

- La principale différence entre une variable et un paramètre est qu'un paramètre peut être passé comme argument à un template.

1.Variables : *xsl: variable*

- XSLT permet de définir des variables pouvant stocker des valeurs.
- Syntaxe

```
<xsl:variable
    name = QName
    select = Expression>
</xsl:variable>
```

- Les variables peuvent être globales (fils de <xsl:stylesheet) ou locales.
- Une variable, en XSLT comme dans tout autre langage, est l'association d'un nom et d'une valeur. Néanmoins, en XSLT, cette association est indestructible : il est impossible de changer la valeur d'une variable, une fois qu'on l'a déterminée.
- L'attribut name détermine le nom de la variable. La valeur est donnée soit par une expression XPath dans l'attribut select soit directement dans le contenu de l'élément xsl:variable

Exemple

```
<xsl:variable name="v1" select="12"/>
```

→ Attribut select avec la valeur (une constante)

```
<xsl:variable name="v1" select="/COURS/ENSEIGNANTS"/>
```

→Attribut select avec une expression XPath

→ Valeur = contenu du fils de ENSEIGNANTS dans l'arbre

Exemple

```
<personne>
<nom>
    <prenom>Jacques</prenom>
    <famille>Martin</famille>
</nom>
</personne>
```

```
<xsl:template match="/">
    <xsl:variable name="nom" select="personne/nom/prenom"/>
    <xsl:copy-of select="$nom"/>
</xsl:template>
```

```
<prenom>Jacques</prenom>
```

Résultat

Le fait qu'une valeur, (ou un fragment d'arbre), soit affectée à une variable ou à un paramètre ne signifie pas qu'elle sera automatiquement insérée dans l'arbre résultat. Il faut insérer cette valeur grâce à l'instruction xsl:value-of ou xsl:copy-of.
Le paramètre select de cette instruction aura alors pour valeur le nom de la variable ou du paramètre précédé du caractère $.

Exemple

```
<xsl:variable name="head">
    <head>
            <link rel="stylesheet" type="text/css" href="style.css" />
    </head>
</xsl:variable>

<xsl:template match="/">
    <html> <xsl:copy-of select="$head"/>
                    <body>...</body>
     </html>
</xsl:template>
```

2. Paramètres : *xsl: param*

- les paramètres représentent un type particulier de variables. Servent à transmettre des valeurs aux règles.
- Syntaxe

```
<xsl:param name="name“
                select="expression">
</xsl:param>
```

- L'élément param peut être enfant de l'élément racine xsl:stylesheet ou des éléments xsl:template
- L'exemple déclare un paramètre bg-color avec une valeur par défaut égale à la chaîne de caractères white

```
<xsl:param name="bg-color" select="'white'"/>
```

- Ou

```
<xsl:param name="bg-color">white<xsl:param/>
```

Passage de Paramètres : *xsl: with-param*

- Transmet un paramètre à un modèle
- Fils de <xsl:apply-templates> et <xsl:call-template>
- Syntaxe

```
<xsl:with-param
        name = QName>
        select= Expression
 </xsl:with-param >
```

  - **Name:** Obligatoire. Les Noms qualifiés du paramètre.
  - **Select:** Une Expression à comparer au contexte actuel. Il n'y a pas de valeur par défaut. En l'absence de contenu, une chaîne vide est générée.

3. Templates nommés et fonctions

- XSL permet de nommer un template et de l'appeler explicitement à n'importe quel endroit

→ Factorisation de code

- Syntaxe: Déclaration de fonction :

```
<xsl:template name="auteur">
        …
 </xsl:template>
```

- Appel de fonction<xsl:call-template>

```
<xsl:call-template name="auteur"/>
```

- On peut passer des paramètres avec xsl:param

3. Templates nommés et fonctions

```
<xsl:template match="/">
        <xsl:call-template name="auteurs">
        </xsl:call-template>
</xsl:template>

<xsl:template name="auteurs">
        <xsl:for-each select="auteur">
                <xsl:value-of select="@nom"/>
                <xsl:text>     </xsl:text>
                <xsl:value-of select="@prenom"/>
        </xsl:for-each>
</xsl:template>
```

