Développer des projets en XML

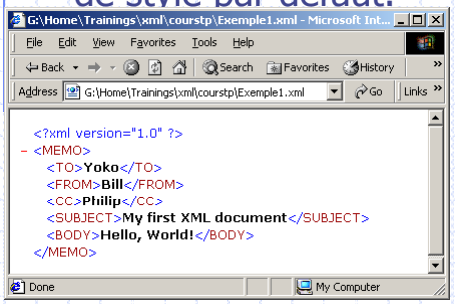
XSLT, langage de transformation de documents XML pour la présentation sur le Web ou convertir en d'autres documents.

Rendu visuel de document XML

- Un document XML, à la base, ne comprend aucune information destinée aux agents utilisateurs appliquant le rendu graphique visuel du document
- Il faut combiner la technologie des feuilles de style avec XML
- Deux technologies existent actuellement :
  - CSS, Cascading Style Sheets
  - XSL, eXtensible Markup Language :
    - XSL-T (tranformation) pour transformer un document.
    - XSL-FO (formating objects) pour générer des sorties papier (non encore normalisé)

Rendu visuel de document XML

- Certains navigateurs (IE5) prévoit une feuille de style par défaut.




```
<?xml version="1.0"?>
<MEMO>
   <TO>Yoko</TO>
    <FROM>Bill</FROM>
    <CC>Philip</CC>
    <SUBJECT>My first XML
document</SUBJECT>
    <BODY>Hello, World!</BODY>
</MEMO>
```


Exemples CSS comparé à XSL-T

Approche différente de CSS et XSL :

```
item {
font-family: Helvetica;
font-size: 10pt;
color: black;
margin-top:0pt;
margin-bottom:6pt;
}
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">
  <html>
    <head><title>UN/SPSC</title></head>
    <body>
      <xsl:apply-templates select="//segment"/>
      </body>
    </html>
  </xsl:template>

<xsl:template match="segment">
  <h1><xsl:value-of select="@ID"/>.
    <xsl:value-of select="title"/></h1>
  <xsl:apply-templates select="family"/>
  </xsl:template>
```

Différence de traitement entre XML+CSS et XML+XSL-T

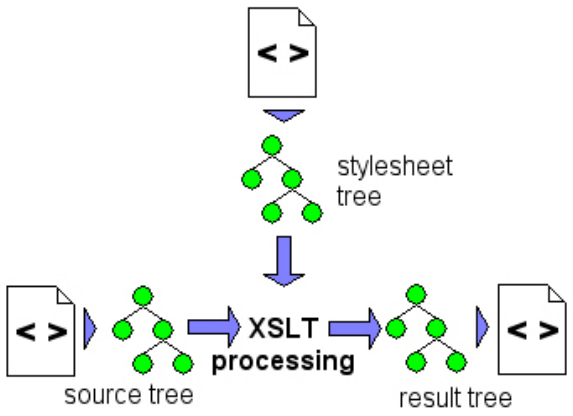
- **Cascading style sheets**
  - applies display
  - easy to learn
  - less powerful

XML + CSS = XML view

- **Extensible style language**
  - transforms
  - full programming language
  - harder to learn

XML + XSL = HTML

Traitement de transformation de XSL-T



Différence entre CSS et XSL-T

- CSS :
  - applique des règles de visualisation
  - facile à étudier
  - langage déclaratif
  - moins puissant comparé à XSL
- XSL-T :
  - applique une transformation de l'arbre d'entrée des éléments (création d 'un arbre résultat)
  - véritable langage de programmation
  - moins simple à apprendre

Utilisation de feuilles de style CSS

- Une règle de style s 'applique à un élément ou à une collection d 'éléments XML et permet de fournir à l 'agent utilisateur, les informations nécessaires pour la visualisation de cet élément
- Pour la spécification des règles de styles dans un document XML, il faut référencer un fichier externet en utilisant l 'instruction processeur suivante :

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

- plusieurs feuilles peuvent être référencées, avec traitement en cascade, proximité = priorité

CSS et XML, exemple

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/css" href="booklist.css"?>
<BOOKLIST>
 <ITEM>
  <CODE>16-048</CODE>
  <CATEGORY>Scripting</CATEGORY>
  <RELEASE_DATE>1998-04-21</RELEASE_DATE>
  <TITLE>Instant JavaScript</TITLE>
  <SALES>375298</SALES>
 </ITEM>
```


CSS et XML, exemple

