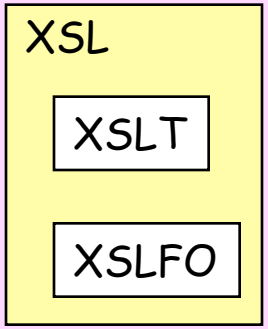
eXtensible Stylesheet Language

Philippe.Poulard@inria.fr

XSLT

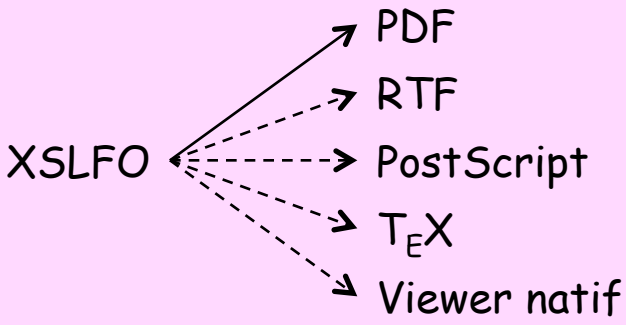
XSLFO

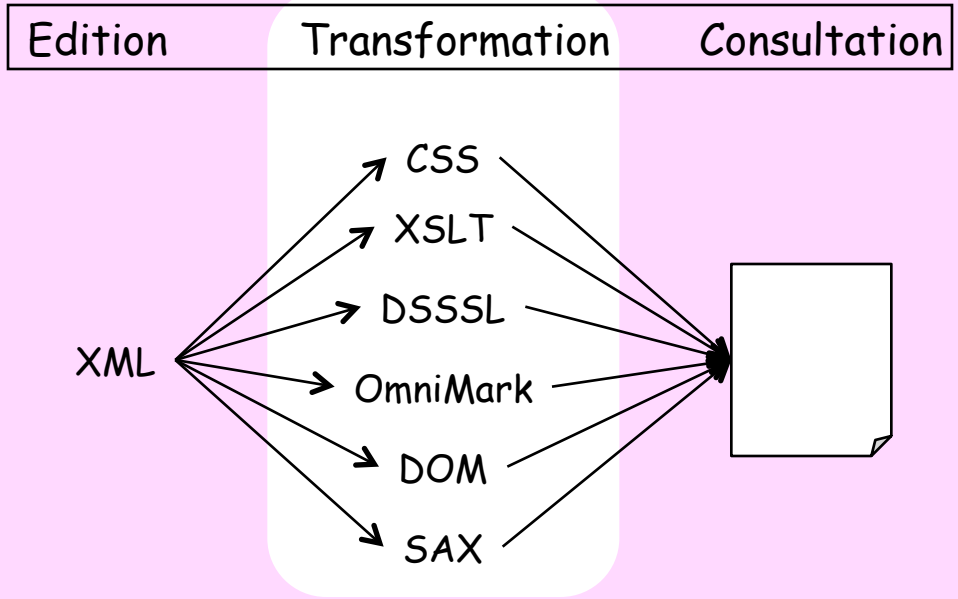
- blocs
- gabarits des pages
- flots de contenus



Transformation

Formattage





Alternatives à XSLT

Processeurs XSLT

Xalan, Saxon
MSXML2.4.x, xsltproc

Convertisseurs FO

FOP (Open Source : `http://xml.apache.org/fop`) XSLFO vers PDF
RenderX's XEP (Commercial : `http://www.renderx.com`) XSLFO vers PDF et PostScript
jfor (Open Source : `http://sourceforge.net/projects/jfor`) XSLFO vers RTF

http://www.w3.org/TR/xslt

XSL Transformation

XSLT utilise une grammaire XML ⟶ Un document XSLT **est un document XML**

XSLT est un langage déclaratif (peu procédural) basé sur des règles

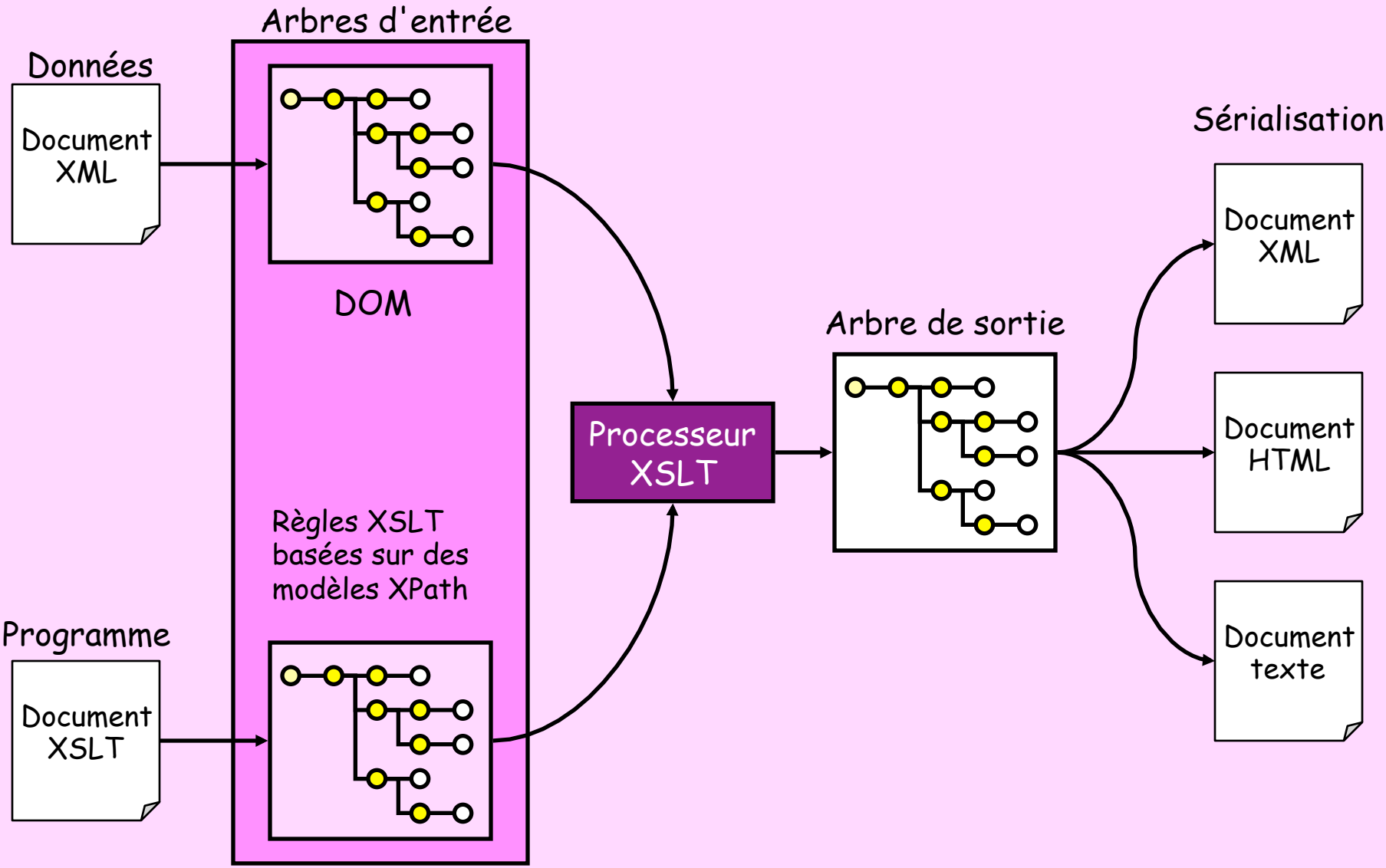
Langage de programmation :

- dédié au traitement des documents XML (transformations de structure)
- basé sur des définitions de règles à appliquer au document source
  Chaque règle produit un résultat

XSLT :

- permet de manipuler des données faiblement typées (booléens, chaînes, nombres, ensembles de nœuds, fragments de résultat)
- fournit un ensemble d'opérations de déclaration et de soumission de règles comme `<xsl:template>`, `<xsl:apply-templates>`, de tri `<xsl:sort>`, de formatage du résultat `<xsl:output>`…
- offre des instructions de contrôle du déroulement du programme (`<xsl:if>`, `<xsl:for-each>`, `<xsl:choose>`…)
- utilise intensivement XPath

Fonctionnement de XSLT



Certains processeurs peuvent traiter en entrée des arbres DOM ou des événements SAX

Certains processeurs peuvent directement délivrer un arbre DOM ou des événements SAX à une application

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
    <Titre>Cours XML</Titre>
    <Auteur>    <Nom>Poulard</Nom>
                <Prénom>Philippe</Prénom>
    </Auteur>
    <Description>Ce cours aborde les concepts de base mis en &#339;uvre dans XML.
    </Description>
</Cours>
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="Cours/Titre"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="Cours/Titre"/></h1>
        <xsl:apply-templates select="Cours/Auteur"/>
        <xsl:apply-templates select="Cours/Description"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="Auteur">
    <p align="right">par <xsl:value-of select="Nom"/>
     <xsl:value-of select="Prénom"/></p>
  </xsl:template>
  <xsl:template match="Description">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

dans le contexte de "/"

dans le contexte de "Auteur"


```
<html xmlns="http://www.w3.org
/1999/xhtml">
  <head>
    <title>Cours XML</title>
  </head>
  <body>
    <h1>Cours XML</h1>
    <p align="right">par
Poulard Philippe</p>
    <p>Ce cours aborde les
concepts de base mis en
&oelig;uvre dans XML. </p>
  </body>
</html>
```

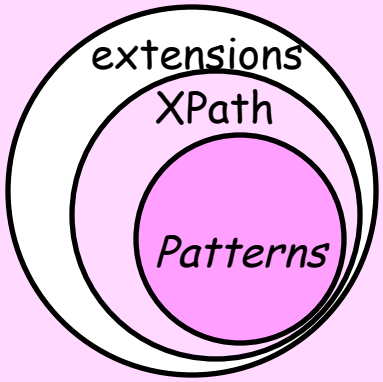
S'utilise dans les attributs : remplacer les < par <

```
<xsl:if test="position() < 5">
```
Non

```
<xsl:if test="position() &lt; 5">
```
Oui

pour les définitions de règles

- On utilise un sous-ensemble de XPath : les *patterns*
- On y ajoute quelques fonctionnalités propres à XSLT



Fonctions XSLT :

- `current()`
- `document()`
- `element-available()`
- `format-number()`
- `function-available()`
- `generate-id()`
- `key()`
- `system-property()`
- `unparsed-entity-uri()`

Contexte d'évaluation de XPath dans XSLT

XSLT défini le contexte d'évaluation de chaque expression XPath :

- le nœud contextuel
- la position du nœud dans le contexte
- la taille du contexte

- les variables
- les librairies de fonctions supplémentaires (`current()`, `document()`, ...)
- la résolution des préfixes

Fonctions définies par le processeur XSLT

```
//geom:arc[ @cos-theta > $custom:limit ]/@xlink:href
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:geom="http://www.foo.com/geometry"
    xmlns:math="http://www.foo.com/maths"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:custom="http://www.bar.org/my-ns"
>
  <xsl:output method="html"/>
  <xsl:variable name="custom:limit" select="number(0.5)"/>
  <xsl:variable name="math:pi" select="number(3.1415926535)"/>
  <xsl:variable name="date" select="'1969-06-10'"/>

      …/…
      <xsl:value-of
         select="//geom:arc[ @cos-theta &gt; $custom:limit ]/@xlink:href">
      …/…

</xsl:stylesheet>
```

Définitions utilisées dans la résolution des espaces de nommage

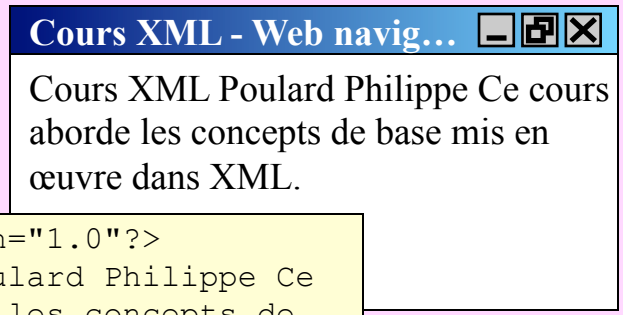
Définitions utilisées dans la résolution des variables

Règles par défaut XSLT


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours niveau="débutant">
    <!-- le meilleur cours sur XML ;o) -->
    <Titre>Cours XML</Titre>
    <Auteur>   <Nom>Poulard</Nom>
               <Prénom>Philippe</Prénom>
    </Auteur>
    <?acheter du pain pour ce soir?>
    <Description>Ce cours aborde les concepts de base mis en &#339;uvre dans XML.
    </Description>
</Cours>
```

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

Une feuille de style XSLT minimaliste permet de produire un résultat.
Ce résultat est obtenu grâce à des règles par défaut.




