Extensible Style sheet Language for Transformation
*X*SLT

Introduction

- CSS peut s'utiliser pour afficher des documents XML
  - Mais nous sommes limités sur la mise en page des informations
  - Le fichier XML n'est pas forcément structuré d'une façon lisible pour CSS
- XSLT : langage permettant de transformer un document XML en tout autre format

XSLT

- Composant d'un langage plus important XSL qui se décompose en 3 partie
  - XSLT
  - XPath
  - Et XSL-FO (Formatting Object) qui permet de générer des documents imprimables (PDF par exemple)
- Avec XSLT et des données XML
  - Il est possible de générer du HTML
  - Un autre fichier XML avec une autre structure
  - Un fichier texte brut
  - ...

Effectuer une transformation XSLT ?

- Besoin de 3 outils
  - Un fichier XML qui contient les données à transformer
  - Une feuille de style XSTL qui contient les règles de transformation
  - Un moteur XSLT qui va exécuter les instructions de la feuille de style
- Moteurs (ou processeur) XSLT disponibles
  - XT, Saxon, Xalan, Sablotron : moteur indépendant
  - MSXML : moteur XSLT d'Internet Explorer à partir du 5.5
  - Cocoon
  - Apache-Xalan

Qu'est ce que XSL ?

- XSLT : langage de transformation. Appliquer des règles de transformation à un document XML
  - Produire à partir du XML : HTML, WML, ...
  - Version 1.0 depuis le 16 Novembre 1999
  - Version 2.0 depuis le 23 Janvier 2007
- XSL-FO : langage de mise en page
  - Produire des documents non XML : PDF, RTF, Postscript
- XPath : utiliser par XSL pour référencer les éléments du document

Structure de document XSL

- Une feuille de style XSL doit correspondre à un document XML bien formé
- Chaque élément XSL doit utiliser l'espace de désignations xsl

  <xsl: >
- La première ligne d'un document XSLT est similaire à celle d'un document XML

  <?xml version="1.0" ?>
- La deuxième ligne est un espace de nom

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

Elément et attribut XSLT

- XSLT fonctionne sur des structures nommées **modèles** ou **templates**

```
<xsl:template match="prenom">Balise prenom identifiée !</xsl:template>
```

- Tout modèle comprend deux éléments importants
  - L'attribut match : Spécifie un modèle dans l'arbre source
  - Le contenu du modèle : Spécifie ce qui devra figurer dans l'arbre cible
- Possibilité d'ajouter des éléments particuliers pour effectuer différentes actions sur l'arbre source
  - <xsl:value-of> permet de récupérer des informations dans l'arbre source et les ajouter dans l'arbre cible

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

Processeur XSLT

- Nous pouvons utiliser 3 processeurs XSLT assez simple
  - un processeur en Java qui nécessite d'avoir installé J2EE 1.4 : XT
  - un processeur en PHP
  - un processeur inclus dans Apache : apache2-modxslt basé sur libxml2
- Pour illustrer les installations des différents processeurs, nous utiliserons les 2 fichiers suivants

Premier exemple

Fichier xsltEx1.xsl :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="html"/>
 <xsl:template match="/">
  <html>
   <body bgcolor="#FFFFFF">
    <h3>Mon premier exemple</h3>
    <xsl:value-of select="exemple/titre"/>
    <br/><xsl:value-of select="exemple/contenu"/>
   </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Fichier xsltEx1.xml :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsltEx1.xsl"?>
<exemple>
<titre>Bienvenue</titre>
<contenu>Bonjour  à tous</contenu>
</exemple>
```

Installation du moteur XT

- XT : Processeur XSLT écrit en Java
  - Disponible à l'adresse http://www.blnz.com/xt
  - Nécessite d'avoir installé J2EE SDK
  - Télécharger les sources de xt, les décompacter et exécuter soit xt.sh sous Linux ou xt.bat sous Windows (qui se trouve dans le répertoire demo)
- Fonctionnement :