```
 <ITEM>
  <CODE>16-105</CODE>
  <CATEGORY>ASP</CATEGORY>
  <RELEASE_DATE>1998-05-10</RELEASE_DATE>
  <TITLE>Instant Active Server Pages</TITLE>
  <SALES>297311</SALES>
 </ITEM>
 <ITEM>
  <CODE>16-041</CODE>
  <CATEGORY>HTML</CATEGORY>
  <RELEASE_DATE>1998-03-07</RELEASE_DATE>
  <TITLE>Instant HTML</TITLE>
  <SALES>127853</SALES>
 </ITEM>
</BOOKLIST>
```

CSS et XML, exemple

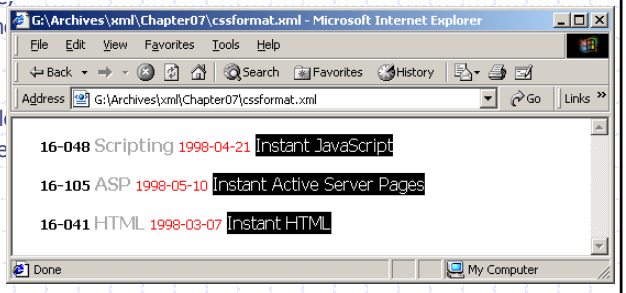
```
ITEM        { display:block; margin:15px }

CODE        { display:inline;
              font-family:Tahoma,Arial,sans-serif;
              font-size:10pt;
              font-weight:bold }

CATEGORY    { display:inline;
              font-family:Tahoma,Arial,sans-serif;
              color:darkgray;
              font-size:12pt;
              font-weight:bold }
```


CSS et XML, exemple

```
RELEASE_DATE { display:inline;
               font-family:Tahoma,Arial,sans-serif;
               color:red;
               font-size:10pt }
TITLE        { display:inline;
               font-family:Tah
               font-size:12pt;
               color:white;
               background-col
SALES        { display:none
```



Introduction à XSL

- XSL, extensible markup language
- But : rendre la structure du document XML indépendante de l'affichage ou autres traitements de rendus graphiques (écran, papier, …)
- La technologie XSL est composée de deux parties :
  - un langage de transformation (XSL-T, déjà normalisé)
  - des spécifications d'un modèle objet de formatage (XSL-FO, non normalisé)
- Ces deux parties sont implantées sous formes de namespaces différents

XSL-T, Langage de transformation

- Namespace : xsl
- Permet de convertir un arbre XML en une autre structure résultat
- Le document résultat est une structure (ou un fichier plat dans la cas "texte ») peut inclure des balises HTML ou d'autres langages, permettant le rendu visuel de ce document
- XSL-T se base sur une autre langage XPath pour sélectionner des éléments dans l 'arbre d 'entrée.

XSL-T, Langage de transformation

- XSL-T est un langage développé pour XML
  - il est plus puissant
  - son champ d'action dépasse la mise en page
  - il se base sur une syntaxe XML
  - les besoins d'un document XML sont différents
    - en particulier, il faut réorganiser le document
- le standard est plus complexe que CSS
  - XSL-T est composé de différent éléments

XSL-T, Langage de transformation

- XSL-T est une application de XML, un langage conçu à partir de XML
- XSL-T est basé sur un mécanisme similaire au feuille de style mais XSL-T est un véritable langage de programmation
- Application complète comprend au minimum 3 documents
  - Données d'entrée
  - feuille de style exprimée en XSL-T
  - La structure résultat (xml, html ou texte)

XSL-FO, spécification de formatage

- Namespace : fo
- Inclut un ensemble d ’élément de formatage
- Permet d'écrire une nouvelle sémantique d'affichage qui pourrait être traitée directement par un moteur d'affichage ou de génération de sorties (navigateur ou autre)
- Non encore normalisé
- voir projet FOP

Le langage XSL-T et les feuilles de style XSL

- Décrivent le processus de création d'une structure en sortie
- Comprend un ou plusieurs modèles appliqué à des éléments et contenant des éléments d ’autres langage et des éléments du document XML en entrée
- Utilise la technique de patterns pour référencer les éléments du document XML.
- Les modèles fournissent une structure au document de sortie (HTML, XML ou texte).

Le langage XSL-T et les feuilles de style XSL

- Les modèles XSL référencent les éléments ou attributs du document XML en entrée via des patterns
- Ne doivent pas être forcement liées à des fichiers XML, XSL peut être utilisé pour générer un nouveau document sans données en entrée

Exemple

- utilisons XML pour convertir une liste de produit en HTML
  - le document source
  - la feuille de style
  - le résultat dans un navigateur
    - dans ce cas, la conversion est faite par le serveur

Catalogue XML (I)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<products>
 <product id="0">
  <name>WizzBang Ultra Word Processor</name>
  <description xml:lang="EN">More words per
   minute than the competition.</description>
  <description xml:lang="FR">Plus de mots à
   la minute que la
   concurrence.</description>
  <image>wordprocessor.jpg</image>
  <price>$799.99</price>
 </product>
```


