

XSLT: transformations en XML

Nous avons vu que les langages XML et XMLSchema permettent de structurer et d'organiser des données. XSLT (*eXtensible Stylesheet Language Transforms*) va nous permettre de définir des transformations pour *modifier* et/ou **présenter** les données des documents XML (cf Figure X.1). Par exemple, on peut

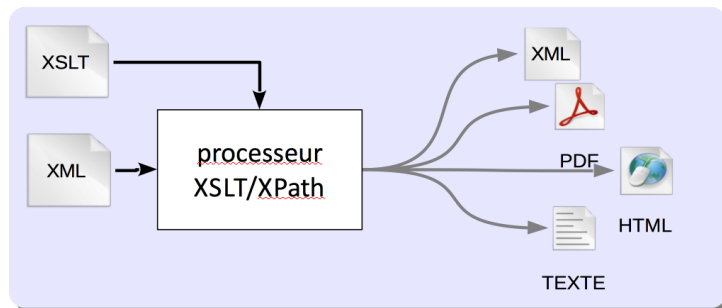


Figure X.1: XSLT permet de transformer un document XML en un document mis en page.

imaginer, comme illustré figure X.2 une transformation XSLT qui transforme un XML en un livre au format pdf.

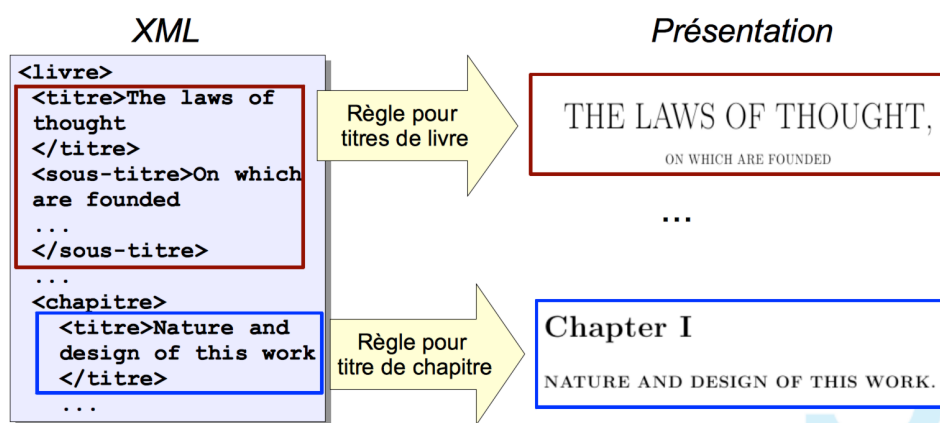


Figure X.2: Exemple de transformation d'un document XML en un livre au format pdf.

XSLT fait partie de l'ensemble XSL (*eXtensible Stylesheet Language*). XSL est composé d'XPath, XSLT et XSL-FO (*Formating Objects* XSL-FO donne la définition de la sémantique de présentation utilisée principalement dans les documents pdf).

XSLT ne permet pas vraiment d'écrire des algorithmes, mais plutôt de spécifier des transformations à appliquer. Il est basé sur un paradigme récursif.



XSLT est beaucoup plus complet que ce que nous allons voir ici.

1 Principes de fonctionnement

Pour transformer un document XML (i.e. une instance XML) en un autre document, XSLT le *transforme* en arbre d'instance (le même arbre que nous avons vu au chapitre III). Il commence par traiter la racine du document puis applique des transformations récursives nœud par nœud. Chaque transformation (aussi appelée *template*) s'appuie sur 3 mécanismes pour écrire le document de sortie:

- *push* permet au processeur d'écrire directement dans le fichier de sortie
- *pull* permet au processeur d'utiliser le fichier d'entrée (XML) pour écrire dans le fichier de sortie
- *navigation* permet au processeur de naviguer dans l'arbre du document d'entrée (XML)

1.a Anatomie d'un document XSLT

XSLT est un langage XML dont la racine est l'élément **stylesheet** (qui contient autant d'attributs **xmlns** que de vocabulaires utilisés, tous préfixés) et le premier sous-élément est **output** (qui permet de préciser le type du fichier de sortie).