Exemple

```
<xsl:template name="faire-un-lien">
   <xsl:param name="href"/>
   <xsl:param name="target"></xsl:param>

   <a href="{$href}" target="{$target}">
      <xsl:apply-templates/>
   </a>
</xsl:template>
```

Template nommé

```
<xsl:call-template name="faire-un-lien">
            <xsl:with-param name="href" select="'ma-page.html'" />
            <xsl:with-param name="target">ALL</xsl:with-param>
</xsl:call-template>
```

Appel de template avec passage de paramètres

Résultat <a href="ma-page.html" target="ALL">

Plan de cette partie

I. Principe de XSLT

II. Règles XSLT

III. Construction de contenu

IV. Les éléments de structure

V. Variables et paramètres XSLT

VI. Indexation du document XML: Clés XSLT

VII. Extensions apportées à XPath

Clés XSLT

- **<xsl:key>** permet de définir une clé, une paire nom-valeur assignée à un élément spécifié d'un document XML.
- Syntaxe

```
<xsl:key
  name = QName
  match = Pattern
  use = Expression
</xsl:key>
```

  - *name:* nom de la clé
  - *match:* le filtre déterminant le nœud (ou la liste de nœuds) auquel la clé est attachée.
  - *use:* une expression XPath indiquant où les valeurs de la clé devront être recherchées.

- Une clé peut être attachée à n'importe quel type de nœud et non uniquement à des éléments comme les attributs de type ID, IDREF et IDREFS.

Clés XSLT

- **Exemple**

L'instruction suivante définit une clé nommée **idR**. Cette clé permet de faire référence à un nœud **livre** à partir de son **auteur** dans une expression XPATH

```
<xsl:key name="idR" match="livre" use="@auteur">
```

- Pour obtenir les nœuds liés à une clé, il faut utiliser la fonction Xpath: **key (string, object)** qui prend en argument le nom d'une clé et en deuxième argument la valeur d'une clé et retourne un ensemble de nœuds correspondants.

- **Exemple**

```
<livres>
        <livre titre="XML par la pratique"  auteur="Thierry Boulanger"/>
         <livre titre=" XML pour édition"  auteur="Bernard Prost"/>
         <livre titre=" Publishing with XML "  auteur=" Bernard Prost "/>
</livres>
```

```
<xsl:key name="idR" match="livre" use="@auteur">
<xsl:template match="/">
        <html>
                <body>
        <xsl: for-each select="key('idR', 'Bernard Prost')">
          <xsl:value-of select="@titre"/>
        </xsl: for-each>
                </body>
        </html>
 </xsl:template>
```

XML pour édition
Publishing with XML

Plan de cette partie

I. Principe de XSLT

II. Règles XSLT

III. Construction de contenu

IV. Les éléments de structure

V. Variables et paramètres XSLT

VI. Clés XSLT

VII. Extensions apportées à XPath

Extensions apportées à XPath

Fonctions sur les nœuds

Fonction	Définition
current()	retourne le nœud courant
key(nom, objet)	retourne la liste de nœuds du document source qui correspondent au filtre défini par le paramètre match de la clé nom et qui contiennent à l'emplacement indiqué par le paramètre use de cette clé la valeur objet
document(uri)	Renvoie le document XML identifié par l'URI
generate_id(noeud)	Renvoie un identifiant unique

Fonctions sur les chaînes de caractères

Fonction	Définition
format-number(nombre, format)	Convertit un nombre en chaine de caractères en contrôlant le format de sortie.

- Et bien d'autres….

Conclusion

- XSLT est un vrai langage de règles pour la transformation de documents
- Basé sur le langage Xpath
- Est un langage fonctionnel et déclaratif

Liens utiles

- XSLT recommandation W3C:http://xmlfr.org/w3c/TR/xslt/

- XSLT version 2.0 *http://www.w3.org/TR/xslt20/*
- Les éléments XSLT: http://msdn.microsoft.com/fr-fr/library/ms256058%28v=vs.80%29.aspx

- Support de Cours en ligne Elisabeth Murisasco

- Jacques Le Maitre, Description et manipulation de documents XML, supports de cours en ligne http://lemaitre.univ-tln.fr/cours.htm