THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Intégration de la technologie XML pour la génération automatique de pages HTML

Wahid, Moussa

Award date:
2002

Facultés Universitaires
Notre-Dame de la Paix

Intégration de la technologie XML pour la génération automatique de pages HTML

Moussa WAHID

Promoteurs : Prof. M. Noirhomme et Dr. A. de Baenst

Mémoire présenté
en vue de
l'obtention du grade
de licencié en
Informatique

Institut d'Informatique
rue Grandgagnage,
21 – 5000 Namur

- Année Académique 2001-2002 -

Errata : la figure 3 à la page 56 est à remplacer par la figure 3 qui suit.

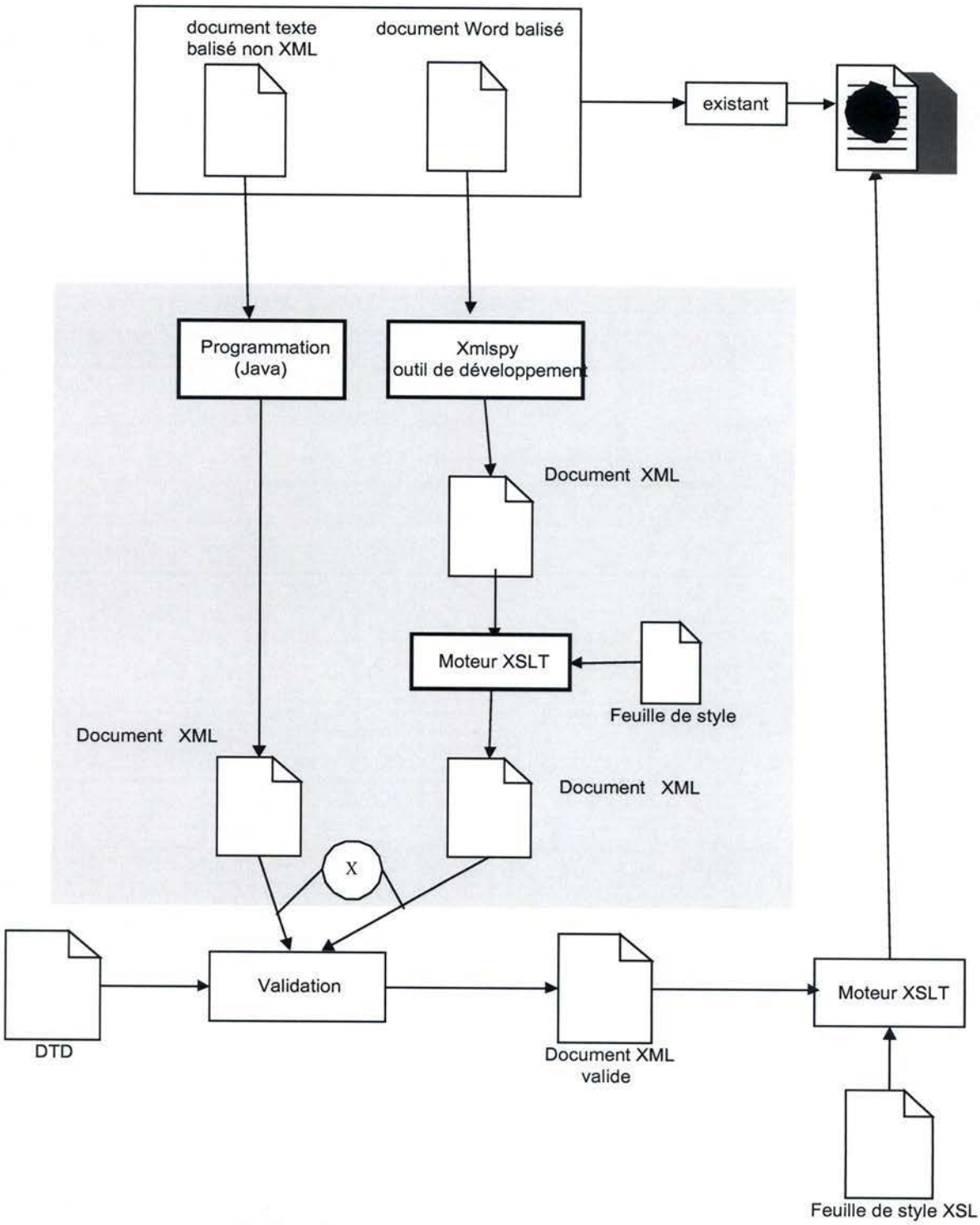


Figure 3 : Diagramme générale du processus de transformation.

Errata : la figure 4 à la page 65 est à remplacer par la figure 4 qui suit.

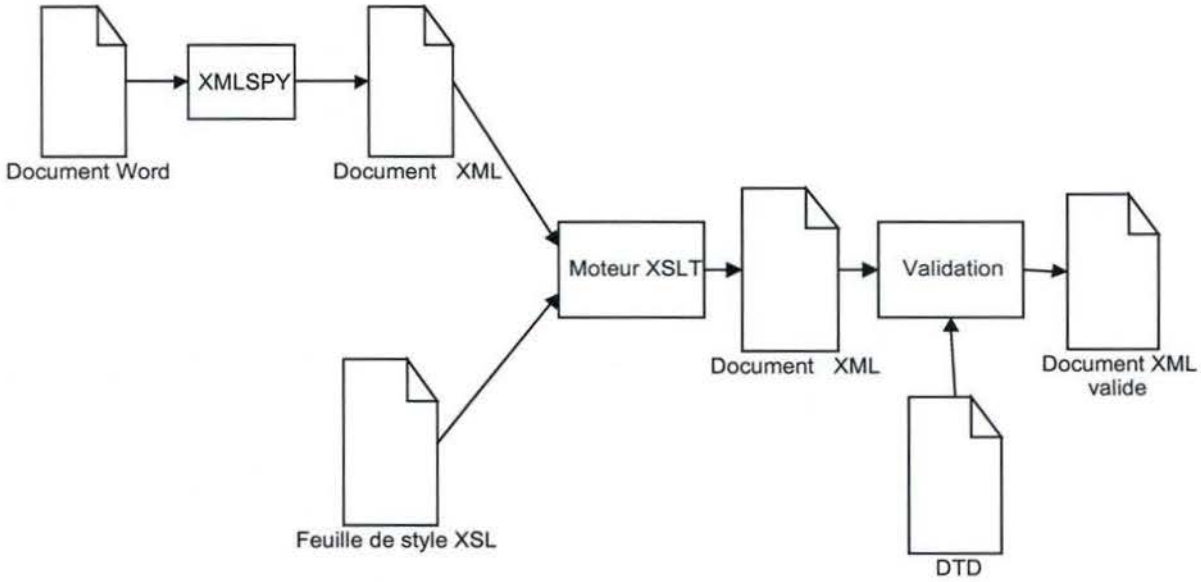


Figure 4 : Schéma de fonctionnement de la transformation d'un document Word en document XML valide.

Remerciements

Je voudrais remercier Madame Anne de Baenst pour m'avoir encadré et encouragé durant la réalisation de ce mémoire et pour avoir fait preuve d'une grande disponibilité. Je voudrais la remercier également pour ces remarques très enrichissantes et constructives sur différents aspects des idées exposées dans ce mémoire, ainsi que pour l'intérêt qu'elle a porté à mon travail.

Je souhaite également remercier Madame M. Noirhomme, professeur à l'Institut d'Informatique, qui m'a donné la possibilité de réaliser ce mémoire.

Mes remerciements s'adressent enfin à toute ma charmante petite famille pour son soutien moral et logistique.

GLOSSAIRE

API	: Application Program Interface
DOM	: Modèle objet de document (Document Object Model)
DTD	: Définition de type de documents (Document Type Definition)
FTP	: Protocole de transport de fichiers (File Transport Protocol)
HTML	: Langage de balisage hypertexte (HyperText Markup Language)
Namespace	: espace de nom
URI	: Uniforme Ressource Identifier
URL	: Universal Resource Locator
SAX	: Simple API for XML
SGML	: Langage général de marquage (Standard Generalized Markup Language)
Xlink	: XML Linking Language
XML	: Langage de balisage extensible (eXtensible Markup Langage)
XSL	: Langage de feuille de style extensible (eXtensible Stylesheet Language)
XSLT	: eXtensible Stylesheet Language Transformations

Tables des matières

Introduction

Le sujet de ce mémoire est né du fait que beaucoup de sites et de pages web sont difficiles à maintenir, à gérer, à modifier et à réutiliser les compétences que ce soit les compétences de programmation, de conception ou même d'analyse. Cependant mettre en place une stratégie de résolutions de ces problèmes oblige le développement et l'utilisation de moyens nécessaires pour permettre la création des composants portables, réutilisables, faciles à utiliser, à modifier et à gérer. Parmi ces composants, on peut citer l'utilisation des technologies XML et Java.

C'est dans ce cadre que nous apportons notre contribution en mettant en place, ou du moins, en proposant une technique qui soit la plus automatique possible pour exploiter ces documents qui sont stockés soit sous forme de fichiers plat qui n'ont aucune structure soit sous forme de fichiers propriétaires comme Word (fichiers binaires) ou sous format HTML.

Afin de bien cerner le problème, nous allons, dans un premier temps, faire un état de l'art en ce qui concerne les technologies XML pour la structuration des données et en particulier l'impact de cette technologie dans la création, la génération des pages et des sites web.

Dans le premier chapitre, nous donnerons un aperçu générale du langage HTML. Ceci permettra de mieux comprendre les limites du langage HTML.

Dans le deuxième chapitre jusqu'au sixième, nous abordons les technologies XML. Nous commencerons par le langage XML, DTD, Schéma XML, Analyseurs (DOM et SAX) et en fin XSL qui décrit les moyens de transformer les document XML en HTML.

Nous poursuivrons en présentant le travail pratique réalisé au cours de ce mémoire, c'est-à-dire la démarche de l'intégration des technologies XML dans la création de pages web automatiquement en partant des transformations des fichiers textes structurés non XML ou des fichiers propriétaires Word en documents XML, puis par application des feuilles de style sur ces derniers pour générer des pages HTML qui seront destinées à être publier sur Internet. Nous verrons les grandes lignes de la démarche proposée, ainsi que les détails de chacune des deux méthodes proposées. En conclusion nous montrerons les apports de l'intégration de XML dans le processus de création des pages et des sites web. Les codes sources de cette implémentation sont, en partie, disponible dans les Annexes.

1.Rappel d 'HTML

1.1 Introduction

HTML (**H**yper **T**ext **M**arkUp **L**anguage) est un langage de description (et non pas un langage de programmation) qui permet de décrire l'aspect d'un document, d'y inclure des informations variées (textes, images, sons, animations, etc.) et d'établir des relations cohérentes entre ces informations grâce aux liens hypertextes.
Les avantages du langage HTML sont nombreux:

1. Il est peux coûteux, en effet un simple éditeur de texte suffit pour écrire un document HTML.
2. Il est relativement facile à aborder.
3. Il représente en outre un bon moyen de dépasser les problèmes de compatibilité entre des systèmes et des formats informatiques différents.

La description d'un document HTML passe par l'utilisation de **Balises** ("Tags" en Anglais). Une balise est délimitée par les signes "<" et ">" entre lesquelles figure la description de la balise. La plupart de temps, on utilise une balise de début est une balise de fin, qui définissent les propriétés de l'intervalle.

Exemple:

*** écriture en gras ***

se visualise par : (résultat dans le navigateur)
écriture en gras

1.2 Origine

Le terme « Markup » se réfère aux marques, aux annotations manuscrites placées par l'auteur sur un document pour préciser à l'imprimeur comment il doit être présenté. Avec l'apparition des ordinateurs, ces marques ont été intégrées dans le texte mais chaque matériel de photocomposition réclamait son propre langage "Markup". Au début des années 80, le GCA (Graphics Communications Association) a mis au point le premier langage de marquage généralisé baptisé GenCode. Au même moment, un comité de normalisation ANSI publiait le standard Generalized Markup Language (GML). En décembre 1986, les deux comités ont unit leurs efforts pour définir le standard SGML (Standard Generalized Markup Language) accepté par l'ISO (International Standard Organization) sous le numéro 8879.
Jusqu'à 1990 les principales applications Internet étaient le courrier électronique, telnet et FTP (File Transport Protocole).
En 1990, Tim Berners-Lee, un physicien du CERN, développa les protocoles du World Wide Web. Il créa le HTML, issu d'un sous ensemble du SGML en s'appuyant sur les travaux de Ted Nelson qui furent à l'origine du terme hypertexte (1965).
C'est en 1993 que l'explosion de l'Internet eu lieu avec la sortie du navigateur MOSAIC.

1.3 SGML et HTML

Le SGML est un standard ouvert qui n'est lié ni à une plate-forme, ni à un constructeur. Les fichiers SGML sont stockés sous forme de simple textes ASCII et peuvent donc être utilisés par n'importe quelle machine.
Le langage SGML est extrêmement pratique pour réaliser des publications à la demande. Le but est de pouvoir, à partir d'un document de base unique, publier différentes versions de présentations s'adressant à des publics différents.
SGML est un langage de description de donnée qui se divise en deux parties : la DTD (Document Type Definition) et les données elles-mêmes. La DTD est une sorte de dictionnaire qui décrit les différents "balises" acceptées dans le document et les relations qui les unissent. Le corps du document contient les données délimitées par les "balises" définies dans la DTD.
Le langage HTML n'est qu'une instance du langage SGML définie par une DTD spécifique.
Un document HTML est un simple fichier texte ASCII qui contient des "balises" HTML.
Comme dans SGML, chaque commande de langage HTML débute par un "<" et se termine par un ">" et les commandes peuvent s'écrire en minuscules ou en majuscules indifféremment.

1.4 Les concepts de base de HTML

Un document HTML est composé de texte et de commandes (*tag en anglais*). Ces commandes permettent de mettre en forme le texte (Titre, caractère gras, italique, image, liens, etc...).

1.4.1 Allure d'un fichier HTML

```
<HTML>
 <HEAD>
  <TITLE>Exemple de structure de document HTML</TITLE>
<META NAME="Author" CONTENT="Albert Camus">
 </HEAD>
<BODY>
...
... Le document HTML
...
</BODY>
</HTML>
```

1.4.2 Les en-têtes HTML

1.4.2.1 <HTML>

Les commandes de début et de fin du document HTML sont :
<HTML> </HTML>.

1.4.2.2 <HEAD>

les balises *<HEAD> </HEAD>* contiennent les informations non affichées concernant le document.

1.4.2.3 <TITRE>

Les balises *<TITRE>*et *</TITRE>* expriment le titre du document qui apparaît dans la barre de titre du navigateur Web.

1.4.2.4 <META>

Les balise *<META ...>* guident les moteurs de recherche pour indexer les pages.

1.4.2.5 <BODY>

Toutes les informations affichées par le navigateur sont contenues entre les balises *<BODY>* et *</BODY>*.

1.4.2.6 Les commentaires < !-- commentaires - ->

Les commentaires peuvent être placés n'importe où dans un document HTML à condition de ne pas être imbriqués. Ils sont placés entre les chaînes de caractères < !— et - ->.

1.4.3 Les délimiteurs HTML

Contrairement à la plupart des systèmes de traitement de texte, les retours à la ligne et les espaces n'ont aucune valeur en HTML. C'est le navigateur Web lui-même qui définira le passage à la ligne en fonction de facteurs comme la taille des polices de caractères, la largeur de la fenêtre de visualisation, etc.
Les délimiteurs permettent de formater le document.

<P> </P>	Cette commande <P> permet le saut de paragraphe
*** ***	Cette commande impose la coupure d'une ligne de texte en rejetant ce qui suit à la ligne suivante.
<NOBR>...</NOBR>	Cette commande permet de mettre du texte sans retour à la ligne.
<WBR>	Cette commande permet de forcer un retour à la ligne dans un texte encadré par *<NOBR>*.
<HR> ***WIDTH*** ***SIZE*** ***ALIGN*** ***NOSHADE***	pour une ligne horizontale de séparation : fait varier la largeur de la ligne soit en % soit en pixel (valeur par défaut 100%) fait varier l'épaisseur de la ligne en pixel (valeur par défaut 1) fait un alignement de la ligne suivant les 3 possibilités : **CENTER** par rapport au centre de la fenêtre, **LEFT** par rapport à la gauche de la fenêtre, **RIGHT** par rapport à la droite de la fenêtre. Le paramètre **ALIGN** a un effet que lorsque la longueur de la ligne est inférieur à 100%. lorsqu'il est présent dans le marqueur <HR> l'effet est une ligne pleine sans ombrage.

1.4.4 Les liens HTML

Une des fonctionnalités la plus agréable du langage HTML est la possibilité de créer des liens vers d'autres documents. Ces documents peuvent être des documents HTML, des images, du son, des films, et des serveurs FTP[1], etc.
Ces liens peuvent être présenter sous forme du texte souligné ou d'image encadrée que l'on appelle **ancre** ou **lien**.
Ce lien peut être soit un fichier locale, soit une URL[2] (Unified Ressource Locator).
Il existe quatre principaux types de liens :

1.4.4.1 Les liens vers un document complet distant

Syntaxe : <A HREF = « URL » ancre </A>
Exemple : pour accéder aux pages de l'Institut Informatique de Namur.

<A HREF = " http://www.info.fundp.be/acceui.html "> Acceuil </A> F.U.N.D.P

1.4.4.2 Les liens vers un document complet local

Syntaxe : <A HREF = « NomDeFichierLocal » ancre </A>
Exemple : pour accéder à une page locale

<A HREF = " /Mes documents/CoursHtml/index.html "> Cliquez-Ici </A>

1.4.4.3 les liens vers une partie d'un document local

Syntaxe :
… <A NAME = "etiquette"> nom </A>… .pour définir un point de branchement.
… <A NAME = "NomDeFichierLoca#etiquette"> ancre </A>… . pour faire le lien.
Exemple : pour accéder à une bibliographie du document local.

<A NAME = "Bibliographie" > Bibliographie : </A>
<A HREF = " References.html#Bibliographie " > Bibliographie </A>

1.4.4.4 les liens vers une partie du document courant

Syntaxe :
<A NAME = "etiquette"> nom </A> pour définir un point de branchement.
<A NAME = "#etiquette"> ancre </A>… . pour faire le lien.
Exemple : pour accéder au début du document courant

<A NAME = "top" > retour au début </A>
<A HREF = "#top" > Reournons au début </A>

---

[1] FTP (File Transport Protocole) : protocole de transport des fichiers
[2] URL : est un moyen de nommer un objet dans le monde WWW.

1.4.5 Les Listes

Il existe différents types de listes ; une balise spécifique définit chacun d'eux.
Les listes numérotées ou ordonnées
Les listes numérotés sont une forme particulière de listes. La numérotation des différentes entrées s'effectues automatiquement.
La commande <ol> définit le début d'une liste numérotée et termine cette définition.
La balise <li> définit une entrée. termine cette définition.

Exemple :

```
<ol>
   <li> Premier point de l'ordre du jour   : Conférence </li>
   <li> Deuxième point de l'ordre du jour : Déjeuner </li>
   <li> Troisième point de l'ordre du jour : Café </li>
   <li> Quatrième point de l'ordre du jour : Débat </li>
</ol>
```

Les listes non numérotées dites listes à puces dont les entrées sont signalées par un signe typographique. Les puces sont des symboles non numérotés (point, flèche, petit carré ou étoile)
La commande <ul> définit le début de la liste à puce et termine cette définition. Le <li> et mentionnent le début et la fin d'une nouvelle entrée dans la liste.

Les options suivantes sont possibles :
1. TYPE=1 : pour une liste numérotée 1,2,3...
2. TYPE=A : pour un repérage type A,B,C...
3. TYPE =a : pour un repérage type a, b, c...
4. TYPE = I : pour une liste numérotée I,II,III,IV...
5. TYPE = i : pour une liste numérotée i, ii, iii, iv...

START = n fait débuter le repérage (chiffres ou lettres) au rang numéro n.
Remarque : le TYPE = 1 est utilisé par défaut.