```
xt fichier_source fichier_style fichier_resultat
```

- Pour l'exemple, 2 fichiers : xsltEx1.xml, xsltEx1.xsl
- Fichier résultat : xsltEx1.html

```
xt xsltEx1.xml xsltEx1.xsl xsltEx1.html
```

Installation du processeur sous PHP4

- Sous Php4
  - installer Apache, Sablotron, Expat, libapache2-mod-php4 et php4-xslt

Pour tester, créer le fichier `test-php4.php` suivant

```
<?php
        $xsl_file="xsltEx1.xsl";
        $xml_file="xsltEx1.xml";
        $xh = xslt_create();
        xslt_set_encoding($xh,"ISO-8859-1");
        /* Traitement du document */
        $result = xslt_process($xh, $xml_file, $xsl_file);
        if ($result) {
        echo $result;
        } else {
          echo "Désolé, ".$xml_file." n'a pu être transformé par ".$xsl_file;
          echo "<br \>La raison est " . xslt_error($xh) . " et ";
          echo " le code d'erreur est " . xslt_errno($xh);
        }
        xslt_free($xh); ?>
```

Installation du processeur sous PHP5

- Sous Php5
  - installer Apache, Sablotron, Expat, php-xslt

Pour tester, créer le fichier `test-php5.php` suivant

```
<?php
        $xsl_file="xsltEx1.xsl";
        $xml_file="xsltEx1.xml";
        $xsl = new XSLTProcessor();
        $xsl->importStyleSheet(DOMDocument::load($xsl_filename));
        echo $xsl->transformToXML(DOMDocument::load($xml_filename));
?>
```

Installation du module xslt sous Apache2

- Installer Apache2 et libapache2-modxslt
- Faire un lien dans `/etc/apache2/mods-enabled/` vers

  `/etc/apache2/mods-available/modxslt.load`
- Ajouter la ligne `AddOutputFilter mod-xslt xml` dans le fichier de configuration d'Apache2 (`/etc/apache2/apache2.conf`)
- **ATTENTION** : avec cette méthode, tout fichier xml sera interprété à la volée grâce à un fichier xsl qui devra exister
  - S'il la ligne

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

    n'existe pas dans le fichier XML, Apache retourne une erreur interne
- Plus d'info à l'adresse `http://www.mod-xslt2.com/`

Utilisation de Firefox version 2

- A partir de Firefox, il vous suffit d'ouvrir le fichier XML
  - Attention : le fichier XSL doit bien être indiqué dans le document XML
- Firefox affiche des fichiers Text et HTML donc
  - Si vous avez fait une méthode

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

    - Pas de problème
  - Si vous avez fait une méthode

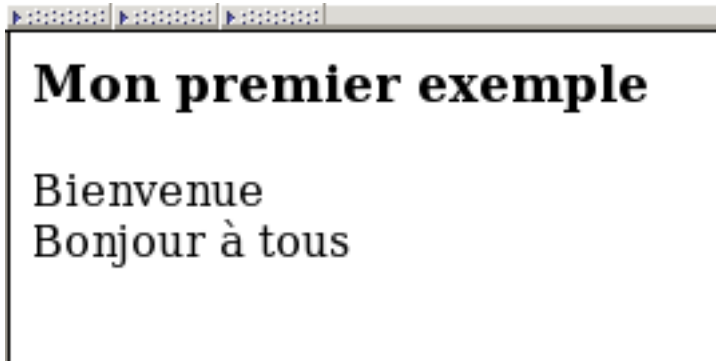
    ```
    <xsl:output method="html"/>
    ```

    - Votre code généré par la transformation XSL devra donc être un code HTML valide ! Sinon, vous aurez une page blanche
    - Insérer, à la main les balises <HTML><BODY>...</BODY></HTML>
- Pour voir le code généré, installer l'extension Web Developer et faire « View Generated Source » car l'affichage du Code Source dans le menu Firefox vous affichera uniquement le document XML d'origine.

Résultat obtenu

xsltEx1.html :