Catalogue XML (II)

```
 <product id="1">
  <name>Super WizzBang Calculator</name>
  <description xml:lang="EN">Cheap and
   reliable with power saving.</description>
  <description xml:lang="FR">Economique et
   fiable avec économie
   d'énerge.</description>
  <image>calculator.jpg</image>
  <price>$5.99</price>
 </product>
```

Catalogue XML (III)

```
  <product id="2">
   <name>WizzBang Safest Safe</name>
   <description xml:lang="EN">Choose the
    authentic WizzBang Safest
    Safe.</description>
   <description xml:lang="FR">Exigez
    l'original!</description>
   <image>safe.jpg</image>
   <price>$1,999.00</price>
  </product>
</products>
```


XPath

- Spécifient les éléments XML auxquels s'applique le modèle XSL
- Fournissent un langage de requête simple pour identifier un nœud (un élément) dans un document xml.
- On appelle contexte le nœud sur lequel s'applique la requête, càd en cours de traitement

XPath

- On repère un nœud par son nom précédé de tous les noms des nœuds parents séparés par "/" en partant de la racine
- Exemple : authors/author/name
- Le pattern représente TOUS les éléments qui correspondent au pattern
- Peuvent contenir des caractères génériques = "*"

XPath

- comme des chemins de fichiers
  - / est le séparateur
    - /products/product
  - @ pointe vers un attribut
    - /products/product/@id
  - condition entre []
    - /products/product/description[@xml:lang='FR']

La racine du document

- /

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<products>
 <product id="0">
 <name>WizzBang Ultra Word Processor</name>
  <description xml:lang="EN">More words per
   minute than the competition.</description>
  <!-- ... -->
  <image>safe.jpg</image>
  <price>$1,999.00</price>
 </product>
</products>
```


Les Produits

- /products/product

```
<product id="0">
  <!-- ... -->
</product>
<product id="1">
  <!-- ... -->
</product>
<product id="2">
  <!-- ... -->
</product>
```

Les noms de produit

- /products/product/name

```
<name>WizzBang Ultra Word Processor</name>
<name>Super WizzBang Calculator</name>
<name>WizzBang Safest Safe</name>
```


Les identifiants des produits

- /products/product/@id

```
0
1
2
```

Les descriptions en français

- /products/product/description[@xml:lang='FR']

```
<description xml:lang="FR">Plus de mots à
  la minute que la
  concurrence.</description>
<description xml:lang="FR">Economique et
  fiable avec économie
  d'énerge.</description>
<description xml:lang="FR">Exigez
  l'original!</description>
```


Le nom du produit 1

- /products/product[@id='1']/name

```
<name>Super WizzBang Calculator</name>
```

Les noms comme texte

- /products/product/name/text()

```
WizzBang Ultra Word Processor
Super WizzBang Calculator
WizzBang Safest Safe
```


L'utilisation du //

- //name

```
<name>WizzBang Ultra Word Processor</name>
<name>Super WizzBang Calculator</name>
<name>WizzBang Safest Safe</name>
```

Combinaison de critères

- //product[description/@xml:lang='EN' and price='$799.99']/name

```
<name>WizzBang Ultra Word Processor</name>
```

- //product[price='$5.99' or price='$799.99']/name

```
<name>WizzBang Ultra Word Processor</name>
<name>Super WizzBang Calculator</name>
```


Négation

- //product[not(price='$5.99')]/name

```
<name>WizzBang Ultra Word Processor</name>
<name>WizzBang Safest Safe</name>
```

- //product[price != '$5.99']/name

```
<name>WizzBang Ultra Word Processor</name>
<name>WizzBang Safest Safe</name>
```

Feuille XSL (I)

```
<?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/REC-html40">

<xsl:output method="html"/>

<xsl:template match="/">
   <HTML><HEAD><TITLE>Shop</TITLE></HEAD>
   <BODY>
      <P>Select a product:</P>
```


Feuille XSL (II)