1.4.6 Style

L'utilisation de différents styles de polices de caractères permet de varier la présentation d'un texte. En HTML, on peut utiliser des styles logiques (l'initiative est laisser au client WWW) où le nom du style indique la nature du fragment de texte à écrire dans ce style ou des styles physiques où le nom de style indique explicitement le style de police que l'on souhaite voir utiliser.

Les commandes de styles logiques sont les suivantes :

<EM> texte</EM>	met le texte généralement en italique.
<STRONG> texte</STRONG>	met le texte généralement en gras.
<CODE>texte</CODE>	pour l'utilisation d'une police à chasse fixe (encombrement des caractères constant).
<SAMP>caractères</SAMP>	séquence de caractères littéraux.
<KBD>saisie</KBD>	pour mettre en évidence une saisie d'utilisateur.

<VAR>variable</VAR>	pour indiquer le nom d'une variable.
<DFN>définition</DFN>	pour mettre en évidence la 1ère utilisation d'un terme.
<CITE>citation</CITE>	pour mettre en évidence une citation.
<ADDRESS>adresse</ADDRESS>	cette commande est généralement utilisée pour indiquer l'auteur d'un document ainsi que le moyen de le contacter ou bien elle donne l'adresse du document. Elle est souvent placée en fin de document.

Les commandes de styles physiques sont les suivantes :

<I>texte</I>	met le texte en italique.
<B>texte</B>	met le texte en gras.
<TT>texte</TT>	pour l'utilisation d'une police à chasse fixe (encombrement des caractères constants).
<U>texte</U>	souligne le texte.
<S>texte</S>	barre le texte.
^{texte}	exposant.
_{texte}	indice.

1.4.7 Les couleurs utilisées dans <BODY>

Par défaut le fond du document est gris clair, les caractères sont noirs, les prises d'hypertextes sont bleues quand elles n'ont jamais été utilisées, violettes dans le cas contraire.

Il est possible de modifier ces couleurs en rajoutant à la commande <BODY> les options suivantes:

BGCOLOR=c	pour le fond du document
TEXT=c	pour la couleur des caractères
LINK=c	pour la couleur des prises d'hypertextes non utilisées
VLINK=c	pour la couleur des prises d'hypertextes utilisées
ALINK=c	pour la couleur des prises d'hypertextes à l'instant de la sélection

La valeur de c est composée de trois nombres hexadécimaux accolés (codés de 00 à FF) présentant le mélange des trois couleurs primaires RGB (Red, Green, Blue). Le nombre obtenu est précédé d'un dièse (#). Le blanc a pour valeur #FFFFFF; le noir a pour valeur #000000.
Toutes les autres couleurs sont obtenues par des dosages dans chacune des composantes RGB.

Exemple de couleur :

c= #FFFFFF	**pour le blanc**
c= #C0C0C0	**pour gris clair**
c= #80FF80	**pour vert clair**
c= #FFFFF8	**pour le jaune clair**
c= #800000	**pour le marron**

Exemple d'utilisation des couleurs :

```
<HTML>
<HEAD>
<TITLE> Essai de couleurs </TITLE>
</HEAD>
<BODY BGCOLOR=#80FFFF TEXT=#000080 LINK=#FF0000>
La couleur du fond est bleu claire, la couleur du texte est bleu. La prise d'hypertexte est
rouge. Retour : cliquer <A HREF="htmldoc.htm"> ici </A>
</BODY>
</HTML>
```

On peut aussi décrire les couleurs par leurs noms.
Exemple d'utilisation des couleur :

```
<HTML>
<HEAD>
<TITLE> Essai de couleurs </TITLE>
</HEAD>
<BODY BGCOLOR="black" TEXT="white" LINK="RED">
   La couleur du fond est noire, la couleur du texte est blanc. La prise d'hypertexte est
   rouge.  Retour  : cliquer <A HREF="htmldoc.htm"> ici </A>
</BODY>
</HTML>
```

1.4.8 Les image dans le texte

Des images peuvent être insérées dans le texte d'un document HTML, et elles sont au format GIF ou JPG.
Elles peuvent servir de prises d'hypertextes :

1. Soit toute l'image réagit à un clic.
2. Soit l'image réagit en fonction de la zone où le clic s'est produit (on parle dans ce cas là d'image cliquable ou réactive).

La commande <*IMG*> permet d'insérer une image dans le texte. La syntaxe de cette commande est la suivante :

<*IMG SRC*="image" *ALIGN*="attribut" *ALT*="titre" *HSPACE*="hh" *VSPACE*="vv" *BORDER*="bb">

image = adresse une image

attribut = "MIDDLE" "TOP" "BOTTOM" "LEFT" ou "RIGHT"

titre = "un titre" qui se substituera à l'image pour les clients ne pouvant ou ne sachant pas afficher des images. En effet, il est beaucoup plus agréable pour ces utilisateurs de voir un message ou un caractère, au lieu de [IMAGE] .

hh = nombre de pixels de séparation à droite et à gauche de l'image. (par défaut hh=0)

vv = nombre de pixels de séparation en haut et en bas de l'image. (par défaut vv=0)

bb = nombre de pixels de l'encadrement de l'image. (par défaut il n'y a pas d'encadrement).

1.4.9 Les tableaux

Les tableaux sont comparables aux listes, en ce sens qu'ils permettent d'organiser l'information.
Pour créer un tableau, les commandes sont les suivantes :

TABLE>...</TABLE>

Pour cette commande les options suivantes sont possibles :

BORDER :	trace un cadre en trait fin
BORDER = n :	trace un cadre en trait de n pixels d'epaisseur.
CELLSPACING =n :	espacement de n pixels entre les cellules.
CELLPADDING =n :	espacement autour de l'écriture dans les cellules.
WIDTH =n ou n% :	largeur en pixels ou largeur relative du tableau.

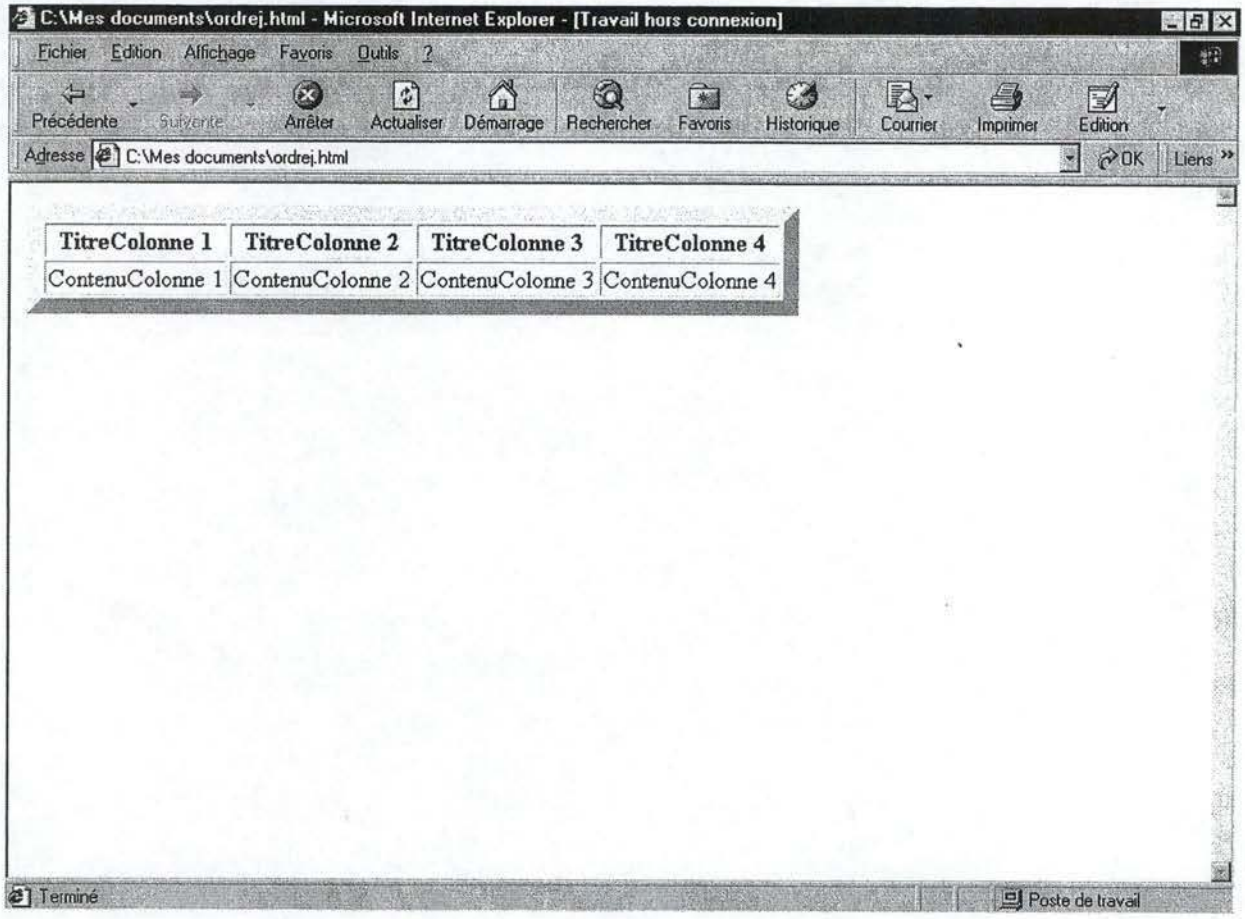
<TR > ...</TR> ces balises définissent le début et la fin d'une ligne.
HTML distingue deux types de cellules : les cellules d'en-tête et les cellules de données.
<TH> ... </TH> encadre une cellule d'en-tête du tableau. (cellule pouvant contenir un texte alphanumérique, une image, une liste, un lien, un autre tableau ou rien).
<TD> ... </TD> encadre une cellule du tableau. (cellule pouvant contenir un texte alphanumérique, une image, une liste, un lien, un autre tableau ou rien)

Exemple :

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<table border= "10" >
  <tr>
     <th> TitreColonne 1 </th>
     <th> TitreColonne 2 </th>
     <th> TitreColonne 3 </th>
     <th> TitreColonne 4 </th>
  </tr>
  <tr>
     <td> ContenuColonne 1 </td>
     <td> ContenuColonne 2 </td>
     <td> ContenuColonne 3 </td>
     <td> ContenuColonne 4 </td>
  </tr>
</table>
```

```
</BODY>
</HTML>
```

Le résultat de l'affichage de la page correspondante est le suivante :



2. Concepts essentiels de XML

2.1 Introduction

XML offre une nouvelle façon d'effectuer un balisage des données. En fait, bien que HTML et XML puissent paraître très similaires, ils sont en réalité extrêmement différents.

2.2 Langage de balisage

Un balisage est tout ce qui est ajouté à un document pour apporter des informations complémentaires. Par exemple, un texte souligné ou mis en gras dans un traitement de texte est une forme de balisage.

HTML est un exemple classique de langage de balisage permettant d'écrire un document destiné à être affiché sur le Web.

2.2.1 Balise et élément

Les balises sont les signes supérieurs et inférieurs (< et >, connus sous le nom de délimiteurs) ainsi que le nom de balise placé entre eux. Voici quelques exemples de balises utilisées en HTML :
<P> est une balise signalant le début d'un nouveau paragraphe ;
<I> est une balise indiquant que le texte qui suit doit être afficher en italique ;
</I> est une balise signalant la fin d'une section de texte affichée en italique.

Le terme d'élément s'applique en revanche aux balises avec leur contenu. Voici un exemple d'élément :

<B> Ici du texte en gras </B>

En terme généraux, les balises sont des étiquettes signalant au navigateur ou à un analyseur d'effectuer quelque chose avec ou sur ce qui est enfermé dans ces balises.

2.2.2 Attributs

Toute balise peut posséder un ou plusieurs attributs. Un attribut se présente sous la forme d'une paire nom/valeur; en effet un élément peut posséder un attribut (doté d'un nom), et l'attribut possède une chaîne placée entre guillemets comme valeur. Cela se présente sous la forme :

<nombalise attribut = "valeur">

2.3 Langage XML

XML, ou **eXtensible Markup Language (langage à balises étendu)**, est un de langage de balisage qui, comme HTML, utilise de façon intensive les balises et les attributs.

En HTML, les **balises utilisées sont prédéfinis** : il existe un ensemble bien défini de balises et d'attributs permettant d'écrire des pages Web.

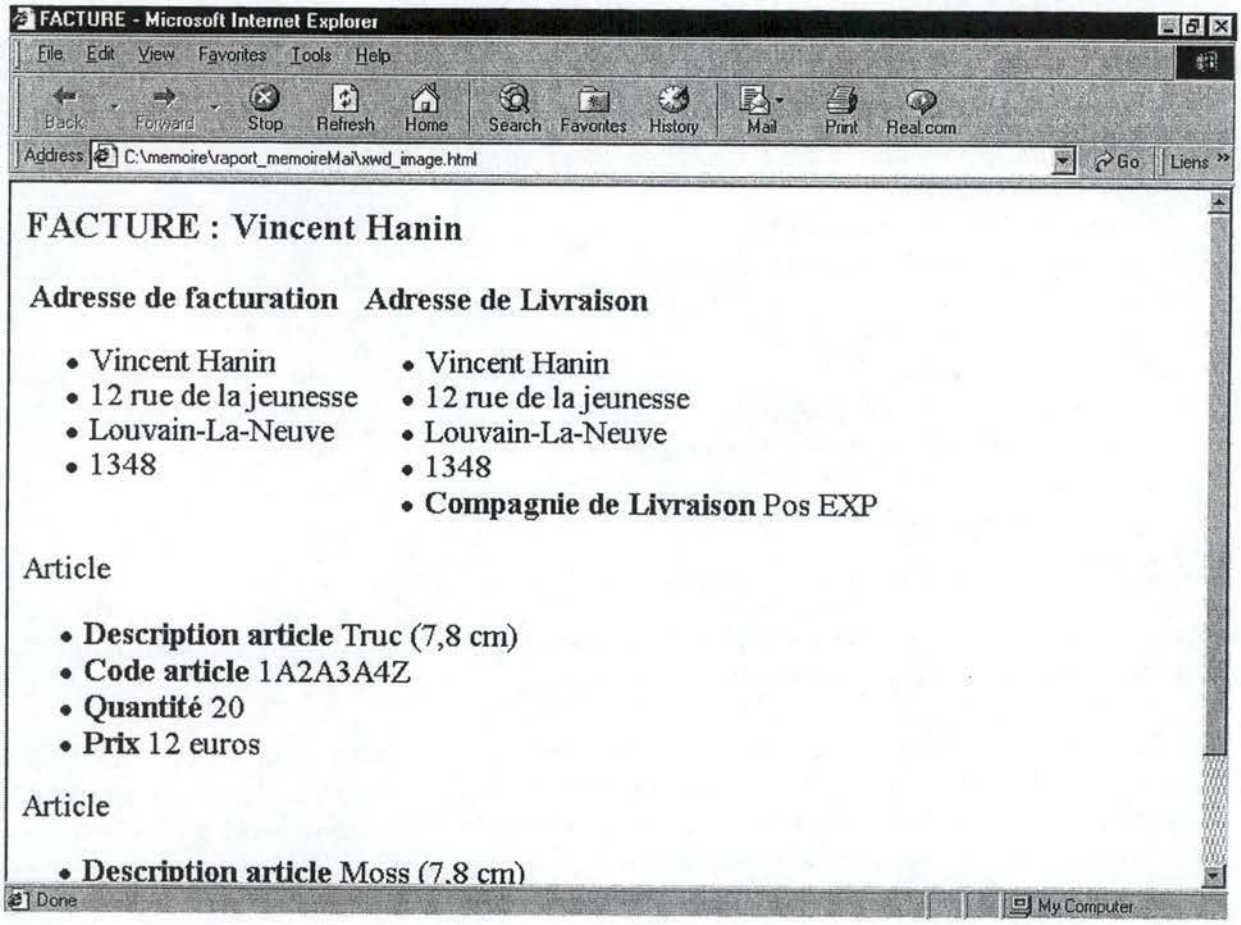
En revanche XML est bien plus souple. Il permet de créer nos propres balises et attributs et ses utilisations possibles dépassent largement un simple affichage dans un navigateur Web. Comme nous pouvons avec XML, créer nos propre balises et attributs, nous avons la possibilité de concevoir un balisage décrivant précisément **le contenu de l'élément (on les appelle les balises de structure)**, plutôt que de n'utiliser que des balises décrivant le moyen d'afficher les données sur une page Web.

XML est une façon de baliser des données afin de les rendre auto-descriptives. Par exemple, on considère le fichier HTML suivant :

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 //EN ">
<HTML>
   <HEAD><TITLE>FACTURE</TITLE></HEAD>
<BODY>
   <H3>FACTURE : Vincent Hanin </H3>
      <TABLE>
     <TR>
        <TD valign = "top">
           <H4>Adresse de facturation</H4>
           <UL>
           <LI> Vincent Hanin </LI>
           <LI>12 rue de la jeunesse</LI>
           <LI>Louvain-La-Neuve</LI>
           <LI>1348</LI>
           </UL>
        </TD>
        <TD valign="top">
           <H4>Adresse de Livraison</H4>
           <UL>
           <LI> Vincent Hanin </LI>
           <LI>12 rue de la jeunesse</LI>
           <LI>Louvain-La-Neuve</LI>
           <LI>1348</LI>
           <LI><B>Compagnie de Livraison</B> Pos                    EXP</LI>
           </UL>
        </TD>
     </TR>
     </TABLE>
      Article
   <UL>
     <LI><B>Description article </B> Truc (7,8 cm) </LI>
     <LI><B>Code  article </B> 1A2A3A4Z </LI>
     <LI><B>Quantité  </B>20 </LI>
     <LI><B>Prix </B> 12 euros</LI>
   </UL>
      Article
   <UL>
     <LI><B>Description article </B> Moss (7,8 cm) </LI>
     <LI><B>Code  article </B> 2Z3Z4Z4A </LI>
     <LI><B>Quantité  </B>10 </LI>
```

```
    <LI><B>Prix </B> 120 euros</LI>
  </UL>
</BODY>
</HTML>
```

Le résultat de l'affichage de ce fichier par le navigateur est le suivant :



Bien que cela soit parfait pour afficher cette page Web, des balises comme **<LI>** ne disent absolument pas qu'elles contiennent des informations sur un produit commandé. Rien dans le balisage HTML n'indique qu'il s'agit d'une facture.

En générale les données possèdent de nombreux utilisateurs potentiels, mais chacun de ces utilisateur peut utiliser des logiciels écrits dans différents langages de programmation, fonctionnant sous différents systèmes d'exploitations (HP-UNIX, LINUX, Windows ...). Au moyen de la technologie XML cela permet de transmettre ces données entre les programmes d'une façon indépendante des plates formes, tout en indiquant à chaque programme comment chaque donnée est balisée.
Comme indiqué précédemment, nous pouvons créer nos propres balises en XML . Alors adoptons à la place des balises HTML des balises XML permettant de décrire les données afin de montrer qu'il s'agit d'une facture et comment cette facture est structurée.

On aurait, par exemple :

```
< ?xml version="1.0" encoding ="ISO-8859-1" ?>
<Facture>
   <nomClient > Vincent Hanin </nomClient >
   <adresseFacturation>12 rue de la jeunesse</adresseFacturation>
   <villeFacturation>Louvain-La-Neuve </villeFacturation>
   <codePostalFacturation>="1348"</codePostalFacturation>
   <adresseLivraison> 12 rue de la jeunesse</adresseLivraison>
   <villeLivraison>"Louvain-La-Neuve" </villeLivraison>
   <codePostalLivraison> "1348"</codePostalLivraison>
   <compagnieLivraison>"POST EXP" </compagnieLivraison>
   <LigneDeCommande
      codeArtcle = "1A2A3A4Z"
      descriptionArticle ="Truc (7,8 cm) "
      quantite = "20"
      prix = "12 euros"/>
   <LigneDeCommande
      codeArtcle = "2Z3Z4Z4A"
      descriptionArticle ="Moss (7,8 cm) "
      quantite = "10"
      prix = "120 euros"/>
</Facture>
```

Dans ce document, l'élément *Facture* représente une facture et l'élément *nomClient* représentele nom du client concerné par cette facture.

