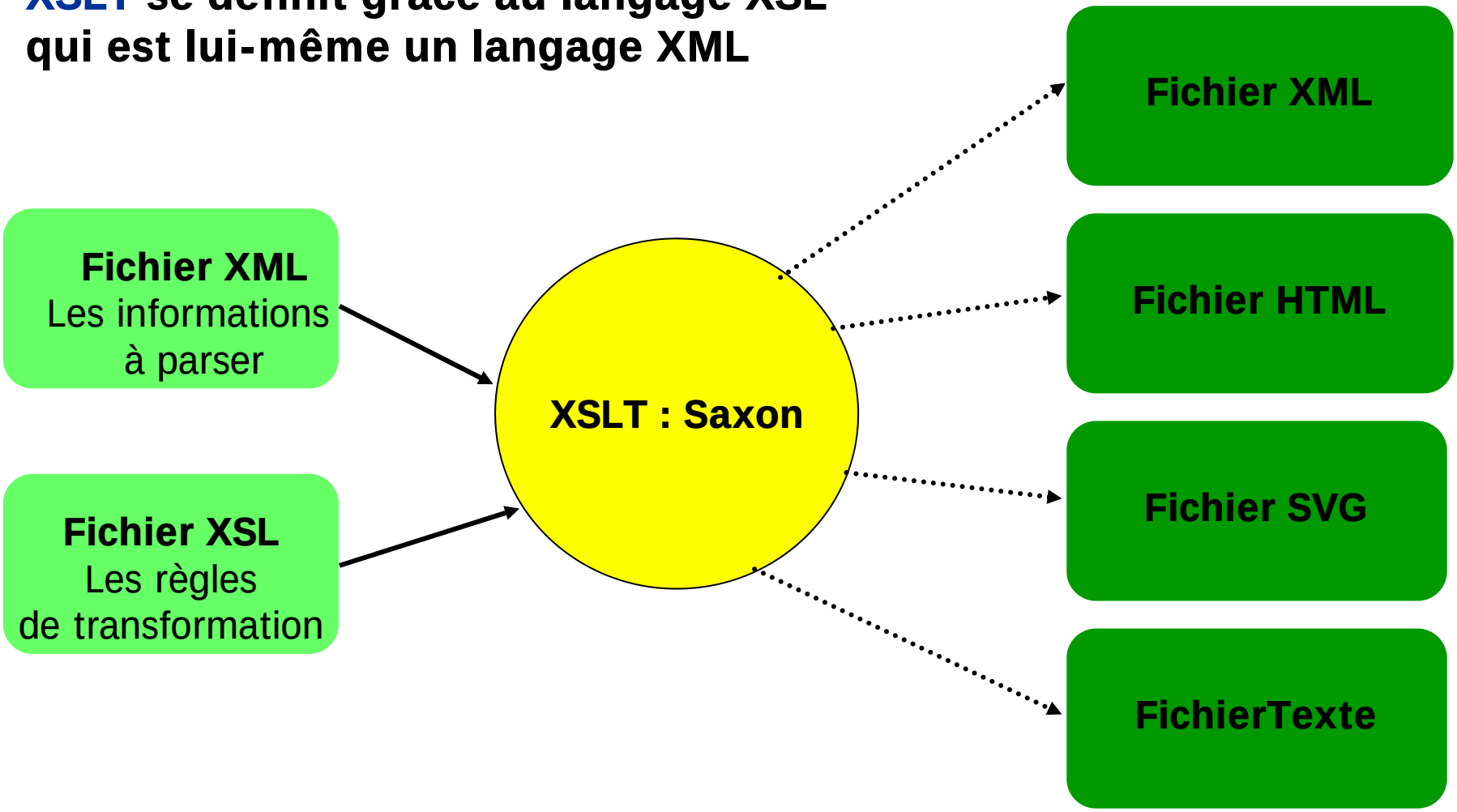
Rappels et approfondissements du langages XSL

