dessus sont de la même veine, et participent aussi à la mise en œuvre du modèle de traitement, tout en étant déjà moins indispensables.

Ce qu'il faut donc retenir, c'est que ce que nous appelons instruction de transformation, ce n'est pas une instruction effectuant en elle-même une transformation quelconque, mais une instruction intégrée au cœur du modèle de transformation.

Les autres instructions, que nous avons classées dans les catégories d'instructions de programmation ou de création, pourraient très bien être supprimées du langage XSLT sans que cela casse le modèle de traitement, qui conserverait toute la puissance de son principe. Ces instructions apportent des facilités complémentaires, et augmentent chacune à leur manière le champ d'application du langage, mais sans rien apporter de fondamentalement nouveau au modèle de traitement, qui à la limite, peut se contenter de l'instruction `xsl:template`, et des deux instructions `xsl:value-of` et `xsl:apply-templates` pour fonctionner.

Instruction xsl:template

Cette instruction a déjà été largement détaillée à l'occasion de la description de l'instruction `xsl:value-of` (voir *Instruction xsl:value-of*, page 102) : nous nous contenterons de rappeler les éléments essentiels et les variantes syntaxiques.

Syntaxe

xsl:template

```
<xsl:template match="... motif ..." />
    <!-- modèle de transformation -->
    ...
    <!-- fin du modèle de transformation -->
</xsl:template>
```

L'instruction `xsl:template` doit apparaître uniquement comme instruction de premier niveau.

Sémantique

Cette instruction, qui définit une règle, est au cœur du fonctionnement du langage XSLT ; on se reportera aux sections *Modèle de traitement*, page 87, *Motifs (patterns)*, page 91 et *Priorités entre règles*, page 100.

Instruction xsl:value-of

Cette instruction a déjà été largement détaillée (voir *Instruction xsl:value-of*, page 102) : nous nous contenterons d'en indiquer les éléments essentiels et les variantes syntaxiques.

Syntaxe

xsl:value-of

```
<xsl:value-of select="... chemin de localisation ..." />
```

L'instruction `xsl:value-of` ne doit pas apparaître en tant qu'instruction de premier niveau.

Le chemin de localisation fourni comme valeur de l'attribut `select` n'est pas limité à certaines formes comme c'est le cas pour le motif (voir *Syntaxe et contrainte pour un motif XSLT*, page 98) ; toute la puissance expressive du langage XPath est ici utilisable.

Règle XSLT typique

Une règle XSLT utilisant l'instruction `xsl:value-of` va typiquement avoir la forme :

```
<xsl:template match="... motif (pattern) ...">
    <!-- modèle de transformation -->
    ...
    mélange de texte et
    d'instructions XSLT de la forme :

        <xsl:value-of select="... chemin de localisation ..." />

    ...
    <!-- fin du modèle de transformation -->
</xsl:template>
```

Sémantique

Lors de l'instanciation du modèle, le motif de la règle est en concordance avec le nœud courant (le nœud en cours de traitement). C'est ce nœud qui fait office de nœud contexte dans l'évaluation du chemin de localisation fourni comme valeur de l'attribut `select`.

Comme son nom l'indique, l'instruction `<xsl:value-of select="..." />` est remplacée lors de l'instanciation du modèle par la valeur textuelle de ce qui est désigné par l'attribut `select`. Il s'agit donc de la valeur textuelle (i.e. sous forme de chaîne de caractères) d'un node-set, ou d'un booléen, ou d'un nombre, ou d'une chaîne de caractères, suivant la valeur renvoyée lors de l'évaluation du chemin de localisation.

Un booléen ou un nombre est converti en chaîne de caractères par appel de la fonction `string()`.

Un node-set comportant en général plusieurs nœuds source, sa valeur textuelle est définie comme étant celle du nœud source qui arrive en premier dans l'ordre de lecture du document. L'instruction est donc finalement remplacée par la valeur textuelle d'un nœud (un seul), notion qui a été définie à la section *Modèle arborescent d'un document XML vu par XPath*, page 30 et suivantes.

Exemple

Saison.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Saison>
    <Concert>
        <Organisation> Anacréon </Organisation>
        <Date>Samedi 9 octobre 1999 <Heure> 20H30 </Heure> </Date>
        <Lieu>Chapelle des Ursules</Lieu>
    </Concert>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mardi 19 novembre 1999 <Heure> 21H </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mercredi 20 novembre 1999 <Heure> 21H30 </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
</Saison>
```

Saison.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0">

    <xsl:output method='text' encoding='UTF-8'/>

    <xsl:template match='/'>
        Date Concert : <xsl:value-of select="Saison/Concert/Date"/>
        Date Théâtre : <xsl:value-of select="Saison/Théâtre[1]/Date"/>
        Date Théâtre : <xsl:value-of select="Saison/Théâtre[2]/Date"/>
    </xsl:template>

</xsl:stylesheet>
```

Appliquée au fichier `Saison.xml` , cette feuille de style produit le résultat suivant :

```
        Date Concert : Samedi 9 octobre 1999  20H30
        Date Théâtre : Mardi 19 novembre 1999  21H
        Date Théâtre : Mercredi 20 novembre 1999  21H30
```

La déclaration `<xsl:output method='text' encoding='UTF-8'/>` permet de spécifier que l'on veut générer un résultat qui sera un simple document texte (encodé en UTF-8), et non pas un document XML balisé comme il se doit. Si l'on supprimait cette déclaration, voici le résultat que l'on obtiendrait :

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

        Date Concert : Samedi 9 octobre 1999  20H30
```

```
        Date Théâtre : Mardi 19 novembre 1999  21H
        Date Théâtre : Mercredi 20 novembre 1999  21H30
```

soit à peu près la même chose, sauf le préambule de fichier XML généré automatiquement, en contradiction avec la suite du fichier qui n'a pas du tout l'air d'un fichier XML.

Variante syntaxique

On peut si l'on veut ajouter un attribut `disable-output-escaping` à l'élément `xsl:value-of`, comme ceci :

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

Cet attribut vaut `no` par défaut, ce qui veut dire que les caractères spéciaux pour XML (comme < ou >) sont sortis sous forme d'entités caractères (`<` ou `>`).

Instruction xsl:apply-templates

Cette instruction a déjà été largement détaillée (voir *Instruction xsl:apply-templates*, page 107) : nous nous contenterons d'en indiquer les éléments essentiels et les variantes syntaxiques.

Syntaxe

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

L'instruction `xsl:apply-templates` ne doit pas apparaître en tant qu'instruction de premier niveau.

Règle XSLT typique

Une règle XSLT utilisant l'instruction `xsl:apply-templates` aura souvent la forme :

```
<xsl:template match="... motif (pattern) ...">
    <!-- modèle de transformation -->
    ... texte ...
    <xsl:apply-templates />
    ... texte ...
    <!-- fin du modèle de transformation -->
</xsl:template>
```

Sémantique

Lors de l'instanciation du modèle, le motif de la règle est en concordance avec le nœud courant (le nœud en cours de traitement) ; l'instruction `<xsl:apply-templates/>` est remplacée par le fragment de document qui résulte du traitement de la liste des enfants du nœud courant. Le détail important, ici, est que cette liste, que nous avions déjà évoquée (voir *Instanciation d'un modèle de transformation relativement à un nœud courant et une*

liste courante, page 90) est constituée avec les éléments enfants du nœud courant, pris dans l'ordre de lecture du document source.

Exemple

Saison.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    version   = "1.0">

    <xsl:output method='text' encoding='UTF-8'/>

    <xsl:template match='/'>
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match='Saison'>
        Manifestations au programme
        <xsl:apply-templates/>
        Réservations 10 jours avant la date.
    </xsl:template>

    <xsl:template match='Concert'>
        Concert : <xsl:value-of select="."/>
    </xsl:template>

    <xsl:template match='Théâtre'>
        Théâtre : <xsl:value-of select="."/>
    </xsl:template>

</xsl:stylesheet>
```

Appliquée au même fichier `Saison.xml` que celui vu précédemment (voir *Exemple*, page 130), cette feuille de style produit le résultat suivant :

```
    Manifestations au programme


    Concert :
         Pygmalion
        Samedi 9 octobre 1999  20H30
        Chapelle des Ursules


    Théâtre :
         Masques et Lyres
        Mardi 19 novembre 1999  21H
        Salle des Cordeliers
```

```
Théâtre :
     Aristophane
    Mercredi 20 novembre 1999  21H30
    Salle des Cordeliers


Réservations 10 jours avant la date.
```

Variante syntaxique select="..."