```
<TABLE BORDER="0">
   <xsl:for-each select="products/product">
   <TR><TD>
   <B><xsl:value-of select="name"/></B><BR/>
   <xsl:value-of
  select="description[@xml:lang='EN']"/><BR/>
   <xsl:value-of select="price"/>
    </TD>
    <TD><IMG><xsl:attribute name="SRC">
          <xsl:value-of select="image"/>
       </xsl:attribute></IMG>
    </TD>
```

Feuille XSL (III)

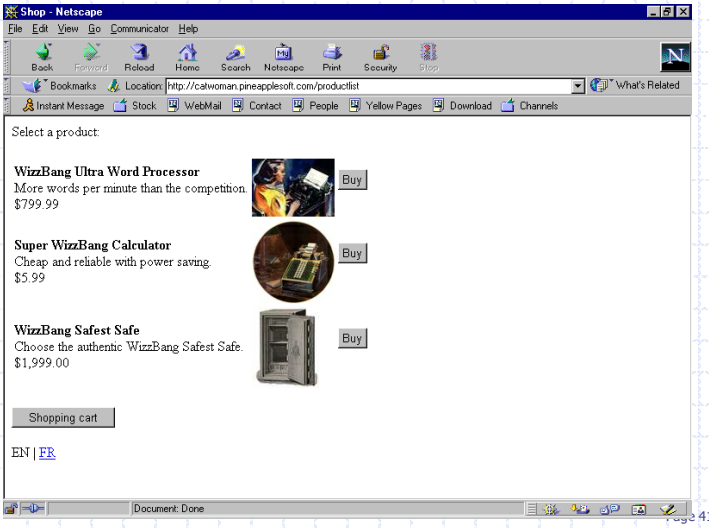
```
<TD><FORM ACTION="shoppingcart" METHOD="POST">
   <INPUT TYPE="HIDDEN" NAME="add">
      <xsl:attribute name="value">
         <xsl:value-of select="@id"/>
      </xsl:attribute></INPUT>
   <INPUT TYPE="SUBMIT" VALUE="Buy"/>
</FORM></TD>
</TR>
</xsl:for-each>
</TABLE>
```


Feuille XSL (IV)

```
<FORM ACTION="shoppingcart">
   <INPUT TYPE="SUBMIT" VALUE="Shopping cart"/>
</FORM>
EN | <A
  HREF="productlist?xsl=/productlist_fr.xsl">FR</A>
</BODY>
</HTML>
</xsl:template>

</xsl:stylesheet>
```

Le résultat



xsl:stylesheet

- le point d'entrée dans la feuille de style
  - version="1.0"
  - XSL namespace
    - http://www.w3.org/1999/XSL/Transform
  - HTML namespace
    - http://www.w3.org/TR/REC-html40

xsl:output

- spécifie le format de sortie
  - xml
  - html
  - text
- cela sélectionne une routine de formatage différente mais la structure est inchangée

xsl:template

- détaille la transformation d'un élément
  - l'élément est sélectionné par le match
  - l'élément est remplacé par le contenu du template
- les éléments HTML suivent la syntaxe XML

xsl:template

- Syntaxe :

```
<xsl:template match="node-context"  >
```

  - match spécifie le contexte à partir duquel le template sera exécuté
  -
```
<xsl:template match="stock">
  <DIV STYLE="font-weight:bold">
    Symbol: <xsl:value-of match="symbol" />,
    Price: <xsl:value-of match="price" />
  </DIV>
</xsl:template>
```


xsl:value-of

- extrait la valeur d'un élément du document
  - l'élément est sélectionné par un XPath
- XPath est similaire au chemin de fichiers
  - / est la racine
  - products/product
    - l'élément product enfant de products
  - pour sélectionner un attribut
    - description[@xml:lang='EN']

xsl:for-each

- boucle sur les différents nœuds d'un jeu de nœuds
- complète xsl:value-of pour les XPath retournant plusieurs éléments
- le tout est assez proche d'un mailing en Word

xsl:attribute

- permet de calculer un attribut dans le document résultat
- uniquement utile lorsqu'il faut calculer la valeur de l'attribut, sinon il suffit d'insérer l'attribut directement

Association document XML et feuille de style XSL

- Syntaxe : instruction processeur (dans le document xml en entrée, dans le prologue)

  <?xml-stylesheet type="text/xsl" href="*File_name*.xsl"?>

  - type peut prendre les valeurs
    - text/xsl = feuille de style XSL
    - text/css = feuille de style CSS
  - href = url de la feuille de style

Les modèles

- Exemple:

```
<?xml version="1.0">
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE>XSL Test</TITLE>
    </HEAD>
    <BODY>
      <B>This is a test of an XSL template.</B>
    </BODY>
  </HTML>
</xsl:template>
</xsl:stylesheet>
```

Processeur XSL

- Le processeur XSL traite l 'arbre en entrée pour produire un autre arbre en sortie
- Chaque élément XML en entrée sont analysés et peuvent faire l 'objet d 'une projection dans l 'arbre résultat
- La terminologie fait également référence à un nœud représentant un élément en cours de traitement

Autres éléments du langage XSL-T

Elément XSL	Description
xsl:comment	Génère un commentaire dans la sortie
xsl:copy	Copie le nœud courant de la source vers la sortie
xsl:element	Crée un élément de nom spécifié dans la sortie
xsl:for-each	Applique un modèle de manière répétitive à une collection d'éléments

Autres éléments du langage XSL-T

Elément XSL	Description
xsl:if	Condition
xsl:choose	Permet l'utilisations de conditions multiples
xsl:when	A utiliser avec xsl: choose
xsl:otherwhise	A utiliser avec xsl:choose
xsl:value-of	Insère la valeur du nœud comme du texte

Exemple de script utilisant XML et XSL-T

- Fichier HTML :

```
<HTML>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="onload">
      var source = new ActiveXObject("Microsoft.xmldom");
      source.load("Lst8_4.xml");
      var style = new ActiveXObject("Microsoft.xmldom");
      style.load("Lst8_5.xsl");
      xslContainer.innerHTML =source.transformNode(style.documentElement);
    </SCRIPT>
    <TITLE>Code Listing 8-6</TITLE>
  </HEAD>
  <BODY>
    <DIV ID="xslContainer"></DIV>
  </BODY>
</HTML>
```

Autre forme

- cette feuille de style est très proche du résultat final
- parfois il est souhaitable de se rapprocher du document d'origine

Feuille XSL (I)

```
<?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/REC-html40">
<xsl:output method="html"/>
<xsl:template match="/">
   <HTML><HEAD><TITLE>Shop</TITLE></HEAD>
   <BODY><P>Select a product:</P>
   <xsl:apply-templates/>
   <FORM ACTION="shoppingcart">
     <INPUT TYPE="SUBMIT" VALUE="Shopping cart"/>
   </FORM>
EN | <A HREF="productlist?xsl=/productlist_fr.xsl">
FR</A></BODY></HTML>
</xsl:template>
```

Feuille XSL (II)

```
<xsl:template match="product">
   <FORM ACTION="shoppingcart" METHOD="POST">
      <INPUT TYPE="HIDDEN" NAME="add">
         <xsl:attribute name="value">
            <xsl:value-of select="@id"/>
         </xsl:attribute>
      </INPUT>
      <xsl:apply-templates/>
      <INPUT TYPE="SUBMIT" VALUE="Buy"/>
   </FORM>
</xsl:template>

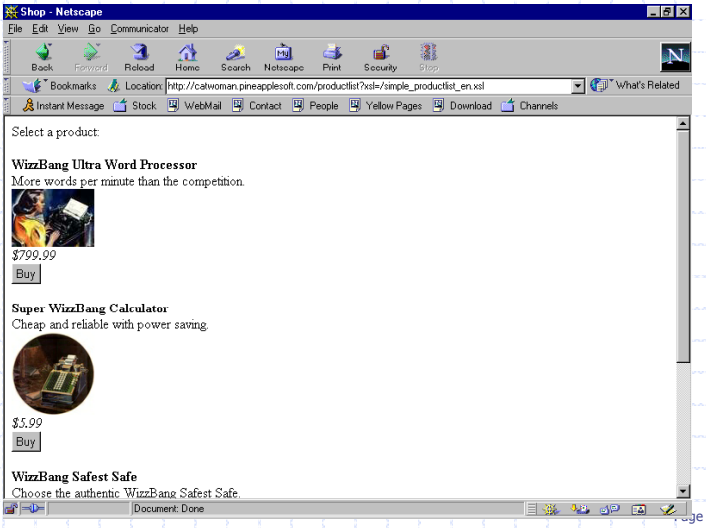
<xsl:template match="name">
   <B><xsl:apply-templates/></B><BR/>
</xsl:template>
```


Feuille XSL (III)