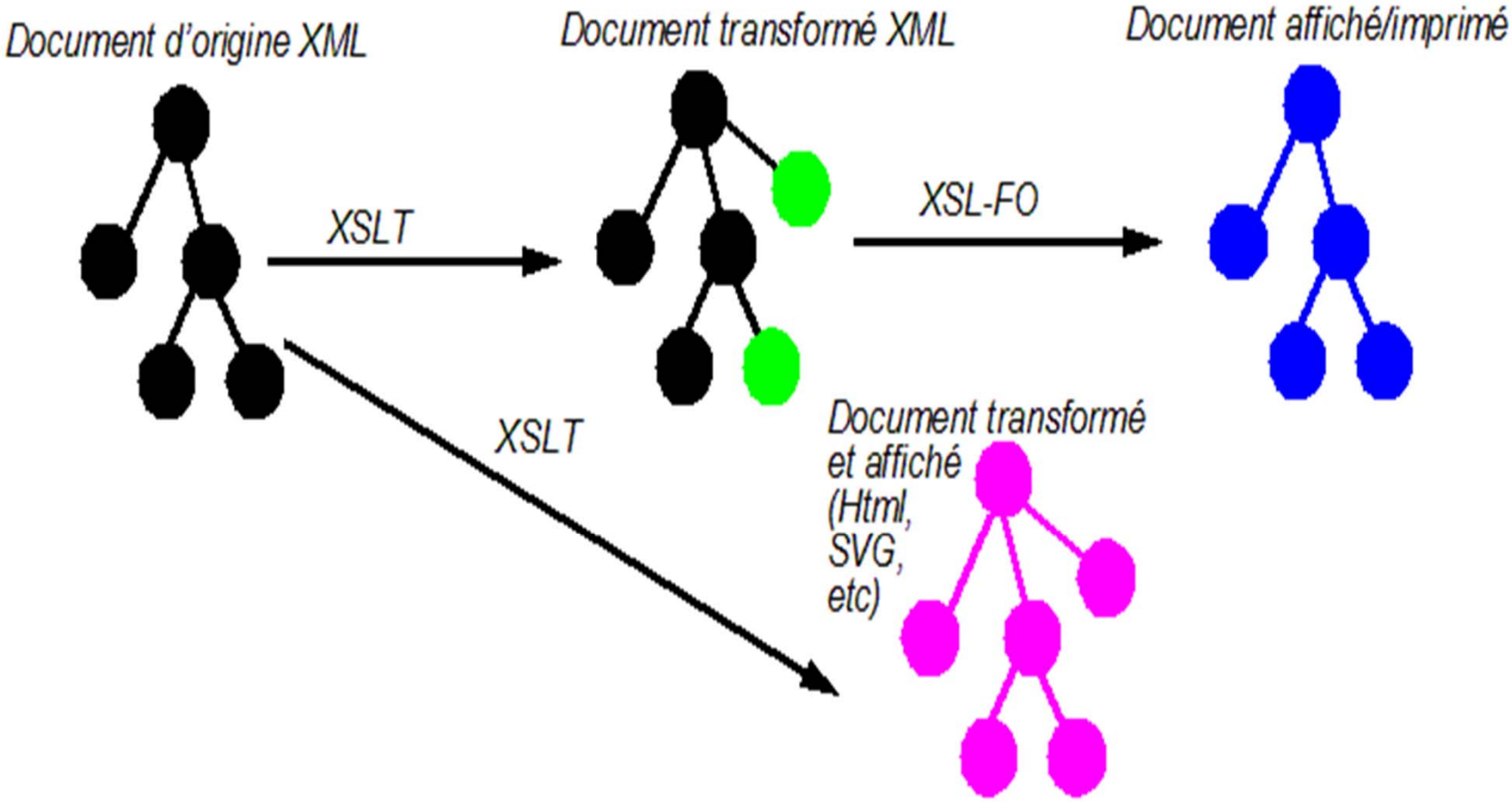
XSL : e**X**tensible **S**tylesheet **L**anguage