On peut si l'on veut ajouter un attribut `select` à l'élément `apply-templates` , comme ceci :

```
<xsl:apply-templates select="... chemin de localisation ..." />
```

L'effet de l'attribut `select` est de modifier la constitution de la nouvelle liste de nœuds à traiter : en l'absence de `select="..."`, cette liste contient tous les enfants directs du nœud courant ; mais si l'attribut `select` est fourni, sa valeur (un chemin de localisation) est calculée, ce qui donne un node-set, et les éléments de ce node-set, pris dans l'ordre de lecture du document XML, vont alors constituer la nouvelle liste de nœuds à traiter.

En principe, le chemin de localisation que l'on fournit pour l'attribut `select` fait partie des descendants du nœud courant, même si la spécification du langage XSLT n'impose pas cette contrainte. En tous cas, c'est une bonne pratique que de se limiter à ce genre de node-set. Si l'on transgresse cette règle de bon sens, on risque d'introduire une récursion infinie dans le fonctionnement du processeur XSLT :

```
<xsl:template match='truc'>
    <xsl:apply-templates select="."/>
</xsl:template>
```

Ici, on impose au processeur XSLT une récursion infinie, puisque l'attribut `select` sélectionne le nœud courant lui-même, qui va donc indéfiniment concorder avec le motif de cette règle, indéfiniment réactivée.

Dans l'exemple que nous venons de voir (voir *Exemple*, page 132), on pourrait vouloir faire apparaître les pièces de théâtre avant les concerts ; cela pourrait se faire ainsi :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    version   = "1.0">

    <xsl:output method='text' encoding='UTF-8'/>

    <xsl:template match='/'>
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match='Saison'>
```

```
        Manifestations au programme
        <xsl:apply-templates select="Théâtre"/>
        <xsl:apply-templates select="Concert"/>
        Réservations 10 jours avant la date.
    </xsl:template>

    <xsl:template match='Concert'>
        Concert : <xsl:value-of select="."/>
    </xsl:template>

    <xsl:template match='Théâtre'>
        Théâtre : <xsl:value-of select="."/>
    </xsl:template>

</xsl:stylesheet>
```

Le résultat obtenu serait alors le suivant :

```
        Manifestations au programme

        Théâtre :
         Masques et Lyres
        Mardi 19 novembre 1999  21H
        Salle des Cordeliers

        Théâtre :
         Aristophane
        Mercredi 20 novembre 1999  21H30
        Salle des Cordeliers

        Concert :
         Pygmalion
        Samedi 9 octobre 1999  20H30
        Chapelle des Ursules

        Réservations 10 jours avant la date.
```

Variante syntaxique mode="..."

On peut si l'on veut ajouter un attribut `mode` à l'élément `apply-templates` , comme ceci :

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

L'emploi de cet attribut va de pair avec la définition de règles XSLT différentes applicables au même élément source, ces règles étant étiquetées par un nom de mode pour les différencier :

```
<xsl:template match='...' mode="mode1">
    ...
</xsl:template>

<xsl:template match='... la même chose ...' mode="mode2">
```

```
    ...
</xsl:template>
```

Lors de la définition d'un modèle de transformation, on peut utiliser l'instruction `xsl:apply-templates` en précisant le mode choisi, ce qui aura pour effet de sélectionner, parmi les différentes règles également applicables, celle dont le mode est égal au mode choisi :

```
<xsl:apply-templates mode="mode1" />
```

La conséquence est qu'un même élément peut être traité plusieurs fois, par des règles différentes, une par mode.

C'est ce qui distingue la notion de mode de celle de priorité :

- Avec la notion de mode, on introduit volontairement des ambiguïtés potentielles en écrivant plusieurs règles simultanément éligibles (le plus souvent, elles ont le même motif, mais ce n'est pas obligatoire). Ces règles simultanément éligibles sont associées chacune à des modes différents afin de pouvoir choisir la bonne par l'intermédiaire d'une instruction `<xsl:apply-templates mode="..."/>`, qui spécifie le mode adéquat en fonction de l'instruction en cours.
- Avec la notion de priorité, on cherche au contraire à éliminer toute ambiguïté, en affectant une fois pour toutes des priorités différentes aux règles qui pourraient éventuellement être simultanément éligibles : à l'exécution, si l'ambiguïté se présente, c'est toujours la même règle qui est choisie (celle de plus haute priorité), et toujours les mêmes qui sont écartées.

En résumé, avec la notion de mode, les différents choix possibles restent ouverts jusqu'au dernier moment, alors qu'avec celle de priorité, on ferme tout dès le départ.

Exemple

Nous conservons le même exemple de fichier XML à traiter ; mais cette fois, imaginons que le texte à produire soit destiné à un service municipal qui a notamment en charge de prévoir le chauffage des salles utilisées. On veut un texte qui puisse être intégré dans une note de service qui annonce les manifestations à venir, et qui récapitule à la fin les directives de chauffage.

Le problème ici est qu'un même élément (par exemple `<Lieu>`) devra être traité deux fois : une fois en tant que donnée d'une manifestation, et une fois en tant que donnée d'une directive de chauffage.

La solution est de définir deux modes de traitement : un mode « annonce » et un mode « logistique ». Le programme XSLT prend la forme suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    version   = "1.0">

    <xsl:output method='text' encoding='UTF-8'/>
```

```
    <xsl:template match='/'>
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match='Saison'>
        Manifestations à venir
        <xsl:apply-templates select="Théâtre" mode="annonce"/>
        <xsl:apply-templates select="Concert" mode="annonce"/>
        Chauffage
        <xsl:apply-templates select="Théâtre" mode="logistique"/>
        <xsl:apply-templates select="Concert" mode="logistique"/>
    </xsl:template>

    <xsl:template match='Concert' mode="annonce">
        Concert : <xsl:value-of select="."/>
    </xsl:template>

    <xsl:template match='Théâtre'  mode="annonce">
        Théâtre : <xsl:value-of select="."/>
    </xsl:template>

    <xsl:template match='Concert' mode="logistique">
        le <xsl:value-of select="Date"/>, <xsl:value-of select="Lieu"/>
    </xsl:template>

    <xsl:template match='Théâtre'  mode="logistique">
        le <xsl:value-of select="Date"/>, <xsl:value-of select="Lieu"/>
    </xsl:template>

    <xsl:template match='Organisation'  mode="logistique">
    </xsl:template>

</xsl:stylesheet>
```

Le résultat obtenu est alors le suivant :

```
        Manifestations à venir

        Théâtre :
         Masques et Lyres
        Mardi 19 novembre 1999  21H
        Salle des Cordeliers

        Théâtre :
         Aristophane
        Mercredi 20 novembre 1999  21H30
        Salle des Cordeliers

        Concert :
         Pygmalion
```

```
Samedi 9 octobre 1999  20H30
Chapelle des Ursules

Chauffage

le Mardi 19 novembre 1999  21H  , Salle des Cordeliers
le Mercredi 20 novembre 1999  21H30  , Salle des Cordeliers
le Samedi 9 octobre 1999  20H30  , Chapelle des Ursules
```

Instruction xsl:for-each

Cette instruction est la cousine de `xsl:apply-templates`, en ce sens que `xsl:apply-templates` et `xsl:for-each` sont les deux seules instructions du langage qui, lors de l'instanciation du modèle qui les héberge, provoquent la création d'une nouvelle liste de nœuds, traitée récursivement (voir *Instanciation d'un modèle de transformation relativement à un nœud courant et une liste courante*, page 90). Bien sûr, les effets de ces deux instructions sont différents, mais néanmoins, elles déclenchent des mécanismes assez semblables.

Bande-annonce

Saison.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Saison>
    <Concert>
        <Organisation> Anacréon </Organisation>
        <Date>Samedi 9 octobre 1999 <Heure> 20H30 </Heure> </Date>
        <Lieu>Chapelle des Ursules</Lieu>
    </Concert>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mardi 19 novembre 1999 <Heure> 21H </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mercredi 20 novembre 1999 <Heure> 21H30 </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
</Saison>
```

Saison.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    version   = "1.0">
    <xsl:output method='text' encoding='ISO-8859-1'/>
```

```
    <xsl:template match='Saison'>
        <xsl:for-each select="Théâtre">
            Date Théâtre : <xsl:value-of select="Date"/>
        </xsl:for-each>
    </xsl:template>

</xsl:stylesheet>
```

Résultat

```
            Date Théâtre : Mardi 19 novembre 1999  21H
            Date Théâtre : Mercredi 20 novembre 1999  21H30
```

Syntaxe

```
<xsl:for-each select="... chemin de localisation ...">
    ...
</xsl:for-each>
```

L'instruction `xsl:for-each` ne doit pas apparaître en tant qu'instruction de premier niveau.