```
<xsl:template match="description[@xml:lang='EN']">
   <xsl:apply-templates/><BR/>
</xsl:template>
<xsl:template match="description[not(@xml:lang='EN')]"/>
<xsl:template match="price">
   <I><xsl:apply-templates/></I><BR/>
</xsl:template>
<xsl:template match="image">
   <IMG><xsl:attribute name="SRC">
         <xsl:apply-templates/></xsl:attribute>
   </IMG><BR/>
</xsl:template>

</xsl:stylesheet>
```

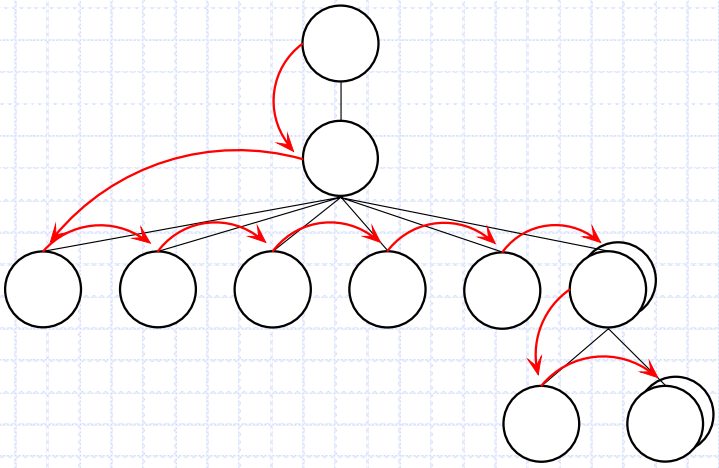
Le résultat



Différences

- il y a plusieurs xsl : template
  - le fonctionnement est récursif
  - le processeur avance sur le document
  - à chaque élément, il applique la feuille de style
- xsl:apply-templates correspond à un appel récursif
- ```
  <xsl:template match="price">
      <I><xsl:apply-templates/></I><BR/>
  </xsl:template>
  ```
  ```

Chemin du processeur



Feuille XSL (I)

```
<?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/REC-html40">

<xsl:output method="html"/>

<xsl:template match="/">
   <HTML>
   <HEAD>
      <TITLE>Shop</TITLE>
   </HEAD>
```

Feuille XSL (II)

```
<BODY>
    <P>Quick access:</P>
       <xsl:apply-templates mode="toc"/>
    <P>Select a product:</P>
    <xsl:apply-templates/>
    <FORM ACTION="shoppingcart">
       <INPUT TYPE="SUBMIT" VALUE="Shopping cart"/>
    </FORM>
    EN | <A
  HREF="productlist?xsl=/productlist_fr.xsl">FR</A>
   </BODY>
   </HTML>
</xsl:template>
```


Feuille XSL (III)

```
<xsl:template match="product" mode="toc">
   <xsl:apply-templates select="name" mode="toc"/>
   <!-- -1 is to account for the "newline" text node-->
   <xsl:if test="not(position()=last() - 1)"> |
  </xsl:if>
</xsl:template>

<xsl:template match="name" mode="toc">
   <A>
      <xsl:attribute name="HREF">
         <xsl:text>#</xsl:text>
         <xsl:value-of select="generate-id(..)"/>
      </xsl:attribute>
```

Feuille XSL (IV)

```
 <xsl:apply-templates/>
   </A>
</xsl:template>

<xsl:template match="product">
   <FORM ACTION="shoppingcart" METHOD="POST">
      <INPUT TYPE="HIDDEN" NAME="add">
         <xsl:attribute name="value">
            <xsl:apply-templates select="@id"/>
         </xsl:attribute>
      </INPUT>
      <xsl:apply-templates select="image"/>
      <xsl:apply-templates select="name"/>
```


Feuille XSL (V)

```
      <xsl:text> </xsl:text>
      <xsl:apply-templates select="price"/><BR/>
      <xsl:apply-templates select="description"/><BR/>
      <INPUT TYPE="SUBMIT" VALUE="Buy"/>
      <BR CLEAR="LEFT"/>
   </FORM><BR/>
</xsl:template>

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

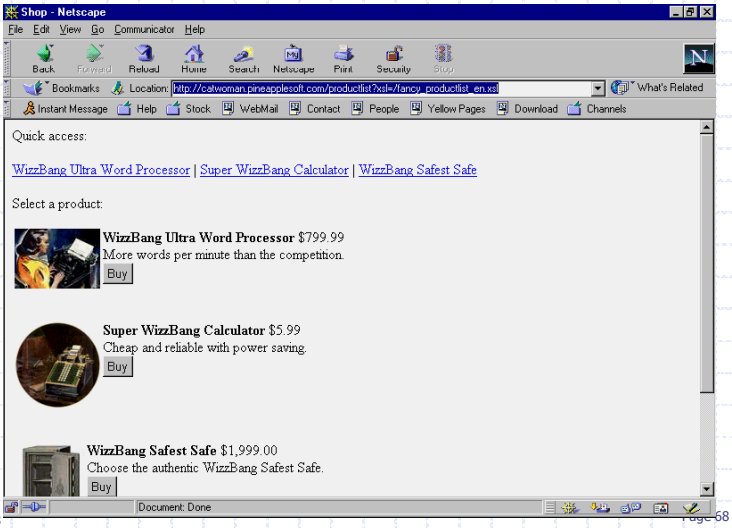
<xsl:template match="description[not(@xml:lang='EN')]"/>
```