Par ailleurs, ce fichier XML étant du texte brut, les données d'un fichier XML sont accessibles à tout langage de programmation et sur toute plate-forme, et il peut facilement être transmis à l'aide de HTTP. Nous pouvons donc utiliser les données balisées en XML de bien plus nombreuses façons que la version HTML. Comme ce n'est que du texte et que nous savons qu'à chaque fois que nous rencontrons un élément *Facture* nous y trouverons des informations sur une facture, les données deviennent bien plus souples.

L'ensemble de ces balises et attributs ainsi rédigés pour baliser les données de la facture est collectivement connu sous le nom de **vocabulaire** XML.

Puisque nous pouvons créer nos propre balises et attributs XML, il nous faut un moyen de définir un vocabulaire, afin de pouvoir le partager avec d'autres en utilisant la même syntaxe. La spécification XML 1.0 utilise les **définitions de type de document** ou **DTD** (Document Type Definition) pour y parvenir. Une DTD définit quel balisage peut être utilisé dans un document qui est supposé respecter ce vocabulaire. Par exemple, elle définit les éléments pouvant être présents dans un document, le nombre d'instance de chaque élément, et quel doit être l'ordre des éléments dans le document. Elle peut spécifier les attributs acceptés par un élément, si certains attributs sont obligatoires, si une valeur par défaut doit être retenue en cas d'absence de valeur, etc.
Ainsi, dans l'exemple précédent de facture, nous pouvons définir le balisage de façon telle que chaque élément *Facture* doive obligatoirement contenir un attribut *nomClient,* un attribut *adresseFacture,* etc.

Lorsqu'un document XML, utilisant un vocabulaire quelconque, respecte les règles de la spécification XML 1.0, il est qualifié de **bien formé.** Lorsqu'un document bien formé respecte les règles de la DTD décrivant ce vocabulaire, il est également qualifié de **valide**.

2.4 Traiter des documents XML

Les documents XML sont traités par un composant logiciel nommé **analyseur,** lisant le document XML comme texte brut. Celui-ci implémente une ou plusieurs API (Application Programming Interface), comme le modèle objet de document **DOM** (Document Object Model), ou **SAX (** Simple API for XML). L'API offre aux programmeurs un ensemble de fonctionnalités pouvant être appelées depuis un programme pour réclamer des informations à l'analyseur alors que celui-ci traite le document. Par exemple, un programme peut demander à l'analyseur de lui fournir le premier enfant de l'élément racine, ainsi que le texte qu'il contient.
Certains analyseurs sont en mesure de vérifier la conformité d'une instance de document XML à une DTD utilisé pour décrire ce vocabulaire et de contrôler si le balisage utilisé respecte le balisage prévu. Les analyseurs dotés de cette fonctionnalité portent le nom d' analyseurs **validateurs**.

2.5 Structure du document XML

Nous avons vu qu'il est possible de créer des balises et des attributs décrivant leur contenu (ce qui compose en générale la plus grande partie du balisage), mais la boîte à outils XML dispose d'autres types de balisage :

- Déclaration XML ;
- Éléments ;
- Attributs ;
- Données textuelles
- Section (CDATA) ;
- Références d'entités.
- Instructions de traitements ;
- Commentaires

2.5.1 Déclaration XML

La déclaration XML est fortement recommandée afin que les applications réceptrices sachent qu'il s'agit d'un document XML et connaissent la version utilisée.

```
<?xml version="1.0"?>
```

La déclaration XML, connue également sous le nom de **prologue XML**, se trouve en en-tête du document. La mention *xml* est écrite en minuscules.

Dans cette déclaration, on peut également définir le langage utilisé pour écrire les données XML. Cela est particulièrement important si les données contiennent des caractères n'appartenant pas au jeu de caractère ASCII anglais. On peut spécifier l'encodage du langage à l'aide de l'attribut optionnel *encoding :*

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

2.5.2 Eléments

Les éléments sont les composants les plus importants des documents XML. Tout document XML doit posséder un élément de base, dans lequel sont imbriqués tous les autres balisages. Le document suivant présente un tel élément, *ExempleDeDocument* :

```
< ?xml version="1.0" encoding="ISO-8859-1"?>
<ExempleDeDocument>
    Ceci est un exemple de document XML.
</ExempleDeDocument>
```

Cet élément de plus haut niveau porte le nom **d'élément document** ou **élément racine.**
Exemple : l'élément ***ExempleDeDocument*** est l'élément racine (ou l'élément root) du document XML ci-dessus.

Les éléments peuvent être utilisés selon l'une de ces deux façons :

- Comme dans cet exemple, avec une balise d'ouverture et une valise de fermeture où l'élément possède trois parties :
  1. Une balise d'ouverture (***<ExempleDeDocument>***)
  2. Suivie d'un contenu (la chaîne ***Ceci est un exemple de document XML.***)
  3. Suivi d'une balise de fermeture (***</ExempleDeDocument>***).

Ou

- Comme élément vide, lorsqu'il n'y a pas de contenu entre les balises d'ouverture et de fermeture : une unique balise est utilisée, avec une barre oblique avant le signe supérieur de fermeture, par exemple :

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

XML étant sensible à la casse, hormis la barre oblique, les éléments doivent donc se correspondre exactement : ***<ExempleDeDocument >*** et ***<exempleDeDocument >*** sont deux balises distinctes.

Toutes les balises s'imbriquent correctement, ce qui signifie que tout chevauchement d'éléments n'est pas autorisé. Ceci est par exemple correct :

```
<DocExemple>
    <DesDonnees> informations </DesDonnees>
</DocExemple>
```

alors que ceci est incorrect :

```
<DocExemple>
    <DesDonnees> informations
</DocExemple>
        </DesDonnees>
```

parce que la balise de fermeture </DesDonnees> se situe après la balise de fermeture </DocExemple>.

2.5.3 Attributs

Les attributs sont placés dans la balise d'ouverture d'un élément, et sont exprimés sous forme de paire **nom/valeur**. La valeur est placée entre apostrophes ou guillemets.
Par exemple :

```
< ?xml version="1.0" encoding ="ISO-8859-1" ?>
<DocExemple Auteur = "David Laurent">
    Ceci est document XML.
</DocExemple>
```

dans ce document exemple, l'élément *DocExemple* possède un attribut associé. Le **nom** de l'attribut est *Auteur* et sa **valeur** est *David Laurent.*

De même, le nom d'attribut n'apparaît qu'une seule fois dans une balise d'ouverture particulière. Ce qui suit n'est pas donc autorisé en XML :

```
<?xml version = "1.0" encoding = "ISO-8859-1" ?>
<DocExemple Auteur = "David Laurent" Auteur = "Vincent Fontaine">
    Ceci est un document XML.
<DocExemple>
```

En revanche, ce qui suit est parfaitement acceptable :

```
< ?xml version = "1.0" encoding = "ISO-8859-1" ?>
<DocExemple >
        <Phrase Auteur = "David Laurent" >
        Ceci est une phrase de Mr David Laurent.
        </Phrase>
        <Phrase Auteur = "Vincent Fontaine">
        Ceci est une phrase de Mr Vincent Fontaine.
        </Phrase>
</DocExemple>
```

Un attribut est associé à un élément (puisqu'il n'apparaît que dans une balise d'ouverture ou d'élément vide, ce qui est toujours une partie d'élément). De même, un attribut ne peut pas contenir les caractères <, &, ' ou".

Enfin, l'ordre dans lequel on place les attributs dans un élément n'importe pas, ce qui signifie que les deux documents suivants sont sémantiquement identique.

```
< ?xml version = "1.0" encoding = "ISO-8859-1"?>
<DocExemple >
<Phrase Auteur = "David Laurent"  DateDeCreation="12/04/02" >
      Ceci est une phrase de Mr David Laurent.
   </Phrase>
</DocExemple>
```

et

```
< ?xml version = "1.0" encoding = "ISO-8859-1" ?>
<DocExemple >
        <Phrase DateDeCreation="12/04/02"  Auteur = "David Laurent" >
   Ceci est une phrase de Mr David Laurent.
   </Phrase>
</DocExemple>
```

2.5.4 Données textuelles

Dans l'exemple ci-dessus, l'élément *Phrase* contient le texte *Ceci est une phrase de Mr David Laurent.* Il s'agit du contenu d'élément, possédant un nom particulier en XML : il est appelé donnée textuelle.
En XML, tout bloc de texte contigu situé dans un élément est traité comme une seule unité par l'analyseur ou en cas de manipulation. Voyons l'exemple suivant :

```
< ?xml version = "1.0" encoding = "ISO-8859-1"?>
<DocExemple >
Ceci est une <Phrase > phrase </Phrase> de Mr David Laurent.
</DocExemple>
```

Dans cet exemple, l'élément *DocExemple* contient trois éléments de données :
Le bloc de texte *Ceci est une*
L'élément *Phrase*
Le bloc de texte de Mr David Laurent.
A son tour, l'élément *Phrase* contient un seule morceau de données :
Le bloc de texte *phrase.*

Une donnée textuelle est une chaîne de caractères qui respecte la syntaxe XML (c'est-à-dire qui ne comporte pas le symbole le et commercial (&) et le symbole inférieur (<)) et peut apparaître n'importe où dans un élément, ou comme valeur d'un attribut. Les symboles & et < sont interprétés par les analyseurs XML comme constituant le début d'une balisage particulière plus précisément une instance d'entité et le début de toute balise bien formée. Si on veut inclure ces caractères dans le document XML, que ce soit comme valeur d'attribut ou dans un bloc de texte, nous utilisons soit une section CDATA soit des entités.

2.5.5 Section CDATA

Si on souhaite incorporer du balisage au document XML, une façon d'y parvenir consiste à l'encapsuler dans une **section CDATA.** Les sections CDATA ne sont pas analysées par les analyseurs; les règles de syntaxe XML ne s'y appliquent pas. Par exemple, le document suivant ne produira pas le résultat souhaité :

```
<?xml version ="1.0" encoding = "ISO-8859-1" ?>
<DeclarationElement>
En XML pour déclarer un élément, on utilise une balise d'ouverture
d'élément : <NomDeBalise>
</DeclarationElement>
```

Lorsque l'analyseur tente d'analyser ce document, il identifie la chaîne *<NomDeBalise>* comme étant le début d'un nouveau élément, puis déclare ne pas pouvoir identifier de balise de fermeture correspondante *</NomDeBalise>*. Une solution serait d'encapsuler le texte en question dans une section CDATA :

```
<?xml version ="1.0" encoding = "ISO-8859-1" ?>
<DeclarationElement>
<![CDATA[En XML pour déclarer un élément, on utilise une balise d'ouverture
d'élément: <NomDeBalise>]]>
</DeclarationElement>
```

Une section CDATA débute par un marqueur de début de CDATA :
< ![CDATA[
et se termine par le marqueur de fin de CDATA :
]]>

2.5.6 Références d'entités

Une référence à une entité sert à inclure une unité d'information dans le document XML. Cela permet par exemple d'accéder aux caractères absents du clavier ou réservés à XML.

```
<?xml version ="1.0" encoding = "ISO-8859-1" ?>
<DeclarationElement>
En XML pour déclarer un élément, on utilise une balise d'ouverture d'élément :
    &lt ;NomDeBalise>
</DeclarationElement>
```

Dans cet exemple, la chaîne *&lt ;* est une référence d'entité. Plus spécifiquement, il s'agit d'une instance de l'entité nommée *lt*. Les références d'entités des documents XML (par opposition à celles des DTD comme on le verra par après) débutent toujours par une esperluette (&) et se terminent par un point virgule. Lorsqu'un analyseur rencontre une référence d'entité, il va dans la table de symboles créée lors de l'analyse de la DTD du document (voir partie DTD plus loin) et en extrait la chaîne adéquate. Si elle existe, il substitue alors cette chaîne à l'instance d'entité. Les entités ainsi traitées portent le nom d'**entités analysables.** Il est également possible de déclarer dans un document XML des **entités non analysables.**

Deux types d'entités analysables peuvent être instanciées dans un document XML :

1. Les entités internes, dont le contenu de remplacement est incorporé dans la DTD du document.
2. Les entités externes, pointant via une URL (Unified Ressource Locator) vers un emplacement contenant le texte de remplacement.

Il existe quelques entités standards définies pour les documents XML.

Entité	Caractère
&lt ;	<
&gt ;	>
&amp ;	&

&apos ;	'
&quot ;	"

Tout analyseur compatible XML reconnaît automatiquement ces entités et les remplace par leur valeur appropriée.

2.5.7 Instructions de traitement

Les instructions de traitement permettent de transmettre des informations aux outils chargés de traiter le document XML. Une instruction de traitement débute par le marqueur de début d'instruction de traitement < ? et se termine par la chaîne ?>.
Une déclaration d'instruction de traitement comporte deux parties : la **cible** de l'instruction de traitement et la chaîne sur laquelle l'instruction opère. Tout ce qui est présent dans la déclaration d'instruction de traitement avant le premier espace vierge est considéré comme étant la cible de l'instruction de traitement, tout ce qui se trouve après l'espace vierge est la chaîne utilisée pour diriger le comportement du processeur. Dans l'exemple suivant :

```
<?xml:stylesheet type = "texte/xsl"» href = "menu.xsl'' ?>
```

la cible est *xml:stylesheet,* et la chaîne d'information complémentaire constitue le reste du texte à l'exclusion du point d'interrogation.
L'instruction de traitement relative à XML, *<?xml ... ?>*, sert principalement à préciser au logiciel d'analyse l'encodage du document. Ces instructions peuvent aussi être utilisées pour indiquer la feuille de styles XSL (eXtensible Stylesheet Language) qui doit être employée pour visualiser le document; par exemple, dans l'exemple ci-dessus, l'instruction de traitement permet de visualiser le document avec la feuille de style *menu.xsl.*

2.5.8 Commentaires

Les commentaires dans un document XML ont exactement la même syntaxe que pour les commentaires de HTML. Le début de la balise de commentaire commence par < ! -- et se termine par la chaîne de marqueur de fin de commentaire --> :

```
<?xml version = "1.0'' encoding = "ISO-8859-1'' ?>
<!-- Ce document est crée le 04/04/02 -->
<EtudiantFiche/>
```

2.6 Espaces de noms

Si on souhaite utiliser plusieurs vocabulaires XML dans un même document, on peut y parvenir à l'aide d'**espaces de noms.** Les espaces de noms sont des mécanismes d'identification permettant d'identifier de manière non ambiguë les éléments et attributs utilisés par chacun des vocabulaires qui l'utilisent. Les espaces de noms ont un effet de granularité fine et permettent d'identifier le vocabulaire auquel appartient chaque élément ou attribut d'un document. L'objectif des espaces de noms est de permettre à plusieurs vocabulaires d'être utilisés simultanément au sein d'un même document.
Pour pouvoir utiliser un espace de noms, on doit d'abord le déclarer :

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

```
<DeclarationLiens xmlns :xlink="http://www.3wc.org/1999/xlink">
   <MonLien xlink :type="simple" xlink :href="MonDoc.xml"/>
</DeclarationLiens>
```

Les espaces de noms sont déclarés en dotant un élément d'un attribut possédant le préfixe *xmlns :*. Un analyseur compatible espace de noms interprète tout attribut possédant ce préfixe comme définition d'espace de noms. La chaîne suivant les deux-points dans le noms de l'attribut est le **préfixe** qui sera utilisé pour déclarer les attributs et les éléments conformément à l'espace de noms défini (il s'agit dans cet exemple de *xlink*). La valeur de l'attribut contient alors le nom de l'espace de noms : un URI (Uniform Resource Identifier) identifiant cet espace de noms. Ce URI est unique et persistant. Dans cet exemple, le préfixe d'espace de noms *xlink :* est déclaré comme correspondant à l'espace de noms http://www.3wc.org/1999/xlink. Un processeur compatible Xlink reconnaît alors que l'élément *Monlien* représente un Xlink simple pointant vers le fichier *MonDoc.xml*, et accomplit toute action nécessaire afin d'indiquer à l'utilisateur la présence de ce lien simple.

Un espace de noms n'est valide que pour l'élément pour lequel il est déclaré. Par exemple, un analyseur lisant le document suivant identifie correctement la présence du lien simple :

```
< ?xml version ="1.0" encoding = "ISO-8859-1"?>
<DeclarationLiens >
   <MonLien xmlns :xlink="http://www.3wc.org/1999/xlink"
xlink :type="simple" xlink :href="MonDoc.xml"/>
</DeclarationLiens>
```

En revanche, le processeur ne reconnaît pas le lien dans l'élément suivant, parce que la déclaration du lien est en dehors de la portée de la déclaration de l'espace de noms :

```
<?xml version ="1.0" encoding = "ISO-8859-1" ?>
<DeclarationLiens >
   <MonTextLien xmlns :xlink="http://www.3wc.org/1999/xlink">
   Ceci est le texte du lien
   </MonTextLien>
<MonLien xlink :type= « simple » xlink :href= « MonDoc.xml »/>
</DeclarationLiens>
```

De préférence, on déclare tous les espaces de noms du document en tant qu'attributs de l'élément **racine** (élément de premier niveau, qui constitue le point d'entrée dans la hiérarchie d'information du document.), afin de garantir que leur portée englobe la totalité du document. Un document XML peut posséder un nombre quelconque d'espace de noms. les espaces de noms sont très importants lors des transformations XSLT (eXtensible Stylesheet Language Transformation,voir chapitre XSLT) : XSLT faisant correspondre à la fois le nom local et l'espace de noms, il faut donc déclarer correctement les espaces de noms dans une feuille de style XSLT pour que celui-ci reconnaisse les éléments correspondants le document.

3.Définitions de documents types : DTD

3.1 Introduction

La DTD (Document Type Definition) est un mécanisme proposé dans la spécification XML 1.0 pour contrôler le contenu pouvant apparaître dans un document XML. Le rôle de la DTD est de définir le vocabulaire et la structure d'un document XML. Une DTD est caractérisée par un ensemble de règles. Elles permettent de spécifier les éléments et leurs attributs, ainsi que leur relations, leurs ordres et leurs fréquences d'apparition dans le document XML. Une DTD est un héritage de SGML, elle n'est pas un document XML.

Certains analyseurs XML sont capables de valider un document XML par rapport à sa DTD, lançant une erreur s'ils détectent le non respect d'une règle de celui-ci. Ce type d'analyseur est appelé analyseur validateur. Si un document XML respecte les règles de la DTD, il est qualifié de document XML valide, plutôt que simplement bien formé.

3.2 Déclarations de DTD

La DTD peut être incluse au fichier XML, on dit alors qu'elle est interne, ou externe et dissociée du fichier XML. La déclaration externe est spécifiée dans le document XML à l'aide d'une déclaration de type de document suivant la syntaxe :

< !DOCTYPE ...>

Exemple :

```
<!DOCTYPE MonDocXML SYSTEM "http://www.monsite.com/MonDocXML.dtd">
```

On pointe ici vers une DTD nommé *MonDocXML*. Le nom de la DTD correspond à celui de l'élément racine [3] du document XML, si bien que l'élément racine des documents XML écrits conformément à cette DTD doit être <MonDocXML>. L'utilisation du mot clé SYSTEM indique que la DTD est un fichier externe, dont l'emplacement est spécifié entre les guillemets. Ce type de DTD est connu sous le terme de DTD **externe**, puisqu'elle se trouve dans un fichier externe.

Une DTD peut également figurer dans une déclaration de type de document, auquel cas elle est désignée par le terme de DTD **interne**, comme :

```
< !DOCTYPE MonDocXML [
< !ELEMENT MonDocXML (#PCDATA)>
]>
```

Dans une DTD interne, toutes les contraintes sur le contenu du document sont spécifiées comme déclaration entre crochets [...].

[3] Chaque document XML bien formé est formé d'un élément unique, qui contient tous les autres.

Lorsqu'un analyseur validateur rencontre une DTD externe, il accède à la ressource indiquée dans l'URI (Uniform Ressource Identifier) et en extrait les contraintes de document. Il se comporte ensuite comme si ces contraintes avaient été déclarées en ligne.
Il est utile de posséder une DTD externe au document lui-même : cela permet à plusieurs documents d'utiliser les mêmes règles sans avoir à répéter le même ensemble de contraintes dans chaque document devant respecter le même ensemble de règles.