```
<?xml version="1.0"?>
Cours XML Poulard Philippe Ce
cours aborde les concepts de
base mis en œuvre dans XML.
```

Règles par défaut

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

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

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

```
<xsl:template match="prénom">
        …/…
</xsl:template>
<xsl:template match="nom">
        …/…
</xsl:template>
<xsl:template match="auteur/nom">
        …/…
</xsl:template>
<xsl:template match="auteur[@rôle='principal']/nom">
        …/…
</xsl:template>
<xsl:template match="rédacteur/nom">
        …/…
</xsl:template>
<xsl:template match="//chapitre//nom">
        …/…
</xsl:template>
```

règles candidates

règle retenue

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

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<livre><titre>La philosophie chinoise</titre>
    <auteur rôle="principal">
        <nom>Doe</nom><prénom>John</prénom>
    </auteur>
    <auteur>
        <nom>Doe</nom><prénom>Jane</prénom>
    </auteur>
    <chapitre><titre>Introduction</titre>
        <p>Le grand <nom>Lao Tseu</nom> a dit : <cite>il faut trouver la voie</cite>
…/…
```

Appel de règles selon le modèle délégatif

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

Sélection explicite de nœuds

```
<xsl:apply-templates"/>
```

Sélection implicite de tous les nœuds fils du contexte (peut contenir commentaires, textes, éléments…)

Chaque item de la collection sélectionnée déclenchera sa propre règle (potentiellement différente)

Le modèle **délégatif** est très évolutif, car ces règles peuvent continuer de fonctionner même en cas de modification de la structure

Si l'élément `title` ne peut contenir que du texte, ces règles produiront le même résultat

Modèle délégatif

```
<xsl:template match="title">
    <h1><xsl:apply-templates/></h1>
</xsl:template>
<xsl:template match="bold">
    <b><xsl:apply-templates/></b>
</xsl:template>
```

Modèle non délégatif

```
<xsl:template match="title">
    <h1><xsl:value-of select="."/></h1>
</xsl:template>
<xsl:template match="bold">
    <b><xsl:value-of select="."/></b>
</xsl:template>
```

Si l'élément `title` est modifié et peut aussi contenir l'élément `bold`, le modèle délégatif produira l'effet souhaité.
Si l'élément `bold` est modifié et peut aussi contenir l'élément `italic`, il suffira d'ajouter une règle pour ce nouvel élément

Déclenchement de règle

Appel de règle

```
<xsl:call-templates name="nom-de-règle"/>
```

Définition de règle

```
<xsl:template name="nom-de-règle">
        …/…
</xsl:template>
```

Le contexte ne change pas

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

```
<xsl:template match="pattern">
        …/…
</xsl:template>
```

```
<xsl:apply-templates select="expression"
          mode="catégorie"/>
```

```
<xsl:template match="pattern"
    mode="catégorie">
        …/…
</xsl:template>
```

Le **contexte** devient celui de l'expression

Expression : désigne un ensemble de nœuds avec XPath
Pattern : condition sur les nœuds pour déclencher la règle

Contexte :
Donné par la fonction `current()`

Pattern : "sous-ensemble" de XPath

- retourne toujours un ensemble de nœud (*node-set*)
- les étapes de localisation ne doivent concerner que les axes `child` ou `attribute` ou `descendent` en utilisant la notation `//`

Quand plusieurs règles sont candidates, un algorithme de sélection permet de choisir celle qui sera appliquée, selon :

- le mode
- la priorité
- la préséance

Priorité explicite :

```
<xsl:template match="pattern"
    priority="x">
        …/…
</xsl:template>
```

Priorité par défaut (selon le *pattern*)

Patterns à une seule étape de localisation :

`foo-élément`	0
`@bar-attribut`	0
`ns-blah:foo-élément`	0
`ns-blah:@bar-attribut`	0
`processing-instruction("foo")`	0
`ns-blah:*`	-0,25
`@ns-blah:*`	-0,25
`*`	-0,5
`@*`	-0,5
`text()`	-0,5
`node()`	-0,5
`comment()`	-0,5
`processing-instruction()`	-0,5

Autres patterns : 0,5

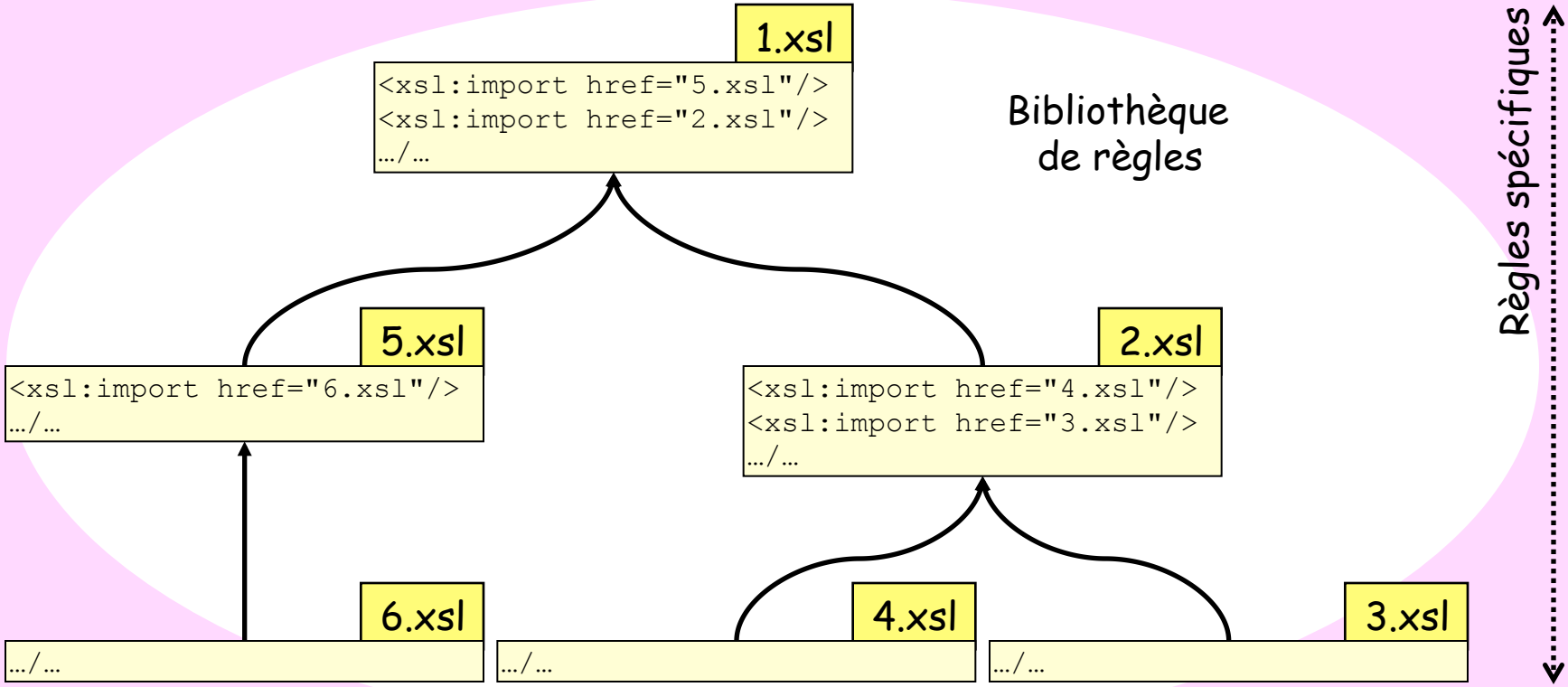
(plus une règle est spécifique, plus sa priorité est élevée)

Plus la valeur est grande, plus la priorité est élevée

Préséance : s'applique aux règles importées par `xsl:import`

Règle de préséance (comme avec les CSS et la directive `@import`) :

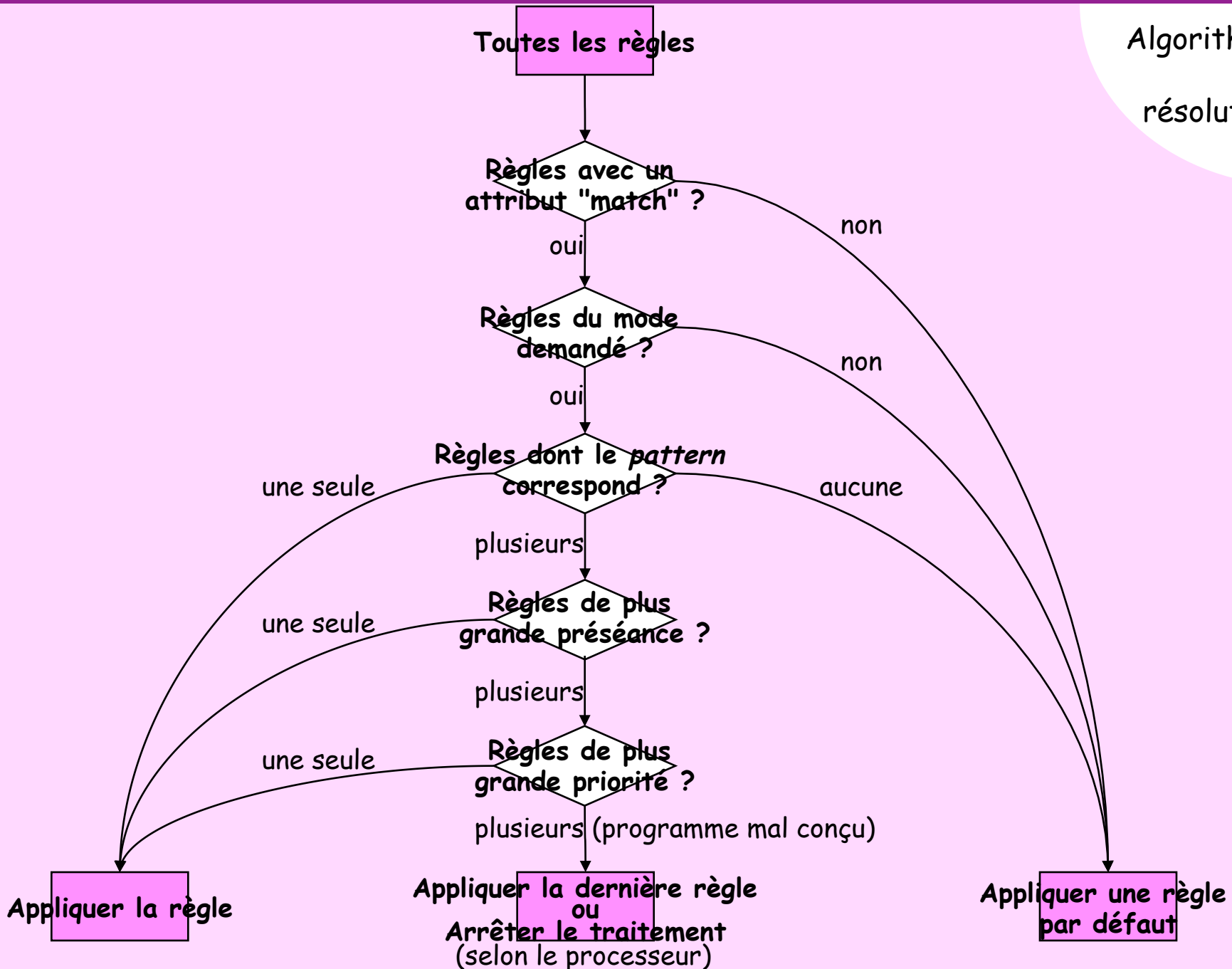
- la dernière règle importée à la primeur sur les autres
- les règles d'un document importateur prévalent sur celles d'un document importé (surcharge des règles)