Feuille XSL (VI)

```
<xsl:template match="image">
   <A>
      <xsl:attribute name="NAME">
         <xsl:value-of select="generate-id(..)"/>
      </xsl:attribute>
      <IMG ALIGN="LEFT">
         <xsl:attribute name="SRC">
            <xsl:apply-templates/>
         </xsl:attribute>
      </IMG>
   </A>
</xsl:template>

</xsl:stylesheet>
```


Le Résultat



mode

- un attribut, permet de parcourir le document en appliquant d'autres templates
  - exemple, génération de la table des matières
  - `<xsl:apply-templates mode="toc"/>`
  - `<xsl:template match="product" mode="toc">`

select

- peut s'appliquer à un apply-templates
  - `<xsl:apply-templates select="image"/>`
- ce n'est pas un value-of : s'il y a un template, il s'exécute

xsl:text

- permet la génération de texte
  - très utile pour gérer précisement les espaces dans un document
  - `<xsl:text>#</xsl:text>`
    `<xsl:text> </xsl:text>`

xsl:if

- exécute le template en fonction d'un test
  - `<xsl:if test="not(position()=last() - 1)"> | </xsl:if>`

xsl:if

- Syntaxe
  <xsl:if expr="*script-expression*
  test="*pattern*" >
- si expr est true et que le pattern de test est vérifié, le contenu de xsl:if est placé dans la sortie.
- Pattern de test

xsl:if

- Exemple :

```
<xsl:for-each select="portfolio/stock">
<TR> <TD>
<xsl:value-of select="symbol"/>
<xsl:if test="@exchange[.='nasdaq']">*</xsl:if>
</TD> <TD>
<xsl:value-of select="name"/></TD> <TD>
<xsl:value-of select="price"/></TD> </TR>
</xsl:for-each>
```

Choix multiples

- pour les choix multiples, choose
-
```
<xsl:choose>
    <xsl:when test="Price/@currency='eur'">
       <xsl:value-of select="Price"/> EUR
    </xsl:when>
    <xsl:when test="Price/@currency='bef'">
       <xsl:value-of select="Price"/> BEF
    </xsl:when>
    <xsl:otherwise>
       <xsl:value-of select="Price"/>
       <xsl:value-of select="Price/@currency"/>
    </xsl:otherwise>
</xsl:choose>
```


xsl:choose

- Syntaxe :
  `<xsl:choose >`
-
```
<xsl:template match="order">
  <xsl:choose>
    <xsl:when test="total[. $lt$ 10]">
      <HR STYLE="color:red"/>
    </xsl:when>
    <xsl:when test="total[. $lt$ 20]">
      <HR STYLE="color:pink"/>
    </xsl:when>
    <xsl:otherwise>
      <BR/>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:apply-templates />
</xsl:template>
```

Fonctions

- fonctions comme text() et not()
- bien d'autres fonctions possibles
  - position() position dans la liste des noeuds
  - last() dernier noeud dans la liste courante
  - generate-id() identifiant unique de l'élément

Template avec Nom

```
<xsl:template name="format-date">
    <xsl:param name="date"/>
    <xsl:value-of
        select="substring($date,1,4)"/>
    <xsl:value-of
        select="substring($date,6,2)"/>
    <xsl:value-of
        select="substring($date,9,2)"/>
</xsl:template>
```

```
<xsl:call-template name="format-date">
    <xsl:with-param name="date"
        select="DeliverBy"/>
</xsl:call-template>
```