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" ...
3     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4     xmlns:ns1="http://www....namespace1...."
5     xmlns:ns2="http://www....namespace2...."
6     .....>
7     <xsl:output method="xml"/> <!-- ou html, ou text, ou autre... -->
8     <!-- Modules de transformation -->
9 </xsl:stylesheet>

```

Dans une feuille XSLT, après la racine, et au même niveaux les uns des autres (i.e. fils directs de la racine) se trouvent les modules. Un module, aussi appelé *template* (modèle) est la description de la transformation à appliquer à un nœud donné de l'entrée XML. Le nombre de modules/*templates* n'est pas limité.

La définition d'un *template* à appliquer sur des cibles d'un document XML s'écrit de la manière suivante:

```

1 <xsl:template match="cible-xpath">
2     ...
3     <!-- code xsl -->
4     ...
5 </xsl:template>

```

Le code défini à l'intérieur du *template* va être appliqué à tous les nœuds du document XML d'entrée qui seront sélectionnés par le chemin *cible-xpath*.

Le premier *template* à être appliqué sur un document doit s'appliquer à sa racine. C'est pourquoi, pour démarrer la transformation, le processeur applique le *template* suivant:

```

1 <xsl:template match="/">
2     <!-- point d'entrée de la transformation -->
3     ...
4 </xsl:template>

```

ici, la *cible-xpath* est / qui désigne la racine, donc l'ensemble du document.

1.b Mécanisme Général

Le document XML d'entrée est traité récursivement à partir de la racine (parcours en profondeur d'abord, comme le sens de lecture du texte). On commence par sélectionner la racine (/), puis le processeur XSLT recherche une règle de transformation pour chaque nœud sélectionné. Lorsqu'un nœud est examiné, le processeur XSLT

- recherche le *template* correspondant au nœud examiné (autrement dit, la règle de transformation qui *match* le nom du nœud).
- si un *template* correspond
 - le nœud examiné devient le nœud contexte (i.e. le nœud courant)
 - le *template* sélectionné est "exécuté" (utilisé comme modèle de transformation pour le nœud contexte).
- sinon XSLT applique le traitement par défaut, c'est-à-dire recopie la donnée texte d'entrée (i.e. le texte contenu dans les éléments) vers la sortie.

Lorsque le *template* est terminé pour le nœud courant, le processeur passe au nœud suivant et ainsi de suite jusqu'à ce que le document XML ait été complètement exploré.

Note : Si aucun *template* n'est défini dans la feuille de style, la transformation XSLT va enlever toutes les balises d'un document XML et recopier le texte dans le document de sortie.

Dans les *templates*, on peut activer les 3 mécanismes *push*, *pull* et *navigation*.

1.c Exemple

Dans la suite du chapitre, on considère l'exemple XML simpliste présenté figure X.3.

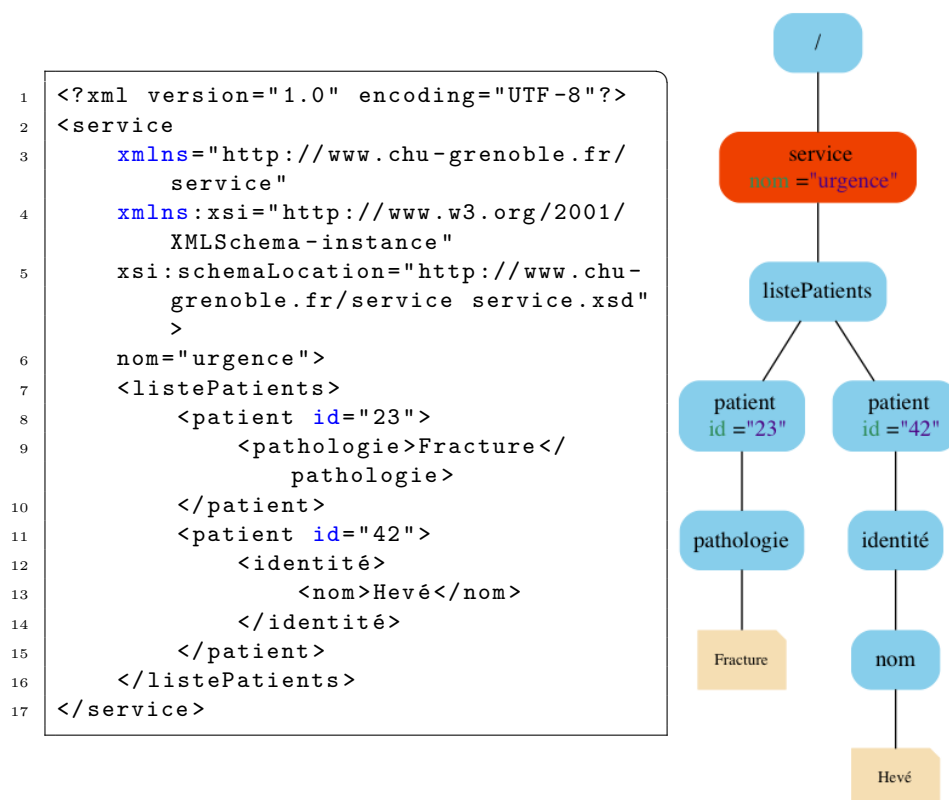


Figure X.3: Exemple de document XML simpliste et arbre d'instance correspondant.

2 Mécanisme *Push*

Le mécanisme *push* sert à écrire dans le document de sortie le texte écrit dans le template. Cela est utile pour écrire du code libre comme du xml, du xhtml, etc.

Par exemple, si l'on applique la transformation X.4 sur l'exemple X.3, on obtient le document X.5.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4   xmlns:chu="http://www.chu-grenoble.fr/service"
5   version="1.0">
6
7   <xsl:output method="html"/>
8
9   <xsl:template match="/">
10     <html>
11       <head>
12         <title>Test</title>
13       </head>
14       <body>
15         Petit texte...
16       </body>
17     </html>
18   </xsl:template>
19
20 </xsl:stylesheet>