```
<html>
<body bgcolor="#FFFFFF">
<h3>Mon premier exemple</h3>Bienvenue<br>Bonjour &agrave;
tous</body>
</html>
```



XSLT

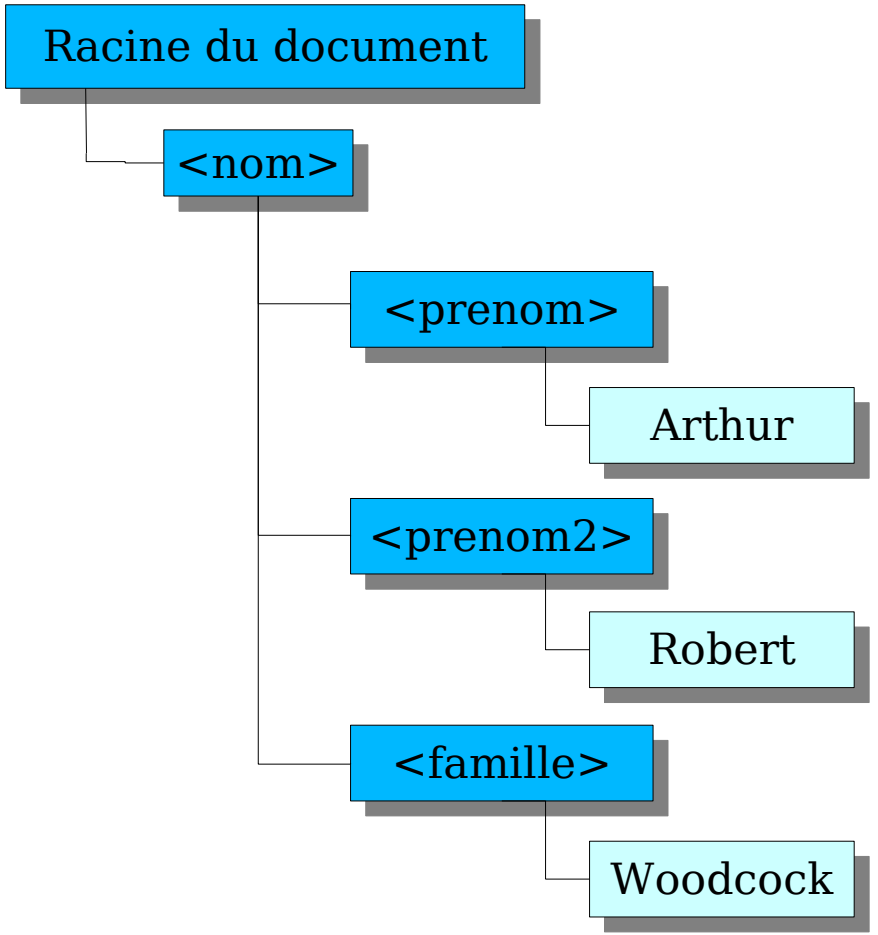
Le langage XPath

XPath

- Langage qui permet d'extraire les différents éléments d'un fichier XML en parcourant l'arbre
- XPatch utilise le concept de **noeud** pour définir les éléments, attributs, commentaires, instructions de traitement, ...
- **Racine du document ?**