Le processus de transformation appelé XSLT se définit grâce au langage XSL qui est lui-même un langage XML



XSL :

- XSLT : langage de transformation
- Xpath : langage de navigation
- XSL-FO : Vocabulaire de mis en forme de documents (audio, ecran, papier)



Des URLs à connaître

Documentation XSLT 2.0 : http://www.w3.org/TR/xslt20/

Documentation XPath : http://www.w3.org/TR/xpath20/

Documentation des fonctions : http://www.w3.org/TR/xpath-functions/

Un moteur XSLT : http://saxon.sourceforge.net

XSL : les paramètres et les variables

- On peut définir des paramètres avec l'instruction
  - <xsl:param name="nomParam">…</xsl:param>

- On peut définir des variables locales à un noeud avec l'instruction
  - <xsl:variable name="nomVar">…</xsl:variable>

- On peut récupérer le contenu d'une variable ou d'un paramètre avec l'expression $nomElement où nomElement est le libéllé de la variable ou du paramètre.

- En général, on définit ces éléments comme globaux au début du source XSL

XSL : Patron de nœud

- La règle xsl:template permet de définir un ensemble de règles qui pourront être exécutées par le processeur.
- On distingue deux sortes de template.
  - les templates qui font référence à des nœuds et qui sont exécutés par le processeur quand il parcourt ces nœuds ou quand ils sont invoqués par la règle xsl:apply-templates.

    <xsl:template match="NomDuNoeud" mode="???">...</xsl:template>
  - les templates qui sont des fonctions. Ils sont appelés explicitement à l'aide de la règle <xsl:call-template et on peut leur passer des paramètres.

    <xsl:template name="NomFonction">...</xsl:template>

    On utilise la balise <param name="...." pour la déclaration de paramètres.

    On utilise la balise <with-param name="..." pour passer des parmètres.

Les règles avec la clause select

La clause select permet de sélectionner

- une partie de l'arbre XML
- le résultat du calcul d'une expression
- la valeur d'un paramètre, d'une variable ou d'un attribut

Elle s'applique à l'une des règles suivantes :

- <xsl:value-of select= " ??? " / >
- <xsl:apply-templates select="???" mode="???"/>
- <xsl:for-each select="???">...</xsl:for-each>
- <xsl:copy-of select="???"/>

Une partie d'un arbre XML est un sous arbre qui est définit avec l'opérateur / et les prédicats définis entre crochets pour filtrer des nœuds.

Exemple select="n1[@att = '1']/n2[$var = '2']/n3"

XSL : Les règles conditionnelles

- On peut définir des règles conditionnelles
  - <xsl:if test="???">...</xsl:if>
  - <xsl:choose>
    <xsl:when test="???">...<xsl:when>
    <xsl:when test="???">...<xsl:when>
    <xsl:otherwise>...</xsl:otherwise>
    </xsl:choose>

- Pour définir la condition, on utilise
  - les opérateurs and, or not(...), < ≤ > ≥
  - les fonctions du nœud courant (contexte) last(), position()
  - toutes les fonctions qui sont proposées dans Xpath

CDATA

- Dans le langage XLS, certains symboles ont un sens particulier comme &, <, etc. On parle alors de métacaractères.
- Si on désire les déspécialiser pour les écrire dans la sortie standard, dans une variable ou dans un paramètre, il faut utiliser l'instruction particulière qui suit :
  <![CDATA[...]]>.
  La zone ... qui est entre crochets peut contenir un texte quelconque y compris les caractères &, <, ... qui sont déspécialisés.
  Exemple : <![CDATA[if(x < 4)]]>

Xpath : Navigation

XPath considère 7 types de nœuds

1. racine
2. éléments
3. nœuds texte
4. attributs
5. espace de noms
6. instructions de traitement
7. commentaires