Les règles surchargées peuvent être évoquées explicitement par : `<xsl:apply-imports/>`

Sélection de règle

Algorithme de résolution



Production de l'arbre de sortie

Production d'éléments

Peut se faire :

- avec des éléments littéraux
- avec des instructions `xsl:element`
- par copie avec les instructions `xsl:copy` et `xsl:copy-of`

```
<xsl:template match="bar">
    <foo>
        …/…
    </foo>
</xsl:template>
```

ou

```
<xsl:template match="bar">
    <xsl:element name="foo">
        …/…
    </xsl:element>
</xsl:template>
```

Arbre de sortie :

```
<foo>
    …/…
</foo>
```

Copie du nœud courant, et de sa valeur

```
<xsl:template match="bar">
    <xsl:copy>
        …/…
    </xsl:copy>
</xsl:template>
```

Ne copie pas les attributs
Copie les espaces de nommage

Copie d'un nœud en profondeur, avec tous les sous-noeuds, et leur valeur

```
<xsl:template match="bar">
    <xsl:copy-of select="."/>
</xsl:template>
```

Production de l'arbre de sortie

Calcul de valeurs

Avec `xsl:value-of`

```
<xsl:text>
<xsl:element>
<xsl:attribute>
<xsl:processing-instruction>
<xsl:comment>

<xsl:attribute-set>
<xsl:copy>
<xsl:copy-of>
<xsl:value-of>
<xsl:number>
```

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

<xsl:template match="livre">
    <tr>
        <td><xsl:value-of select="titre"/></td>
        <td><xsl:value-of select="auteur/nom"/></td>
        <td><xsl:value-of select="auteur/prénom"/></td>
    </tr>
</xsl:template>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<livres>
    <livre>
        <auteur>
            <nom>Victor</nom>
            <prénom>Hugo</prénom>
        </auteur>
        <titre>Les misérables</titre>
    </livre>
    <livre>
        <titre>Germinal</titre>
        <auteur>
            <nom>Emile</nom>
            <prénom>Zola</prénom>
        </auteur>
</livre>
</livres>
```

```
<table border="1">
    <tr>
        <td>Les misérables</td>
        <td>Victor</td>
        <td>Hugo</td>
    </tr>
    <tr>
        <td>Germinal</td>
        <td>Emile</td>
        <td>Zola</td>
    </tr>
</table>
```

Web navigator

Les misérables	Victor	Hugo
Germinal	Emile	Zola

A éviter : `<xsl:value-of disable-output-escaping="yes" select="string('<')"/>`

Production d'attributs

Peut se faire :

- avec des éléments littéraux
- avec des instructions `xsl:attribute`
- avec des jeux d'attributs prédéfinis

```
<xsl:template match="bar">
    <foo param="value">
        …/…
    </foo>
</xsl:template>
```

```
<xsl:template match="bar">
    <foo>
        <xsl:attribute name="param">value</xsl:attribute>
        …/…
    </foo>
</xsl:template>
```

```
<xsl:template match="bar">
    <xsl:element name="foo" use-attribute-set="foo-attributes">
        …/…
    </xsl:element>
</xsl:template>
```

Contenus d'attributs calculés

```
<image name="photo.jpg">
```

```
<img>
  <xsl:attribute name="src">
    <xsl:text>img/</xsl:text>
    <xsl:value-of select="image/@name" />
  </xsl:attribute>
</img>
```

```
<img src="img/photo.jpg"/>
```

Attribute Value Template

Forme abrégée

```
<img src="img/{image/@name}"/>
```

```
<xsl:text>
<xsl:element>
<xsl:attribute>
<xsl:processing-instruction>
<xsl:comment>

<xsl:attribute-set>
<xsl:copy>
<xsl:copy-of>
<xsl:value-of>
<xsl:number>
```

Production d'attributs

Ensembles d'attributs

Avec `xsl:attribute-set`

```
<xsl:template match="chapter/title">
    <fo:block xsl:use-attribute-set="title-style">
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
<xsl:template match="section/title">
    <fo:block xsl:use-attribute-set="title-style">
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
<xsl:attribute-set name="title-style">
    <xsl:attribute name="font-size">12pt</xsl:attribute>
    <xsl:attribute name="font-weight">bold</xsl:attribute>
    <xsl:attribute name="text-align">center</xsl:attribute>
</xsl:attribute-set>
```

```
<fo:block font-size="12pt"
          font-weight="bold"
          text-align="center">
    …/…
</fo:block>
```

Il est possible d'utiliser à la fois sur le même élément :

- des définitions d'ensembles d'attributs
- et des définitions d'attribut uniques

Production de texte

Avec `xsl:text`

Permet de contrôler avec précision les contenus des éléments de sortie

```
<xsl:template match="/">↲
  → (c) Philippe Poulard↲
</xsl:template>
```

⇒

```
↲
  → (c) Philippe Poulard↲
```

```
<xsl:template match="/">↲
  → <xsl:text>(c) Philippe Poulard</xsl:text>↲
</xsl:template>
```

⇒

```
(c) Philippe Poulard
```

Les nœuds blancs qui ne sont pas inclus dans l'élément `xsl:text` ne produisent pas de nœud texte dans l'arbre de sortie

Lorsqu'on produit une sortie HTML, on ne se soucie pas des nœuds blancs puisque HTML les réduit à un seul espace. Sauf pour les éléments comme `pre`.

A éviter :

```
<xsl:template match="/">
    <xsl:text disable-output-escaping="yes">&lt;malformé&gt;</xsl:text>
</xsl:template>
```

⇒

```
<malformé>
```

Production de commentaire

Avec `xsl:comment`

```
<xsl:template match="foo">
    <!-- début du commentaire -->
    <xsl:comment> commentaire produit en sortie </xsl:comment>
    <!-- fin du commentaire -->
</xsl:template>
```

```
<!-- commentaire produit en sortie -->
```

(sans commentaires)

Production d'instructions de traitement

Avec xsl:processing-instruction

```
<xsl:template match="/">
    <xsl:processing-instruction name="xml-stylesheet">
        <xsl:text>type="text/xsl" href="style.xsl"</xsl:text>
    </xsl:processing-instruction>
    <xsl:apply-templates />
</xsl:template>
```

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

Exécution conditionnelle

Avec `xsl:if`


```
<?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="/">
        <html>
            <body>
                <table border="1">
                    <xsl:apply-templates select="équipes/équipe"/>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="équipe">
        <tr>
            <xsl:if test="position() mod 2 = 0">
                <xsl:attribute name="bgcolor">yellow</xsl:attribute>
            </xsl:if>
            <td><xsl:apply-templates/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

Que se passe-t-il si on remplace par `<xsl:apply-templates/>` ?

```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes>
    <équipe cn="fr">OM</équipe>
    <équipe cn="it">Milan AC</équipe>
    <équipe cn="fr">PSG</équipe>
    <équipe cn="en">Arsenal</équipe>
    <équipe cn="it">Juventus</équipe>
</équipes>
```

Alternative

Avec `xsl:choose`

```
<xsl:if>
<xsl:choose>
<xsl:for-each>

<xsl:apply-templates>
<xsl:call-template>
<xsl:apply-imports>
<xsl:message>
<xsl:fallback>
```

```
<?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="/">
        <html>
            <body>
                <table border="1">
                    <xsl:apply-templates/>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="équipe">
        <tr>
            <td>
                <xsl:apply-templates/>
                (<xsl:choose>
                    <xsl:when test="@cn='fr'">France</xsl:when>
                    <xsl:when test="@cn='en'">Angleterre</xsl:when>
                    <xsl:when test="@cn='it'">Italie</xsl:when>
                    <xsl:otherwise>???</xsl:otherwise>
                </xsl:choose>)
            </td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes>
    <équipe cn="fr">OM</équipe>
    <équipe cn="it">Milan AC</équipe>
    <équipe cn="fr">PSG</équipe>
    <équipe cn="en">Arsenal</équipe>
    <équipe cn="it">Juventus</équipe>
</équipes>
```

Boucle

Avec `xsl:for-each`

```
<?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="/">
        <html>
            <body>
                <table border="1">
                    <xsl:for-each select="équipes/équipe">
                        <tr>
                            <td>
                                <xsl:apply-templates/>
                                (<xsl:choose>
                                    <xsl:when test="@cn='fr'">France</xsl:when>
                                    <xsl:when test="@cn='en'">Angleterre</xsl:when>
                                    <xsl:when test="@cn='it'">Italie</xsl:when>
                                    <xsl:otherwise>???</xsl:otherwise>
                                </xsl:choose>)
                            </td>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Instructions de contrôle du programme

Tri

xsl:apply-templates
xsl:sort

ou

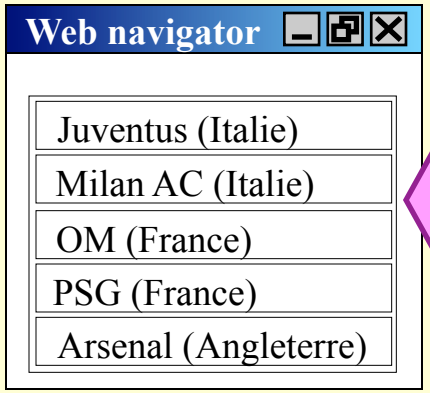
xsl:for-each
xsl:sort

```
<xsl:apply-templates select="équipes/équipe">
  <xsl:sort select="@cn" order="descending"/>
  <xsl:sort select="."/>
</xsl:apply-templates>
```


<xsl:if>
<xsl:choose>
<xsl:for-each>

<xsl:apply-templates>
<xsl:call-template>
<xsl:apply-imports>
<xsl:message>
<xsl:fallback>


```
<?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="/">
        <html>
            <body>
                <table border="1">
                    <xsl:for-each select="équipes/équipe">
                        <xsl:sort select="@cn" order="descending"/>
                        <xsl:sort select="."/>
                        <tr>
                            <td>
                                <xsl:apply-templates/>
                                (<xsl:choose>
                                    <xsl:when test="@cn='fr'">France</xsl:when>
                                    <xsl:when test="@cn='en'">Angleterre</xsl:when>
                                    <xsl:when test="@cn='it'">Italie</xsl:when>
                                    <xsl:otherwise>???</xsl:otherwise>
                                </xsl:choose>)
                            </td>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