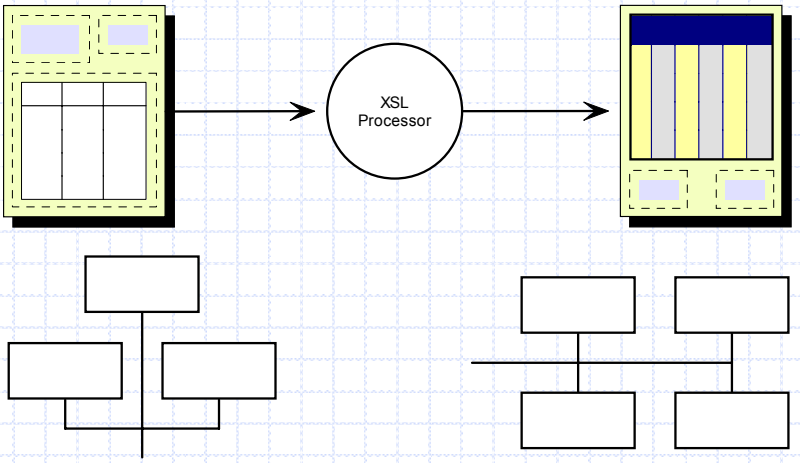
Notion d'axe

- child::para
  - para
- attribute::name
  - @name
- /child::doc/child::chapter[position()=5]/child::section[position()=2]
  - /doc/chapter[5]/section[2]

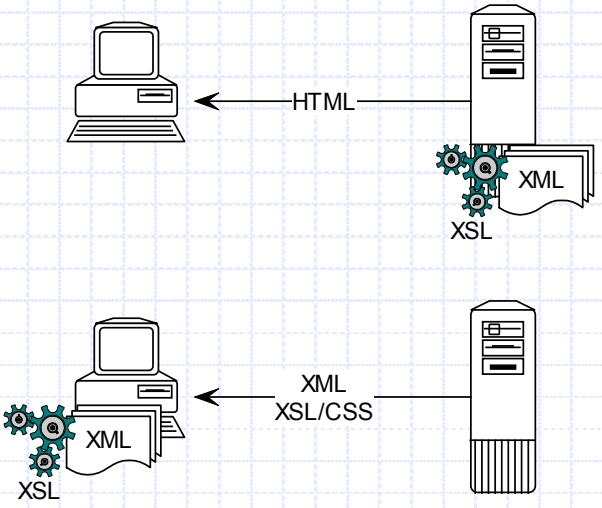
Axes

- child
- descendant
- parent
- following-sibling
- preceding-sibling
- following
- preceding
- attribute
- namespace
- self
- descendant-or-self
- ancestor-or-self

Conversion avec XSL



Client ou serveur



Document “éléments”

```
<?xml version="1.0"?>
<products>
   <product><name>XML Editor</name>
      <price>499.00</price></product>
   <product><name>DTD Editor</name>
      <price>199.00</price></product>
   <product><name>XML Book</name>
      <price>19.99</price></product>
   <product><name>XML Training</name>
      <price>699.00</price></product>
</products>
```


Même information

- l’information, y compris l’information de structure, peut être stockée à l’aide d’éléments ou d’attributs
  - pour preuve, on passe de l’un à l’autre via une transformation non-destructrice
    - c’est à dire une transformation réversible
  - une feuille de style crée un document avec attributs
    - et la feuille inverse existe

Transformation non-destructrice

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="products">
   <products>
      <xsl:apply-templates/>
   </products>
</xsl:template>
<xsl:template match="product">
   <product>
      <xsl:attribute name="name">
         <xsl:value-of select="name"/>
      </xsl:attribute>
      <xsl:attribute name="price">
         <xsl:value-of select="price"/>
      </xsl:attribute>
   </product>
</xsl:template>
```


Document “attributs”

```
<?xml version="1.0"?>
<products>
   <product name="XML Editor"
            price="499.00"/>
   <product name="DTD Editor"
            price="199.00"/>
   <product name="XML Book"
            price="19.99"/>
   <product name="XML Training"
            price="699.00"/>
</products>
```

Transformation inverse

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="products">
   <products>
      <xsl:apply-templates/>
   </products>
</xsl:template>
<xsl:template match="product">
   <product>
      <name>
         <xsl:value-of select="@name"/>
      </name>
      <price>
         <xsl:value-of select="@price"/>
      </price>
   </product>
</xsl:template>
</xsl:stylesheet>
```


En conclusion

- du point de vue de la structure, le choix est neutre, donc arbitraire
- il faut donc avoir recours à d'autres critères
  - du point de vue de la manipulation, les attributs offrent une "localité" de manipulation qui manque aux éléments
  - mais les éléments sont généralement perçus comme plus simples pour un débutant