- **name** : élément **name**
- ***** : n'importe quel élément
- **namespace:name :** élément **name dans l'espace des noms namespace**
- **namespace:*** : n'importe quel élément dans l'espace des noms **namespace**
- **comment()** : noeud commentaire
- **text()** : noeud texte
- **processing-instruction() :** instruction de traitement
- **processing-instruction('target') :** instruction de traitement pour l'application **target** indiquée
- **node() :** n'importe quel type noeud où qu'il soit

XPath : fonctions utilisées dans les prédicats

- **sur les noeuds**
  - retourne un nombre : last(), position(), count(node-set), id(object)
  - retourne une chaîne : localname(node-set?), namespace-uri(node-set?),name(node-set?)
- **sur les chaînes de caractères (texte)**
  - retourne une chaîne : string(object?), concat(string, string, string*),substring-before(string, string), substring-after(string, string), translate (string, string, string)
  - retourne un booléen : contains(string, string), starts-with(string, string)
  - retourne un nombre : string-length(string)
- **booléen** retourne un booléen **:** boolean(object), true(), false(), lang(string)
- **numérique** retourne un nombre : number(object?), sum(node-set), floor(number), ceiling(number), round(number)

Syntaxe : entre crochet *après le test du noeud*

Exemple : **paragraph[last()]**

Xpath : Les axes

XPath propose les axes suivants :

•ancestor	Ancestors of the context node
•ancestor-or-self	Ancestors, including the context node
•attribute	Attributes of the context node (abbreviated "@")
•child	Children of the context node (the default axis)
•descendant	Descendants of the context node
•descendant-or-self	Descendants, including the context node (abbreviated "//")
•following	Elements which occur after the context node, in document order
•following-sibling	Elements which occur after the context node, in document order and have the same parent as the context node
•preceding	Elements which occur before the context node, in document order (returned in reverse-document order)
•preceding-sibling	Elements which occur before the context node, in document order (returned in reverse-document order) and have the same parent as the context node
•namespace	The namespace nodes of the context node
•parent	The parent of the context node (abbreviated "..")
•self	The context node (abbreviated ".")

Xpath : Exemple de prédicats

- **nodetest[1]** le premier nœud
- **nodetest[position()=last()]** le dernier nœud
- **nodetest[position() mod 2 = 0]** les nœuds pairs
- **element[@id="foo"]** élément(s) dont l'attribut id a la valeur "foo "
- **element[not(@id)]** les éléments qui n'ont pas d'attribut id
- **author[firstname="Norman"]** les éléments author qui ont des fils firstname dont le contenu est "Norman"
- **author[normalize-space(firstname)="Norman"]** les éléments author qui ont des fils firstname dont le contenu est "Norman" (sans tenir compte des espaces)

XPath : Exemple - Document XML

```
<...> -----> preceding
<...> -----> preceding
   <chapter> -----> ancestor
        <section> -----> ancestor
             <paragraph> ------------------> self
                  <figure>...</figure> -----> descendant
             </paragraph>
             <paragraph> -----> following-sibling
                 ...
              </paragraph>
        </section>
        <section> -----> following
            ...
        </section>
    </chapter> -----> following
<...> -----> following
```

XPath : Les axes *suivant* et *précédent*

```
<<xsl:for-each select="liste-de-termes/terme[not(@rubrique = preceding-
   sibling::terme/@rubrique)] ">
   <xsl:sort select="@rubrique"/>
   <xsl:variable name="rubrique"><xsl:value-of select="@rubrique"/></xsl:variable>
   <xsl:if test="@rubrique">
   <hr/>
   <b><xsl:value-of select="@rubrique"/></b>
   <hr/>
   </xsl:if>
   <xsl:for-each
         select="//liste-de-termes/terme[@type ='concept' and (@rubrique = $rubrique)]">
               <xsl:sort select="nom-complet" />
         <a href="#concept{@id}"
               class="NonSouligne1" style="padding-left:5px;">
               <xsl:value-of select="nom-complet" />
         </a>
         <hr />
   </xsl:for-each>
</xsl:for-each>
```

XSL : Utilisation des fonctions et opérateurs

- XSL propose une grande variété de fonctions et les opérateurs classiques.