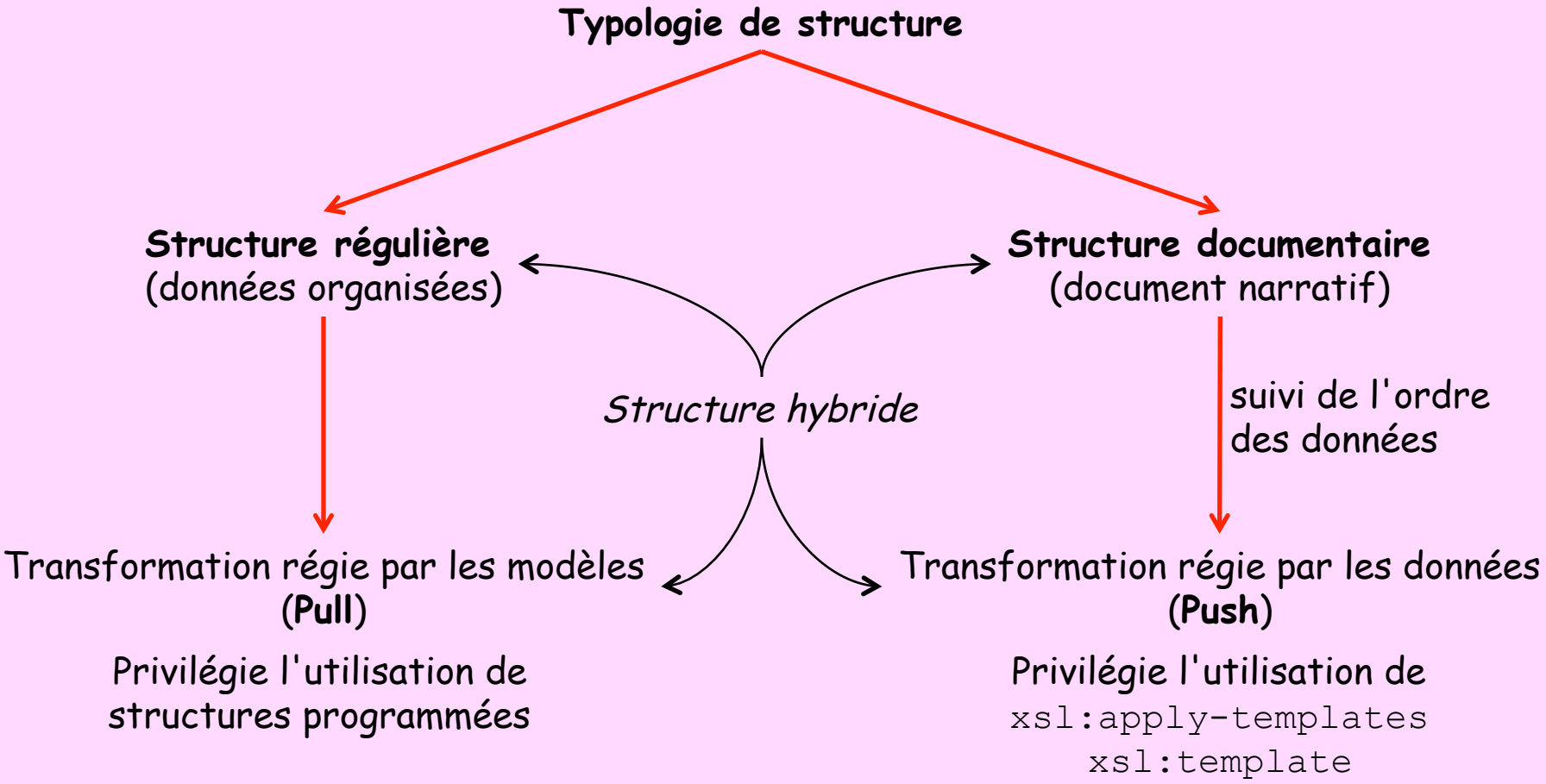



```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes>
    <équipe cn="fr">OM</équipe>
    <équipe cn="it">Milan AC</équipe>
    <équipe cn="fr">PSG</équipe>
    <équipe cn="en">Arsenal</équipe>
    <équipe cn="it">Juventus</équipe>
</équipes>
```

Typologie de structures

```
<commande id="kh34jl">
    <article id="4l34m" prix="120" qté="5"/>
    <article id="df40t" prix="210" qté="1"/>
    <article id="lnk23" prix="330" qté="4"/>
</commande>
```

```
<vers>Maître <b>corbeau</b> sur un
arbre perché tenait dans son bec un
fromage</vers>
<vers>Maître <i>renard</i> par l'odeur
alléché …/…</vers>
```



La plupart des documents sont des structures hybrides. Les structures purement documentaires ou régulières se trouvent dans des cas particuliers

Transformations régies par les données

Structures

- contenant des ensembles d'éléments hétérogènes
- avec des éléments optionnels
- récursives
- dont l'ordre est important

Exemple : que se passe-t-il si on triait un poème dans l'ordre alphabétique de ses vers ?

A l'opposé, une structure régulière, telle qu'une commande, peut être :

- non triée,
- triée par date
- triée par nom d'item
- triée par prix
- …

Procédures

xsl:call-template
xsl:with-param
xsl:param

```
<?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="/">
    <html>
      <body>
        <table border="1">
          <xsl:for-each select="équipes/équipe">
            <tr>
              <td>
                <xsl:apply-templates/>
                (<xsl:call-template name="label-nation">
                   <xsl:with-param name="code-nation" select="@cn"/>
                 </xsl:call-template>)
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template name="label-nation">
    <xsl:param name="code-nation"/>
    <xsl:choose>
      <xsl:when test="$code-nation='fr'">France</xsl:when>
      <xsl:when test="$code-nation='en'">Angleterre</xsl:when>
      <xsl:when test="$code-nation='it'">Italie</xsl:when>
      <xsl:otherwise>???</xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

<xsl:if>
<xsl:choose>
<xsl:for-each>

<xsl:apply-templates>
<xsl:call-template>
<xsl:apply-imports>
<xsl:message>
<xsl:fallback>

Les *template* nommés fournissent une structure de programmation équivalente aux **procédures** des langages procéduraux. Ils acceptent le passage de paramètres

Passage de paramètres au programme

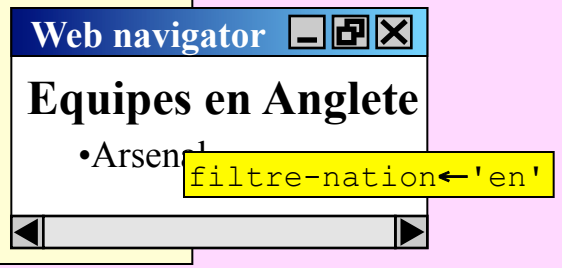
Paramètres xsl:param

Permet d'agir de l'extérieur

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="filtre-nation" select="'fr'"/>
  <xsl:template match="/">
    <html>
      <body>
        <h1>Equipes en
          <xsl:call-template name="label-nation">
            <xsl:with-param name="code-nation" select="$filtre-nation"/>
          </xsl:call-template >
        </h1>
        <ul>
          <xsl:for-each select="équipes/équipe[@cn=$filtre-nation]">
            <li><xsl:apply-templates/></li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
  <xsl:template name="label-nation">
    <xsl:param name="code-nation"/>
    <xsl:choose>
      <xsl:when test="$code-nation='fr'">France</xsl:when>
      <xsl:when test="$code-nation='en'">Angleterre</xsl:when>
      <xsl:when test="$code-nation='it'">Italie</xsl:when>
      <xsl:otherwise>???</xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

Valeur modifiable par programme externe

Le mode d'intéraction dépend du processeur et du langage



Définition de variables

xsl:variable

On peut assigner à une variable :

- le résultat d'une expression XPath (*boolean*, *number*, *string*, *node-set*)
- du contenu (*result-tree-fragment*)

(Représentation sérialisée d'un résultat)

Exemples

```
<xsl:variable name="organisme" select="'FIFA'">
```
string

```
<xsl:variable name="âge-du-capitaine" select="number( 22 )">
```
number

```
<xsl:variable name="gagné" select="true()">
```
boolean

```
<xsl:variable name="équipes-françaises" select="//équipes/équipe[@cn='fr']">
```
node-set

```
<xsl:variable name="libellé-nation">
    <xsl:call-template name="label-nation">
        <xsl:with-param name="code-nation" select="@cn"/>
    </xsl:call-template>
</xsl:variable>
```
type dépendant du résultat de l'exécution de la règle nommée

```
<xsl:variable name="lieu">
    Stade de France
    <nation>fr</nation>
</xsl:variable>
```
RTF

Utilisation de variables

S'utilise dans les expressions XPath

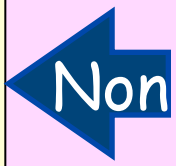
```
<xsl:value-of select="$organisme"/>
<xsl:for-each select="joueur[@age &lt; $âge-du-capitaine]">
<xsl:apply-templates select="$équipes-françaises/joueurs/joueur"/>
<xsl:value-of select="$libellé-nation"/>
<xsl:value-of select="$lieu"/>
```

Portée :

- variables globales : définies par des éléments `xsl:variable` de premier niveau, fils de `xsl:stylesheet`.et
  ⇒ Visibles dans tout le programme
- variables locales : définies dans les corps de règle.
  ⇒ Visibles dans tous les nœuds `following-sibling` et `descendant` du nœud `xsl:variable`

On ne peut pas définir 2 variables ayant le même nom si leurs portées se chevauchent (*shadowing*) à moins que l'une d'elles soit une variable globale ou que les 2 soient globales sans la même **préséance**.

```
<xsl:variable name="i" select="number(1)"/>
<xsl:for-each select="équipe">
    <xsl:variable name="i" select="number($i + 1)"/>
</xsl:for-each>
<p>Il y a <xsl:value-of select="$i"/> équipes.</p>
```



Provoque une erreur

Result tree fragment

```
<xsl:variable name="lieu">
    Stade de France
    <nation>fr</nation>
</xsl:variable>
```

Ne peut pas s'utiliser comme un *node-set*

```
<xsl:apply-templates select="$lieu/nation"/>
```

Non

Provoque une erreur

Les processeurs XSLT utilisent souvent une extension non prévue dans le standard qui permette de convertir un *result-tree-fragment* en *node-set*.

```
<xsl:apply-templates select="node-set($lieu)/nation"/>
```

Son utilisation dépend du processeur

Xalan-Java

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xalan="http://xml.apache.org">
…/…
        <xsl:apply-templates select="xalan:nodeset($lieu)/nation"/>
…/…
```

MSXML2.4.x

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt">
…/…
        <xsl:apply-templates select="msxml:node-set($lieu)/nation"/>
…/…
```

Message

`xsl:message` permet de renvoyer un message à la console, ou dans les logs, ou dans une boîte de dialogue…

```
<xsl:choose>
    <xsl:when test="$code-nation='fr'">France</xsl:when>
    <xsl:when test="$code-nation='en'">Angleterre</xsl:when>
    <xsl:when test="$code-nation='it'">Italie</xsl:when>
    <xsl:otherwise>
        <xsl:message terminate="yes">
            Code nation inconnu
        </xsl:message>
    </xsl:otherwise>
</xsl:choose>
```

Erreurs

`xs:fallback` s'utilise dans un contexte ou la cible de déploiement des programmes XSLT n'est pas connue du développeur

```
<?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="/">
    <xsl:for-each select="//équipe/@cn">
      <xsl:if test="not(../following-sibling::*[@cn=current()])">
        <xsl:result-document format="html" href="equipe-{.}.html">
          <xsl:fallback>
            <xsl:message terminate="yes">
              Impossible de créer un fichier.
              L'instruction &lt;xsl:result-document&gt; n'est pas implémentée
            </xsl:message>
          </xsl:fallback>
          <html>
            <body>
…/…
```

Indexation

L'indexation avec `xsl:key` permet d'augmenter les performances de certains traitements

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="code-nation" select="'fr'"/>
  <xsl:key name="équipes-par-nation" match="équipe" use="@cn"/>
  <xsl:template match="équipes">
    <html>
      <body>
        <ul>
          <xsl:for-each select="key('équipes-par-nation',$code-nation)">
            <li><xsl:value-of select="équipe"/></li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

On peut retrouver les équipes d'une nation, mais on peut difficilement obtenir la liste des nations → méthode de Muench

```
<xsl:for-each select="équipe/@cn[not(. = ../preceding-sibling::équipe/@cn)]">
  <li><xsl:value-of select="."/></li>
</xsl:for-each>
```

Peu performant : oblige le processeur à "regarder en arrière" pour chaque item
Peut-être préjudiciable pour les longues listes.

Génération d'ID

Fonction `generate-id(node)`

Retourne un identifiant unique (dans le document) pour le nœud passé en argument, ou le premier nœud du *node-set* passé en argument, ou du nœud courant s'il n'y a pas d'argument

---

La méthode de Muench (*Muenchian method, by Steve Muench*) :

- permet de grouper des données
- permet de sélectionner des valeurs distinctes

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:key name="group" match="équipe" use="@cn"/>
  <xsl:template match="équipes">
    <html>
      <body>
        <ul>
          <xsl:for-each select="équipe[generate-id()=generate-id(key('groupe',@cn)[1])]">
            <li><xsl:value-of select="@cn"/></li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

ou

```
<xsl:for-each select="équipe[count(.|key('groupe',@cn)[1])=1]">
```

Dans un *node-set*, les nœuds sont uniques

Eléments XSLT de premier niveau

Sous la racine `xsl:stylesheet`, on trouve :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
          <xsl:namespace-alias/>
          <xsl:output/>
          <xsl:preserve-space/>
          <xsl:strip-space/>

          <xsl:attribute-set/>
          <xsl:key/>
          <xsl:decimal-format/>
          <xsl:param/>
          <xsl:variable/>

          <xsl:import/>
          <xsl:include/>

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

directives de traitement du processeur

définitions d'ensembles de données

directives d'importation

définitions de règles

`xsl:transform` et `xsl:stylesheet` sont synonymes

Contrôle des éléments en entrée :