Règle XSLT typique

Une règle XSLT utilisant l'instruction `xsl:for-each` sera souvent employée comme ceci :

```
<xsl:template match="... motif (pattern) ...">
    <!-- modèle de transformation englobant -->
    ... texte ou instructions XSLT ...
    <xsl:for-each select="...">
        <!-- modèle de transformation propre au for-each -->
         ... texte ou instructions XSLT ...
        <!-- fin du modèle de transformation -->
    </xsl:for-each>
    ... texte ou instructions XSLT ...
    <!-- fin du modèle de transformation englobant -->
 </xsl:template>
```

Sémantique

Remarque

Il n'est peut-être pas inutile de préciser tout d'abord que l'instruction `<xsl:for-each>` **n'est pas** une instruction pour effectuer une boucle, et n'a donc rien à voir avec la boucle `for(..;..;..)` que l'on trouve en C ou en Java. En particulier, la notion de compteur qui s'incrémente est une notion très étrangère à XSLT : XSLT est un langage qui ne permet pas de faire évoluer la valeur d'une variable ; rappelons que XSLT s'apparente aux langages fonctionnels (comme ML ou Caml) lorsqu'il s'agit de faire de l'algorithmique.

Lors de l'instanciation du modèle englobant, le motif de la règle est en concordance avec le nœud courant (le nœud en cours de traitement) ; l'instruction `<xsl:for-each>`, faisant partie de ce modèle de transformation englobant, est remplacée par le fragment de document qui résulte du traitement de la liste des nœuds sélectionnés par son attribut `select="..."`. L'instruction `<xsl:for-each>` est la seule, avec `<xsl:apply-templates>`, à induire la construction d'une nouvelle liste de nœuds, et à lancer un traitement sur cette liste. Comme dans la variante syntaxique `<xsl:apply-templates select="...">`, la liste des nœuds à traiter est établie d'après la valeur du chemin de localisation fourni dans l'attribut `select="..."`. En fait, la seule différence appréciable entre `<xsl:for-each select="...">` et `<xsl:apply-templates select="...">` est que pour `xsl:for-each`, la règle à appliquer à chaque nœud de la liste n'est pas recherchée parmi l'ensemble des règles du programme XSLT, comme dans le cas de `<xsl:apply-templates select="...">`, mais au contraire, le même modèle de transformation est appliqué uniformément à chacun d'entre eux.

Le corps de l'instruction `<xsl:for-each>` (i.e. l'ensemble des éléments fils de l'élément `<xsl:for-each>`) constitue le modèle de transformation uniformément appliqué à chacun des nœuds sélectionnés.

Le fragment de document qui résulte du traitement de cette liste est tel que cela été défini dans le modèle de traitement : voir *Modèle de traitement*, page 87.

Exemple

Saison.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    version   = "1.0">
    <xsl:output method='text' encoding='ISO-8859-1'/>

    <xsl:template match='Saison'>
        <xsl:for-each select="Théâtre">
            Date Théâtre : <xsl:value-of select="Date"/>
        </xsl:for-each>
    </xsl:template>

</xsl:stylesheet>
```

Saison.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Saison>
    <Concert>
        <Organisation> Anacréon </Organisation>
        <Date>Samedi 9 octobre 1999 <Heure> 20H30 </Heure> </Date>
        <Lieu>Chapelle des Ursules</Lieu>
    </Concert>
```

```
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mardi 19 novembre 1999 <Heure> 21H </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mercredi 20 novembre 1999 <Heure> 21H30 </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
</Saison>
```

Résultat

```
            Date Théâtre : Mardi 19 novembre 1999  21H
            Date Théâtre : Mercredi 20 novembre 1999  21H30
```

Autre sémantique

L'interprétation naturelle de `<xsl:for-each>` est celle d'une instruction permettant la répétition d'une même transformation sur plusieurs éléments. Bien sûr, pour que ceci ait un sens, il faut qu'il y ait effectivement une structure régulière, répétitive d'éléments à traiter, et que ce fait soit connu au moment où l'on écrit le programme XSLT.

L'explication de la sémantique de cette instruction fait intervenir deux modèles de transformation, l'un englobant, l'autre étant le modèle de transformation propre au `<xsl:for-each>` :

```
<xsl:template match="... motif (pattern) ...">
    <!-- modèle de transformation englobant -->
    ... texte ou instructions XSLT ...
    <xsl:for-each select="...">
        <!-- modèle de transformation propre au for-each -->
         ... texte ou instructions XSLT ...
        <!-- fin du modèle de transformation -->
    </xsl:for-each>
    ... texte ou instructions XSLT ...
    <!-- fin du modèle de transformation englobant -->
 </xsl:template>
```

Mais on peut aussi considérer un seul modèle de transformation (ce n'est qu'une question de point de vue) :

```
<xsl:template match="... motif (pattern) ...">
    <!-- modèle de transformation -->
    ... texte ou instructions XSLT ...
    <xsl:for-each select="...">
         ... texte ou instructions XSLT ...
    </xsl:for-each>
    ... texte ou instructions XSLT ...
    <!-- fin du modèle de transformation -->
 </xsl:template>
```

La seule modification concerne les commentaires : rien n'est donc changé pour le processeur XSLT. Pourtant cette présentation éclaire différemment la sémantique du `<xsl:for-each>`.

Le modèle de transformation est ici vu comme un modèle unique, composé de trois parties :

```
<xsl:template match="... motif (pattern) ...">
    <!-- modèle de transformation -->
    <!-- première partie -->
    ... texte ou instructions XSLT ...
    <!-- fin première partie -->
    <xsl:for-each select="...">
        <!-- deuxième partie -->
         ... texte ou instructions XSLT ...
        <!-- fin deuxième partie -->
    </xsl:for-each>
    <!-- troisième partie -->
    ... texte ou instructions XSLT ...
    <!-- fin troisième partie -->
    <!-- fin du modèle de transformation -->
 </xsl:template>
```

Dès lors, le `<xsl:for-each>` apparaît comme une instruction qui change temporairement le nœud courant pendant l'instanciation du modèle de transformation, et ce point de vue peut être encore renforcé si l'on imagine que le `select="..."` du `<xsl:for-each>` ne sélectionne qu'un seul élément.

L'instruction `<xsl:for-each>` aurait aussi bien pu s'appeler `<xsl:change-current-node>`.

En effet, lors de l'instanciation des première et troisième parties du modèle de transformation, le motif de la règle est en concordance avec le nœud courant (le nœud en cours de traitement). L'instanciation de la deuxième partie se fait avec un autre nœud courant, celui sélectionné par l'attribut `select="..."` de l'instruction `<xsl:for-each>`.

Exemple

On met ici en œuvre un exemple qui reprend l'idée du modèle de transformation en trois partie, la deuxième donnant lieu à une instanciation relativement à un (ou plusieurs) nœud(s) courant(s) différent(s) du nœud courant en concordance avec le motif ; le fichier XML est le même : `Saison.xml` (voir *Exemple*, page 139)

Saison.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
    version   = "1.0">

    <xsl:output method='text' encoding='ISO-8859-1'/>
```

```
    <xsl:template match='Concert'>
        Après le concert
        <xsl:value-of select="Organisation"/> du <xsl:value-of select="Date"/>,
        il y aura encore les spectacles suivants :
        <xsl:for-each select="/Saison/Théâtre">
            Théâtre (<xsl:value-of select="Organisation"/>),
            le <xsl:value-of select="Date"/>
        </xsl:for-each>
        Rappel des salles :
        <xsl:value-of select="Lieu"/>
    </xsl:template>

    <xsl:template match='Organisation'>
    </xsl:template>

    <xsl:template match='Date'>
    </xsl:template>

    <xsl:template match='Heure'>
    </xsl:template>

    <xsl:template match='Lieu'>
        <xsl:value-of select="."/>
    </xsl:template>

</xsl:stylesheet>
```

La première règle, dont le motif concorde avec le nœud `<Concert>`, déclare un modèle de transformation en trois parties.

Première partie

```
Après le concert
<xsl:value-of select="Organisation"/> du <xsl:value-of select="Date"/>,
il y aura encore les spectacles suivants :
```

Deuxième partie

```
Théâtre (<xsl:value-of select="Organisation"/>),
le <xsl:value-of select="Date"/>
```

Troisième partie