- XSL n'est pas un langage impératif. L'appel d'une fonction ou d'un opérateur ne se fait que dans une clause select ou une clause de test.

```
<!-- On fait la somme des variables a et b -->
<xsl:value-of select="$a + $b"/>

<!-- On test si le nœud courant (contexte) est le
   dernier -->
<xsl:if test="position() = last()">
```

XSL : Les fonctions de chaînes

- La fonction concat permet de concaténer des chaînes

  Le nombre de chaînes à concaténer est variable.

  La fonction renvoie une nouvelle chaîne qui est donc la concaténation des chaînes passées en paramètres.

  ```
  <!-- On construit un nom de fichier avec l'extension .gif -->
  <xsl value-of select="concat($nom,'.gif')"/>
  ```

- La fonction substring permet d'extraire une sous chaîne dans une chaîne. Elle a trois paramètres : la chaîne, la position de début et le nombre de caractères à extraire.

  ```
  <!-- Dans un numéro de fixe passé en attribut  on extrait les chiffres pour la région -->
  <xsl:value-of select="substring(@noTel,1,2)"/>
  ```

- Si on omet le 3ème paramètre dans un appel de substring, alors la fonction renvoie toute la chaîne à partir de la position de début indiqué par le deuxième paramètre.

XSL : Les fonctions de chaînes

- La fonction substring-before(txt,token) permet d'extraire de la chaîne txt, la sous chaîne située avant la chaîne token.
- La fonction substring-after(txt,token) permet d'extraire de la chaîne txt, la sous chaîne située après la chaîne token.
- La fonction string-length(txt) renvoie la longueur de la chaîne txt.
- La fonction translate(txt,avant,apres) renvoie une copie de la chaîne txt dans laquelle tous les caractères avant ont été remplacés par les caractères apres. Si la chaîne apres est plus courte que la chaîne avant alors des caractères ne seront pas translatés.

XSL : La fonction document

- Cette fonction permet en plus du document XML d'entrée de charger en mémoire vive un autre document.
- Le document doit être un arbre XML valide.
- On extrait tout le fichier

```
<!-- On renvoie le contenu du fichier monFichier.xml -->
<xsl:value-of select="document('monFichier.xml')" />
```

- On extrait une partie du fichier

```
<!-- On renvoie une partie du contenu du fichier
    monFichier.xml -->
<xsl:value-of
    select="document('monFichier.xml')/pere/fils[@att =
    '???']" />
```

XSL : Formats text et number

- XSL est un langage non typé.
- Toute information est par défaut une chaîne de caractères.
- Cela pose un certain nombre de problèmes quand le contenu de l'information est du numérique.
- Dans la clause xsl:sort il faut préciser si ce que l'on veut trier est du texte ou du numérique avec l'attribut data-type.

  Si en entrée, on a la série 5, 13 et 12 :

  - avec <xsl:sort data-type="text"/> on aura en sortie 12, 13 et 5
  - avec <xsl:sort data-type="number"/> on aura en sortie 5, 12 et 13

XSL : Formats text et number

- Dans une clause test si les valeurs à tester sont des numériques il faut utiliser la fonction format-number.
- Cette fonction reçoit deux paramètres : la valeur et le format pour transformer cette valeur.
- Dans le format on utilise les caractères 0 et #. 0 signifie qu'un chiffre non significatif sera remplacé par 0. # signifie qu'un chiffre non significatif ne sera pas remplacé.

  format-number(53.51, "000.###") retourne 053.51

  format-number(53.51, "000.000") retourne 053.510

  format-number(53.51, "#.000") retourne 53.510
- <xsl:if test="format-number($prix,"000.00") < '012.25'">
- Par défaut la virgule est le séparateur de groupe et le point le séparateur de décimal
- format-number(5351,"#,###") retourne 5,351