```
<nom>
   <prenom>Arthur</prenom>
   <prenom2>Robert</prenom2>
   <famille>Woodcock</famille>
</nom>
```

  - En XPath, c'est /
  - Ainsi, /nom/prenom signifie sélectionner tous les éléments <prenom>, enfant des éléments <nom>, enfant de la racine



Localisation spécifique

- *Rechercher l'élément ville dont le texte est Reims ?*
  - Utilisation des crochets []
  - client[ville] identifie tout élément <client> possédant un élément <ville>
  - client[@numéro] identifie tout élément <client> possédant un attribut <numero>
  - client[ville = 'Reims'] identifie tout élément <client> possédant un élément enfant <ville> dont le texte est Reims
  - client[.='Reims'] identifie tout élément <client> dont la valeur est Reims
- Opérateur de descente récursive
  - Se fait à l'aide de // . Permet de rechercher un noeud de l'arbre, quelque soit sa place. Attention : ralenti la recherche !
    - *Exemple :* //nom recherche tous les éléments <nom> du document, quelque soit leur place. Ainsi, /client/nom et /client/produit/nom seront sélectionnés
- Que permet d'identifier //client[produit/@id='12'] ?
  - Tous les <client> ayant un élément <produit> dont l'attribut id est 12
- Si on veut sélectionner tous les <client> ayant comme enfant un élément <produit>, qui a lui-même des éléments enfants avec l'attribut id égal à 12
  - //client[produit/*/@id='12']

Exemple

- Exercez-vous à partir des 2 fichiers suivants en changeant l'expression en gras:

exemple1.xml :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="exemple1.xsl"?>
<clients>
        <client>
                <produit id="10"> Clou </produit>
                <ville> Reims </ville>
        </client>
        <client>
                <produit id="11"> Marteau </produit>
                <ville> Reims </ville>
        </client>
        <client>
                <produit>
                        <nom id="11"> Tournevis </nom>
                </produit>
                <ville> Amiens </ville>
        </client>
        <client>
                <produit id="12">
                        <nom id="11"> Vis </nom>
                </produit>
        </client>
</clients>
```

exemple1.xsl :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
 <xsl:output method="html"/>

 <xsl:template match="/">
        <html><body>
  <xsl:for-each select="clients/client/produit[@id=11]">
   <xsl:value-of select="text()"/><br />
  </xsl:for-each>
        </body></html>
 </xsl:template>

</xsl:stylesheet>
```

Les fonctions XPath

- XPath possède un certain nombre de fonctions
- Toutes les fonctions se terminent par ()
- Des paramètres peuvent éventuelles être placés entre les ()
- Plusieurs types de fonction :
  - Fonctions de noeud
  - Fonctions de position
  - Fonctions de booléennes
  - Fonctions numériques
  - Fonctions de chaînes

Fonctions de noeud

- name() : retourne nom du noeud contextuel

  <xsl:value-of select= "name()" />

- processing-instruction() : retourne toute instruction de traitement

  <xsl:template match="processing-instruction()"> ... </xsl:template>

- comment() : retourne les commentaires

  <xsl:template match="comment()"> ... </xsl:template>

  - Attention : les commentaires ne sont pas forcément toujours envoyer à l'application. Cela dépend du parseur XML !

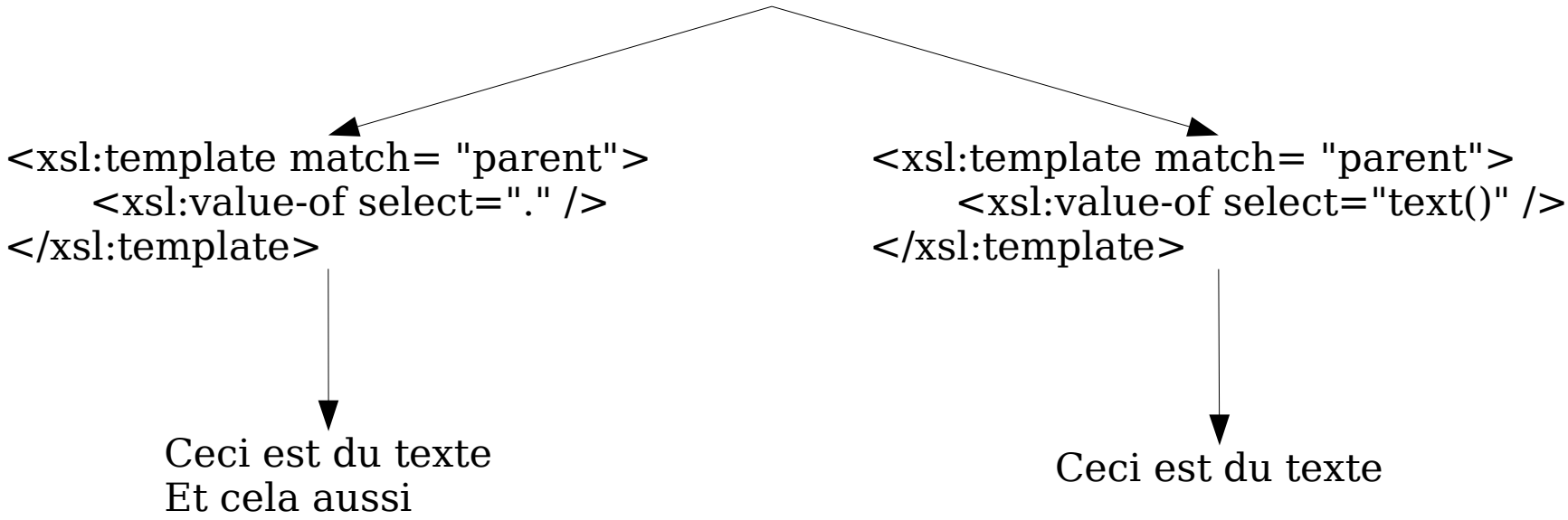
- text() : Retourne le contenu PCDATA d'un noeud

  <xsl:value-of select="text()" />

Fonctions de noeud

- Exemple d'utilisation de la fonction text() :