3.3 Déclaration d'élément

La déclaration des types éléments permet de décrire le vocabulaire utilisable dans un document XML. Elle permet aussi de contraindre les éléments utilisés dans l'application et de spécifier l'ordre dans lequel ces derniers doivent apparaître. Ils correspondent aux balises du document XML. Un élément est désigné par un nom et il peut contenir d'autres éléments, des données ou être sans données.
Un élément est déclaré à l'aide de la syntaxe suivante :

< !ELEMENT NomDeElement (modéleContenu)>

où *NomDeElement* est le nom de l'élément et *modèleContenu* ce que peut contenir cet élément. Dans cet exemple on déclare un élément nommé *MonDocXML*. Celui-ci ne contient que du texte : cela est défini à l'aide de la syntaxe #PCDATA.

Exemple :

<!ELEMENT NomDocXML (#PCDATA)>

Il existe cinq types différents de contenu d'élément pouvant être déclarés dans une déclaration d'élément :
Contenu d'élément;
Contenu texte seul ;
Contenu mixte;
Modèle de contenu vide (EMPTY);
Modèle de contenu quelconque (ANY).

3.3.1 Contenu d'éléments

Dans le premier type de déclaration d'élément, l'élément est défini comme ne contenant que d'autres éléments. La déclaration spécifie l'ordre et les contraintes de nombre de chacun des éléments contenus. Par exemple, la déclaration :

<!ELEMENT Etudiant(NomEtudiant, PrenomEtudiant, DateNaissanceEtudiant, AnneeEtude, NomUniversite)>

stipule que, dans l'élément Etudiant, les éléments NomEtudiant, PrenomEtudiant, DateNaissanceEtudiant , AnneeEtude et NomUniversite doivent apparaître chacun exactement une fois, dans cet ordre. Si bien que, pour la DTD *Etudiant.dtd* suivante :

<!ELEMENT Etudiant(NomEtudiant, PrenomEtudiant, DateNaissanceEtudiant, NomUniversite)>
<!ELEMENT NomEtudiant (#PCDATA)>
<!ELEMENT PrenomEtudiant (#PCDATA)>
<!ELEMENT DateNaissanceEtudiant (#PCDATA)>

```
<!ELEMENT NomUniversite (#PCDATA)>
```

Le document XML exemple qui suit respecte la DTD :

```
<?xml version ="1.0" stadalone ="no"?>
<!DOCTYPE Etudiant SYSTEM «Etudiant.dtd»>
<Etudiant>
        <NomEtudiant >Nom de l'etudiant </NomEtudiant>
        <PrenomEtudiant > Prenom de l'étudiant </PrenomEtudiant>
        <DateNaissanceEtudiant> Date de naissance de l'étudiant
        </DateNaissanceEtudiant>
        <NomUniversite>Nom de l'Université </NomUniversite>
</Etudiant>
```

Ce n'est pas le cas des trois exemples suivants :

```
< ?xml version ="1.0" stadalone ="no" ?>
< !DOCTYPE Etudiant SYSTEM "Etudiant.dtd">
<Etudiant>
        <NomEtudiant>Nom de l'etudiant </NomEtudiant>
        <PrenomEtudiant> Prenom de l'étudiant </PrenomEtudiant>
</Etudiant>
< !—l'élément DateNaissanceEtudiant et NomUniversite sont manquants -->
```

```
<?xml version ="1.0" stadalone ="no" ?>
<!DOCTYPE Etudiant SYSTEM "Etudiant.dtd">
<Etudiant>
        < DateNaissanceEtudiant > Date de naissance de l'étudiant </
        DateNaissanceEtudiant >
        < NomEtudiant >Nom de l'etudiant </ NomEtudiant >
        < PrenomEtudiant > Prenom de l'étudiant </ PrenomEtudiant >
        < NomUniversite >Nom de l'Université </ NomUniversite >
</Etudiant>
< !—les élément ne sont pas dans le bon ordre -->
```

```
< ?xml version ="1.0" stadalone ="no" ?>
< !DOCTYPE Etudiant SYSTEM "Etudiant.dtd">
<Etudiant>
        < NomEtudiant >Nom de l'etudiant </ NomEtudiant >
        < NomEtudiant >Nom de l'etudiant </ NomEtudiant >
        < PrenomEtudiant > Prenom de l'étudiant </ PrenomEtudiant >
        < DateNaissanceEtudiant > Date de naissance de l'étudiant
        </ DateNaissanceEtudiant >
        < NomUniversite >Nom de l'Université </ NomUniversite >
</Etudiant>
< !—trop d'éléments NomEtudiant - ->
```

Il est également possible de définir un ensemble d'éléments, dont seul l'un peut (doit) être présent. Cela est indiqué en séparant les options à l'aide du caractère de barre verticale |. De ce fait, la déclaration

```
< !ELEMENT Couleur ( blanc | rouge | noire | jaune) >
```

stipule que l'élément Couleur peut contenir un élément blanc, un élément rouge, un élément noir ou un élément jaune, mais seulement l'un d'entre eux.

Les éléments enfants spécifiés dans la déclaration d'élément peuvent aussi comporter des suffixes de cardinalité. Ceux-ci indiquent le nombre d'occurrences de chaque élément pouvant apparaître à cet emplacement du contenu d'élément. Il existe quatre opérateurs d'occurrences :

Opérateur	Signification
?	Facultatif (peut être présent 0 ou une fois) Exemple : *< !ELEMENT Personne (nom, prenom, telephone, portable ?) >* définit un type d'élément Personne contenant les types d'éléments nom, prenom, telephone, et de manière optionnelle portable, dans cette ordre.
*	Facultatif multiple (peut être 0 fois ou plus) Exemple : *< !ELEMENT Personne (nom, prenom, telephone*) >* définit un type d'élément *Personne* contenant les types d'éléments *nom*, *prenom* et zéro ou plusieurs fois le type d'élément *telephone*.
(pas de suffixe)	Obligatoire (doit être présent exactement une fois)
+	Obligatoire multiple (doit être présent 1 ou plus) Exemple : *< !ELEMENT Personne (nom, prenom, telephone+) >* définit un type d'élément *Personne* contenant les types d'éléments *nom*, *prenom* et une ou plusieurs fois le type d'élément *telephone*.

Des éléments enfant peuvent en outre être rassemblés dans la déclaration d'élément. ils peuvent être regroupés soit dans une liste ordonnée, soit dans une liste de choix. Ces regroupements peuvent aussi comporter les opérateurs d'occurrences présentés dans le tableau précédent. Voir les exemples ci-dessous.

Exemple 1 :

```
<!Element MonDoc (A*, B+ ,C ?)>
```

les fragments XML suivants pour l'élément MonDoc sont tous valides :

```
<MonDoc>
<A> Du texte </A>
```

```
<B> Encore du texte </B>
</MonDoc>
```

```
<MonDoc>
   <B> Encore du texte </B>
   <B> Encore du texte </B>
   <B> Encore du texte </B>
</MonDoc>
```

```
<MonDoc>
        <A> Du texte </A>
        <A> Du texte </A>
        <B> Encore du texte </B>
        <C> Et encore du texte </C>
</MonDoc>
```

Exemple 2 :

```
< !Element MonDoc ((A | B) ,C )>
```

Cet exemple stipule que l'élément *MonDoc* doit d'abord contenir un élément *A* ou un élément *B*, ensuite être suivi d'un élément *C*.
Les deux exemples suivants sont donc les seuls contenus valides de *MonDoc* :

```
<MonDoc>
        <A> Du texte </A>
        <C> Et encore du texte </C>
</MonDoc>
```

Exemple 3 :

```
< !Element MonDoc ((A+ , B ?) | C* )>
```

Dans cet exemple, *MonDoc* doit contenir au moins un élément *A* suivi facultativement d'un élément *B*, ce groupe pouvant être remplacé par zéro ou plusieurs éléments *C*.
Les exemples qui suivent sont valides :

```
<MonDoc>
        <A> Du texte </A>
        <A> Du texte </A>
        <A> Du texte </A>
</MonDoc>
```

```
<MonDoc>
        <A> Du texte </A>
        <B> Encore du texte </B>
</MonDoc>
```

3.3.2 Contenu texte seul

Lorsqu'un élément ne peut contenir que du texte, un tel élément est défini comme suit :

```
< !Element MonDoc   (#PCDATA)>
```

Exemple :

```
<!Element Etudiant (Nom, Prenom, Adresse, CodePostal , Université, AnneeEtude)>
<!Element Nom (#PCDATA)>
<!Element Prenom (#PCDATA)>
<!Element Adresse (#PCDATA)>
<!Element  CodePostal (#PCDATA)>
<!Element Universite (#PCDATA)>
<!Element AnneeEtude (#PCDATA)>
```

le document XML qui suit est valide.

```
<Etudiant>
   <Nom> Vincent </Nom>
   <Prenom> Fontaine </Prenom>
   <Adresse>clos de la jeunesse n°45 Namur<Adresse>
   <CodePostal>5000</CodePosal>
   <Universte>Facultés Universitaires Notre-Dame de La Paix<Universite>
   <AnneeEtude> 2ème Licence Informatique </AnneeEtude>
</Etudiant>
```

3.3.3 Contenu mixte

Un élément peut être également déclaré comme renfermant un mélange d'autres éléments et du texte. Tout élément ainsi déclaré peut contenir n'importe lesquels des éléments définis dans la liste de contenu, dans un ordre quelconque, du texte se trouvant placé entre eux. Une déclaration d'élément à contenu mixte se présente comme suit :

```
< !ELEMENT MonDoc (#PCDATA | A | B | C )*>
< !ELEMENT  A  (#PCDATA) >
< !ELEMENT  B  (#PCDATA)>
< !ELEMENT  C  (#PCDATA)>
```

L'expression #PCDATA doit obligatoirement figurer en tête dans la liste séparée par les barres verticales (symbole OU) : il signale que l'élément peut comporter du texte. Les autres éléments de la liste peuvent être présents ou non. On remarque qu'il n'existe aucune contrainte sur l'ordre ou le nombre d'occurrences des éléments. C'est la seule déclaration licite pour du contenu mixte : on n'est pas autorisé à définir l'emplacement ou le nombre des occurrences des différents sous-éléments d'un élément à contenu mixte. Les fragments présentés ci-dessous sont donc tous valides :

```
<MonDoc>
   Voici du <A>texte</A> et des<B> éléments</B>ensembles.
</MonDoc>
```

```
<MonDoc>
   <C/> <C/> <C/> Plusieurs éléments de C.
</MonDoc>
```

```
<MonDoc>
   Cet élément est totalement dépourvu d'éléments enfants.
</MonDoc>
```

Il est recommandé d'éviter le modèle de contenu mixte, lorsque cela est possible, dans la conception de structures XML.

3.3.4 Modèle de contenu EMPTY

Le modèle de contenu EMPTY permet de déclarer un élément vide qui n'a pas de contenu. Il peut cependant posséder des attributs. Un élément vide est déclaré comme suit :

```
<!Element MonDoc EMPTY>
```

Tout élément ainsi déclaré doit se présenter sous l'une des deux formes suivantes :

```
<MonDoc/>
<MonDoc></MonDoc>
```

3.3.5 Modèle de contenu ANY

Si un élément est déclaré comme possédant un modèle de contenu ANY, cela indique qu'il peut contenir n'importe quel contenu du moment que les éléments enfants respectent leur propre modèle de contenu, comme défini ailleurs dans la DTD. Un tel élément est déclaré comme suit :

```
< !ELEMENT MonDoc ANY>
```

Les exemples suivant sont valides pour cette déclaration :

```
<MonDoc>
Voici n'importe quoi.
</MonDoc>
<MonDoc>
   <A><B><C><D><E></E></D></C></B></A>
</MonDoc>
```

```
<MonDoc>
   <A>Ceci</A><B>est </B><C>bien</C><D>balisé</D>
</MonDoc>
```

3.4 Déclaration d'attribut

La déclaration d'attribut permet de définir les attributs devant ou pouvant apparaître dans un élément donné de la DTD afin de le completer ou de l'enrichir. La syntaxe générale est la suivante :

< !ATTLIST nom-élément définition-attribut*>

Une définition d'attribut définition-attribut ressemble à ce qui suit :

Nom-attribut type-attribut déclaration-par-défaut

Exemple de DTD avec une déclaration d'attribut :

```
<!ELEMENT MonDoc EMPTY>
< !ATTLIST MonDoc
        Texte CDATA #REQUIRED>
```

La DTD ci-dessus stipule que l'élément *MonDoc*, qui doit être vide, possède un attribut obligatoire nommé *Texte* pouvant recevoir toute valeur chaîne de caractères. Ainsi le fragment suivant est valide :

```
<MonDoc Texte ="mon document"/>
```

Les différents types d'attribut pouvant être définis dans une DTD sont les suivants :

```
CDATA
ID, IDREF ou IDREFS
ENTITY ou ENTITIES
NMTOKEN
```

3.5 Ensemble des valeurs énumérées

Une autre fonctionnalité intéressante des déclarations d'attributs dans les DTD est la possibilité de contrôler les valeurs autorisées pour cet attribut. Cela est très utile si on possède des points de données correspondant à un ensemble de valeurs parfaitement défini. Par exemple :

```
< !ELEMENT MonDoc EMPTY>
< !ATTLIST MonDoc
   Couleur ( Bleu | Rouge | Vert ) #REQUIRED>
```

Comme une déclaration d'ensemble de valeurs énumérées une liste est fournie, séparée par une barre verticale. Pour ce fragment de DTD, les fragment de document suivant est valide :

```
<MonDoc Couleur ="Bleu"/>
```

Tandis que ce n'est pas le cas de celui-ci :

```
<MonDoc Couleur ="Jaune"/>
```

Il s'agit d'un des moyens de limiter des valeurs autorisées pour un point de données dans une DTD.

3.6 Déclarations d'entité

Les entités permettent de mettre en place un système de substitution de contenu (texte ou autre) analysable ou non par l'analyseur XML. Cela permet d'associer une symbolique à une information récurente tout au long du document de manière homogène.
Il existent deux types d'entités pouvant être déclarées :
- Les entités internes
- Les entités externes

3.6.1 Entités internes

Les entités internes contiennent leur valeur de remplacement dans leur déclaration. Lorsqu'un analyseur rencontre une référence vers l'entité interne spécifiée, il lui substitue la valeur de remplacement trouvée dans la déclaration de cette entité.
Par exemple :

```
<!ENTITY MonSite "Construction">
<!ELEMENT Concerne (#PCDATA)>
```

```
<Concerne>
   Ce site est actuellement au stade de &MonSite ;.
</Concerne>
```

Lorsque l'analyseur lit ce document, il substitue la chaîne présente dans la déclaration d'entité à la chaîne de référence d'entité « &MonSite ; ». Pour un processeur le document ressemble à :

```
<Concerne>
   Ce site est actuellement au stade de Construction.
</Concerne>
```

3.6.2 Entités externes

Par contre les entités externes font référence à des ressources externes au contexte du document XML.

```
< ! ENTITY SonFichier SYSTEM " http://mon.url.versLeFichier/Fichier.xml">
```

En déclarant ainsi des entités, leur contenu est récupéré et substitué en lieu et places de leurs références. Elles sont connues sous le nom d'entités analysables car elles doivent respecter les règles XML. Il est également possible de déclarer des entités non analysables, comme cela a été mentionné précédemment.

On dispose de la DTD suivante, nommée *facture.dtd*

```
<!ENTITY FactureLigneDeCommande SYSTEM "LigneDeCommande.xml">
<!ELEMENT Facture (NomClient, LigneDeCommande+)>
```

```
<!ELEMENT NomClient (#PCDATA)>
<!ELEMENT LigneDeCommande (Article, Quantité, Prix)>
<!ELEMENT Article (#PCDATA)>
<!ELEMENT Quantité (#PCDATA)>
<!ELEMENT Prix (#PCDATA)>
```

ainsi que le document suivant , nommé *facture.xml*

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<!DOCTYPE Facture SYSTEM "facture.dtd" >
<Facture>
   <NomClient>Julien Fontaine</NomClient>
   &FactureLigneDeCommande;
</Facture>
```

Si le document *LigneDeCommandes.xml* contient ce qui suit :

```
<LigneDeCommande>
   <Article> bouquins</Article>
   <Quantite>20</Quantite>
   <Prix> 300</Prix>
</LigneDeCommande>

<LigneDeCommande>
   <Article> boulons</Article>
   <Quantite>150</Quantite>
   <Prix> 100</Prix>
</LigneDeCommande>
```

alors, après substitution, le document ressemblera à :

```
< ?xml version = "1.0" encoding = "ISO-8859-1" ?>
< !DOCTYPE Facture SYSTEM "facture.dtd">
<Facture>
   <NomClient>Julien Fontaine</NomClient>
   <LigneDeCommande>
      <Article> bouquins</Article>
      <Quantite>20</Quantite>
      <Prix> 300</Prix>
   </LigneDeCommande>

   <LigneDeCommande>
      <Article> boulons</Article>
      <Quantite>150</Quantite>
      <Prix> 100</Prix>
   </LigneDeCommande>
</Facture>
```

3.6.3 Entité paramètres

On peut également utiliser des entités pour construire les éléments ou leurs attributs dans une DTD. Ces entités portent le nom d'entités paramètres. En voici un exemple :

```
<!ENTITY % ChoixCouleur1 " Rouge ">
<!ENTITY % ChoixCouleur2 " Vert ">
<!ENTITY % ChoixCouleur3 " Bleu ">
<!ELEMENT MonDoc EMPTY>
<!ATTLIST MonDoc (%ChoixCouleur1 ; | %ChoixCouleur2 ;| % ChoixCouleur3 ;)>
```

Pour un analyseur validateur la DTD ressemble à :

```
<!ELEMENT MonDoc EMPTY>
<!ATTLIST MonDoc (Rouge | Vert | Bleu)>
```

3.7 Les Limites des DTD

Les DTD proviennent en réalité de SGML (Standard Generalized Markup Language). Il s'agissait de contraindre le langage et rien n'a changé avec XML.
Pour du texte, il est envisageable de laisser une certaine liberté aux auteurs, mais dès qu'il s'agit de travailler avec des données, il est important d'être un peu plus rigoureux

Les DTD ne permettent pas :
-de fournir un ensemble de type de données comme par exemples les chaînes, les données booléennes ou les nombres à virgule flottante,
- ne sont pas des document XML,
-n'offrent pas des types de données permettant de créer des types définis par l'utilisateur avec définition de contraintes.
Les DTD ne résolvent pas les problèmes des collisions des noms des éléments et ne supporte pas les espaces de noms.

4. Schéma XML

4.1 Introduction

Le langage DTD a des inconvénients : syntaxe différente de XML, pas de définition de types, pas de notion d'espace de noms (d'où risque de collisions entre symboles importés et symboles définis localement). Par contre un schéma XML permet de spécifier le même genre d'information qu'une DTD, c'est à dire la syntaxe d'une classe de document XML, en définissant les éléments et les attributs ainsi que les contraintes sur les valeurs de ceux-ci.

4.2 Déclaration

Un schéma XML est déclaré comme suit :

```
<xsd :schema xmlns :xsd= « http://www.w3.org/2001/XMLSchema »>
...
</xsd :schema>
```

Le préfixe ***xsd*** est choisie arbitrairement et va permettre d'accéder, dans l'élément *xsd:schema*, à tous les symboles définies dans la syntaxe des schémas.

4.3 Principes

Les spécifications sur les Schémas sont divisées en deux parties : une sur les *structures* [recommandation XML Schema Part 1: Structures] et une sur les *types de données* [recommandation XML Schema Part 2: Datatypes].

4.3.1 Les types de données

Le document Schéma XML fournit deux types de données principaux.

- Les types de données primitifs, non définis à partir des autres types. Ils constituent la base fondamentale des Schémas XML et sont à l'origine de tous les autres types. Ainsi, String, Boolean, Float, Double, ID, IDREF et Entity sont des exemples de types de données primitifs dans les schémas XML.
- Les types de données dérivés qui sont définis à partir des types existants. Les nouveaux types sont crés soit d'un type primitif soit d'un autre type dérivé. Ainsi, le type CDATA de Schéma XML, qui représente une chaîne de caractères pouvant comporter des espaces, dérive du type primitif String qui constitue son type de base.

Ces types de données nous permettent d'élaborer des types simples et des types complexes.

4.3.1.1 Types simples.

Un élément ne contenant pas de sous-élément ni d'attributs est considéré comme de type simple .

Exemple :