```
<équipe xml:space="preserve">
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
          <xsl:preserve-space elements="équipe joueur"/>
          <xsl:strip-space elements="équipes joueurs"/>
</xsl:stylesheet>
```

Contrôle des éléments en sortie :

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

```
<xsl:text disable-output-escaping="yes"></xsl:text>
```

```
<xsl:text>&#x9;</xsl:text>
```

```
<xsl:text>&#xA;</xsl:text>
```

```
<xsl:text>&#x20;</xsl:text>
```

```
<xsl:text> </xsl:text>
```
espace insécable

Un nœud texte utilisé seul permet de s'assurer qu'aucun autre contenu ne sera produit :

```
<xsl:template match="foo">↲
  → <xsl:text>ici</xsl:text>↲
<xsl:template>
```

↓

production du texte "ici" sans ↲ → avant ni ↲ après

Normalisation des contenus d'éléments (les valeurs d'attributs sont toujours normalisées) : `normalize-space()`

Espaces de nommage dans XSLT

```
<?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="/">
        <html>
            <body>
                <table border="1">
                    <xsl:apply-templates select="équipes/équipe"/>
                </table>
            </body>
        </html>
    </xsl:template>
    …/…
</xsl:stylesheet>
```

Ne sélectionne rien

Quand on utilise un espace de nommage dans un document, les feuilles de style qui le traitent doivent aussi le déclarer…

```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes xmlns="http://www.fifa.fr/championnat">
    <équipe cn="fr">OM</équipe>
    <équipe cn="it">Milan AC</équipe>
    <équipe cn="fr">PSG</équipe>
    <équipe cn="en">Arsenal</équipe>
    <équipe cn="it">Juventus</équipe>
</équipes>
```

…en utilisant obligatoirement un préfixe (sinon, les données ne sont pas adressables par XPath)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:foot="http://www.fifa.fr/championnat">
    <xsl:template match="/">
        <html>
            <body>
                <table border="1">
                    <xsl:apply-templates select="foot:équipes/foot:équipe"/>
                </table>
            </body>
        </html>
    </xsl:template>
    …/…
</xsl:stylesheet>
```

Espaces de nommage dans XSLT

Tous les espaces de nommage déclarés sont reproduits en sortie, sauf :

- celui associé au processeur XSLT
- ceux exclus nominativement avec `exclude-result-prefixes`

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:foot="http://www.fifa.fr/championnat"
  exclude-result-prefixes="foot">
    <xsl:template match="/">
        <html>
            <body>
                <table border="1">
                    <xsl:apply-templates select="foot:équipes/foot:équipe"/>
                </table>
            </body>
        </html>
    </xsl:template>
    …/…
</xsl:stylesheet>
```

Exclusion de plusieurs déclarations d'espaces de nommage :

```
exclude-result-prefixes="foot joueurs matchs"
```

Exclusion de la déclaration d'espace de nommage par défaut :

```
exclude-result-prefixes="#default"
```

Les alias utilisés dans la feuille de style peuvent être renommés dans le document produit :

```
<xsl:namespace-alias
    stylesheet-prefix="pàv"
    result-prefix="planche-à-voile"/>
```

Formattage des nombres et numérotation

Formattage des nombres

`<xsl:decimal-format/>`

`format-number()`

La représentation des nombres est variable selon les pays

```
<xsl:decimal-format name="european" decimal-separator=',' grouping-separator='.'/>
<xsl:value-of select="format-number(24535.2, '###.###,00', 'european')"/>
```

⇩

`24.535,20`

Numérotation

```
<h2>Sommaire</h2>
<p>
    <xsl:for-each select="section">
        <xsl:number level="multiple" format="1." count="section"/>
        <a href="#{generate-id()}">
            <xsl:value-of select="title" />
        </a>
    <xsl:for-each>
</p>
```

`<xsl:number/>`

Permet de numéroter les tables des matières et indexes

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<content>
  <section><title>Objectifs</title>
    <p>…/…</p>
  </section>
  <section><title>Prérequis</title>
    <p>…/…</p>
  </section>
  <section><title>Manipulat
    <section><title>Démonta
      <p>…/…</p>
    </section>
    <section><title>Nettoya
      <p>…/…</p>
    </section>
  </section>
```

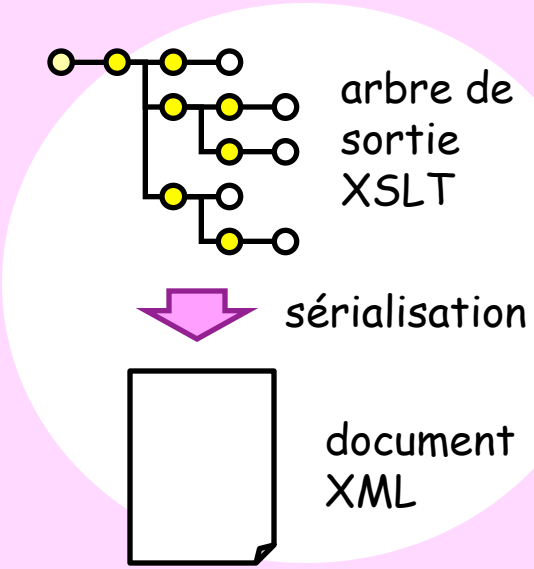
Web navig…

Sommaire

1 Objectifs
2 Prérequis
3 Manipulation
3.1 Démontage
3.2 Nettoyage

Contrôle de la sérialisation de l'arbre de sortie

```
<xsl:output
    method = "html"
    version = "4.0"
    encoding = "iso-8859-1"
    omit-xml-declaration = "yes"
    standalone = "yes"
    doctype-public = "-//W3C//DTD XHTML 1.0 Transitional//EN"
    doctype-system = "DTD/xhtml1-transitional.dtd"
    cdata-section-elements = "script style"
    indent = "yes"
    media-type = "text/html"
/>
```



`<xsl:output method="xml"/>`

`<xsl:output method="text"/>`

Il est recommandé d'utiliser `xsl:text` pour produire le résultat

`<xsl:output method="html"/>`

La mise au format HTML dépend des implémentations et des autres paramètres.
On peut obtenir la réécriture des caractères non ASCII par des appels d'entités (`&eacute;` au lieu de `é`)…

`<xsl:output method="maMethode"/>`

A programmer soit-même…

Exemple : `<xsl:output method="pdf"/>`

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns="http://www.w3.org/1999/xhtml">
    <xsl:template match="/">
        <html>
            <head>
                <title><xsl:value-of select="Cours/Titre"/></title>
            </head>
            <body>
                …/…
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

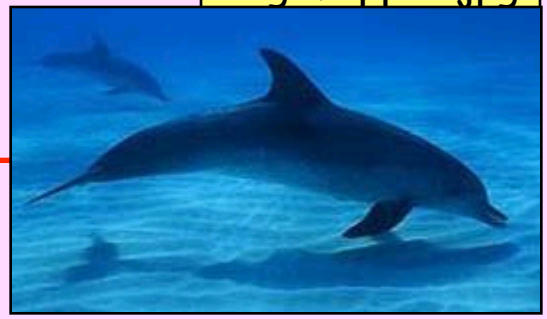
Syntaxe "simplifiée" de la feuille de style

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<html xsl:version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title><xsl:value-of select="Cours/Titre"/></title>
    </head>
    <body>
        …/…
    </body>
</html>
```

Les entités binaires

Fonction `unparsed-entity-uri(node)`

img/flipper.jpg



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE zoo [
    <!ATTLIST dauphin photo ENTITY>
    <!NOTATION jpg SYSTEM "images/jpeg">
    <!ENTITY flipper SYSTEM "img/flipper.jpg" NDATA jpg>
]>
<zoo>
    <aquarium>
        <dauphin id="jhgtr13" photo="flipper">
            <nom>Flipper</nom>
            …/…
</zoo>
```

```
<xsl:template match="dauphin">
    <h2><xsl:value-of select="nom" /></h2>
    <p>
        <xsl:if test="@photo">
            <img src="{unparsed-entity-uri(@photo)}"/>
        </xsl:if>
    </p>
</xsl:template>
```

```
<h2>Flipper</h2>
<p>
    <img src="img/flipper.jpg"/>
</p>
```

Web navigator

Flipper

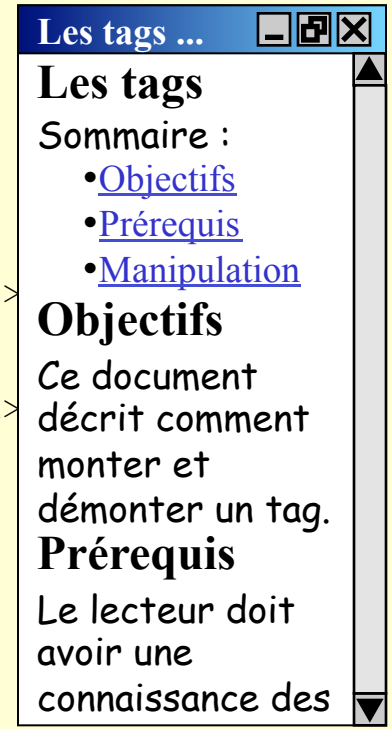
Fonction

`generate-id(node)`

Construction d'une table des matières

```
<?xml version="1.0" encoding="ISO
…/…
<content><title>Les tags</title>
  <section><title>Objectifs</titl
    <p>Ce document décrit comment
       monter et démonter un tag.
    </p>
  </section>
  <section><title>Prérequis</titl
    <p>Le lecteur doit avoir une
       connaissance des tags appr
    </p>
    <p>En particulier, il doit ma
       les tags bleus et les tags
       de compétition.
    </p>
  </section>
  <section><title>Manipulation</t
    <info>Penser à vérifier les t
      avant toute manipulation.
    </info>
    <step>Enlever l'eau sale du t
    <step>Dévisser le capot infér
    <step>Cliquer sur OK.</step>
  </section>
  …/…
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns="http://www.w3.org/TR/xhtml1/strict">
  <xsl:template match="content">
    <html>
      <head>
        <title><xsl:value-of select="title"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="title"/></h1>
        <h2>Sommaire</h2>
        <ul>
          <xsl:for-each select="//section">
            <li><a href="#{generate-id(title)}">
              <xsl:value-of select="title"/>
            </a></li>
          </xsl:for-each>
        </ul>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="/content/title"/>
  <xsl:template match="title">
    <h2><a name="{generate-id()}">
          <xsl:value-of select="."/></a>
  </xsl:template>
  <xsl:template match="section">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="p">
    <p><xsl:apply-templates/></p>
  </xsl:template>
  <xsl:template match="b">
```



Intégration de sources de données multiples

La fonction `document()` permet de réaliser des transformations simultanément sur plusieurs sources de données

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rapport>
  <analyse>
    <p>Cette année, la <b>croissance</b>
    est au rendez-vous,
    …/…
    </p>
  <analyse>
</rapport>
```

ventes.xml

```
<?xml version="1.0"
    encoding="ISO-8859-1" ?>
<ventes>
  <charcuterie>23</charcuterie>
  <fromages>45</fromages>
  <primeurs>78</primeurs>
</ventes>
```

```
<?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="/">
    <xsl:variable name="mes-ventes"
      select="document('ventes.xml')">
    <xsl:apply-templates select="$mes-ventes/ventes"/>
    <xsl:apply-templates select="rapport"/>
  </xsl:template>
</xsl:stylesheet>
```

Rapport - Web Explorer

- charcuterie : 15%
- fromages : 31%
- primeurs : 54%

Cette année, la **croissance** est au rendez-vous …

La fonction `document()` utilisée avec l'argument `''` (chaîne vide) permet d'obtenir la feuille de style elle-même, et donc d'y adresser des informations

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:date="*** Traitement des dates ***">
    <date:noms-mois>
        <date:mois court="jan">janvier</date:mois>
        <date:mois court="fév">février</date:mois>
        …/…
        <date:mois court="déc">décembre</date:mois>
    </date:noms-mois>

    <xsl:template name="date:nom-mois">
        <!--retourne le nom du mois en clair à partir du numéro de mois-->
        <xsl:param name="mois" select="0"/>
        <xsl:value-of select="document('')/*/date:noms-mois/date:mois[$mois]"/>
    </xsl:template>
</xsl:stylesheet>
```

```
<xsl:variable name="année" select="substring-before($date, '-')"/>
<xsl:variable name="mois"  select="substring-before(substring-after($date, '-'), '-')"/>
<xsl:variable name="jour"  select="substring-after(substring-after($date, '-'), '-')"/>
<xsl:variable name="nom-mois">
    <xsl:call-template name="date:nom-mois">
        <xsl:with-param name="mois" select="number($mois)"/>
    </xsl:call-template>
</xsl:variable>
<xsl:value-of select="$jour" />
<xsl:value-of select="$nom-mois" />
<xsl:value-of select="$année" />
```