```
<parent>Ceci est du texte
    <enfant>Et cela aussi</enfant>
</parent>
```



Fonctions de position (1)

- position() : permet d'obtenir la position du noeud dans un ensemble de noeuds, selon l'ordre du document

```
<personnel>
    <personne>a</personne>
    <personne>b</personne>
</personnel>
```

Pour sélectionner le noeud 2

```
<xsl:template match="/personnel/personne[position()=2]">
```

Existence d'une écriture abrégée

```
<xsl:template match="/personnel/personne[2]">
```

Fonctions de position (2)

- last()
  - Retourne le dernier noeud d'un ensemble de noeud

```
<xsl:template match="/personnel/personne[position() = last()]">
```

- count()
  - Retourne le nombre de noeuds d'un ensemble de noeuds

```
<xsl:value-of select="count(personne)"/>
```

Fonctions numériques

- number() : convertit un texte PCDATA en valeur numérique
  - <element>256</element> contient la chaîne de caractère 2,5 et 6. La fonction number(element) permet de considérer cette chaîne comme un nombre
- sum() : additionne les valeurs numériques d'un ensemble de noeuds

```
<noeuds>
    <noeud>1</noeud>
    <noeud>2</noeud>
    <noeud>3</noeud>
</noeuds>
```

  - <xsl:value-of select="sum(/noeuds/noeud)"> retourne la somme de tous les éléments <noeud>

Fonctions booléennes

- boolean() : évalue une expression Xpath comme étant *true* ou *false*
  - Si l'expression est une valeur numérique : 0 correspond à *false* et toute autre valeur à *true*
  - Si la valeur est une chaîne, elle est *true* si sa longueur est supérieur à 0
  - Si la valeur est un ensemble de noeuds, elle est *true* lorsqu'elle n'est pas vide, et *false* dans le cas contraire
- not() : convertit en son contraire
- true() et false() : retourne *true* et *false*

Fonctions de chaînes (1)

- string() : convertit toute valeur en chaîne
- string-length() : retourne la longueur d'une chaîne
- concat() : concaténation de plusieurs chaînes
  - concat('ma chaîne', ' et ', 'une autre chaîne')
- contains() : indique si une chaîne en contient une autre
  - contains("une chaîne à tester", "une ch") retourne true
  - contains("une chaîne à tester","Une ch") retourne false car toutes les fonctions de chaînes sont sensibles à la casse
- starts-with : indique si une chaîne débute par une seconde chaîne
  - starts-with("Ceci est une chaîne","Ceci") retourne true
  - starts-with("Ceci est une chaîne","une ch") retourne false

Fonctions de chaînes (2)

- substring() extrait un nombre de caractères déterminé d'une chaîne
  - 1er paramètre : la chaîne dont doivent être extraits les caractères
  - 2ème paramètre : la position dans la chaîne où débute l'extraction
  - Facultatif : le nombre de caractères à extraire
- *Exemple :*
  - substring('Ceci est une chaîne quelconque',14) retourne « chaîne quelconque »
  - substring('Ceci est une chaîne quelconque',14,6) retourne « chaîne »
- substring-after() retourne tous les caractères après la première occurrence du caractère donné en 2ième paramètre
  - substring-after('Ceci est une chaîne','e') retourne « ci est une chaîne »