```
<auteur>NomAuteur</auteur>.
```

4.3.1.2 Types complexes :

Un élément contenant des sous-élément, des attributs ou les deux à la fois est considéré de type complexe.
Exemple :

```
<personne>
        <nom> Nom</nom>
        <prenom> Prénom</prenom>
</personne>
```

4.3.2 Structure

D'un point de vue structurel, un schéma XML se définit comme étant l'assemblage, sous forme d'arbre XML, d'un ensemble de composants. Ils en existent plusieurs, dont les principaux sont :
- Composants de définition de types, les simples ou les complexes.
Ils permettent de typer les éléments et les attributs pour leur affecter un modèle de contenu. Ainsi une adresse de livraison et une adresse de facturation relèverons toutes deux d'un même type adresse.

- Composants de déclarations : éléments, attributs et notations.
Comme avec les DTD, ces composant permettent de déclarer des éléments avec leur modèles de contenus ainsi que les attributs. Les déclarations peuvent se reposer sur les composants de définition de types.

- Composants groupes : pour la factorisation d'informations de modèles de contenus ou d'attributs.

Ces composants permettent de définir des modèles qui seront souvent réutilisés, soit pour définir des composants de déclaration, soit pour définir des composants de définition de types. Dans les DTD, ces groupes sont formalisés par des entités paramètres, qui permettent, par exemple, de définir tous les composants blocks(listes, remarques, paragraphes, etc.) nécessaires lors de la définitions de modèles de contenus.

4.4 Exemple simple de schéma

On souhaite créer un schéma pour cette exemple *Commande* :

```
<Commande>
        <Client
   nom=" Fontaine "
   prenom= " Albert "
   AdresseElectronique = " fontaine.albert@yahoo.com " />
   <Facture type= " string " />
</Commande>
```

La DTD correspondante pour imposer des contraintes au document ci-dessus, pourrait ressembler à celle-ci :

```
< !ELEMENT Commande (Client, Facture)
< !ELEMENT Client>
< !ATTLIST Client
   nom   CDATA  #REQUIRED
   prenom  CDATA #REQUIRED
   AdresseElectronique CDATA  #IMPLIED>
< !ELEMENT Facture (#PCDATA)>
```

Le schéma XML correspondante est le suivant :

```
<xsd :schema xmlns :xsd= " http://www.w3.org/2001/XMLSchema " >
  <xsd :element name = " Commande " type= " TypeCommande " />
  <xsd :complexType  name = " TypeCommande " >
   <xsd :sequence>
      <xsd :element  name = " Client " type = " TypeClient " >
      <xsd :element  name = " Facture " type = " string " >
   </xsd :sequence>
  </xsd :complexType>
  <xsd :complexType  name = " TypeClient " >
   <attribute name = " nom " type = " string "  use = " required ">
   <attribute name = " prenom " type = " string " use = " required " >
   <attributte name = " AdresseElectronique " type = " string "  use = "required">
   </xsd :complexType>
</xsd :schema>
```

On note qu'on a également définit le **type** de l'élément *Commande* et non son modèle de contenu.
L'association d'un nom à un type constitue la déclaration d'un élément. La nature de ce type est spécifié à part au sein d'un élément ***complexType***.
Un élément dont le contenu est de type simple peut être définit dans un élément tel que *Facture*. En cas de type complexe, les éléments doivent être d'abord déclarés à l'aide de deux attributs nommés respectivement ***name*** et ***type***. L'attribut ***name*** correspond au nom de l'élément. Quant à ***type***, il est utiliser pour associer le nom avec la description du contenu de son type.

On examine maintenant l'élaboration du document en commençant par l'élément racine du schéma.
L'élément <***schema***> contient l'espace de nom pour la spécification du schéma.

```
<xsd :schema xmlns :xsd= " http://www.w3.org/2001/XMLSchema " >
...Déclaration de schéma
</xsd :schema>
```

Ensuite vient la déclaration de l'élément racine *Commande* :

```
<xsd :element name = " Commande »  type= « TypeCommande " />
```

En définissant le nom de l'élément à l'aide de l'attribut ***name***, nous avons également un attribut ***type*** dont la valeur est *TypeCommande*. En effet, l'élément *Commande* n'a ni contenu textuel ni type intégré : il dispose de deux enfants dont l'un a plusieurs attributs. Nous définissons le contenu de cet élément dans <***complexType***> :

```
<xsd :complexType  name = « TypeCommade »>
<xsd :sequence>
      <xsd :element  name = « Client » type = « TypeClient »>
      <xsd :element  name = « Facture » type = « string »>
   </xsd :sequence>
</xsd :complexType>
```

Cet exemple illustre la manière de définir le type *TypeCommande*, qui est dans ce cas, le modèle de contenu de l'élément *Commande*. L'attribut ***name*** de la balise d'ouverture <***complexType***> renferme la même valeur que celle de l'attribut type déclaré dans l'élément *Commande*. Par une telle complémentarité, les modèles de contenu d'éléments peuvent être définis.
L'élément ***sequence*** indique que les éléments enfants contenus en son sein doivent être présents dans le document et apparaître les uns à la suite les autres. Si leur présence était facultative ou si leur ordre d'apparition pouvait différer, ils figureraient dans un élément ***choice*** et non dans ***sequence***.
Deux autres déclarations d'élément viennent ensuite pour *Client* et *Facture*. *Client* contient plusieurs attributs et constitue donc un type complexe. Par conséquent, nous devons ajouter une définition de type de l'élément *Client*, ce qui spécifiera son contenu. Voici le modèle de contenu de l'élément *Client* :

```
<xsd :complexType  name = «TypeClient»>
   <attribute name = " nom " type = " string " use = " required "
   <attribute name = " prenom " type = " string " use = " required " >
   <attributte name = " AdresseElectronique " type = " string " use =" required">
</xsd :complexType>
```

Tout les attributs faisant partie de l'élément *Client* sont de type simple. Aussi, il suffit de les ajouter à la déclaration d'attribut correspondante, laquelle prend elle-même deux attributs :
name pour déclarer le nom de l'attribut ;
type pour définir son type de données.

4.5 Conclusions

Les schémas XML permettent plus de solidité en vue de définir et d'imposer des contraintes aux documents et aux bases de données.
En particulier, nous pouvons décrire des types de données définis par l'utilisateur pour limiter le contenu des éléments et celui des attributs. Et nous pouvons aussi créer nos propres types de données dérivés.
En conclusion, les schémas XML offrent une autre solution bien plus puissante et descriptive en comparaison des DTDs.

5. Analyseurs du XML : les API DOM et SAX

5.1 Introduction

XML définit la structure du document uniquement, ce qui permet d'une part de pouvoir définir séparément la présentation de ce document et d'autre part d'être capable de récupérer les données présentes dans le document pour les utiliser.
Toutefois la récupération des données encapsulées dans le document nécessite un outil appelé ***analyseur*** (en anglais ***parser***), permettant de parcourir le document et d'en extraire les informations qu'il contient.
Dans le cas de XML, l'analyseur permet de créer une structure hiérarchique des données contenus dans le document XML.

On distingue deux types d'analyseurs XML :
- Les analyseurs validants permettant de vérifier qu'un document XML est conforme à sa DTD.
- Les analyseurs non validants se contentant de vérifier que le document XML est bien formé, c'est à dire respectant la syntaxe XML de base.

Les analyseurs XML sont également divisés selon l'approche utilisées par l'interface API (Application Program Interface) qu'ils utilisent pour traiter les documents. Il existe actuellement deux types d'approches :

- Les API utilisant une approche **hiérarchique :** les analyseurs utilisant cette technique construisent une structure hiérarchique (un arbre) contenant des objets représentant les éléments du document. La principale API utilisant cette approche est **DOM** (Document Object Model).
- Les API basés sur un mode **événementiel** permettent de réagir à des événements (comme le début d'un élément, la fin d'un élément) et de renvoyer le résultat à l'application utilisant cette API. **SAX** (Simple API for XML) est la principale API utilisant l'aspect événementiel.

5.2 DOM

5.2.1 Qu'est ce que DOM ?

Comme il est possible de créer ses propres vocabulaires XML, et des instances de documents se conformant à ces vocabulaires, il faut disposer d'une façon standard d'interagir avec ces données. Le DOM procure un modèle objet pouvant modéliser n'importe quel document XML, quelle que soit sa structure et permettant d'accéder à son contenu.

Bien que le DOM soit un modèle objet, celui-ci est abstrait : le DOM n'est pas lui même un programme, et la spécification n'indique pas comment implémenter les interfaces qu'il expose. En fait, la spécification DOM se borne à définir un ensemble d'interfaces de programmation d'applications (API), établissant comment un logiciel compatible DOM doit permettre l'accès à un document et la manipulation de son contenu.
Le DOM permet par exemple de :
- Créer des documents et des parties de documents ;
- Parcourir le document ;
- Déplacer, copier et supprimer des parties de document ;

- Ajouter ou modifier des attributs.

5.2.2 Naviguer dans l'arbre

Les documents XML peuvent être représentés comme des arbres d'informations en raison de leur structure hiérarchique. Les relations entre ces nœuds ont tendance à être exprimés comme celles d'un arbre généalogique, en termes de *parent/enfant, ancêtre/descendant,* etc. Le DOM expose des propriétés permettant de naviguer dans l'arbre en utilisant ce type de terminologie. Parmi ces propriétés, on trouve *parentNode, firstChild, lastChild, previousSibling* et *nextSibling,* qui renvoient toutes un nœud, et la propriété *chieldNodes* qui renvoie une liste de nœuds (*NodeList*).

Parcourir l'arbre est indispensable afin de pouvoir atteindre le nœud sur lequel nous voulons traiter, qu'il s'agisse seulement de récupérer une valeur, de mettre à jour son contenu, d'ajouter quelque chose à cette position dan la structure du document ou bien de le supprimer.

Les nœuds n'ont pas nécessairement des descendants (le cas des attributs) même ceux qui peuvent en avoir. Dans ce cas les propriétés censées de retourner les enfants renvoient *null* (ou une *NodeListe* vide pour la propriété *childNodes).*

5.2.3 Interface Node

Node est l'interface la plus importante du DOM. Presque tous les objets mentionnés ici étendent les fonctionnalité de cette interface, ce qui est logique puisque tout élément XML est un nœud.

L'objet ***Node*** permet de réaliser trois actions clés :

1. **Traverser l'arbre.** Afin d'interroger, ou d'y apporter des modifications.
2. **Extraire des informations sur le nœud.** En interrogeant l'objet ***Node*** à l'aide des méthodes disponibles sur cette interface, on peut obtenir des informations : le type du nœud, ses attributs, son nom et sa valeur.
3. **Ajouter, supprimer et actualiser des nœuds.** Si on veut modifier la structure du document, on peut ajouter, supprimer ou remplacer des nœuds.

Le tableau ci-dessous présente les propriétés disponible avec les objets ***Node.***

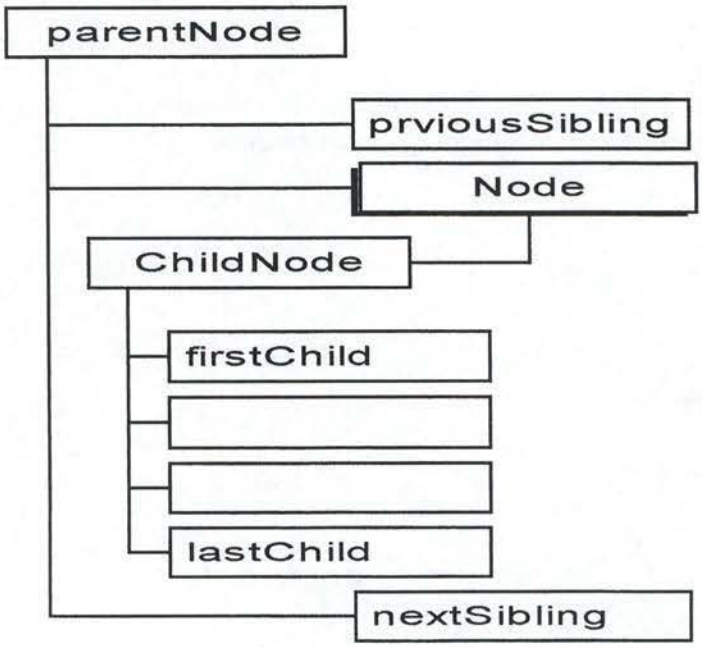
Propriété	**Description**
nodeName	Le nom du nœud. renvoie différentes valeurs en fonction du *nodeType*
nodeValue	La valeur du nœud. renvoie différentes valeurs en fonction du *nodeType*
nodeType	Le type du nœud.
parentNode	Le nœud qui est le parent du nœud
childNodes	Un *nodeList* contant tous les enfants du nœud. S'il n'a pas d'enfants, c'est un *nodeList* vide.
firstChild	Le premier enfant de ce nœud. S'il n'y en a pas renvoie NULL.
lastChild	Le dernier enfant de ce nœud. S'il n'y en a pas renvoie NULL.
previousSibling	Le nœud précédent au même niveau que ce nœud. S'il n'y en a pas renvoie NULL.
nextSibling	Le nœud suivant au même niveau que le nœud concerné. S'il n'y en

	a pas renvoie NULL.
attributes	Un NameNodeMap contenant les attributs de ce nœud. S'il n'y a en pas renvoie NULL.
ownerDocument	Le document auquel appartient le nœud.
namesSpaceURI	L'URI d'espace des noms de ce nœud. Si aucun espace de noms n'est spécifié, renvoie NULL.
prefixe	Le préfixe d'espace de noms de ce nœud. Si aucun espace de noms n'est spécifié, renvoie NULL.

Le tableau ci-dessous présente les méthodes de l'objet *Node*.

Méthode	**Description**
insertBefore (newChild,refChild)	Insère le nœud *newChild* avant le nœud *refChild*. Si *refChild* est null, insère le nœud à la fin de la liste. Renvoie le nœud insèré.
remplaceChild(newChild,oldChild)	Remplace *oldChild* par *newChild*. Sert à mettre à jour des enregistrements existants. Renvoie *oldChild*.
removeChild(oldChild)	Supprime *oldChild* de la liste et le renvoie.
appendChild(newChild)	Ajoute *newhild* à la fin de la liste et le renvoie.
hasChildNodes()	Renvoie true si le nœud possède au moins un enfant, false, sinon.
cloneNode(deep)	Renvoie une copie de ce nœud. Si le paramètre boolèen *deep* est true; copie de façon récursive le sous arbre du nœud. Dans le cas contraire, ne copié que le nœud lui même.

Le diagramme suivant représente un nœud (la case Node) ainsi que les relations entre ce nœud et les autres nœuds de l'arbre.



Ce diagramme de la figure précédente, montre les relations en utilisant les terminologies DOM.

La propriété *childNode* permet d'accéder aux enfants d'un nœud de la case *Node*. Pour extraire le premier ou le dernier enfant, il existe un moyen plus simple que de parcourir cette propriété. Si un nœud ne comporte qu'un seule enfant, les propriétés *firstChild* et *lastChild* renvoient celui-ci.

La propriété *parentNode* renvoie le nœud parent du nœud courant. Les propriétés *previousSibling* et *nextSibling* renvoient les deux nœuds enfants du nœud parent de part et d'autre du nœud traité.

5.3 SAX

5.3.1 Qu'est-ce que SAX ?

SAX (Simple API for XML) est un autre outil de résolution des problèmes relatifs à XML. SAX est une API tout comme DOM et peut être implémenté dans un analyseur.

La différence essentielle entre SAX et DOM est que SAX est une interface pilotée par des événements qui exige la déclaration de certaines méthodes pouvant «capturer» des événements depuis l'analyseur.

Lorsqu'un analyseur fondé sur DOM analyse un document XML, il stocke en mémoire une représentation du document sous forme d'arborescence. SAX, en revanche, envoie le document sous forme de flux de début à la fin, en examinant les différents éléments qu'il rencontre pendant le passage du document. Pour chaque élément structurel du document, SAX appelle une méthode qu'on a rendu disponible.

Soit le document XML suivant :

```
<Personne >
        <NomPersonne >Fontaine</NomPersonne >
        <PrenomPersonne>Vincent </PrenomPersonne>
</Personne >
```

Une interface événementielle telle que l'API SAX permet de créer des événements à partir de la lecture du document ci-dessus. Les événements générés seront :

```
startdocument
startelement:    Personne
startelement:    NomPersonne
characters:      Fontaine
endelement:      NomPersonne
startelement:    PrenomPersonne
characters:      Vincent
endelement:      PrenomPersonne
endelement:      Personne
enddocument
```

Chacun des événements déclenchés demande une implémentation de l'interface SAX afin de travailler avec l'information fournie par l'analyseur. Le début d'un élément est capturé par la méthode *startElement*, la fin par la méthode *endElement* et ainsi de suite.

5.3.2 Interfaces SAX

L'API SAX définit les quatre interfaces suivantes :
DocumentHandler possédant des méthodes renvoyant des événements relatifs au document :
startDocument() renvoyant un événement lié à l'ouverture du document
startElement() renvoyant un événement lié à la rencontre d'un nouvel élément
characters() renvoyant les caractères rencontrés
endElement() renvoyant un événement lié à la fin d'un élément
endDocument() renvoyant un événement lié à la fermeture du document

ErrorHandler possédant des méthodes renvoyant des événements relatifs aux erreurs ou aux avertissements
DTDHandler renvoie des événements relatifs à la lecture de la DTD du document XML
EntityResolver permet de renvoyer une URL lorsqu'une URI est rencontrée

5.4 Conclusion

Les analyseurs utilisant l'interface DOM souffrent du fait que cette API impose de construire un arbre en mémoire contenant l'intégrité des éléments du document quelque soit l'application. Ainsi pour de gros documents DOM devient insuffisant.
Aunsi, de plus en plus d'applications se tournent vers les API événementielles telles que SAX permettant de traiter uniquement ce qui est nécessaire.

6. Transformations XML (XSLT)

6.1 Introduction

XML est un langage qui permet de gérer des données de manière structurée. Un des ces points forts est la possibilité de modifier la structure de ces données, notamment grâce au langage de feuille de style extensible XSL. Ce dernier est en fait constitué de deux langages : XSL lui-même (eXtensible Stylesheet Language) et XSLT (XSL Transformations).

6.2 Qu'est-ce que XSLT ?

XSLT est un langage de haut niveau pour la définition de transformations XML. En règle générale une transformation part d'un document d'entrée XML et produit en sortie un autre document XML ou autre (HTML, XML, PDF, texte brute, etc.).
Le diagramme ci-dessous représente le processus de transformation :

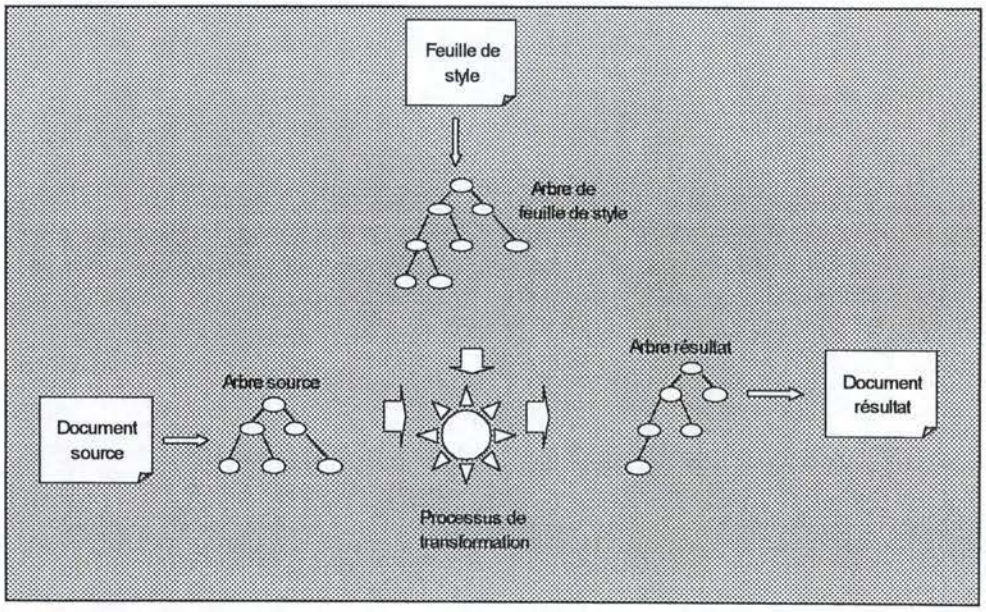


Figure 1 : Processus de transformation avec XSLT.

Le traitement s'effectue en trois étapes distinctes :
- Un analyseur XML part du document source XML et le transforme en une représentation arborescente.
- Le processeur XSLT, suivant les règles exprimées dans une feuille de style, transforme cet arbre en un autre.
- Un « serialiseur » prend l'arbre résultat et le transforme en un document XML ou autre.

Ces trois morceaux logiciels sont rassemblés dans un même produit.
La structure arborescente utiliser par XSLT est très proche de celle du modèle DOM.

6.3 Principe des feuilles de styles

La feuille de style regroupe un ensemble de règles de transformation qui vont s'appliquer à un document XML. Le but est de modifier la structure XML de base afin d'en créer une nouvelle.

En résumé : on prend une structure XML, on la transforme grâce à XSLT et on obtient un nouveau fichier XML ou autres (HTML, PDF, etc.).

6.4 Syntaxe des feuilles de style

Une feuille de style utilise le langage XSLT, qui est basé sur XML. Ainsi, comme pour un document XML, elle débute par la déclaration suivante :

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

Il s'agit de la même déclaration que celle d'un document XML. L'attribut version déclare la version du langage XML avec lequel XSLT est conforme. L'attribut encoding déclare le type de jeu de caractère utilisé par le document : ISO-8859-1 est le codage correspondant aux langue d'Europe de l'Ouest.
Ensuite on trouve les instructions indiquant le début et la fin de la feuille de style :