```

Figure X.4: Exemple d'un *template* XSLT qui utilise le mécanisme *push* (en rouge).

```

1 <html>
2 <head>
3 <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
4 <title>Test</title>
5 </head>
6 <body>
7   Petit texte...
8 </body>
9 </html>

```

Figure X.5: Résultat obtenu en appliquant le *template* X.4 sur l'exemple X.3.

Le *template* de la feuille de style X.4 (lignes 4 à 13) sélectionne la racine (via /) puis écrit dans le document de sortie (mécanisme *push*) le texte html contenu dans la feuille de style. On peut donc mettre en forme et écrire directement dans le document de sortie grâce au mécanisme *push*.

Le mécanisme *push* s'applique également lorsque l'on appelle les sous-*templates*. Par exemple, si l'on applique la transformation X.6 sur l'exemple X.3, on obtient le document X.7.

On peut noter dans le résultat X.7 que la feuille X.6 contient, en plus des instructions *push*, de la navigation. En effet, le *template* `<xsl:template match="patient">` lignes 16 à 18 sélectionne les nœuds *patient*.

Le *template* `<xsl:template match="/">` lignes 4 à 14 écrit le texte dans la sortie, puis appelle les *templates* sur les sous-nœuds avec l'instruction `<xsl:apply-templates/>` ligne 11. Le *template* `<xsl:template match="patient">` lignes 16 à 18 s'applique sur chaque nœud *patient* et écrit Trouvé un patient ! à chaque fois qu'il est appliqué.

Remarque. Attention, il est très important :

- d'inclure tous les espaces de nom (les vocabulaires) utilisés dans la feuille XSLT.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4   xmlns:chu="http://www.chu-grenoble.fr/service"
5   version="1.0">
6
7   <xsl:output method="html"/>
8
9   <xsl:template match="/">
10     <html>
11       <head>
12         <title>Test</title>
13       </head>
14       <body>
15         Petit texte...
16         <xsl:apply-templates/>
17       </body>
18     </html>
19   </xsl:template>
20
21   <xsl:template match="patient">
22     Trouvé un patient !
23   </xsl:template>
24
25 </xsl:stylesheet>

```

Figure X.6: Exemple d'un *template* XSLT appelant un sous-*template*.