```
Rappel des salles :
<xsl:value-of select="Lieu"/>
```

La première et la troisième seront instanciées relativement au nœud courant `<Concert>`, alors que la deuxième sera instanciée plusieurs fois, relativement à différents nœuds courants (successivement tous les éléments `<Théâtre>` enfants de la racine `<Saison>`).

Le résultat est qu'à chaque fois, les instructions

```
<xsl:value-of select="Organisation"/>
```

et

```
<xsl:value-of select="Date"/>
```

de cette deuxième partie sont instanciées différemment en fonction du nœud courant actif. La troisième partie est instanciée sous la forme :

```
Rappel des salles :
Chapelle des Ursules
```

et c'est la règle

```
<xsl:template match='Lieu'>
        <xsl:value-of select="."/>
</xsl:template>
```

qui complète la liste des salles. Appliquée au fichier `Saison.xml`, cette feuille de style produit le résultat suivant :

```
        Après le concert  Anacréon  du Samedi 9 octobre 1999  20H30
        ,
        il y aura encore les spectacles suivants :

            Théâtre ( Masques et Lyres ), le Mardi 19 novembre 1999  21H

            Théâtre ( Masques et Lyres ), le Mercredi 20 novembre 1999  21H30

        Rappel des salles :
        Chapelle des Ursules



        Salle des Cordeliers




        Salle des Cordeliers
```

La mise en page, et notamment la répartition des sauts de ligne n'est pas extraordinaire, mais nous verrons plus loin comment régler ce genre de problème.

Cet exemple met en jeu les règles par défaut, puisque aucune règle n'est fournie pour les éléments `<Saison>` et `<Théâtre>`.

Pour l'élément `<Saison>` la règle

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

s'applique, et relance le traitement sur les nœuds enfants `<Concert>`, `<Théâtre>`, et `<Théâtre>`. Pour l'élément `<Concert>` une règle explicite est fournie. Pour les éléments `<Théâtre>` la règle par défaut

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

s'applique, et relance le traitement sur les nœuds enfants `<Organisation>`, `<Date>`, et `<Lieu>`, ces éléments étant pris en charge par des règles explicites.

Instruction xsl:sort

Bande-annonce

L'instruction `xsl:sort` est une instruction de tri qui ne s'emploie que comme complément à `xsl:apply-templates` ou `xsl:for-each` : elle sert à trier le node-set sélectionné par l'une de ces deux instructions. L'exemple ci-dessous montre un `xsl:sort` accompagnant un `xsl:for-each`.

CDtheque.xml

```
<?xml version="1.0" encoding="UCS-2" standalone="yes"?>

<CDthèque>

    <Compositeurs>

        <Compositeur>
            <nom> Couperin </nom>
            <prénom> Louis </prénom>
            <actifVers> 1670 </actifVers>
        </Compositeur>

        <Compositeur>
            <nom> Simpson </nom>
            <prénom> Thomas </prénom>
            <actifVers> 1610 </actifVers>
        </Compositeur>

        <Compositeur>
            <nom> Faugues </nom>
            <prénom> Guillaume </prénom>
            <actifVers> 1460 </actifVers>
        </Compositeur>

        <Compositeur>
            <nom> Aristophane </nom>
            <prénom> fils de Philippos d'Athènes </prénom>
            <actifVers> -410 </actifVers>
        </Compositeur>

        <Compositeur>
            <nom> Simpson </nom>
            <prénom> Christopher </prénom>
```

```
            <actifVers> 1640 </actifVers>
        </Compositeur>

    </Compositeurs>

</CDthèque>
```

CDtheque.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='text' encoding='ISO-8859-1' />

    <xsl:template match="Compositeurs">
        <xsl:for-each select="Compositeur">
            <xsl:sort select="nom"/>
            <xsl:value-of select="nom"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Résultat

```
      Aristophane  Couperin  Faugues  Simpson  Simpson
```

Variante de tri :

CDtheque.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='text' encoding='ISO-8859-1' />

    <xsl:template match="Compositeurs">
        <xsl:for-each select="Compositeur">
            <xsl:sort select="actifVers" data-type="number"/>
            <xsl:value-of select="nom"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Résultat

```
      Aristophane  Faugues  Simpson  Simpson  Couperin
```

Syntaxe

```
<xsl:sort/>
```

L'instruction `xsl:sort` ne doit pas apparaître en tant qu'instruction de premier niveau, et doit apparaître dans le modèle de transformation d'un `xsl:for-each` ou d'un `xsl:apply-templates`.

Règle XSLT typique

Une règle XSLT utilisant l'instruction `xsl:sort` sera souvent employée comme ceci :

```
<xsl:template match="... motif (pattern) ...">
    ...
    <xsl:for-each select="...">
        <xsl:sort/>
        <!-- modèle de transformation propre au for-each -->
         ... texte ou instructions XSLT ...
        <!-- fin du modèle de transformation -->
    </xsl:for-each>
    ...
</xsl:template>
```

ou encore comme ceci :

```
<xsl:template match="... motif (pattern) ...">
    ...
    <xsl:apply-templates>
        <xsl:sort/>
    </xsl:apply-templates>
    ...
</xsl:template>
```

Sémantique

L'instruction `<xsl:sort/>` ne s'emploie pas seule ; elle est en fait liée aux deux instructions `<xsl:for-each>` et `<xsl:apply-templates>`, et ne peut s'employer autrement que comme associée à l'une de ces deux instructions.

En l'absence d'une instruction `<xsl:sort/>`, les deux instructions `<xsl:for-each>` et `<xsl:apply-templates>` constituent une liste des éléments à traiter, basée sur l'ordre naturel de lecture du document XML.

L'instruction `<xsl:sort/>` intervient donc pour modifier l'ordre des éléments de cette liste : par défaut (c'est-à-dire en l'absence d'attributs propres à `<xsl:sort/>` permettant de spécifier les paramètres du tri à effectuer), les éléments de cette liste sont ordonnés suivant l'ordre lexicographique de la valeur textuelle de chaque élément.

L'instruction `<xsl:sort/>` est toujours vide ; elle peut juste être complétée par différents attributs que nous verrons plus loin.

Utilisée en association avec `<xsl:for-each>`, elle doit nécessairement se trouver placée avant le début du modèle de transformation inclus dans ce `<xsl:for-each>`, comme on le voit dans l'exemple ci-dessus.

Note

Employée dans un `xsl:for-each` la fonction `position()` renvoie le numéro d'ordre du nœud courant au sein du node-set sélectionné par l'attribut select du `xsl:for-each`. Si cette instruction `xsl:for-each` comporte une instruction `xsl:sort`, la numérotation considérée est celle du node-set réordonné par le tri.

Exemple

Nous reprenons l'exemple relatif à l'instruction `<xsl:apply-templates>` , tel qu'il est traité à la section *Instruction xsl:apply-templates*, page 131. Le fichiers XML est le même que celui déjà utilisé comme exemple (voir *Exemple*, page 132) :

Saison.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Saison>
    <Concert>
        <Organisation> Pygmalion </Organisation>
        <Date>Samedi 9 octobre 1999 <Heure> 20H30 </Heure> </Date>
        <Lieu>Chapelle des Ursules</Lieu>
    </Concert>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mardi 19 novembre 1999 <Heure> 21H </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
    <Théâtre>
        <Organisation> Aristophane </Organisation>
        <Date>Mercredi 20 novembre 1999 <Heure> 21H30 </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
</Saison>
```

Le fichier XSLT est lui aussi le même, à ceci près que l'on ajoute une instruction `<xsl:sort/>` à l'instruction `<xsl:apply-templates>` :

Saison.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method='text' encoding='UTF-8'/>

    <xsl:template match='/'>
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match='Saison'>
        Manifestations au programme
        <xsl:apply-templates> <xsl:sort/> </xsl:apply-templates>
        Réservations 10 jours avant la date.
    </xsl:template>

    <xsl:template match='Concert'>
        Concert : <xsl:value-of select="."/>
    </xsl:template>

    <xsl:template match='Théâtre'>
        Théâtre : <xsl:value-of select="."/>
```

```
    </xsl:template>

</xsl:stylesheet>
```

Résultat

```
        Manifestations au programme




        Théâtre :
         Aristophane
        Mercredi 20 novembre 1999  21H30
        Salle des Cordeliers

        Théâtre :
         Masques et Lyres
        Mardi 19 novembre 1999  21H
        Salle des Cordeliers

        Concert :
         Pygmalion
        Samedi 9 octobre 1999  20H30
        Chapelle des Ursules