```
<xsl :stylesheet xmlns :xsl= "http://www.w3.org/1999/XSL/Transform">
Ici, les règles de transformations.
</xsl :stylesheet>.
```

La première instruction comprend la déclaration d'un espace de noms, qui débute par *xmlns* :. Ici, l'espace de noms est *xsl* et est rattaché à l 'URL www.w3.org/1999/XSL/Transform.

En effet, le principe de XML étant que l'on peut donner n'importe quel nom aux éléments, il est donc possible d'obtenir deux éléments ayant le même nom en XML et en XSLT. Pour remédier à ce type de collision, tous les éléments propres à XSLT sont rattachés à un espace de noms.

6.5 Règles de transformation

Les règles de transformation utilisent des instructions XSLT pour convertir les données XML de base en une structure nouvelle XML résultat.

6.5.1 Instructions XSLT

Les instructions XSLT constituent un sous-ensemble d'élément de XSLT : fondamentalement celles qui peuvent être utilisés directement comme parties d'un corps de modèle.
Les instructions les plus utilisées définies en XSLT sont les suivantes :

<xsl :apply-imports>
<xsl :apply-templates>
<xsl :attribute>
<xsl :for-each>
<xsl :if>
<xsl :call-template>
<xsl :message>
<xsl :choose>
<xsl :number>
<xsl :comment>
<xsl :processing-instruction>
<xsl :copy>
<xsl :text>
<xsl :copy-of>
<xsl :value-of>
<xsl :element>
<xsl :variable>
< xsl :template>

6.5.1.1 <xsl :template> : définir un modèle et un contexte.

Le modèle *template* permet de déterminer, à partir du XML de base, le contexte dans lequel nous allons travailler. Ce contexte repose sur un ou plusieurs critères. Si des éléments du XML de base satisfont intégralement ces critères ou conditions (patterns), alors ils feront partie du contexte. Ces conditions sont exprimées dans l'attribut *match*, comme suit :

<xsl :template match= "conditions ">.

Leur syntaxe repose sur le langage XPath. XPath indique au processeur XML ou XSLT comment s'orienter dans le document XML. En voici des exemples :

/ (barre oblique).

Désigne la racine du XML de base, c'est-à-dire l'élément de plus haut niveau de l'arborescence XML ainsi que ces éléments fils.

L'utilisation seule de cette condition englobe tous les éléments de l'arborescence XML.

L'exemple « Pere/Fils » sélectionne les éléments « Fils » qui ont pour élément père « Pere » (lui même fils du contexte courant).

. (point)

Désigne le contexte courant. S'il est utilisé dans le contexte « Titre », par exemple, alors l'emploi de « . » signifiera « Titre ».

.. (double point)

Signifiera que l'on remonte d'un niveau dans l'arborescence XML.

Texte()

Sélectionne les nœuds texte. Il s'agit du contenu d'un élément.

| (barre verticale)

Exprime la condition OU. Par exemple « Noire | Blanche » sélectionnera l'ensemble des éléments fils dont le nom est « Noire » ou « Blanche » au premier niveau du contexte courant.

Titre[end()]

Sélectionne le dernier élément « Titre » dans l'ensemble des éléments fils au niveau du contexte courant.

[Titre]
Sélectionne n'importe quel élément qui a pour élément fils « Titre ».
@
Sélectionne les attributs indiqués. Exemple : « @class ».
@*
Sélectionne tous les attributs.
Titre[@ class]
Sélectionne, dans le contexte courant, l'élément fils « Titre » possédant un attribut « class ».
Titre[@ class='Item']
Sélectionne, dans le contexte courant, l'élément fils "Titre" dont l'attribut "class" a la valeur "Item".

6.5.1.2 <xsl :apply-templates> : appliquer les transformations.

Cette instruction provoque le traitement d'un ensemble de nœuds sélectionnés, chaque nœud utilisant la règle modèle (template) adéquate d'après les motifs de correspondance et les priorités.

Lorsque <xsl :template> est utilisé pour traiter un ensemble de nœuds, il n'est pas obligatoire qu'il y ait exactement une règle modèle correspondante à chaque nœuds. Il peut n'y en avoir aucune, comme il peut y en avoir plusieurs.
Cette instruction possède deux attributs, tous deux facultatifs. L'attribut *select* est une expression XPath définissant l'ensemble des nœuds à traiter : par défaut, il s'agit des enfants du nœud courant. L'attribut *mode* donne le nom d'un mode de traitement : seuls les éléments <xsl :template> possédant le même nom de mode constituent des candidats valables pour les correspondances.

6.5.1.3 <xsl :for-each> : affiner le contexte et le traitement.

L'instruction *<xsl :for-each>* sert à définir le traitement à accomplir sur chaque membre d'un ensemble de nœuds. L'ensemble des nœuds à traiter doit être défini par une expression XPath dans l'attribut *select*. Le traitement lui-même est défini par le corps de modèle contenu dans l'élément *<xsl :for-each>*.
Cette exemple crée un nœud attribut dans l'arbre résultat, correspondant à chaque élément enfant du nœud courant de l'arbre source :

```
<xsl :for-each select = " * "
   <xsl :attribute name = "{name()} ">
      <xsl :value-of select = ". ">
   </xsl :attribute>
</xsl :for-each>
```

6.5.1.4 <xsl :element> : générer un élément.

L'instruction <xsl :element> écrit un nœud élément dans l'arbre résultat. Le contenu de l'instruction <xsl :element> est un corps de modèle, instancié afin de construire le contenu de l'élément généré.
Cette instruction possède comme attributs *name* et *namespace*. L'attribut *name* donne le nom du nouvel élément, tandis que l'attribut *namespace* donne l'URI d'espace de noms.

Les attributs *name* et *namespace* sont tous deux interprétés comme modèles de valeur d'attributs(Attribute Value Templates). L'exemple suivant crée un élément *<html>* en utilisant un URI d'espace de noms transmise comme paramètre :

```
<xsl :element name= " html " namespace= " {$html-namespace} ">
  <head>
   <title><xsl :value-of select= " titre "/></title>
  </head>
  <body>
   <xsl :call-template name= " gerer-body ">
  </body>
<xsl :element>
```

6.5.1.5 <xsl :attribute> : générer un attribut.

Cette instruction a pour effet d'écrire un nœud attribut dans l'arbre résultat. Cela n'est possible que si la dernière chose écrite était un élément ou un autre attribut. L'instruction <xsl :attribut> possède deux attributs *name* et *namespace*.

6.5.1.6 <xsl :if> : définir un traitement conditionnel.

Cette instruction effectue une action si la condition est vrai. Il n'existe pas de branchement *else*. *<xsl :if>* possède comme attribut obligatoire *test*, qui définit la condition à tester sous la forme d'une expression XPath dont le résultat est une valeur booléenne. L'élément *<xsl :if>* contient un corps de modèle instancié si la condition est vrai.

6.5.1.7 <xsl :choose> : choisir parmi plusieurs alternatives.

Cette instruction sert à effectuer un traitement conditionnel. L'élément *<xsl :choose>* ne possède pas d'attribut. Il contient une séquence d'un ou plusieurs éléments *<xsl :when>*, facultativement suivi par un élément *<xsl :otherwise>*. Chaque élément *<xsl :when>* spécifie une condition, et le premier dont la condition est satisfaite est exécuté. Si aucune des conditions n'est remplie, c'est l'élément *<xsl :otherwise>* qui sera utilisé. Par exemple :