```

1 <html>
2 <head>
3 <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
4 <title>Test</title>
5 </head>
6 <body>
7     Petit texte...
8
9
10
11     Trouvé un patient !
12
13
14     Trouvé un patient !
15
16
17 </body>
18 </html>

```

Figure X.7: Résultat obtenu en appliquant le *template* X.6 sur l'exemple X.3.

- de préfixer TOUS les espaces de nom (pas de vocabulaire par défaut en XSLT)

3 Mécanisme *Pull*

Le mécanisme *pull* récupère (pioche) des données dans le fichier XML source en utilisant des expressions XPath. Il est utilisé pour transformer/analyser/présenter les données et est inclus dans les *templates*.

La récupération des données se fait principalement à l'aide de l'instruction `xsl:value-of`. L'instruction `<xsl:value-of select="expression xpath"/>` sélectionne la valeur (ensemble de nœuds ou bien valeur textuelle/chiffre/booléen, etc. de l'expression XPath. Cette expression peut être soit absolue (partir de

la racine de l'arbre), soit relative et s'appuyer sur le nœud courant, c'est-à-dire le nœud qui est en train d'être traité par le *template*.

Par exemple, si l'on applique la transformation X.8 sur l'exemple X.3, on obtient le résultat X.9.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4   xmlns:chu="http://www.chu-grenoble.fr/service"
5   version="1.0">
6
7   <xsl:output method="html"/>
8
9   <xsl:template match="/">
10    <html>
11      <head>
12        <title>Test</title>
13      </head>
14      <body>
15        <h1>Service <xsl:value-of select="service/@nom"/></h1>
16        Il y a <xsl:value-of select="count(//patient)"/> patients.
17      </body>
18    </html>
19  </xsl:template>
20
21 </xsl:stylesheet>

```

Figure X.8: Exemple de transformation avec le mécanisme *pull*.

```

1 <html>
2 <head>
3 <META http-equiv="Content-Type"
4   content="text/html; charset=UTF
5   -8">
6 <title>Test</title>
7 </head>
8 <body>
9 <h1>Service urgence</h1>
10  Il y a 2 patients.
11 </body>
12 </html>

```

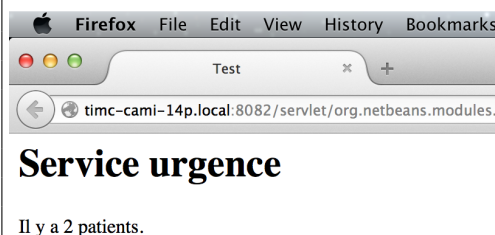


Figure X.9: Résultat obtenu en appliquant le *template* X.8 sur l'exemple X.3.

Dans ce document X.8, la cible de la *template* lignes 4 à 14 cible la racine. Le nœud courant concerne donc tout le document. L'expression *pull* ligne 10 `<xsl:value-of select="service/@nom"/>` utilise un chemin XPath relatif à la racine qui désigne l'attribut `nom` de l'élément `service`. La valeur écrite dans le document résultat est donc la valeur de cet attribut, soit **urgence**.

L'expression *pull* ligne 11 `<xsl:value-of select="count(//patient)"/>` utilise la méthode XPath *count* qui renvoie le nombre de nœuds sélectionné dans une expression XPath. L'expression XPath en question `//patient` est une expression absolue qui sélectionne tous les éléments `patient` du nœud courant. En l'occurrence, il y en a 2.

4 Mécanisme de navigation

Le langage XSLT parcourt le document selon l'arbre d'instance, en profondeur d'abord (dans l'ordre des balises du texte). On peut cependant guider cette navigation pour être plus efficace en utilisant les instructions

- `xsl:apply-templates`
- `xsl:call-template`

`xsl:apply-templates` permet d'appliquer récursivement des *templates* sur des sous-nœuds choisis. Par exemple, la feuille de transformation [X.10](#) saute de la racine aux nœuds `patient`¹.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4   xmlns:chu="http://www.chu-grenoble.fr/service"
5   version="1.0">
6
7   <xsl:output method="html"/>
8
9   <xsl:template match="/">
10     <html>
11       <head>
12         <title>Test</title>
13       </head>
14       <body>
15         <h1>Service <xsl:value-of select="service/@nom"/></h1>
16         Il y a <xsl:value-of select="count(//patient)"/> patients.
17         <xsl:apply-templates select="service/listePatients/patient"/>
18       </body>
19     </html>
20   </xsl:template>
21
22   <xsl:template match="patient">
23     <p>patient id=<xsl:value-of select="@id"/></p>
24   </xsl:template>
25
26 </xsl:stylesheet>

```

Figure X.10: Sélection des nœuds `patient` et appel de la *template* correspondante.