        Réservations 10 jours avant la date.
```

Nous reprenons le déroulement du processus de traitement sur cet exemple (voir *Constitution d'une liste ne contenant que la racine*, page 108), qui ne change en rien, sauf une fois arrivé à l'étape représentée par la figure 3-21, que l'on rappelle ici (voir figure 4-1).

Sans l'instruction `<xsl:sort/>`, la constitution, par `<xsl:apply-templates>`, de la liste des éléments à traiter, se fait dans l'ordre de lecture du document.

Avec l'instruction `<xsl:sort/>`, la constitution, par `<xsl:apply-templates>`, de la liste des éléments à traiter, se fait dans l'ordre lexicographique de la valeur textuelle des éléments (voir figure 4-2).

La valeur textuelle des trois éléments `<Théâtre/>`, `<Théâtre/>`, `<Concert/>` est obtenue comme expliqué à la section *Nœud de type element*, page 31, et aux figures 3-23, 3-25, 3-27 de la section *Instanciation du modèle de transformation*, page 118. Ici, les mots *Pygmalion*, *Masques et Lyres*, *Aristophane* imposent leur ordre lexicographique à leurs trois éléments respectifs.

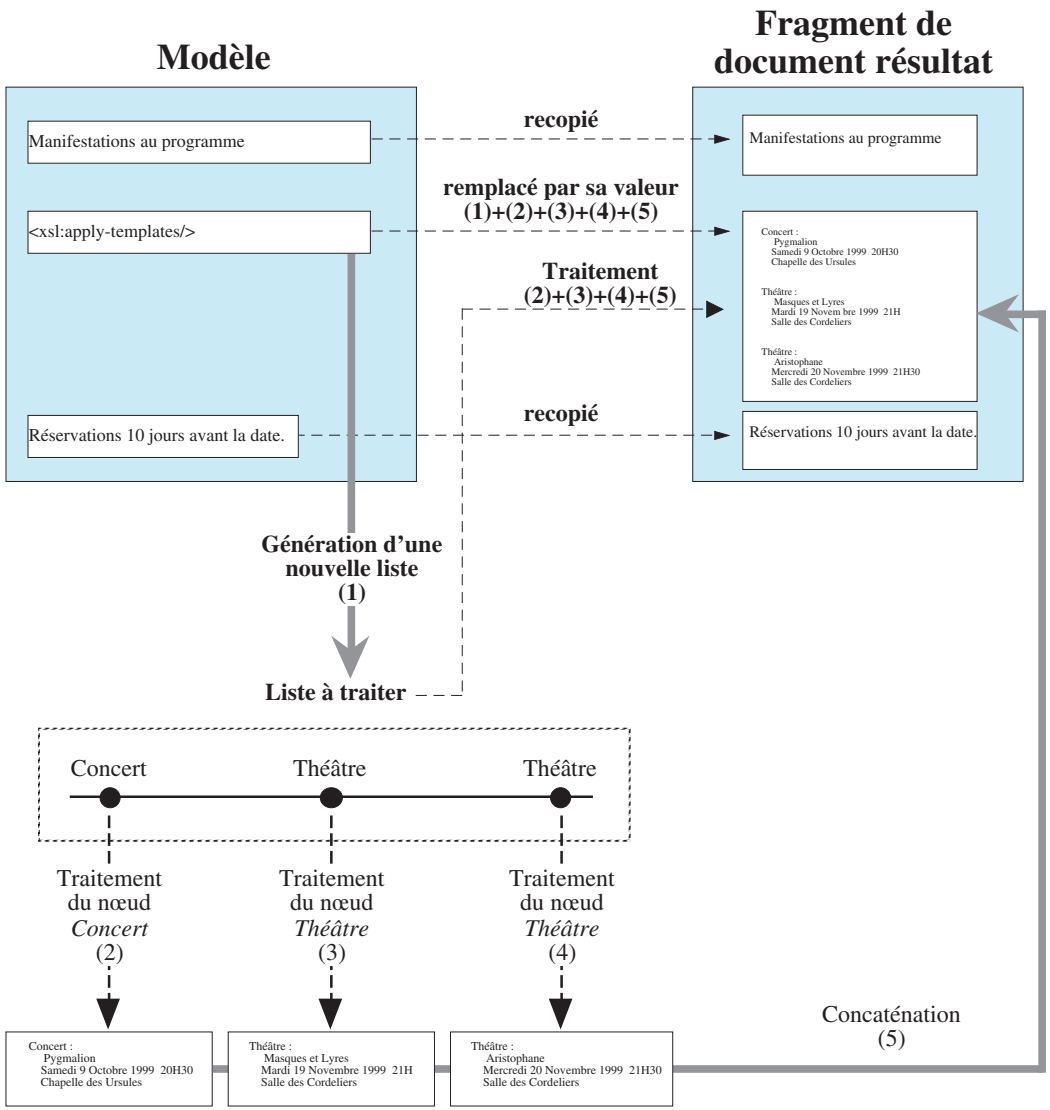

Figure 4-1

Instanciation du modèle de transformation (sans <xsl:sort/>).

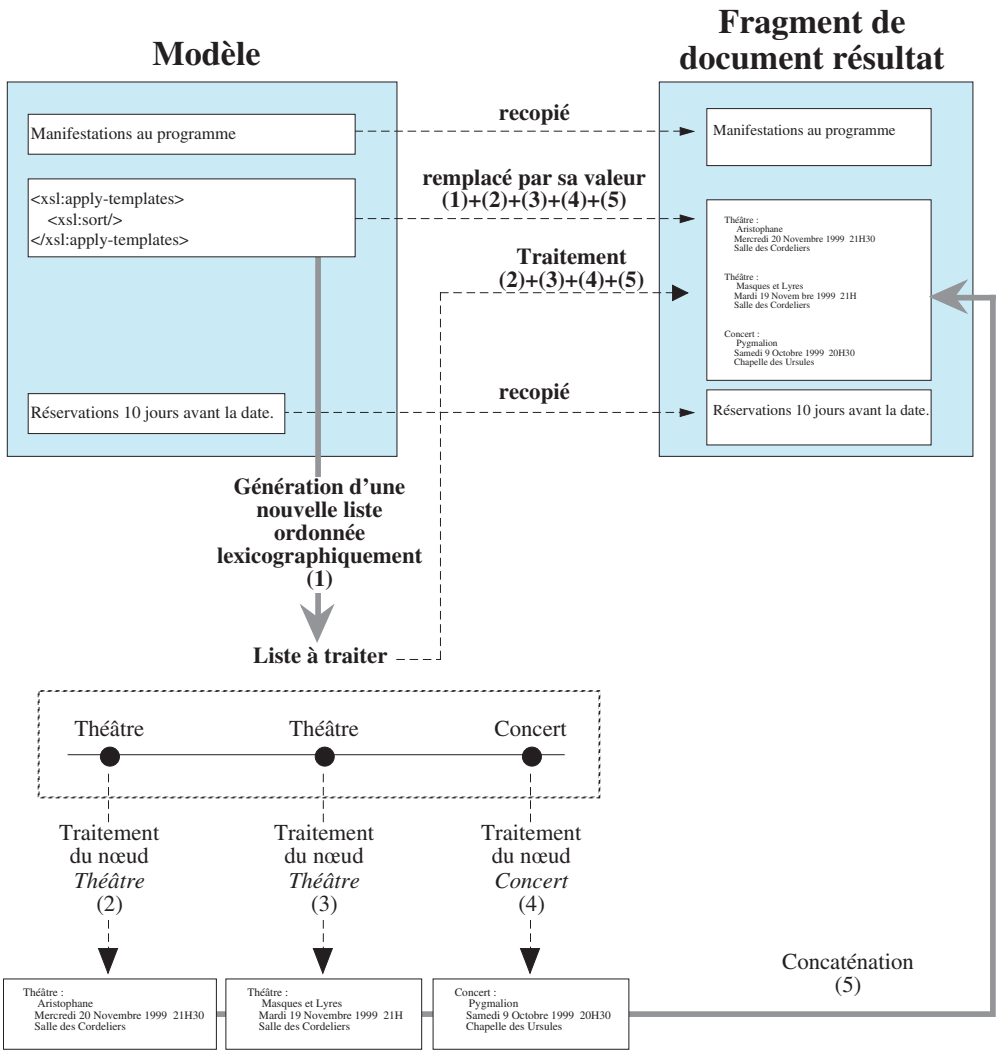

Figure 4-2

Instanciation du modèle de transformation (avec <xsl:sort/>).

Variantes syntaxiques

On peut, si l'on veut, ajouter à l'instruction `<xsl:sort/>` les attributs suivants :