- substring-before()
  - substring-before('Ceci est une chaîne','s') retourne « Ceci e »

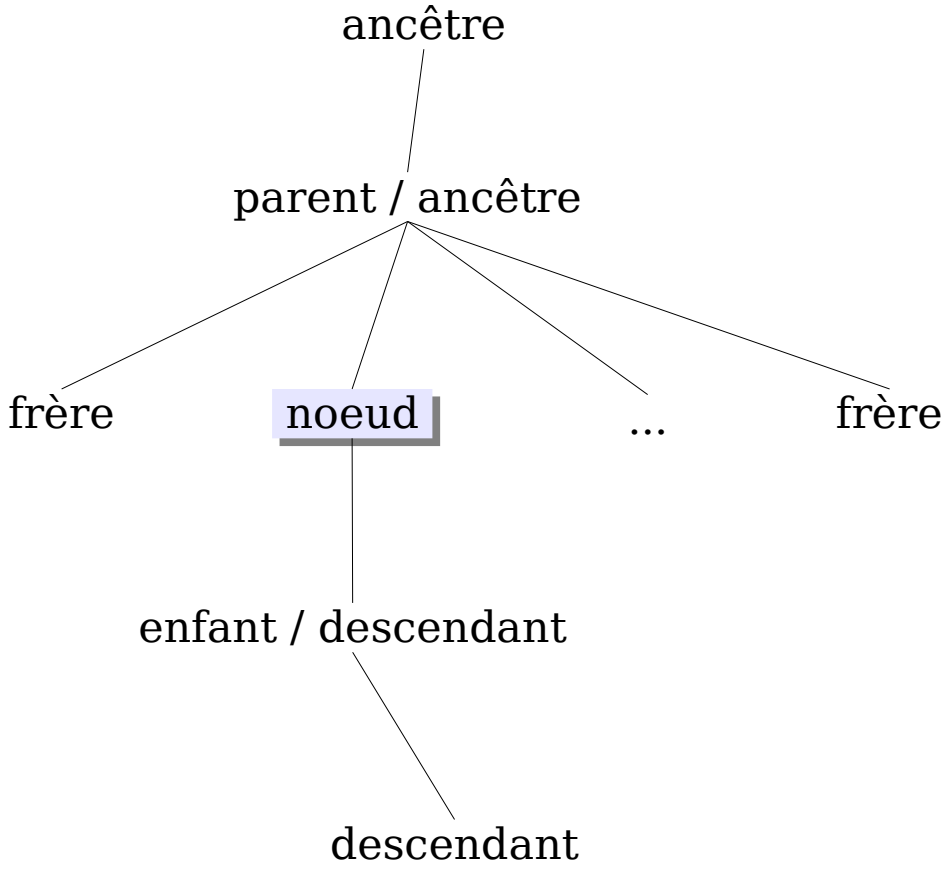
Fonctions de chaînes (3)

- translate() : permet de remplacer des caractères d'une chaîne de caractère
  - translate('John','ohn','OHN') retourne JOHN
    - Elle remplace le 'o' par 'O', 'h' par 'H' et 'n' par 'N'
- Fonctionnement de translate() :
  - Balayage caractère par caractère de la première chaîne
  - Recherche de ce caractère dans la deuxième et remplacement par le caractère placé au même index dans la troisième chaîne.
  - Si ce caractère n'existe pas dans la 2ème chaîne, il n'est pas remplacer
  - Si ce caractère existe dans la 2ème chaîne mais pas dans la 3ième, il est supprimer
- Exemple :
  - translate('+abc+','abc','def') retourne « +def+ »
  - translate('bonjour à tous',' ','+') retourne « bonjour+à+tous »
  - translate('bonjour+à+tous','b+','B') retourne « Bonjouràtous »

Noms d'axes

- Depuis un noeud, il est possible de se promener dans l'arbre complet à partir du noeud courant en spécifiant un noeud d'axe
- Descendre dans l'arbre
  - descendant spécifie tous les enfants du noeud courant
  - descendant-or-self contient le noeud courant ainsi que tous ses enfants, également noté //
- Remonter dans l'arbre
  - parent spécifie le parent du noeud courant
  - ancestor spécifie le parent, mais aussi ses grand-parents, ...
  - ancestor-or-self spécifie le noeud courant et ses « ancêtres »
- Les frères
  - following-sibling et preceding-sibling spécifient respectivement les noeuds frères suivant le noeud courant et les noeuds précédents
- Les attributs : obtenu soit par @, soit par attribute

Illustration



Exemple utilisé dans la suite

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="commande.xsl"?>
<commande>
 <client>
    <nom>Société A</nom>
    <adresse>3, rue des tourterelles</adresse>
    <ville>Reims</ville>
  <produit>
    <nom>Carte mère Asus</nom>
    <code>CMASUSA7V</code>
    <prixunit>40</prixunit>
    <quantite>1</quantite>
    <prixtotal>40</prixtotal>
  </produit>
  <produit>
    <nom>Processeur XP 2600+</nom>
    <code>XP26+</code>
    <prixunit>50</prixunit>
    <quantite>1</quantite>
    <prixtotal>50</prixtotal>
  </produit>
  <produit>
    <nom>Boitier Morex</nom>
    <code>BTMo54</code>
    <prixunit>30</prixunit>
    <quantite>1</quantite>
    <prixtotal>30</prixtotal>
  </produit>
 </client>
 <client>
    <nom>Société B</nom>
    <adresse>21 allée des bras casssés
    </adresse>
    <ville>Bricol Ville</ville>
  <produit>
    <nom>Carte Mère Abit</nom>
    <code>CMABITKT266</code>
    <prixunit>50</prixunit>
    <quantite>1</quantite>
    <prixtotal>50</prixtotal>
  </produit>
  <produit>
    <nom>Dvd Pioneer</nom>
    <code>Dvd103S</code>
    <prixunit>40</prixunit>
    <quantite>2</quantite>
    <prixtotal>80</prixtotal>
  </produit>
  <produit>
    <nom>Souris Logitech</nom>
    <code>SoLogiOpt</code>
    <prixunit>20</prixunit>
    <quantite>5</quantite>
    <prixtotal>100</prixtotal>
  </produit>
 </client>
</commande>
```