```
<xsl :choose>
   <xsl :when test = "ang('ar') "> Salam</xsl :when>
   <xsl :when test = "lang('fr') "> Bienvenu</xsl :when>
   <xsl :when test = "lang('en') "> Welcome</xsl :when>
   <xsl :otherwise>Erreur !</xsl :otherwise>
</xsl :choose>
```

7.Etude de cas : génération automatique de pages HTML

7.1 Introduction

Le langage HTML mélange présentation et information. Aucun programme n'est capable d'extraire sans risque d'erreur l'information utile d'une page HTML : cette information est imbriquée de façon complexe avec des instructions définissant son affichage. Ce langage n'est pas **extensible**, parce que HTML ne propose qu'un nombre limité d'instructions dont l'objet principale est la mise en forme de documents délivrés grâce au Web. HTML utilise des balises et des attributs relevant uniquement de la présentation et n'ayant aucun rapport avec la structure des documents. Il n'y a donc pas de description, ni de structuration de données. Ces handicaps de HTML limitent les champs d'utilisations de ces documents, à savoir l'extension, les modifications et la portabilité.

Par contre, XML apporte une réponse aux limitations d'HTML : XML définit l'information à l'aide d'instructions qui servent à décrire et à structurer l'information elle-même et non sa présentation. En plus XML est une technologie intégrable. En effet, il offre aux systèmes et aux applications existantes de nouvelles fondations sans remettre en cause les investissements importants qui ont déjà été réalisés. Il permet aussi de compléter les autres technologies pour permettre de construire des applications offrant de nouveaux services aux utilisateurs. XML s'est imposé aux cours de ces dernières années comme un format universel de description et d'échange de données entre applications informatiques hétérogènes.

En résumé, XML est un métalangage[4]de définition de types de documents. XML définit les règles de niveaux supérieurs (c'est-à-dire la syntaxe) de tout nouveau type de document. Ce langage peut servir à toutes les étapes du développement, à savoir le stockage, le transport, le traitement et l'affichage des données.

Dans le cadre du projet de recherche SAPHIR[5], un site Web a été réalisé dont la partie principale concerne un corpus de fiches de recommmandations qui s 'adresse aux concepteurs Web afin de favoriser l'accès à Internet au plus grand nombre, et ce y compris aux personnes présentant des handicaps et aux personnes âgées.

Il a été décidé de construire toutes les fiches sur une même structure basée sur un certain nombre de rubriques prédéfinis soit obligatoires soit optionnelles.

Pour faciliter la tâche de préparation et de suivi du contenu des fiches efféctué en grande partie par des non-informaticiens, l'encodage a été réalisé au moyen de fichiers textes en adoptant un balisage prédéfini simplifié des rubriques à l'aide de titres (voir Annexe 1[6]).

---

[4] Un métalangage est un langage qui fixe les règles valables pour d'autres langages qui ont sont dérivés

[5] Projet de recherche effectué au sein de l'équipe « Multimidia » de l'Institut d'Informatique des FUNDP en collaboration avec le CREATIC de l'ULB et financé par la Région Wallonne.

[6] Le texte complet est actuellement accessible à l'adresse http://www.info.fundp.ac.be/Saphir/R9Fiches.pdf

Pour intégrer les fiches dans le site Web, la procédure adopté a consisté à générer de façon semi-automatique des pages HTML à partir du document texte balisé non XML en utilisant des outils spécialisés tel que Dreamweaver de Macromedia. La figure qui suit décrit l'existant :

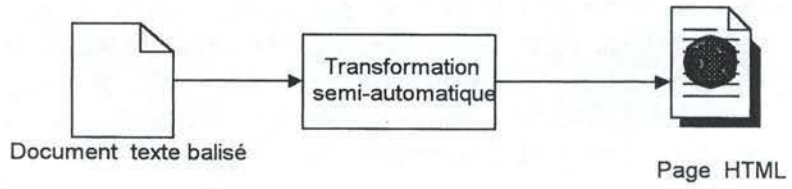


Figure 2 : Procédure existante de la génération des pages HTML.

7.2 Démarche suivie

Le but de ce sujet de mémoire est l'automatisation de la procédure de génération des pages HTML (ou des sites Web) à partir d'un ensemble de fichiers textes structurés non XML ou de fichiers propriétaires Word. On veut pouvoir extraire des données ou analyser ce genre de fichiers existants. Il s'agit de fournir un moyen simple qui permet de rendre les données « standard » et utilisables par plusieurs applications : « Standard » veut dire rendre ces données indépendantes de la plate-forme et telles qu'elles aient un format texte structuré d'une manière hiérarchique. Ce format standard est le format XML.

Dans notre étude de cas, nous présentons dans deux démarches de transformations. La première démarche se base sur la programmation Java et la deuxième se base sur la combinaison d'un outil de développement XMLSPY[7] et d'une feuille de styles XSL[8]. Dans le but de visualiser, pour la présentation de documents, nous allons utiliser le langage HTML qui est le langage par excellence pour l'affichage de pages dans les navigateurs Internet.

Nous partons au départ d'un document qui n'est pas XML car il est plus aisé aux utilisateurs d'encoder les sources à l'aide de logiciels de traitement de texte comme Word. Les balises originales peuvent être celles de Word ou être un balisage classique par titre et sous titres dans des fichiers textes etc.

La première approche consiste à transformer les documents **textes** balisé **NON** XML qui sont à notre disposition en documents balisés XML en utilisant une approche de programmation (programme en Java) qui transforme le fichier texte non XML en document XML. La seconde approche consiste à partir des fichiers en format Word et à les transformer en document XML en deux étapes : nous utilisons l'outil de développement XMLSPY qui permet de transformer un document Word en document XML brut, et ensuite nous appliquons une feuille de style à ce document généré pour produire un autre document XML final correspondant au format souhaité.

L'étape suivante s'attache à vérifier si les documents générés par les deux procédés décrits ci-dessus sont bien formés et valides, bien **formés** parce qu'ils respectent la syntaxe XML et

[7] Voir le site http://www.xmlspy.com
[8] XSL (eXtensible Stylsheet Language)

valides car ils sont en conformité avec notre DTD[9] (ou du Schéma[10] XML). Cette étape de validation contrôle la conformité grammaticale du document par rapport à un vocabulaire et à sa grammaire associée qui sont décrits dans la DTD.

La dernière étape permet alors de transformer notre document XML bien formé et valide en document HTML également par l'application d'une feuille de styles XSL. Cette transformation peut aller vers d'autres formats textes que le HTML, à savoir le PDF, XML ou d'autres. En fait, il est permis d'aller vers tous les langages moins riches sémantiquement que le XML.

Dans la section suivante nous présentons le diagramme générale des processus de transformations utilisés ainsi que la description détaillée de chaque étape participant à ce processus. Ce diagramme illustre les processus de transformation des fichiers textes balisés non XML et des fichiers Word (fichiers binaires) en format XML puis en format HTML en utilisant les technologies XML.

7.3 Diagramme général du processus de transformation

Nous présentons dans ce diagramme les différents étapes qui contribuent à la transformation décrite plus haut :

[9] La DTD spécifie l'organisation des types de données dans un fichier (voir le chapitre DTD) dans la première partie.

[10] Un schéma spécifie le même genre d'information qu'une DTD, c'est-à-dire la syntaxe d'une classe de documents XML, en définissant les éléments et attributs ainsi que les contraintes sur les valeurs de ceux-ci.

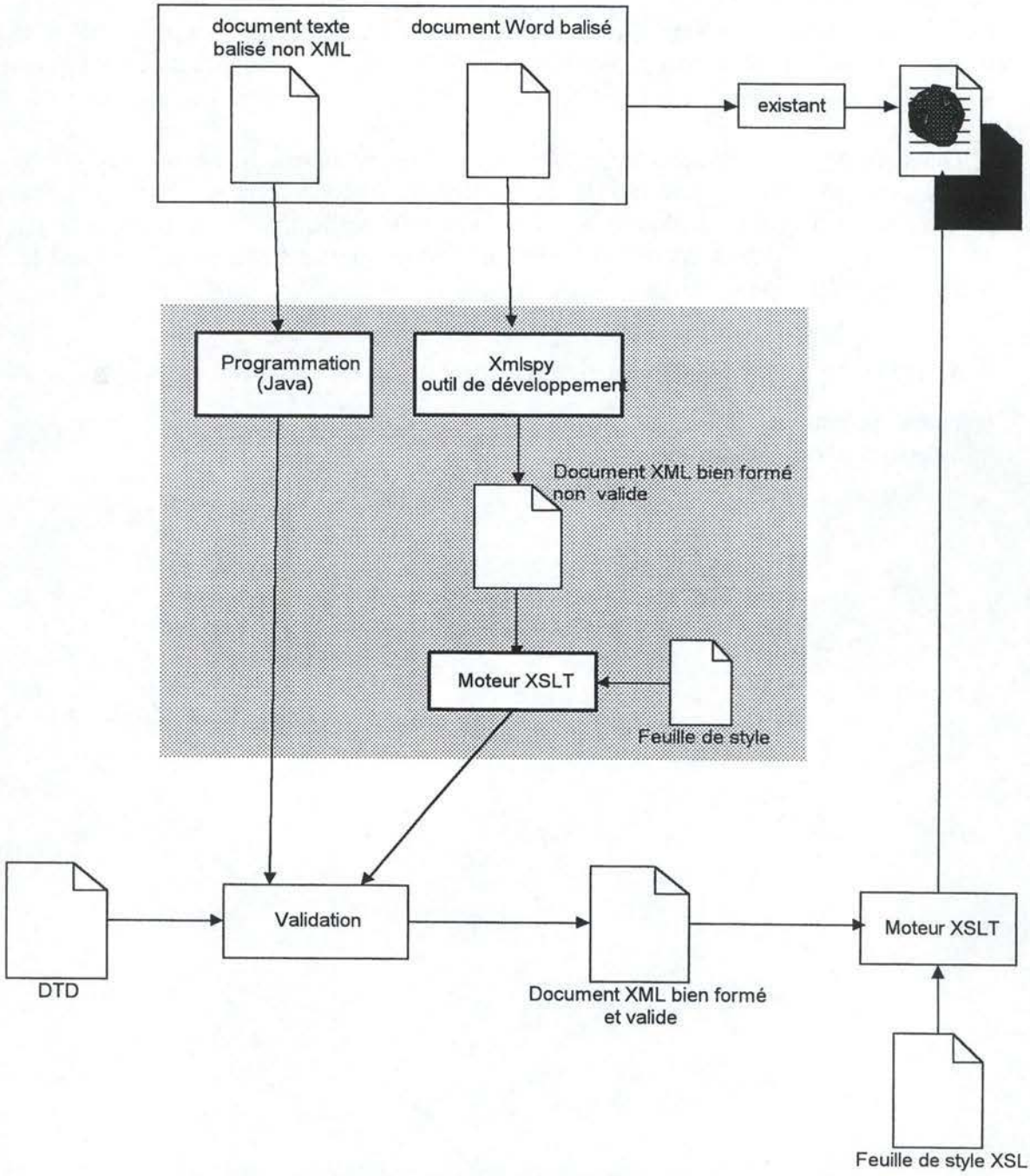


Figure 3 : Diagramme générale du processus de transformation.

7.4 Etape 1 : Des fichiers textes ou Word ou vocabulaire XML

Cette étape concerne les fichiers textes et les fichiers Word. Dans ce paragraphe nous allons décrire la différence entre un fichier texte et un fichier binaire.

Actuellement, on trouve deux moyens de stocker les données sur un ordinateur : soit à l'intérieur de fichiers dits « binaires » soit à l'intérieur de fichiers textes. Nous allons voir en détail, ces deux moyens de stockage.

7.4.1 Fichiers binaires

Un fichier binaire est une suite de bits, de « 0 » ou de « 1 ». cela conduit généralement lorsqu'on ouvre ce genre de fichiers avec une application non compatible à une erreur générée par l'application, ou bien à un affichage complètement incompréhensible. Les fichiers binaires ne peuvent être ouverts ou crées que par des applications pouvant les comprendre ou les générer. Un des défauts de ce genre de fichiers : ils sont propriétaires et peu de programmes peuvent les lire ou les créer. De même, un problème de compatibilité peut survenir dans bien des cas (souvent des problème de versions). Par contre, un fichier binaire est plus facile à lire ou à écrire par l'ordinateur qu'un fichier texte. La gestion de donnée peut se faire bien plus rapidement.

7.4.2 Fichiers textes

Un fichier texte ne constitue, en réalité, rien d'autre qu'un fichier binaire structuré d'une manière que l'on pourrait qualifier d'« universelle ». N'importe quelle application est théoriquement capable de le lire. Les bits des fichiers textes sont en fait regroupés en petits ensembles qui permettent de coder la présentation de symboles (chiffres, lettres etc.). Les programmes savent tous lire les fichiers textes, car ils tentent tous de regrouper ces bits de la même manière. Par contre les fichiers binaires ont un agencement de bits qui dépend totalement du programme pouvant les produire ou les lire. Un fichier texte n'est donc pas propriétaire et peut être lu par un très grand nombre d'application au prix d'une rapidité souvent inférieur à un fichier binaire. N'importe quel utilisateur peut donc ouvrir un fichier texte et en récupérer son contenu correctement. Le partage de l'information se révèle ici possible, et ce, très facilement.

Un désavantage d'un fichier texte réside dans la nature brute du texte qu'il contient, c'est-à-dire qu'on ne peut y trouver du texte en gras, des images des couleurs etc. Il n'y a aucune mise en forme. Au début de l'Internet, les pages n'étaient constituées que de textes bruts. C'est dans ce contexte qu'on a essayé d'imaginer un langage capable de posséder les avantage du binaire (notamment pour la mise en forme) combiné avec la souplesse de lecture des fichiers textes.

Plusieurs moyens peuvent être utilisés pour mettre la mise en forme d'un fichier texte. Il serait possible, par exemple de faire précéder une phrase devant être en couleur bleu par une suite de symboles indiquant que « la phrase qui suit doit être en bleu » et le nombre de caractères à colorer dans cette teinte : système simple, mais relativement difficile à lire. Donc le système de balisage est un meilleur choix ; on entoure notre phrase de deux balises qui indiquent que le contenu qu'elles délimitent doit être en bleu. Schématiquement, cela consiste à transformer ce texte simple :

...texte. Notre phrase à mettre en bleu. ...encore du texte

en celui-ci :

*...texte. **<bleu>**Notre phrase à mettre en bleu **</bleu>**. ...encore du texte*
Dans cette exemple, la balise ouvrante est <bleu>, et indique que le texte qui suit doit être en bleu. La balise fermante </bleu> indique que l'enrichissement stylistique prend fin.

Dans cette étude de cas, nous disposons en entrée de fichiers textes balisés non XML et de fichiers Word qui font parties des types de fichiers décrits ci-dessus.
Les fichiers textes que nous possédons ont la structure suivante :

Fiche Numéro :
Degré :
..du texte..
Titre :
..du texte..
Description courte :
..du texte..
Groupe concerné :
..du texte..
Priorité :
..du texte..
Description détaillée :
..du texte..
Rapport avec l'ergonomie :
..du texte..
Illustration :
..du texte..
Place dans le processus de développement :
..du texte..
Implémentation optimale :
..du texte..
Informations complémentaires :
..du texte..
Exemples :
..du texte
Recommandations associées :
..du texte..
Mots Clés :
..du texte..
Références :
..du texte..

Un exemple de fiche complet se trouve en Annex 1.

Les titres des paragraphes en gras dans ce document texte ci-dessus constituent les balises des fichiers textes non XML initiaux.
Le tableau ci-dessous décrit la correspondance entre ces balises et les balises XML qui leur sont associées. Cette correspondance nous permet de construire notre vocabulaire XML qui sera utilisé dans notre DTD. Ce vocabulaire sera aussi utilisé dans les documents XML générés par transformation des fichiers non XML et des fichiers Word.

Les balises des fichiers textes non XML	Les balises des documents XML
Fiche Numéro :	<TFicheNumero> <Fnumber/> </TFicheNumero>
Degré :	<TDegre> <TitreDegre /> <TypeDegre /> </TDegre>
Titre :	<TDocument> <TitreDoc /> <Titre /> </TDocument>
Description courte :	<TDescriptionCourte> <TitreDescriptionCourte /> <PragDC /> </TDescriptionCourte>
Groupe concerné :	<TGroupeConcerne> <TitreGroupeConcerne /> <ParaGroupeConcerne /> </TGroupeConcerne>
Priorité :	<TPriorite> <TitrePriorite /> <ValeurPriorite /> </TPriorite>
Description détaillée :	<TDescriptionDetaillee> <TitreDescriptionDetaillee /> <ParaDescriptionDetaillee /> </TDescriptionDetaillee>
Rapport avec l'ergonomie :	<TRapportAvecErgonomie > <TitreRapportAvecErgonomie/> <ParaRapportAvecErgonomie/> </TRapportAvecErgonomie >
Illustration :	<TIllustration> <TitreIllustration /> <ParaIllustration /> </TIllustration>
Place dans le processus de développement :	<TPlaceDansProcessusDDeveloppement>

	<TitrePlaceDansProcessusDDeveloppement /> <ParaPlaceDansProcessusDDeveloppement /> </TPlaceDansProcessusDDeveloppement>
Implémentation optimale :	<TImplementationOptimale> <TitreImplementationOptimale /> <ParaImplementationOptimale /> </TImplementationOptimale>
Informations complémentaires :	<TInformationComplementaire> <TitreInformationComplementaire /> <ParaInformationComplementaire /> </TInformationComplementaire>
Exemples :	<TExemples> <TitreExemples /> <ParaExemples /> </TExemples>
Recommandations associées :	<TRecommandationsAssociees> <TitreRecommandationsAssociees /> <ParaRecommandationsAssociees /> </TRecommandationsAssociees>
Mots Clés :	<TMotsCles> <TitreMotsCles /> <ParaMotsCles /> </TMotsCles>
Références :	<TReferences> <TitreReferences /> <ParaReferences /> </TReferences>

7.5 Etape 2 : Création de la DTD

L'ensemble des balises ainsi rédigées pour baliser notre document texte non XML en document XML constitue notre vocabulaire XML.

Puisqu'en XML, nous pouvons créer nos propres balises, il faut un moyen permettant de définir un vocabulaire, afin de pouvoir aussi le partager avec d'autres en utilisant la même syntaxe. En XML, une DTD définit quel balisage peut être utilisé dans un document qui est supposé respecter ce vocabulaire. En effet, la DTD définit les éléments pouvant être présent dans un document XML, le nombre d'instances de chaque élément, et l'ordre des éléments dans le document. Elle peut spécifier les attributs acceptés par un élément, si certains attributs sont obligatoires, si une valeur par défaut doit être retenue en cas d'absence de valeur, etc.

Notre DTD est écrite sur base d'identification de chaque titre de paragraphe au sein de notre fichier texte non XML. Chaque titre identifié devient un élément dans notre DTD. Et dans chaque titre de paragraphe, il y a des sous-paragraphes représentées par du texte. Ces sous paragraphes deviennent des éléments enfants de ces éléments identifiés précédemment. Et ensuite, il y a la création des relations qui existent entre les éléments et leur enfants, à savoir le nombre d'occurrence, le caractère obligatoire ou facultatif, etc.

La DTD qui représente notre vocabulaire est la suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT SafirDocument (TFicheNumero, TDegre, TDocument, TDescriptionCourte,
TGroupeConcerne, TPriorite, TDescriptionDetaillee, TRapportAvecErgonomie, TIllustration,
TPlaceDansProcessusDDeveloppement, TImplementationOptimale,
TInformationComplementaire, TExemples*, TRecommandationsAssociees, TMotsCles,
TReferences)>
<!ELEMENT TFicheNumero (Fnumber)>
<!ELEMENT Fnumber (#PCDATA)>
<!ELEMENT TDegre (TitreDegre, TypeDegre)>
<!ELEMENT TitreDegre (#PCDATA)>
<!ELEMENT TypeDegre (#PCDATA)>
<!ELEMENT TDocument (TitreDoc, Titre)>
<!ELEMENT TitreDoc (#PCDATA)>
<!ELEMENT Titre (#PCDATA)>
<!ELEMENT TDescriptionCourte (TitreDescriptionCourte, PragDC*)>
<!ELEMENT TitreDescriptionCourte (#PCDATA)>
<!ELEMENT PragDC (#PCDATA )>
<!ELEMENT TGroupeConcerne (TitreGroupeConcerne, GroupeConcerne*,
ParaGroupeConcerne*)>
<!ELEMENT TitreGroupeConcerne (#PCDATA)>
<!ELEMENT GroupeConcerne (#PCDATA)>
<!ELEMENT ParaGroupeConcerne (#PCDATA)>
<!ELEMENT TPriorite (TitrePriorite, ValeurPriorite)>
<!ELEMENT TitrePriorite (#PCDATA)>
<!ELEMENT ValeurPriorite (#PCDATA)>
<!ELEMENT TDescriptionDetaillee (TitreDescriptionDetaillee, ParaDescriptionDetaillee*)>
<!ELEMENT TitreDescriptionDetaillee (#PCDATA)>
<!ELEMENT ParaDescriptionDetaillee (#PCDATA)>
<!ELEMENT TRapportAvecErgonomie (TitreRapportAvecErgonomie,
ParaRapportAvecErgonomie*)>
<!ELEMENT TitreRapportAvecErgonomie (#PCDATA)>
<!ELEMENT ParaRapportAvecErgonomie (#PCDATA)>
<!ELEMENT TIllustration (TitreIllustration, ParaIllustration*)>
<!ELEMENT TitreIllustration (#PCDATA)>
<!ELEMENT ParaIllustration (#PCDATA)>
<!ELEMENT TPlaceDansProcessusDDeveloppement
(TitrePlaceDansProcessusDDeveloppement, ParaPlaceDansProcessusDDeveloppement*)>
<!ELEMENT TitrePlaceDansProcessusDDeveloppement (#PCDATA)>
<!ELEMENT ParaPlaceDansProcessusDDeveloppement (#PCDATA)>
<!ELEMENT TImplementationOptimale (TitreImplementationOptimale,
ParaImplementationOptimale*)>
```

```
<!ELEMENT TitreImplementationOptimale (#PCDATA)>
<!ELEMENT ParaImplementationOptimale (#PCDATA)>
<!ELEMENT TInformationComplementaire (TitreInformationComplementaire,
ParaInformationComplementaire*)>
<!ELEMENT TitreInformationComplementaire (#PCDATA)>
<!ELEMENT ParaInformationComplementaire (#PCDATA)>
<!ELEMENT TExemples (TitreExemples*, ParaExemples*)>
<!ELEMENT TitreExemples (#PCDATA)>
<!ELEMENT ParaExemples (#PCDATA)>
<!ELEMENT TRecommandationsAssociees (TitreRecommandationsAssociees,
ParaRecommandationsAssociees*)>
<!ELEMENT TitreRecommandationsAssociees (#PCDATA)>
<!ELEMENT ParaRecommandationsAssociees (#PCDATA)>
<!ELEMENT TMotsCles (TitreMotsCles, ParaMotsCles*)>
<!ELEMENT TitreMotsCles (#PCDATA)>
<!ELEMENT ParaMotsCles (#PCDATA)>
<!ELEMENT TReferences (TitreReferences, ParaReferences*)>
<!ELEMENT TitreReferences (#PCDATA)>
<!ELEMENT ParaReferences (#PCDATA)>
```

Cette DTD décrit que nous disposons de **SafirDocument** comme élément racine[11]. Il contient une séquence d'éléments TFicheNumero, TDegre, TDocument, TDescriptionCourte, TGroupeConcerne, TPriorite, TDescriptionDetaillee,...TReference, qui sont obligatoires et apparaissent une seule fois dans le document sauf l'élément TExemples* qui est facultatif et peut apparaître zéro ou plusieurs fois dans le document XML (c'est le rôle du signe *). La chaîne de caractères *TFicheNumero* est défini comme étant un type élément contenant un *Fnumber* et les *Fnumber* comme un type élément ne contenant que du texte, cela est décrit par :

<!ELEMENT TFicheNumero (Fnumber)>
<!ELEMENT Fnumber (#PCDATA)>.

Cette DTD représente le modèle des contenus. Ce modèle garanti la cohérence des données. Donc chaque document XML généré peut être validé, ce qui certifie qu'il est conforme à son type.

7.6 Etape 3 : Transformations de base

Cette étape de transformations contient les deux techniques mises en œuvre pour transformer les fichiers textes balisés non XML brut ou en format Word en un document XML. La première technique consiste à utiliser un programme écrit en Java qui prend comme entrée un fichier texte balisé non XML et produit en sortie un document XML. Par contre, la deuxième technique se fait en deux étapes : La première étape consiste à transformer un fichier Word (fichier binaire) en un document texte balisé XML en utilisant l'outil de développement XMLSPY. La deuxième étape consiste à appliquer une feuille de style sur ce fichier généré précédemment pour produire un document XML conforme à notre DTD décrite

[11] Chaque document XML possède un élément racine unique qui représente le point d'entrée dans l'arbre. Car chaque document XML a une représentation en arbre.

précédemment. Les documents XML en sortie sont **valides** [12]et bien **formés**[13]. Ces deux procédés sont décrits en détail ci-dessous.

7.6.1 Transformation par l'approche de programmation.

7.6.1.1 Le choix du langage de programmation

Le langage de programmation utilisé dans cette étape est le langage orienté objet Java. Le choix de ce langage est dû au fait que des similitudes très proches ont toujours exister entre XML et Java. XML et Java partagent le même principe de portabilité : XML assure la potabilité des données et Java assure la portabilité du code, ce qui donne une grande flexibilité aux applications construites sur base de XML et Java.

7.6.1.2 Description du Programme

Ce programme a pour but de transformer des fichiers textes en documents XML qui respectent les règles de notre DTD décrite précédemment.
Le programme prend comme entrée un fichier texte non XML avec la structure adéquate et produit en sortie un document XML. Ce document XML est crée au fur et à mesure que le programme rencontre un titre de paragraphe du document en entrée. En effet, le programme commence d'abord par créer le prologue XML à savoir
<?xml version="1.0" encoding="UTF-8" ?>
ensuite l'élément document ou l'élément racine : **<SafirDocument>.**
Puis, quand il trouve la chaîne de caractères **Fiche Numéro,** il génère les balises suivantes :
<TFicheNumero>
<Fnumber>Fiche Numéro </Fnumber>
</TFicheNumero>
et ainsi de suite jusqu'à la fin du fichier où il ajoute la balise de fermeture de l'élément racine **</SafirDocument>.** Ainsi nous obtenons un document XML semblable suivant :

```
<?xml version="1.0" encoding="UTF-8" ?>
<SafirDocument>
<TFicheNumero>
<Fnumber>Fiche Numéro </Fnumber>
</TFicheNumero>
<TDegre>
<TitreDegre>Degré :</TitreDegre>
<TypeDegre>...texte... </TypeDegre>
</TDegre>
<TDocument>
<TitreDoc>Titre :</TitreDoc>
<Titre>...texte...</Titre>
</TDocument>
<TDescriptionCourte>
<TitreDescriptionCourte>Description courte :</TitreDescriptionCourte>
<PragDC>..du texte...</PragDC>
```

[12] Lorsqu'un document XML, utilisant un quelconque vocabulaire, respecte les règles de la spécification XML 1.0, il est qualifié de bien **formé**.
[13] Lorsqu'un document bien **formé** respecte les règles de la DTD décrivant ce vocabulaire, il est également qualifié de **valide**.

</TDescriptionCourte>
<TGroupeConcerne>
<TitreGroupeConcerne>**Groupe concerné :**</TitreGroupeConcerne>
<ParaGroupeConcerne>…du texte…</ParaGroupeConcerne>
<ParaGroupeConcerne>…du texte… </ParaGroupeConcerne>
</TGroupeConcerne>
<TPriorite>
<TitrePriorite>**Priorité :**</TitrePriorite>
<ValeurPriorite>…texte… </ValeurPriorite>
</TPriorite>
<TDescriptionDetaillee>
<TitreDescriptionDetaillee>**Description détaillée :**</TitreDescriptionDetaillee>
<ParaDescriptionDetaillee>…du texte… </ParaDescriptionDetaillee>
<ParaDescriptionDetaillee>…texte…</ParaDescriptionDetaillee>
</TDescriptionDetaillee>
<TIllustration>
<TitreIllustration>**Illustration :**</TitreIllustration>
<ParaIllustration>… du texte … </ParaIllustration>
<ParaIllustration> </ParaIllustration>
</TIllustration>
<TPlaceDansProcessusDDeveloppement>
<TitrePlaceDansProcessusDDeveloppement>**Place dans le processus de développement :**</TitrePlaceDansProcessusDDeveloppement>
<ParaPlaceDansProcessusDDeveloppement> </ParaPlaceDansProcessusDDeveloppement>
<ParaPlaceDansProcessusDDeveloppement>…texte..
</ParaPlaceDansProcessusDDeveloppement>
</TPlaceDansProcessusDDeveloppement>
<TImplementationOptimale>
<TitreImplementationOptimale>**Implémentation optimale :**</TitreImplementationOptimale>
<ParaImplementationOptimale>…texte… </ParaImplementationOptimale>
</TImplementationOptimale>
<TInformationComplementaire>
<TitreInformationComplementaire> **Informations complémentaires :**
</TitreInformationComplementaire>
<ParaInformationComplementaire>...texte...</ParaInformationComplementaire>
</TInformationComplementaire>
<TExemples>
<TitreExemples>**Exemples :**</TitreExemples>
<ParaExemples>…texte… </ParaExemples>
</TExemples>
<TRecommandationsAssociees>
<TitreRecommandationsAssociees>**Recommandations associées :**</TitreRecommandationsAssociees>
<ParaRecommandationsAssociees>…texte.. </ParaRecommandationsAssociees>
</TRecommandationsAssociees>
<TMotsCles>
<TitreMotsCles>**Mots Clés :**</TitreMotsCles>
<ParaMotsCles>…texte… </ParaMotsCles>
</TMotsCles>

```
<TReferences>
<TitreReferences>Références :</TitreReferences>
<ParaReferences>...texte... </ParaReferences>
</TReferences>
</SafirDocument>
```

Le code source du programme Java se trouve en Annexe 2.