```
<xsl:sort select="..."     order="..."
          case-order="..." lang="..."  data-type="..."/>
```

Ces attributs permettent de préciser les différents paramètres du tri, que nous allons maintenant passer en revue.

Attribut *select*

Signification

L'attribut `select` est essentiel, car il définit la clé de tri, c'est-à-dire la chaîne de caractères à extraire de l'élément en cours de placement, sur laquelle portera le tri.

Valeur possible

On peut fournir une expression XPath quelconque, qui est évaluée, comme il se doit, relativement à un nœud contexte et à une liste contexte.

Le nœud contexte est le nœud en cours de placement, et la liste contexte est la liste de nœuds (conservée dans son état originel), produite par l'instruction directement englobante, c'est-à-dire, suivant les cas, soit par l'instruction `<xsl:apply-templates/>`, soit par l'instruction `<xsl:for-each/>`.

Cette expression ne donne pas forcément une chaîne de caractères comme résultat (on obtient même un node-set dans le cas le plus général), mais s'il le faut, la fonction `string()` lui est appliquée, et la chaîne de caractères qui en résulte constitue alors la clé de tri.

> **Note**
>
> Il y a en fait deux listes en jeu : celle avant et celle après le tri. Le fait que la liste contexte soit celle conservée dans son état originel, donc avant le tri, rend possible l'utilisation de la fonction *position()* au sein de l'expression XPath.

Valeur par défaut

La valeur par défaut est égale à `string(.)`, qui n'est rien d'autre que la valeur textuelle du nœud courant.

Autres attributs

Les autres attributs sont des paramètres qui permettent de régler la façon dont le tri est effectué.

- `order`

  L'attribut `order` définit l'ordre du tri (ascendant ou descendant) ; il peut prendre l'une des deux valeurs `ascending` ou `descending`, et sa valeur par défaut est `ascending`.

Note

Techniquement, sont acceptées ici des valeurs d'attribut ordinaires (des chaînes de caractères, comme on en a l'habitude), mais aussi, et de façon assez exceptionnelle en XSLT, des *Attribute Value Template*, notion qui sera introduite beaucoup plus loin (voir *Descripteur de valeur différée d'attribut (Attribute Value Template)*, page 269).

Mais on pourra ignorer cette remarque en première lecture, puisque les valeurs d'attributs ordinaires sont acceptées.

- **`case-order`**

  L'attribut `case-order` définit la relation d'ordre entre les lettres minuscules et majuscules ; il peut prendre l'une des deux valeurs `upper-first` ou `lower-first`, et sa valeur par défaut dépend de la langue utilisée.

- **`lang`**

  L'attribut `lang` définit la langue utilisée, et par là même, les conventions de tri propres à cette langue ; sa valeur est un des codes de langue définit par XML, et sa valeur par défaut dépend de l'environnement de traitement.

- **`data-type`**

  L'attribut `data-type` définit la nature de la clé, afin de savoir comment comparer deux clés. Essentiellement, cette nature peut être de type texte (comparaison de chaînes de caractères) ou numérique (comparaison de nombres) ; sa valeur est donc soit `text`, soit `number`. Une autre valeur possible est un code spécial propre à une implémentation particulière de XSLT, qui offre cette valeur en extension, et indique naturellement ce qu'elle signifie. On peut penser qu'un code `date` serait très utile, car rien n'a été prévu en standard pour comparer des dates lors d'un tri, alors que ni `text` ni `number` ne conviennent.

  La valeur par défaut est `"text"`.

Exemple

On dispose d'une base de données de CDthèque, constituée d'informations sur les compositeurs, les œuvres, les enregistrements, etc.

Un extrait de cette base pourrait ressembler à ceci :

CDtheque.xml

```
<?xml version="1.0" encoding="UCS-2" standalone="yes"?>

<CDthèque>

    <Compositeurs>

        <Compositeur>
            <nom> Couperin </nom>
            <prénom> Louis </prénom>
            <actifVers> 1670 </actifVers>
```

```
        </Compositeur>

        <Compositeur>
            <nom> Simpson </nom>
            <prénom> Thomas </prénom>
            <actifVers> 1610 </actifVers>
        </Compositeur>

        <Compositeur>
            <nom> Faugues </nom>
            <prénom> Guillaume </prénom>
            <actifVers> 1460 </actifVers>
        </Compositeur>

        <Compositeur>
            <nom> Aristophane </nom>
            <prénom> fils de Philippos d'Athènes </prénom>
            <actifVers> -410 </actifVers>
        </Compositeur>

        <Compositeur>
            <nom> Simpson </nom>
            <prénom> Christopher </prénom>
            <actifVers> 1640 </actifVers>
        </Compositeur>

    </Compositeurs>

</CDthèque>
```

Pour éditer les compositeurs par ordre alphabétique de leur nom, on peut écrire la feuille de style suivante :

CDtheque.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='text' encoding='ISO-8859-1' />

    <xsl:template match="Compositeurs">
        <xsl:for-each select="Compositeur">
            <xsl:sort select="nom"/>
            <xsl:value-of select="nom"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Le résultat obtenu est le suivant :

```
      Aristophane  Couperin  Faugues  Simpson  Simpson
```

Le tri se fait ici avec toutes les valeurs par défaut des attributs `order`, `case-order`, `lang`, et `data-type`.

Supposons maintenant que nous voulions faire un tri par les dates, nous pouvons alors extraire le contenu de l'élément `<actifVers>` pour en faire une clé de tri. Afin de montrer la différence entre tri numérique et tri alphanumérique, nous avons ajouté une entrée « Aristophane » dans le fichier XML donné.

Note

Aristophane était un Grec, actif vers le IVe siècle avant JC, qui nous a laissé un peu de musique, gravée sur des dalles de marbre conservées à Delphes. Cette musique a réellement été enregistrée sur CD (plus récemment).

CDtheque.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='text' encoding='ISO-8859-1' />

    <xsl:template match="Compositeurs">
        <xsl:for-each select="Compositeur">
            <xsl:sort select="actifVers" data-type="number"/>
            <xsl:value-of select="nom"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Le résultat obtenu est le suivant :

```
Aristophane  Faugues  Simpson  Simpson  Couperin
```

On remarquera que ce qui est appelé ici tri sur les dates est en fait un tri sur des entiers, car il n'y a pas encore, en XSLT, de type de donnée normalisé correspondant à une date.

Si on avait omis de préciser l'attribut `data-type`, sa valeur par défaut (`text`) aurait été utilisée, ce qui aurait donné le résultat suivant :

```
Faugues  Simpson  Simpson  Couperin  Aristophane
```

Tri à clés multiples

Il est possible de préciser plusieurs clés de tri : quand deux éléments ont même valeur par la première clé de tri, on les départage avec la deuxième clé, et ainsi de suite. Dans notre exemple, on peut utiliser une deuxième clé de tri sur le prénom, pour affiner le tri sur le nom qui donne un doublon (*Simpson*).

Voyons ce que donne l'édition du nom + prénom sans clé de tri secondaire (remarquer la présence d'un « » ; « » *xsl:value-of*) :

CDtheque.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='text' encoding='ISO-8859-1' />

    <xsl:template match="Compositeurs">
        <xsl:for-each select="Compositeur">
            <xsl:sort select="nom"/>
            <xsl:value-of select="nom"/>
            <xsl:value-of select="prénom"/>;
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Le résultat obtenu est le suivant :

```
 Aristophane  fils de Philippos d'Athènes ;
     Couperin  Louis ;
     Faugues  Guillaume ;
     Simpson  Thomas ;
     Simpson  Christopher ;
```

Nous ne cherchons pas pour l'instant à expliquer la présentation obtenue, notamment la présence des sauts de ligne, qui ne se manifestaient pas dans les exemples précédents.

On observe que les deux *Simpson* ne sont pas classés dans l'ordre de leur prénom. On ajoute alors une deuxième clé de tri :

CDtheque.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='text' encoding='ISO-8859-1' />

    <xsl:template match="Compositeurs">
        <xsl:for-each select="Compositeur">
            <xsl:sort select="nom"/>
            <xsl:sort select="prénom"/>
            <xsl:value-of select="nom"/>
            <xsl:value-of select="prénom"/>;
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Le résultat obtenu est le suivant :