```

1 <html>
2 <head>
3 <META http-equiv="Content-Type"
4   content="text/html; charset=UTF
5   -8">
6 <title>Test</title>
7 </head>
8 <body>
9 <h1>Service urgence</h1>
10   Il y a 2 patients.
11   <p>patient id=23</p>
12   <p>patient id=42</p>

```




Figure X.11: Résultat obtenu en appliquant la *template* [X.10](#) sur l'exemple [X.3](#).

On peut également faire référence à un *template* sans passer par XPath, en l'appelant par son *nom*. On utilise alors l'instruction `<xsl:call-template name="nomDeLaTemplate"/>`. On déclarera alors la *template* grâce à l'expression `<xsl:template name="nomDuTemplate"/>`.

¹Le résultat de cette transformation est présenté Figure [X.11](#)

4.a Application/Exécution des *templates*

Par défaut, les *templates* ne sont pas exécutés (appliqués). Le *template* principal est généralement `<xsl:template match="/">` qui est le seul qui est exécuté automatiquement. En effet, lors de la transformation d'un document, le processeur XSLT commence implicitement par `<xsl:apply-templates select="/">`. Ensuite, tout dépend du code du *template* principal et du contenu du document source.

L'application/exécution d'un *template* dépend donc

- de la correspondance `select/match`
- des instructions d'appel direct

4.b Paramètres de *template*

Un *template* peut recevoir des paramètres

```
1 <xsl:template match="...">
2   <xsl:param name="p1"/>
3   <xsl:param name="p2"/>
4   ...
5 </xsl:template>
```

Les paramètres sont passés dans le *template* appelant en utilisant l'élément `xsl:with-param`.

```
1 <xsl:apply-templates select="...">
2   <xsl:with-param name="p2" select="val/fonc"/>
3   <xsl:with-param name="p1" select="expression xpath"/>
4   ...
5 </xsl:apply-templates>
```

Le type des variables peut être un *Node-Set* (ensemble de nœuds) ou bien un type prédéfini (*string*, *number*, *boolean*).

4.c Note sur le format de sortie

Sur la figure X.7, on peut remarquer de nombreux espaces dans le texte de sortie du XSLT figure X.6. Certains de ces espaces viennent de la mise en page du document XSLT, notamment sur le mécanisme *Push*, mais également du mécanisme de *Navigation*.

Par exemple, si l'on applique la même transformation X.6 sur le document X.12 dans lequel on a ajouté du texte pour les retours à la ligne, on obtient le document X.13.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <service
3   xmlns="http://www.chu-grenoble.fr/service"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://www.chu-grenoble.fr/service service.xsd"
6   nom="urgence">retour à la ligne numéro 1
7   <listePatients>retour à la ligne numéro 2
8     <patient id="23">retour à la ligne numéro 3
9       <pathologie>Fracture</pathologie>retour à la ligne numéro 4
10    </patient>retour à la ligne numéro 5
11    <patient id="42">retour à la ligne numéro 6
12      <identité>retour à la ligne numéro 7
13        <nom>Hevé</nom>retour à la ligne numéro 8
14      </identité>retour à la ligne numéro 9
15    </patient>retour à la ligne numéro 10
16  </listePatients>retour à la ligne numéro 11
17 </service>
```

Figure X.12: Exemple X.3 sur lequel on a ajouté du texte pour les retours à la ligne.


```

1 <html>
2 <head>
3 <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
4 <title>Test</title>
5 </head>
6 <body>
7     Petit texte...
8     retour à la ligne numéro 1
9     retour à la ligne numéro 2
10
11     Trouvé un patient !
12     retour à la ligne numéro 5
13
14     Trouvé un patient !
15     retour à la ligne numéro 10
16     retour à la ligne numéro 11
17 </body>
18 </html>

```

Figure X.13: Résultat obtenu en appliquant le *template* X.6 sur l'exemple X.12.

Ceci s'explique par le fait que la balise `<xsl:apply-templates/>` appelle récursivement les *templates* de la feuille XSLT sur tous les *nœuds* du document XML et pas seulement sur les *éléments*. C'est-à-dire que les nœuds texte sont aussi sélectionnés. Or, comme ici, aucun *template* n'est défini pour les nœuds texte, c'est le comportement par défaut qui s'applique, c'est-à-dire que le texte des nœuds texte est simplement recopié dans le fichier de sortie.