Exemple d'utilisation des noeuds d'axes

- Représenter l'arbre de l'exemple précédent (commande.xml)
- Donner ce que produit le code suivant sur l'exemple précédent

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/commande">
   <xsl:for-each select="child::client/*">
        <xsl:value-of select="name()"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="text()"/>
        <xsl:text>
        </xsl:text>
   </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Solution

nom Société A
adresse 3, rue des tourterelles
ville Reims
produit
produit
produit
nom Société B
adresse 21 allée des bras cassés
ville Bricol Ville
produit
produit
produit

Exercice 1

- Donner le code XSLT permettant d'afficher le texte de tous les éléments nom contenu dans le fichier commande.xml, c'est à dire la sortie suivante :

Société A

Carte mère Asus

Processeur XP 2600+

Boitier Morex

Société B

Carte Mère Abit

Dvd Pioneer

Souris Logitech

Solution

- Donner le code XSLT permettant d'afficher le texte de tous les éléments nom contenu dans le fichier commande.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>
    <xsl:template match="/commande">
        <xsl:for-each select="descendant::nom">
            <xsl:value-of select="text()"/>
            <xsl:text>
            </xsl:text>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
OU
<xsl:for-each select="client//nom">
        <xsl:value-of select="text()"/><br/>
 </xsl:for-each>
```