Transformation à l'identique :

copie.xslt

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="@* | node()">
    <xsl:copy>
        <xsl:apply-templates select="@* | node()" />
    </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

Utile :

- pour changer l'encodage
- pour sérialiser un arbre DOM
- à l'import dans une feuille de style qui fait de l'adaptation de structure

Exemple : conversion des attributs en éléments

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:import href="copie.xslt"/>
<xsl:output method="xml" version="1.0" indent="yes" encoding="ISO-8859-1"/>
<xsl:template match="@*">
    <xsl:element name="{local-name(.)}" namespace="{namespace-uri(..)}">
        <xsl:value-of select="." />
    </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Une autre feuille de style de copie mais qui ne permet pas de faire de l'adaptation de structure :

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

Typage incontrôlable

```
<xsl:variable name="expressionXPath">/doc/section[3]/section[2]</xsl:variable>
…/…
<xsl:apply-templates select="$expressionXPath" />
```

Non

Cet exemple ne fonctionne pas
- la variable `expressionXPath` est une **chaîne de caractères**
- la valeur de l'attribut `select` doit être un **expression XPath**
- il n'y a pas de fonction de **transtypage** d'une **chaîne de caractères** en une **expression XPath**

Fragmentation des résultats impossible

Il n'est **pas possible** de **scinder la sortie du processeur XSLT** en plusieurs tranches
Il n'est **pas possible** dans XSLT de **spécifier la nature du flux de sortie**
(pour écrire dans un fichier par exemple)

Cela doit être réalisé par des **processus externes** à XSLT,
ou par des **extensions propriétaires** :

Apache Xalan

```
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
          extension-element-prefixes="redirect">
…/…
<xsl:template match="…/…">
          <redirect:write select="$filename">
                    <!-- toutes les sorties sont écrites dans le fichier $filename -->
          </redirect>
</xsl:template>
```

Association d'une feuille de style à un document

Comment ? Par l'utilisation d'une **instruction de traitement** spécifique :
`xml-stylesheet`

Convient : à XSLT et aux CSS

CSS : rendu pauvre (dans l'ordre du document, sur les éléments seulement)

Syntaxe : (à inclure dans le document XML)

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

PI interprétée par les navigateurs, qui transforment le source XML avec la feuille de style spécifiée

Pour associer plusieurs feuilles de style en fonction du média :

```
<?xml-stylesheet href="mystyle.xsl" type="text/xsl"?>
<?xml-stylesheet href="mystyle.wml.xsl" type="text/xsl" media="wap"?>
```

Ce qu'on ne peut pas faire :

- passer des paramètres
- brancher un URIResolver
- maîtriser la sérialisation (PDF, GIF, PNG, JPEG)
- transformer un nœud autre que le document

A éviter : les feuilles de styles sont applicables à une classe de documents (DTD)
Ce qui est commun doit être factorisé : déporter l'association d'une feuille de style au niveau du processus de publication des documents, au lieu du document lui-même.

Bibliothèques de règles existantes :

- manipulation de chaînes
- dates

…

http://xsltsl.sourceforge.net

XSLT *design patterns* :

- *Muenchian method*
- *Kaysian method*
- *Wendel Piez method*
- *Oliver Becker's method*

Extensions :

"standardisation" des extensions

http://www.exslt.org

bibliothèques mathématiques, manipulation de chaînes...

Performances

éviter d'utiliser `//` (qui est un gouffre pour les performances) si on connaît le chemin

Réseau

éviter de surcharger le résultat avec de l'indentation inutile, qui ne devrait servir qu'à des fins de débuggage

```
<xsl:output method="html" indent="no"/>
```

Organisation :

Modulariser les programmes :

- utiliser `xsl:import` et `xsl:include`
- distinguer les règles fonctionnelles des règles de formattage
- utiliser les espaces de nommage pour les collections de règles

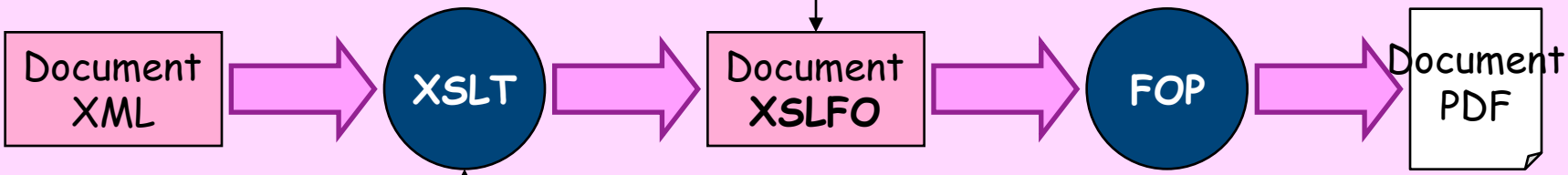
Formattage

http://www.w3.org/TR/xsl/

XSLFO est une application XML (un jeu de balises) qui permet de décrire la mise en page d'un document.
S'inspire de CSS.

Chaîne de traitement

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
          …/…
</fo:root>
```




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

Formatting Objects Processor

http://xmlgraphics.apache.org/fop/

```
<?xml version="1.0" ?>
<xsl:template match="/">
  <fo:root>
```

Layout

```
<fo:layout-master-set>
```

Content of document

```
<fo:page-sequence……>
    <xsl:apply-templates
                    select="…"/>
</…>
</xsl:template>
<xsl:template match="…">
    …
</xsl:template>
```

Gabarit des pages XSLFO

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
        <!-- gabarit des pages -->
    </fo:layout-master-set>
    <fo:page-sequence master-reference="content">
        <!-- data -->
    </fo: page-sequence >
</fo:root>
```

Un seul gabarit dans XSLFO 1.0

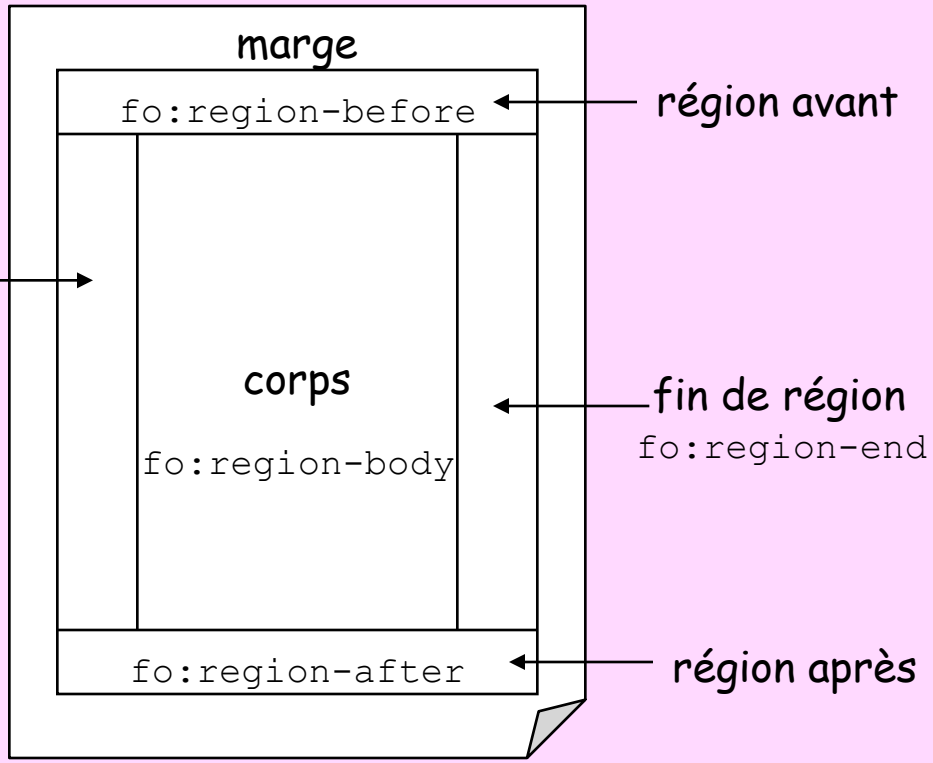
```
<fo:simple-page-master
          margin-right="1cm"
          margin-left="1cm"
          margin-bottom="1cm"
          margin-top="1cm"
          page-width="21cm"
          page-height="29.7cm"
          master-name="content"
>
</fo:simple-page-master>
```

5 régions dont la disposition est déterminée par le sens de l'écriture :

- Hébreu
- Chinois traditionnel

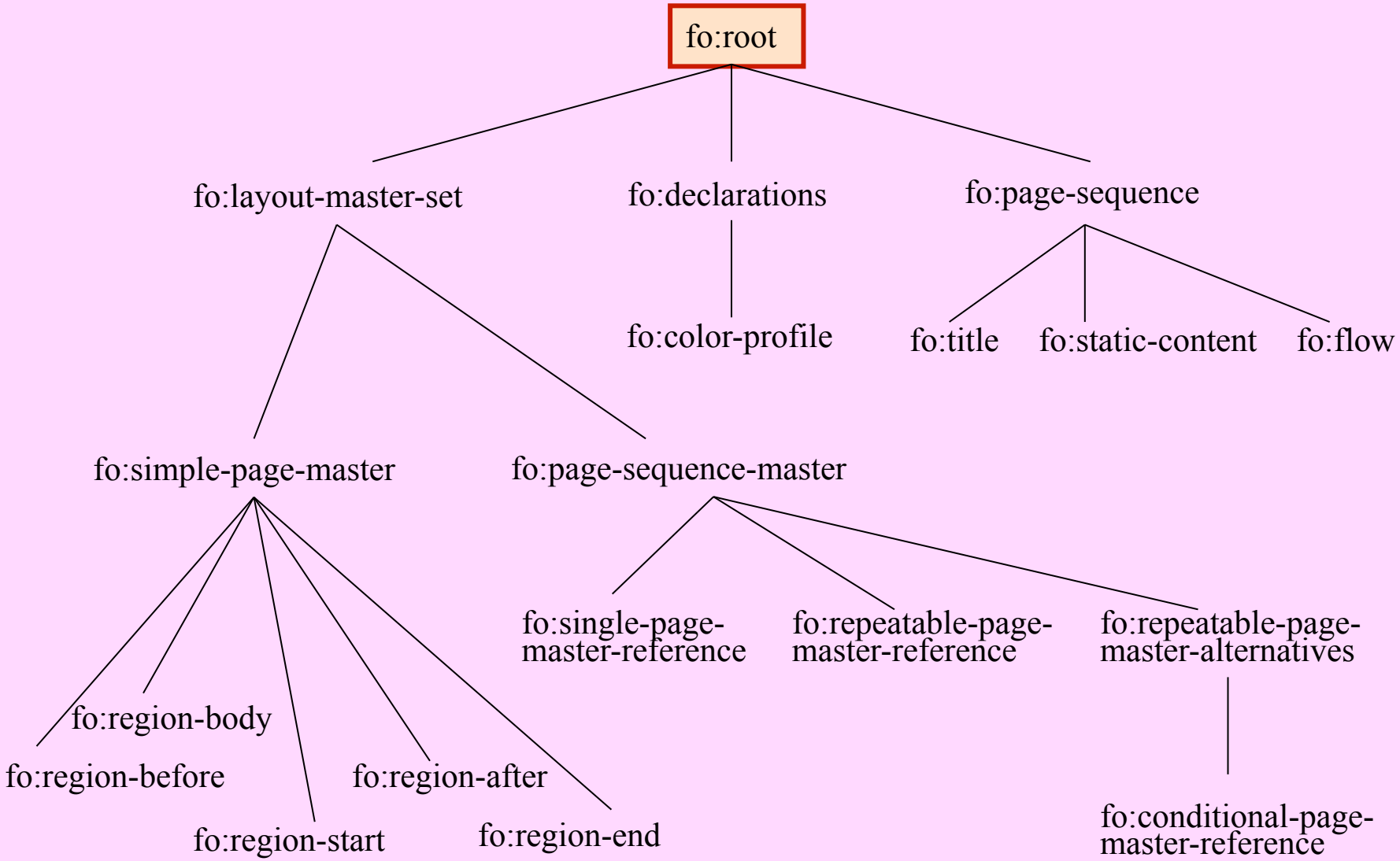
➔ une disposition spécifique des régions est prévue