Pour éviter cela, deux stratégies sont possibles:

- Redéfinir le comportement par défaut des nœuds texte, par exemple en définissant le *template* `<xsl:template match="text()"/>` qui du coup ne va rien écrire sur le fichier de sorties dans le cas de tous les nœuds texte
- Forcer la navigation à n'appliquer les *templates* que sur les sous-éléments et non sur tous les nœuds en remplaçant l'appel `<xsl:apply-templates/>` par `<xsl:apply-templates select="unCheminXPathQuiNeSélectionneQueDesSousElements"/>`

5 Programmation

Il existe d'autres instructions XSLT qui permettent de manipuler les données dans les *templates*. Ces instructions sont similaires à des instructions algorithmiques

- variable
- instructions conditionnelles
- boucles de parcours
- branchements multiples



Le langage XSLT est récursif. Plus exactement, le processeur XSLT (qui est dans une API du langage appelant (Java, C++, C#, JS ...)) fonctionne de manière récursive ; c'est lui qui va exécuter les boucles pour vous ; vous n'avez donc pas à demander explicitement des boucles (comme `foreach`). Le fonctionnement qui doit être adopté lors de la rédaction d'une feuille XSLT est le fonctionnement récursif d'appels de *templates* via XPath = vous avez juste à spécifier les templates pour les nœuds que vous voulez voir pris en charge. Les instructions *algorithmiques* ne doivent être utilisées qu'en dernier recours !

nom	description	exemple
variable	une variable a un nom et un type (string, number, boolean ou node-set). Sa valeur est donnée par <code>\$nomDeLaVariable</code> .	<pre><xsl:variable name="nb">15</xsl:variable> <xsl:variable name="p2" select="patient[2]"/></pre>
tri	On peut également trier la sélection des éléments lorsque l'on applique des <i>templates</i> . <code><xsl:sort select="critère xpath de tri"/></code>	<pre> <xsl:apply-templates select="//patient"> <xsl:sort select="@id" order="descending"/> </xsl:apply> </pre>
condition	Une condition permet d'exécuter une partie de <i>template</i> seulement lorsqu'une condition est vérifiée (Attention, il n'y a pas de <i>else</i>). <code><xsl:if test="..."></code> ... <code></xsl:if></code>	<pre><xsl:if test="age < 6"/> ... </xsl:if> <xsl:if test="\$var != 5"/> ... </xsl:if></pre>
boucle	Une boucle permet d'exécuter le même traitement pour un ensemble de nœud. Attention, cette instruction ne doit être utilisée qu'en dernier recours, l'instruction <code>xsl:apply-templates</code> doit être utilisée en priorité. <code><xsl:for-each select="expression xpath"></code> ... <code></xsl:for-each></code>	<pre><xsl:for-each select="patient[@id>50]"> ... </xsl:for-each></pre>
branchements	Lorsqu'une condition comporte plusieurs possibilité, on peut utiliser un branchement multiple. <code><xsl:choose></code> <code><xsl:when test="...">...</xsl:when></code> <code><xsl:when test="...">...</xsl:when></code> <code><xsl:otherwise>...</xsl:otherwise></code> ... <code></xsl:choose></code>	<pre><xsl:choose> <xsl:when select="@id < 20">...</xsl:when> <xsl:when select="@id > 20 and @id < 100">...</xsl:when> <xsl:otherwise>...</xsl:otherwise> ... </xsl:choose></pre>

On peut également réduire la dépendance au document d'entrée des instructions *push* grâce à l'élément `xsl:text` qui permet d'écrire du texte sans tenir compte du format d'entrée.

Enfin, l'élément `xsl:element` permet de créer facilement des élément (xml) dans le document de sortie. Par exemple,

```
1 <xsl:element name="maBalise">
2   Contenu de l'élément
3 </xsl:element>
```

donne en sortie le code suivant:

```
1 <maBalise>
2   Contenu de l'élément
3 </maBalise>
```

On peut également ajouter des attributs à cet élément grâce à l'élément `xsl:attribute`. Par exemple

```
1 <xsl:element name="img">
2   <xsl:attribute name="src">
3     <xsl:value-of select="photofile"/>
4   </xsl:attribute>
5   <xsl:attribute name="alt">
6     <xsl:value-of select="nom"/>
7   </xsl:attribute>
8 </xsl:element>
```

appliqué sur le morceau de document suivant:

```
1 <facebook>
2   <name>Golade Larry</name>
3   <photofile>fgl123.jpg</photofile>
4 </facebook>
```

donne le résultat suivant:

```
1 
```