```
 Aristophane  fils de Philippos d'Athènes ;
     Couperin  Louis ;
     Faugues  Guillaume ;
     Simpson  Christopher ;
     Simpson  Thomas ;
```

Instruction xsl:copy-of

Cette instruction est la cousine de `<xsl:value-of/>` ; mieux : partout où `<xsl:value-of/>` est utilisée, on pourrait la remplacer par `<xsl:copy-of/>` sans que cela change le résultat obtenu. Bien sûr, la réciproque est fausse, et remplacer une occurrence quelconque de `<xsl:copy-of/>` par `<xsl:value-of/>` peut changer le résultat.

Bande-annonce

L'instruction `<xsl:copy-of select="..."/>` est instanciée sous la forme d'une copie conforme des éléments sélectionnés.

Concert.xml

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
<Concert>

    <Date>Jeudi 17 janvier 2002, 20H30</Date>
    <Lieu>Chapelle des Ursules</Lieu>

    <Interprètes>
        <Interprète>
            <Nom> Jonathan Dunford </Nom>
            <Instrument>Basse de viole</Instrument>
        </Interprète>

        <Interprète>
            <Nom> Silvia Abramowicz </Nom>
            <Instrument>Basse de viole</Instrument>
        </Interprète>
    </Interprètes>

</Concert>
```

Concert.xsl

```
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='xml' encoding='ISO-8859-1' indent='yes' />

    <xsl:template match="Interprètes">
        <Musiciens>
        <xsl:copy-of select="Interprète"/>
        </Musiciens>
    </xsl:template>

    <xsl:template match="text()"></xsl:template>

</xsl:stylesheet>
```

Résultat

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

      <Nom> Jonathan Dunford </Nom>

      <Instrument>Basse de viole</Instrument>

   </Interprète>
   <Interprète>

      <Nom> Silvia Abramowicz </Nom>

      <Instrument>Basse de viole</Instrument>

   </Interprète>
</Musiciens>
```

Syntaxe

```
<xsl:copy-of select="... chemin de localisation ..."/>
```

L'instruction `xsl:copy-of` ne doit pas apparaître en tant qu'instruction de premier niveau.

Règle XSLT typique

Une règle XSLT utilisant l'instruction `xsl:copy-of` sera souvent employée comme ceci :

```
<xsl:template match="... motif (pattern) ...">
    ...
    <xsl:copy-of select="..."/>
    ...
</xsl:template>
```

Sémantique

Lors de l'instanciation du modèle, le motif de la règle est en concordance avec le nœud courant (le nœud en cours de traitement). C'est ce nœud qui fait office de nœud contexte dans l'évaluation du chemin de localisation fourni comme valeur de l'attribut `select`.

L'instruction `<xsl:copy-of>`, comme son nom l'indique, effectue une copie du résultat de l'évaluation du chemin de localisation.

Si ce résultat est une chaîne de caractères, un booléen, ou un nombre, `xsl:copy-of` et `xsl:value-of` ont exactement le même effet (voir *Sémantique*, page 129).

Si le résultat est un node-set, l'effet est très différent : le node-set est sérialisé.

La sérialisation d'un node-set est une opération très simple (malgré son nom un peu impressionnant), qui a pour résultat un fragment de document constitué de la juxtaposition (ou concaténation) des fragments de documents obtenus en *sérialisant* successivement chacun des nœuds du node-set, pris dans l'ordre de lecture du document source (voir figure 4-3).

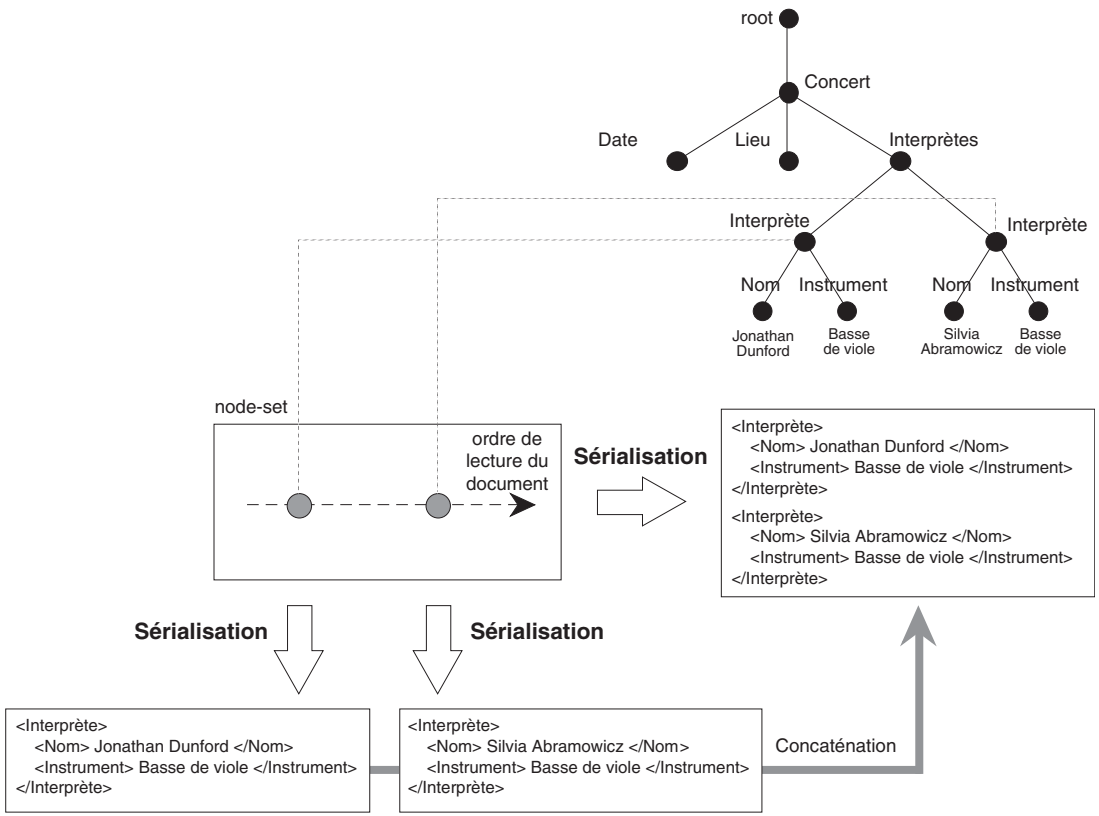

Figure 4-3

Sérialisation d'un node-set.

La sérialisation d'un nœud, quant à elle, est une opération encore plus simple qui consiste à voir ce nœud en tant que racine d'un sous arbre, et à sérialiser ce sous-arbre.

Nous avons déjà vu cela, (voir *Construction - sérialisation*, page 84) mais rappelons que la sérialisation d'un arbre (ou d'un sous-arbre), est l'opération inverse de sa construction à partir d'un document (ou d'un fragment de document) : la construction prend un (fragment de) document et créé le (sous) arbre équivalent ; la sérialisation redonne le (fragment de) document d'origine (voir figure 4-4).

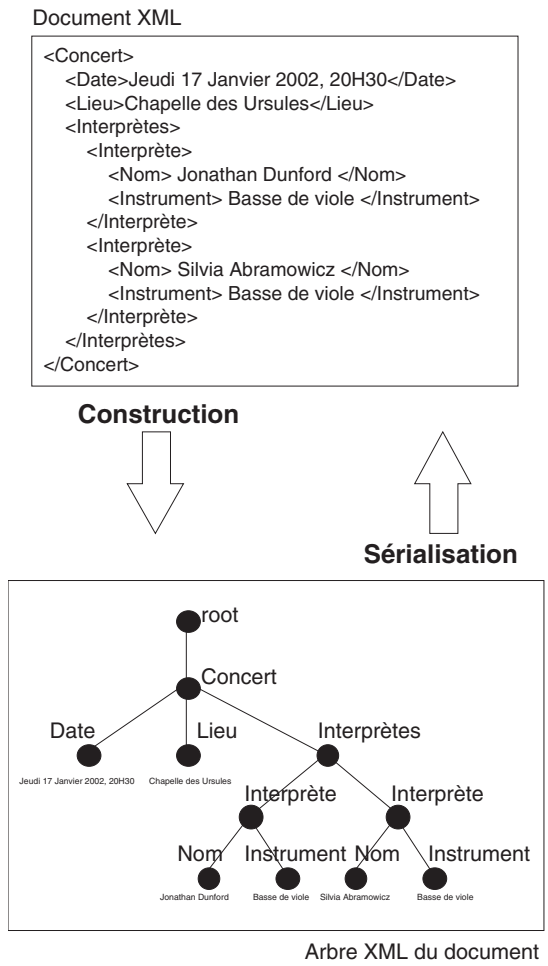

Figure 4-4
Sérialisation d'un arbre.

Remarque

Si l'on raisonne en terme d'arbre XML et non pas en terme de document, il est équivalent de dire que l'instruction `<xsl:copy-of>` provoque un duplicata récursif d'un ensemble de sous-arbres : chaque nœud du node-set est dupliqué de telle sorte qu'il soit récursivement identique au nœud original ; on obtient alors un duplicata certifié conforme de chacun des sous arbres ayant pour racines les nœuds originaux du node-set de départ.

Note

Deux nœuds récursivement identiques sont deux nœuds que rien ne permet de distinguer :ils ont les même attributs, les mêmes namespaces, et les mêmes nœuds enfants, qui sont eux-mêmes récursivement identiques deux à deux.

Exemple trivial

Cet exemple explicite la programmation de la sérialisation (montrée à la figure 4-3). Il est trivial en ce sens qu'il ne fait rien de très intéressant.

Concert.xml

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
<Concert>

    <Date>Jeudi 17 janvier 2002, 20H30</Date>
    <Lieu>Chapelle des Ursules</Lieu>

    <Interprètes>
        <Interprète>
            <Nom> Jonathan Dunford </Nom>
            <Instrument>Basse de viole</Instrument>
        </Interprète>

        <Interprète>
            <Nom> Silvia Abramowicz </Nom>
            <Instrument>Basse de viole</Instrument>
        </Interprète>
    </Interprètes>

</Concert>
```