page rectangulaire, largeur et hauteur fixes, marges

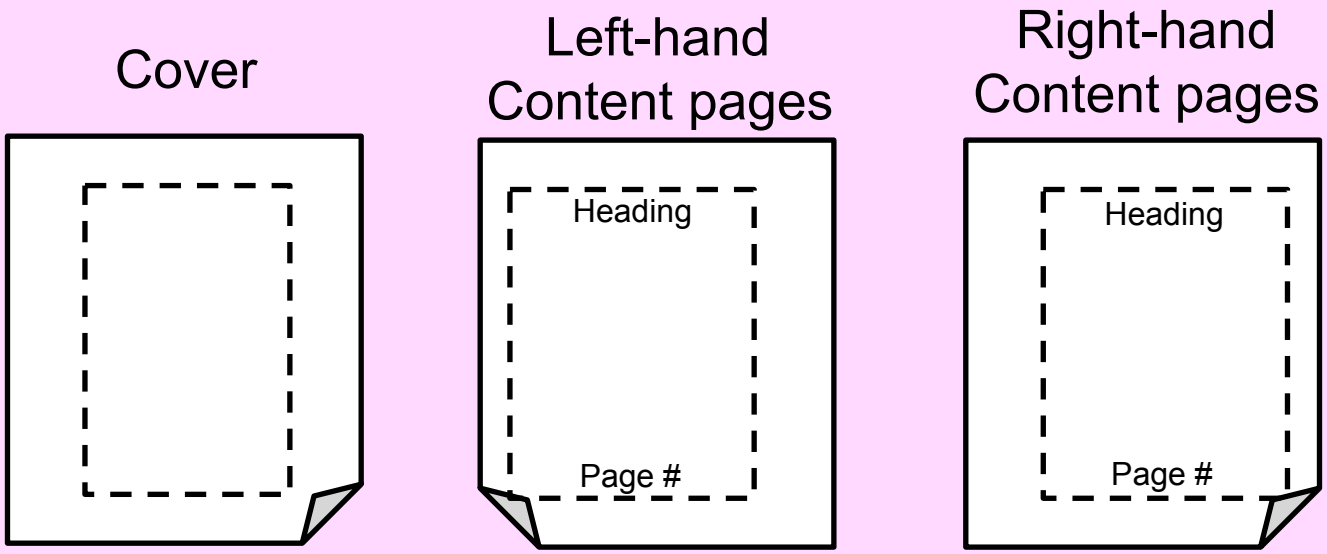


L'attribut `region-name` des éléments `fo:region-xxx` permet de nommer les régions (par défaut, leur nom est `xsl-region-xxx`).
L'attribut `extent` permet de définir la taille des régions

Arbre XSLFO


fo:root
fo:layout-master-set
fo:declarations
fo:page-sequence
fo:color-profile
fo:title
fo:static-content
fo:flow
fo:simple-page-master
fo:page-sequence-master
fo:region-body
fo:region-before
fo:region-after
fo:region-start
fo:region-end
fo:single-page-master-reference
fo:repeatable-page-master-reference
fo:repeatable-page-master-alternatives
fo:conditional-page-master-reference

Exemple




```
<fo:layout-master-set>
    <fo:simple-page-master master-name="cover"…/>
    <fo:simple-page-master master-name="leftPage"…/>
    <fo:simple-page-master master-name="rightPage"…/>
    <fo:page-sequence-master master-name="contents"…/>
</fo:layout-master-set>
```

Agencement des pages :

- <fo:simple-page-master>
  Définition des types de pages
- <fo:page-sequence-master>
  Définition des séquences d'enchaînement

```
<fo:simple-page-master
  master-name = name of the type of the page
  page-height = Page height
  page-width = Page width
  %margin-properties-CSS; = top, bottom, left, right, …
  reference-orientation = 0 | 90 |180 |270
  writing-mode = lr-tb | rl-tb | tb-rl>
  Content : (fo:region-body, fo:region-before?,
       fo:region-after?, fo:region-start?,
        fo:region-end?)
</fo:simple-page-master>
```

Enchaînement des pages

```
<fo:page-sequence-master master-name = name of sequence>
  Content : (single-page-master-reference|
                  repeatable-page-master-reference|
                  repeatable-page-master-alternatives)+
</fo:page-sequence-master>
```

Exemple

```
<fo:page-sequence-master master-name="contents">
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference
          master-name="leftPage" odd-or-even="even"/>
    <fo:conditional-page-master-reference
          master-name="rightPage" odd-or-even="odd"/>
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```

```
<fo:simple-page-master
        master-name="leftPage"
        page-height="12cm"
        page-width="12cm"
        margin-top="0.5cm"
        margin-bottom="0.5cm"
        margin-left="1cm"
        margin-right="0.5cm">
    <fo:region-body margin-top="3cm"/>
    <fo:region-before extent="1cm"/>
    <fo:region-after extent="1cm"/>
</fo:simple-page-master>
```

```
<fo:page-sequence
  master-reference = reference to a simple-page-master
                                or page-sequence-master>
  Content : (title?,static-content*,flow)
</fo:page-sequence>
```

Remplissage du contenu du document :

- L'attribut `master-reference` fait référence à un élément `<fo:simple-page-master>` ou `<fo:page-sequence-master>` du même nom.

Exemple

Static content

Flow

```
<fo:page-sequence master-reference="contents">
    <fo:static-content flow-name="xsl-region-before">
        …
    </fo:static-content>

    <fo:static-content flow-name="xsl-region-after">
        …
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body">
        …
    </fo:flow>

</fo:page-sequence>
```

Remplissage :

- <fo:block> équivalent HTML de
- Le formattage est spécifié par des propriétés CSS en tant qu'attribut XML
- <fo:inline> équivalent HTML de pour formatter une portion de texte dans un bloc
- Les propriétés CSS peuvent être aggrégées

```
<fo:page-sequence master-reference="contents">

  <fo:static-content flow-name="xsl-region-before">
    <fo:block font-family="Times" font-size="16pt">
          Chapître 1
    </fo:block>
  </fo:static-content>

  <fo:static-content flow-name="xsl-region-after">
    <fo:block>
          Page <fo:page-number/>
    </fo:block>
  </fo:static-content>

  <fo:flow flow-name="xsl-region-body">
    <fo:block>
        <fo:inline color="red" font-weight="bold">
          XSLFO
        </fo:inline>
        est un langage XML de formattage de document.
</fo:block>
  </fo:flow>
</fo:page-sequence>
```

<fo:block font="Times 16pt">

Insère le numéro de page courant

```
<fo:flow flow-name="xsl-region-body">
          <fo:block>
                    <!-- ici il y a du contenu -->
          </fo:block>
</fo:flow>
```

Un "bloc" par mise en forme spécifique

L'extrapolation des propriétés de `fo:block` à partir de CSS est souvent triviale

CSS

```
.foo {
          display: block;
          font-family: Arial;
          font-size: 20pt;
          font-weight: bold;
          text-align: center
}
```

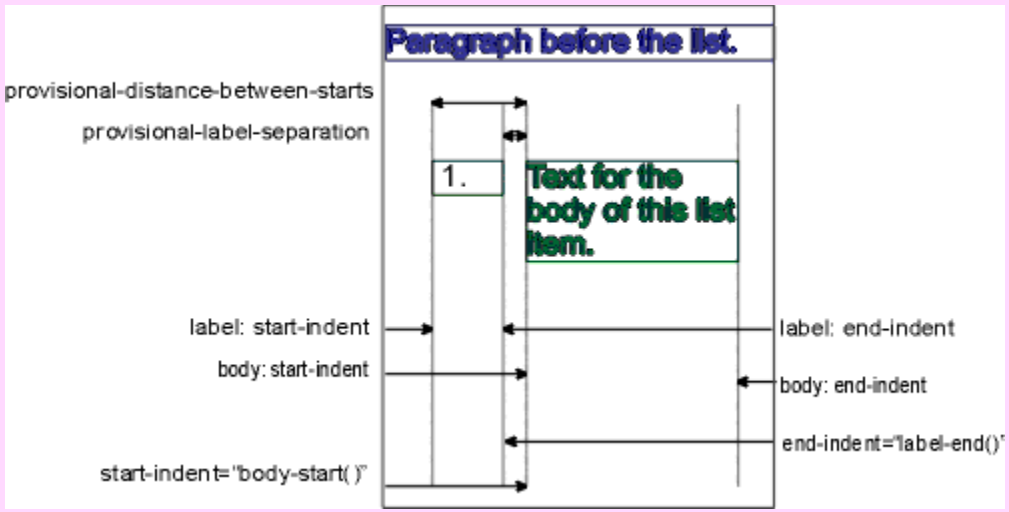
XSLFO

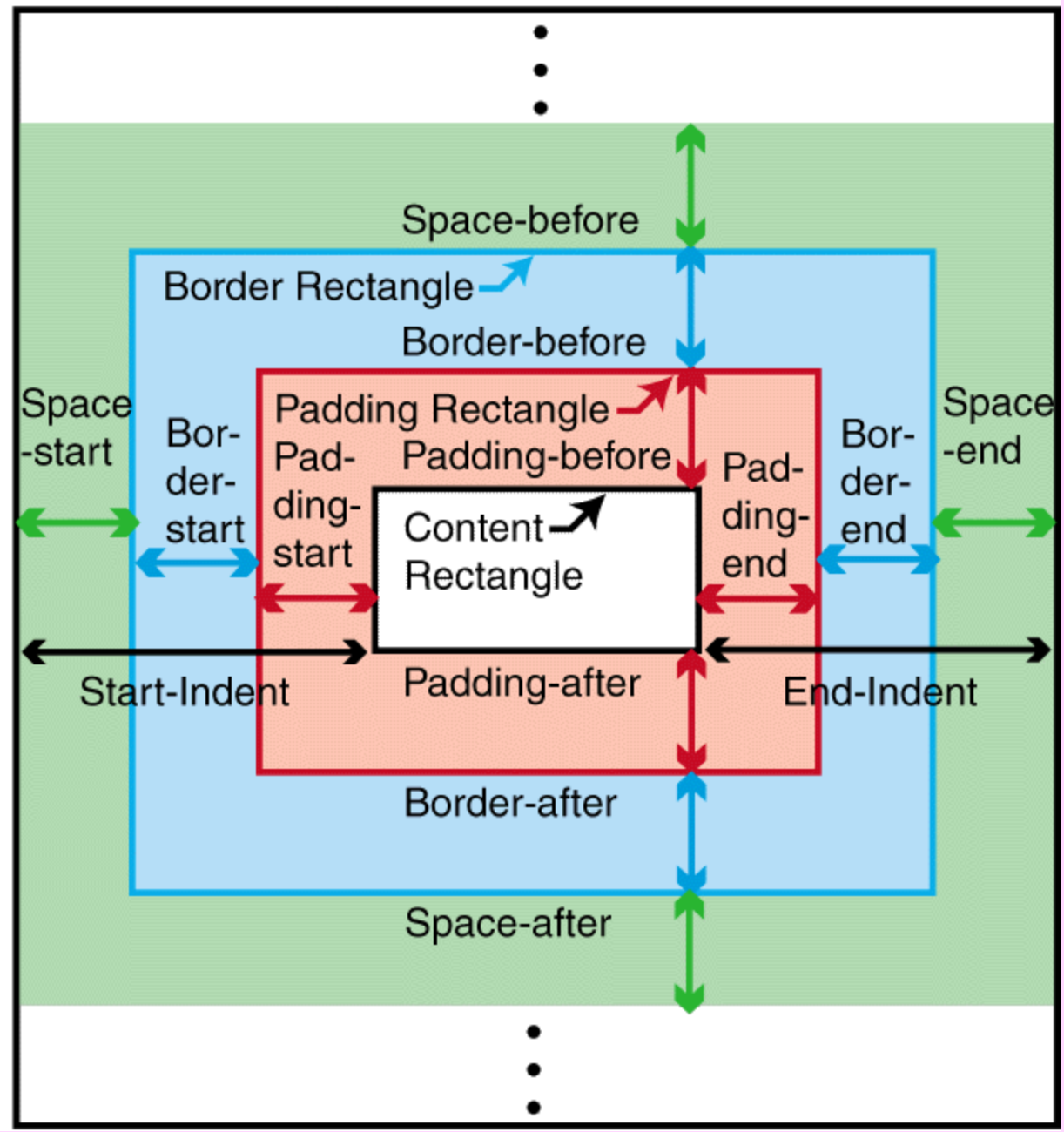
```
<fo:block
          font-family="Arial"
          font-size="20pt"
          font-weight="bold"
          text-align="center"
>
```

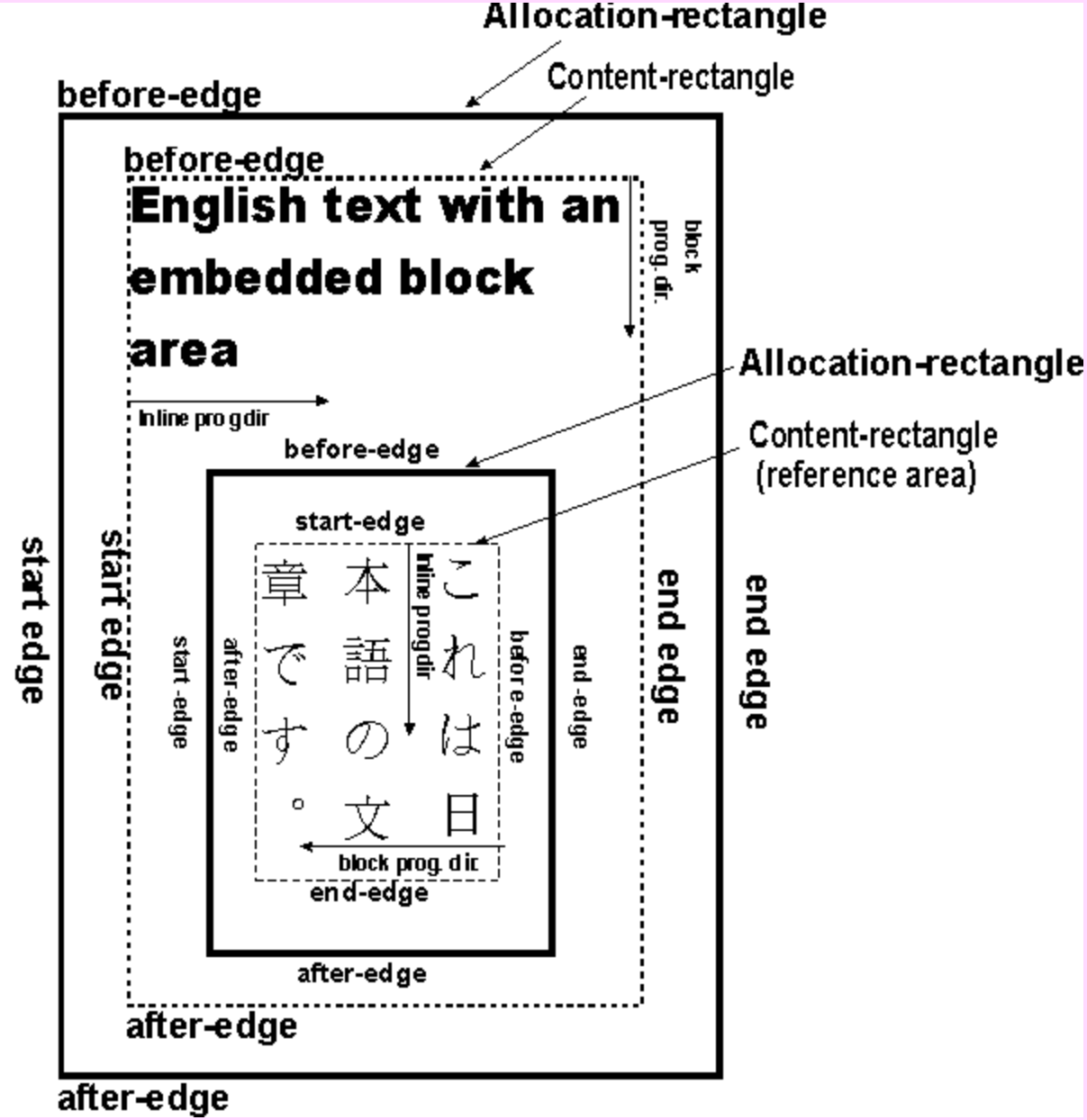
Autres éléments :

- listes

```
fo:list-block
    └── fo:list-item
            ├── fo:list-item-label
            └── fo:list-item-body
```




Space-before
Border Rectangle
Border-before
Space
-start
Bor-
der-
start
Padding Rectangle
Pad-
ding-
start
Padding-before
Content
Rectangle
Pad-
ding-
end
Bor-
der-
end
Space
-end
Start-Indent
Padding-after
End-Indent
Border-after
Space-after


Allocation-rectangle
Content-rectangle
before-edge
before-edge
English text with an embedded block area
block prog. dir.
Allocation-rectangle
Inline prog dir
Content-rectangle
(reference area)
before-edge
start-edge
これは日本語の文章です。
Inline prog dir
block prog. dir
end-edge
start-edge
after-edge
before-edge
end-edge
after-edge
start edge
start edge
end edge
end edge
after-edge
after-edge

Liens

```
<fo:block>
    <fo:basic-link external-destination="www.inria.fr" color="blue">
            www.inria.fr
    </fo:basic-link>
</fo:block>
```

Images

```
<fo:block text-align="center">
    <fo:external-graphic
        src="file:images/logo.gif" width="50pt"/>
</fo:block>
```

Contrôle des sauts de page

```
<fo:block text-align="start" font-size="14pt"
          font-family="serif"
          break-before="odd-page">
            …
</fo:block>
```

Numérotation des pages

```
<fo:static-content flow-name="xsl-region-after">
    <fo:block font-family="Helvetica" font-size="10pt" text-align="center">
        Page - <fo:page-number/>
    </fo:block>
</fo:static-content>
```

Notes de bas de page

```
<fo:block>
    Footer <fo:footnote>
             <fo:inline baseline-shift="super" font-size="smaller">(1)</fo:inline>
             <fo:footnote-body>
                 <fo:list-block provisional-label-separation="0pt"
                                provisional-distance-between-starts="18pt"
                                space-after.optimum="6pt">
                     <fo:list-item>
                         <fo:list-item-label end-indent="label-end()">
                             <fo:block>(1)</fo:block>
                         </fo:list-item-label>
                         <fo:list-item-body start-indent="body-start()">
                             <fo:block>Bla bla</fo:block>
                         </fo:list-item-body>
                     </fo:list-item>
                 </fo:list-block>
             </fo:footnote-body>
           </fo:footnote>
</fo:block>
```

Références

```
<fo:external-graphic id="inriaPic" src="url('www.inria.jpg')"/>
...
Voir page : <fo:page-number-citation ref-id="inriaPic"/>)
```

Inclusion SVG dans XSLFO


```
<fo:block>
  <fo:instream-foreign-object>
    <svg:svg
          xmlns:svg="http://www.w3.org/2000/svg"
          xml:space="preserve"
          width="152mm" height="116mm"
          preserveAspectRatio="meet meet"
          viewBox="0 0 751.859 576.421">
          <svg:g id="Layer_x0020_1" transform="scale(0.575,0.575)">
            …
          </svg:g>
    </svg:svg>
  </fo:instream-foreign-object>
</fo:block>
```

Exemple XSLFO

```
<?xml version="1.0" encoding="UTF-8" ?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master margin-bottom="1cm" margin-left="1.5cm" margin-right="1.5cm"
        margin-top="1.5cm" master-name="page-initiale">
      <fo:region-body margin-bottom="1cm" margin-top="3cm" />
      <fo:region-before extent="3cm" region-name="page-initiale.before" />
      <fo:region-after extent="0.5cm" region-name="page-initiale.after" />
    </fo:simple-page-master>

    <fo:page-sequence-master master-name="tout">
      <fo:repeatable-page-master-alternatives>
        <fo:conditional-page-master-reference
            master-reference="page-initiale" page-position="first" />
        <fo:conditional-page-master-reference
            master-reference="page-impaire" odd-or-even="odd" />
        <fo:conditional-page-master-reference
            master-reference="page-paire" odd-or-even="even" />
      </fo:repeatable-page-master-alternatives>

<fo:flow flow-name="xsl-region-body">
   <fo:block font-size="10pt" font-weight="bold" space-after.optimum="4mm"
       space-before.optimum="4mm" text-align="justify">Le règne animal offre maints
exemples de sociétés capables de réaliser des prouesses par simple coopération entre des
centaines voire des milliers d'individus : les fourmis, les termites, ou encore les
araignées sociales. Ces dernières et leurs superbes toiles ont inspiré des chercheurs en
informatique, avides eux aussi de prouesses, mais sur ordinateur.</fo:block>
  <fo:block font-size="10pt" font-weight="bold" space-after.optimum="4mm"
      space-before.optimum="4mm" text-align="justify">The animal kingdom offers many
examples of societies whose achievements are possible uniquely and simply through the
cooperation of tens, hundreds, or even thousands of individuals. Ants, termites, or
certain species of social spiders. The webs of the latter were a source of inspiration for
computer scientists in image processing.</fo:block>
  <fo:block space-after.optimum="10mm">
    <fo:block font-family="sans-serif" font-size="14pt" font-weight="bold"
```

```
public class Xslt {

    Source src;
    Source xslt;
    OutputStream outStream;

    public void setXML(Source src, Source xslt, OutputStream outStream) {
        this.src = src;
        this.xslt = xslt;
        this.outStream = outStream;
    }

    public void start() throws IOException {
        try {
            TransformerFactory factory = TransformerFactory.newInstance();
            Transformer transformer = factory.newTransformer(xslt);
            Result res = getResult();
            transformer.transform(src, res);
        } catch (TransformerException e) {
            throw new IOException(e);
        } finally {
            outStream.close();
        }
    }

    protected Result getResult() throws TransformerException {
        Result res = new StreamResult(outStream);
        return res;
    }
}
```

Transformation PDF en Java

```
public class Pdf extends Xslt {

    @Override
    protected Result getResult() throws TransformerException {
        try {
            FopFactory fopFactory = FopFactory.newInstance();
            Fop fop = fopFactory.newFop(MimeConstants.MIME_PDF, outStream);
            fop.getUserAgent().setURIResolver(RESOLVER);
            Result res = new SAXResult(fop.getDefaultHandler());
            return res;
        } catch (FOPException e) {
            throw new TransformerException(e);
        }
    }
}
```

Obtenir FOP dans Maven


```
<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>fop</artifactId>
    <version>1.1</version>
    <exclusions>
        <exclusion>
            <artifactId>avalon-framework-api</artifactId>
            <groupId>org.apache.avalon.framework</groupId>
        </exclusion>
        <exclusion>
            <artifactId>avalon-framework-impl</artifactId>
            <groupId>org.apache.avalon.framework</groupId>
        </exclusion>
    </exclusions>
</dependency>
<!-- these two are to correct issues in fop dependency -->
<dependency>
    <groupId>avalon-framework</groupId>
    <artifactId>avalon-framework-api</artifactId>
    <version>4.2.0</version>
</dependency>
<dependency>
    <groupId>avalon-framework</groupId>
    <artifactId>avalon-framework-impl</artifactId>
    <version>4.2.0</version>
</dependency>
```