Concert.xsl

```
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output  method='xml' encoding='ISO-8859-1' indent='yes' />

    <xsl:template match="Interprètes">
        <Musiciens>
        <xsl:copy-of select="Interprète"/>
        </Musiciens>
    </xsl:template>

    <xsl:template match="text()"></xsl:template>

</xsl:stylesheet>
```

Résultat

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

      <Nom> Jonathan Dunford </Nom>

      <Instrument>Basse de viole</Instrument>

   </Interprète>
   <Interprète>

      <Nom> Silvia Abramowicz </Nom>

      <Instrument>Basse de viole</Instrument>

   </Interprète>
</Musiciens>
```

Autre exemple

Etant donné que l'instruction `<xsl:copy-of>` sert à dupliquer un (ou des) sous arbre(s) XML, il semble assez évident qu'elle sera surtout utile lors de transformations XML vers XML. Il y a néanmoins un autre cas d'utilisation assez fréquent, dès lors que l'on manipule des variables. On en verra des exemples à la section *Pattern n° 2 – Fonction*, page 396.

Dans l'exemple qui suit, il s'agit d'expurger un document XML, pour en retirer de l'information :

BaseProduits.xml

```
<?xml version="1.0" encoding="UCS-2" standalone="yes"?>
<BaseProduits>

   <LesProduits>

      <Livre ref="vernes1" NoISBN="193335" gamme="roman" media="papier">
         <refOeuvres>
            <Ref valeur="20000lslm"/>
         </refOeuvres>
         <Prix valeur="40.5" monnaie="FF"></Prix>
         <Prix valeur="5" monnaie="£"/>
      </Livre>

      <Livre
         ref="boileaunarcejac1" NoISBN="533791" gamme="roman" media="papier">
         <refOeuvres>
            <Ref valeur="liatlc.bn"/>
         </refOeuvres>
```

```
         <Prix valeur="30" monnaie="FF"/>
         <Prix valeur="3" monnaie="£"/>
      </Livre>

      <Enregistrement
         ref="marais1" RefEditeur="LC000280" gamme="violedegambe" media="CD">
         <refOeuvres>
            <Ref valeur="marais.folies"/>
            <Ref valeur="marais.pieces1685"/>
         </refOeuvres>
         <Interprètes>
            <Interprète nom="Jonathan Dunford">
               <Role xml:lang="fr"> Basse de viole </Role>
               <Role xml:lang="en"> Bass Viol </Role>
            </Interprète>
            <Interprète nom="Sylvia Abramowicz">
               <Role xml:lang="fr"> Basse de viole </Role>
               <Role xml:lang="en"> Bass Viol </Role>
            </Interprète>
            <Interprète nom="Benjamin Perrot">
               <Role xml:lang="fr"> Théorbe et guitare baroque </Role>
               <Role xml:lang="en"> Theorbo and baroque guitar </Role>
            </Interprète>
            <Interprète nom="Freddy Eichelberger">
               <Role xml:lang="fr"> Clavecin </Role>
               <Role xml:lang="en"> Harpsichord </Role>
            </Interprète>
         </Interprètes>
         <Titre xml:lang="fr"> Les Folies d'Espagne et pièces inédites </Titre>
         <Titre xml:lang="en"> Spanish Folias and unedited music </Titre>
         <Prix valeur="140" monnaie="FF"/>
         <Prix valeur="13" monnaie="£"/>
      </Enregistrement>

      <Matériel
         ref="HarKar1" refConstructeur="XL-FZ158BK"
         gamme="lecteurCD" marque="HarKar">
         <refCaractéristiques>
            <Ref valeur="caracHarKar1"/>
         </refCaractéristiques>
         <Prix valeur="4500" monnaie="FF"/>
         <Prix valeur="400" monnaie="£"/>
      </Matériel>

   </LesProduits>

   <!-- ... etc : le fichier continue avec d'autres éléments -->

</BaseProduits>
```

Ce fichier XML rassemble des éléments d'une base de données de produits, et l'on veut écrire un programme XSLT permettant de créer un fichier des livres uniquement.

Rien de plus simple :

BaseProduits.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

   <xsl:output  method='xml' encoding='ISO-8859-1' indent='yes' />

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

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

   <xsl:template match="text()"></xsl:template>

</xsl:stylesheet>
```

Et voici ce qu'on obtient :

livres.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Livres>
<Livre ref="vernes1" NoISBN="193335" gamme="roman" media="papier">
         <refOeuvres>
            <Ref valeur="20000lslm"/>
         </refOeuvres>
         <Prix valeur="40.5" monnaie="FF"/>
         <Prix valeur="5" monnaie="£"/>
      </Livre>
<Livre ref="boileaunarcejac1" NoISBN="533791" gamme="roman" media="papier">
         <refOeuvres>
            <Ref valeur="liatlc.bn"/>
         </refOeuvres>
         <Prix valeur="30" monnaie="FF"/>
         <Prix valeur="3" monnaie="£"/>
      </Livre>
</Livres>
```

Dans l'instruction :

```
<xsl:output  method='xml' encoding='ISO-8859-1' indent='yes' />
```

l'attribut `indent` permet d'obtenir une indentation (pas très convaincante, il est vrai) ; si on l'avait omis, voici ce qu'on aurait obtenu :

livres.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Livres><Livre ref="vernes1" NoISBN="193335" gamme="roman" media="papier">
         <refOeuvres>
            <Ref valeur="20000lslm"/>
         </refOeuvres>
         <Prix valeur="40.5" monnaie="FF"/>
         <Prix valeur="5" monnaie="£"/>
      </Livre><Livre ref="boileaunarcejac1" NoISBN="533791" gamme="roman"
                     media="papier">
         <refOeuvres>
            <Ref valeur="liatlc.bn"/>
         </refOeuvres>
         <Prix valeur="30" monnaie="FF"/>
         <Prix valeur="3" monnaie="£"/>
      </Livre></Livres>
```

Au fait, pourquoi avoir intégré à ce programme XSLT la règle montrée ci-dessous ?

```
<xsl:template match="text()"></xsl:template>
```

Pour voir son utilité, commençons par voir ce que l'on obtiendrait si on la supprimait :

BaseProduits.xsl

```
<?xml version="1.0" encoding="UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

   <xsl:output  method='xml' encoding='ISO-8859-1' indent='yes' />

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

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

</xsl:stylesheet>
```

livres.xml

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

```
<Livre ref="vernes1" NoISBN="193335" gamme="roman" media="papier">
    <refOeuvres>
        <Ref valeur="20000lslm"/>
    </refOeuvres>
    <Prix valeur="40.5" monnaie="FF"/>
    <Prix valeur="5" monnaie="£"/>
</Livre>

<Livre ref="boileaunarcejac1" NoISBN="533791" gamme="roman" media="papier">
    <refOeuvres>
        <Ref valeur="liatlc.bn"/>
    </refOeuvres>
    <Prix valeur="30" monnaie="FF"/>
    <Prix valeur="3" monnaie="£"/>
</Livre>
```

```
            Basse de viole
            Bass Viol


            Basse de viole
            Bass Viol


            Théorbe et guitare baroque
            Theorbo and baroque guitar


            Clavecin
            Harpsichord


     Les Folies d'Espagne et pièces inédites
     Spanish Folias and unedited music
```

```
</Livres>
```

Ce fatras de textes (après la dernière balise fermante `</livre>`) et de lignes blanches intempestives fait réellement partie du résultat obtenu ; c'est la règle par défaut

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

qui en est responsable.