7.6.2 Transformation par XMLSPY et XSLT.

Dans cette partie, nous avons utilisé l'outil de développement XMLSPY uniquement pour transformer les fichiers Word (des fichiers binaires) en fichiers XML non valides. Cet outil est utile parce qu'il permet de lire les fichiers binaires et les traiter. Les fichiers générés par cet outil seront transformer par le biais d'une feuille de style en documents XML respectant notre vocabulaire décrit précédemment. En effet, les documents générés par XMLSPY ne sont pas valides, c'est-à-dire, qu'ils ne respectent pas les règles décrites dans notre DTD. Pour pallier ce problème, nous avons défini une feuille de style que nous appliquons sur ces fichiers générés pour pouvoir générés des documents XML valides comme souhaités.

La Figure 4 décrit les étapes de cette technique de transformation.

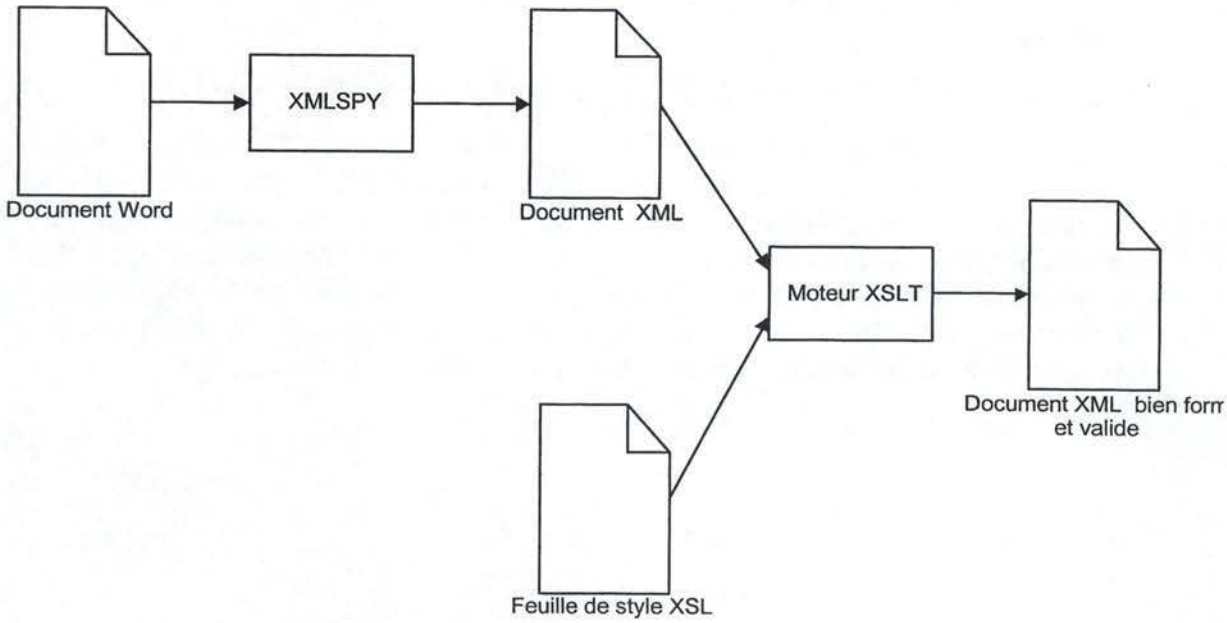


Figure 4 : Schéma de fonctionnement de la transformation d'un document Word en document XML valide.

La feuille de styles XSL permet de transformer un document XML en un autre document XML. Le principe de XSLT consiste à identifier les différents nœuds du document XML de départ, puis à appliquer des règles de transformation pour chaque nœud identifié. Cette transformation nécessite trois éléments : le document XML initial fourni par XMLSPY, une feuille de style XSL et enfin un programme (le moteur XSLT) qui va prendre le fichier XML et la feuille de style XSL et générer un fichier transformé.

Le moteur XSLT utilisé dans cette transformation est « Instant Saxon ». Cet analyseur est un composant logiciel (écrit en Java) permettant d'accéder simplement aux données encapsulées dans un fichier XML. « Instant Saxon » fait partie de la famille SAX[14].
Les feuilles de styles XSL représentent des documents particuliers s'appliquant sur d'autres documents XML. Les feuilles XSL précisent les opérations nécessaires applicables sur les balises d'un document XML. Les transformations réalisées sont décrites par des règles ou modèles, appelés « templates ».
Une « template » est un élément qui va comparer la valeur de son attribut « match » avec le nom de chaque balise du document XML. S'il y a correspondance, alors le contenu de « template » est interprété.
Les listings du document généré par XMLSPY sont disponible dans l'annexe 3 et le listing de la feuille de style dans l'annexe 4.

7.7 Etape 4 : Validation du document XML intermédiaire

Cette étape consiste à valider les documents XML générés à l'étape précédente. Ces documents XML sont plus riches sémantiquement que les documents d'origines. Cette richesse est due à l'ajout des balises sémantiques dans le texte exprimant la signification de la chaîne de caractères qu'elles délimitent. Les documents ne donnent ni indication de structure ni indication de formatage. Ils sont indépendants de la plate-forme et n'importe quelle application peut les traiter et les lire. Mais surtout ces documents sont **bien formés** et **valides** pour notre DTD.

7.8 Etape 5 : Génération de la présentation HTML

Dans cette étape, nous transformons les documents XML générés précédemment en documents HTML en utilisant une feuille de style XSL, qui permet de stipuler des transformations à appliquer au contenu du document XML pour en faire un document affichable, c'est-à-dire de transformer le code XML en code HTML. Dans l'étape 3, nous avons utilisé une feuille de style pour pouvoir générer un document XML à partir d'un autre document XML. Par contre dans cette étape, l'utilisation du feuille de style permet la génération d'un document HTML à partir d'un document XML.

[14] SAX (Simple API for XML) [voir chapitre d'analyseurs XML SAX et DOM].

La Figure 5 décrit cette étape :

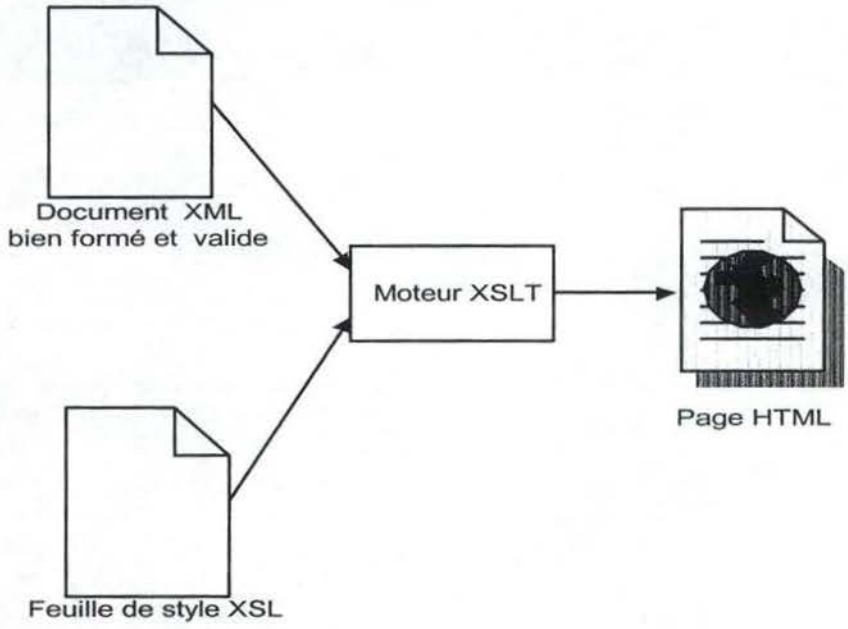


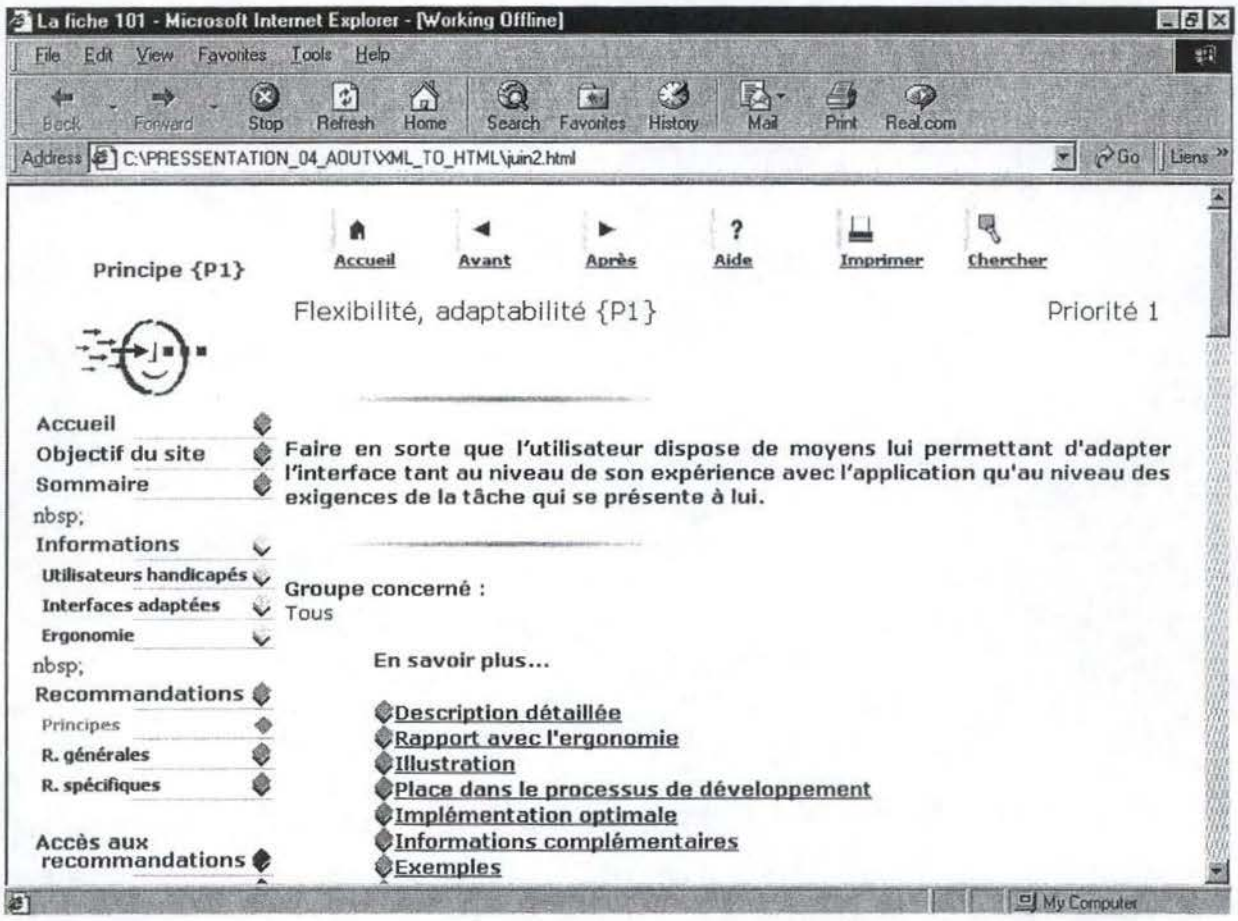
Figure 5 : Processus de transformation d'un document XML valide en HTML.

Le but de cette dernière partie est d'utiliser la flexibilité des documents XML pour pouvoir transformer un document XML en un document HTML. Ce dernier sera destiné à être afficher par un navigateur Web.
En XML, les données sont formatées par rapport à leur sémantique et non par rapport à leur présentation. Pour pouvoir mettre en forme ces données sur le Web, nous allons utiliser le langage HTML. La feuille de style donc, va construire une présentation en HTML, dans laquelle nous n'aurons plus qu'a intégrer les données provenant du code XML.
Cette étape consiste à transformer les balises sémantiques du document XML en d'autres sortes de balises qui sont prédéfinis et qui sont uniquement destinées au formatage visuel et graphique des données.
Le résultat de cette feuille de style est d'intégré les données du fichier XML dans une structure HTML et de produire du code HTML lisible par le navigateur.

Cette étape représente le résultat final du processus de transformation. Elle présente les documents HTML générés par ce processus de transformation. Ces documents sont distinées à être affichés par un navigateur.

Chaque fiche est présentée sous forme d'une page Web complexe. En effet ces pages suivent le même style de présentation qui a été définie dans l'existant.

La figure ci-dessous est un exemple de fiche sous forme de page Web.



Conclusion

Le travail présenté dans ce mémoire visait à migrer des fichiers texte plat sans aucune structure XML ou des fichiers propriétaires Word vers une structure XML arborescente et facile à analyser.

Cette transformation est basée sur l'ajout de la sémantique dans ces fichiers en intégrant des balises qui structurent les données contenues dans ces fichiers. Cette transformation est réalisée par programmation ou par l'utilisation d'un outil de développement XMLSPY qui offre la possibilité de lire et d'interpréter les fichiers binaires Word.

Cette migration a été développé en vue d'une application concrète : répondre aux besoins en matière de génération automatique des pages HTML à partir des fichiers de départ. Il existait déjà d'une autre technologie pour créer des pages HTML d'une manière semi-automatique à partir de ces fichiers : Dreamweaver par exemple.

Nous avons donné un aperçu détaillé, montrant les avantages mais surtout les limites importantes du langage HTML et la difficulté d'analyser le code source de ces fichiers HTML.

1- difficulté d'analyser par programme le code des fichiers HTML,
2- difficulté de modification ou de mise à jour de ces fichiers,
3- absence de balises de structuration de données,
4- balises destinés uniquement pour la présentation visuelle,
5- difficulté de séparer présentation et contenu informationnel des pages HTML…etc.

Notre méthode a été développé principalement en vu de combler les lacunes des fichiers textes plats ou des fichiers binaires Word et les fichiers HTML. En effet la transformations des fichiers textes plats ou des fichiers binaires Word en document XML élargit le champ d'utilisation de ces documents.

L'apport de cette transformation est l'intégration de la technologie XML à l'existant et surtout la flexibilité et la portabilité des ces documents.

Utilisant cette transformation, nous avons implémenté une feuille de style XSLT permettant de transformer un document XML en document HTML. Le dernier chapitre donne le détail de l'existant et présente nos solutions.

Bibliographie

[1] XML www.w3.org/TR/REC-xml
babel.alis.com/web_ml/xml/REC-xml.fr.html

[2] XSL www.w3.org/TR/WD-xsl

[3] DOM
Niveau 1 : www.w3.org/TR/REC-DOM-Level-1
Niveau 2 : www.w3.org/TR/DOM-Level-2

[4] SAX www.megginson.com/SAX/SAX2

[5] HTML www.w3.org/TR/WD-html40-970708
www.la-grange.net/w3c/html4 (version française)

[6] XPATH www.w3.org/TR/xpath

[7] XSLT XSL Transformations, version 1.0. www.w3.org/TR/xslt

[8] NAMESPACES www .w3.org/TR/REC-xml-names

[9] XML-SCHEMA www.w3.org/TR/REC-xml-schema

[10] A. Michard. XML langage et applications. Editions Eyrolles, 2001

[11] K. Williams, M.Brundage ..., M.Vanmane. XML et les bases de données. Editions Eyrolles, 2001.

[12] Brett Spell. Professional Java Programming. Editions Wrox.

[13] J.VanderdonckT, Guide ergonomique des interfaces hommes-machine, Presses Universitaires de Namur, Namur, 1994.

[14] B. La Forge. Entraprise Application Integration with XML and Java, 2001 Prentice Hall PTR.

Annexe 1 : Modèle de fichier texte balisé non XML

Fiche Numéro 103
Degré :
Principe
Titre :
Compatibilité, cohérence externe {P3}
Description courte :
Faire en sorte que les applications et interfaces concernant des objets présents dans le " monde réel " les représentent, selon les cas et les possibilités, d'une manière identique, proche ou analogique, tant sur le plan du contenu informationnel que sur celui des actions et de leurs effets attendus et observables. Ceci impose que les interfaces et les techniques de dialogue soient cohérentes avec les autres outils dont se sert l'utilisateur. Il importe donc d'éviter une opposition entre les règles de fonctionnement du monde réel et les règles d'utilisation de l'application.
Groupe concerné :
Tous
En ce qui concerne les utilisateurs handicapés, la compatibilité doit s'évaluer d'une manière spécifique pour chaque groupe d'utilisateur. Elle ne sera pas du même ordre pour des usagers à vision correcte ou non-voyants. Le respect de ce principe est capital pour favoriser l'accessibilité au plus grand nombre.
Priorité :
1
Description détaillée :
La compatibilité désigne la méthode d'intégration des différents éléments du système tels que les utilisateurs, les tâches à exécuter, les informations reçues ou envoyées, l'environnement,...
Elle porte sur trois aspects :
La présentation et la représentation des informations et des données sur les interfaces utilisateurs : aussi proches que possible de leur représentation dans le monde réel non informatique (structure, formats, labels, nomenclature, disposition relative, interprétation, couleurs, formes, respect des codes habituels et conventions les concernant,...)
Les actions à entreprendre dans l'activité à l'écran : aussi logiques et proches que possible de celles que l'on appliquerait sur les objets physiques appartenant au monde réel (comme compléter les rubriques d'un formulaire dans l'ordre souhaité par l'utilisateur, éviter d'utiliser des instructions arbitraires intermédiaires,...)
Les effets des actions : instantanés (et non différés) permettant à l'utilisateur d'évaluer le résultat de ce qu'il a entrepris, de l'accepter ou de le corriger éventuellement (effets visibles et non différés des actions entreprises, réglages directs sur les objets à régler ou sur les instruments de réglage,...)
Rapport avec l'ergonomie :
L'utilisateur de systèmes informatiques à une représentation opérative du monde réel. Il connaît bien les actions à exécuter sur les objets du monde réel ayant leurs caractéristiques propres. Si chaque élément d'information correspondant aux objets du monde réel doit être recodé ou transformé pour être identifié et compris dans le système informatique, il en résulte une charge mentale de travail importante (alors que la puissance des systèmes informatiques permet de prendre ces recodifications intermédiaires en charge d'une manière transparente). Ces retraductions et réinterprétations sont souvent perçues par l'utilisateur final comme arbitraires et fastidieuses par rapport à la tâche. Outre la fatigue et le stress qu'elles causent, elles engendrent également d'importants risques d'erreurs.

En fait, tout vient de ce que le monde virtuel de l'informatique est une réduction du monde réel, réduction obtenue par application d'une vision du fonctionnement de ce monde au travers d'une logique pouvant être informatisée. Cette logique informatique est elle-même soumise à ses propres contraintes et, de ce point de vue, elle cherchera à influer sur ce qui se passe dans le monde réel, comme par exemple, au travers de ce principe d'économie qui fait qu'on " ne peut plus faire tout ce qu'on faisait avant qu'on n'utilise l'ordinateur ". Cette recherche relative d'économie entraîne une représentation caricaturale du fonctionnement du monde réel dans le monde informatique en présentant à l'utilisateur une dynamique des cas et de leur traitement simplifiée.

On parle également d'activités expertes pour désigner la compétence d'un usager dans son domaine d'activité du réel. L'analyse de l'activité vise à provoquer des observations permettant de dégager les éléments pertinents du travail. Une action dans le monde réel comporte, dans la majorité des cas, un effet directement observable. Tout homme est habitué à ces mécanismes qui constituent les fondements mêmes des actions quotidiennes. De même à une action est associé un temps "naturel" d'action. Si le temps de réaction du système informatique est habituellement sans lien avec ce temps naturel, surtout pour des opérations simples et élémentaires, il perturbera l'utilisateur.

Les personnes âgées préfèrent d'avantage que les personnes jeunes ce qui leur est familier. La technologie leur sera plus acceptable si les éléments de l'interface utilisateur leur sont déjà familiers. On se servira alors de correspondants dans le monde réel (tels ceux présents sur les panneaux de commande d'un téléviseur, d'une chaîne hi-fi, d'un téléphone, d'une voiture, d'un ATM,...). Les personnes âgées seront également plus à l'aise si elles peuvent anticiper le fonctionnement d'un dispositif sur base de leur expérience (il fonctionnera de la même manière que celle connue et sans surprise). Le non-respect de ce dernier facteur psychologique constitue souvent une raison pour laquelle les personnes âgées refusent d'utiliser des moyens informatiques. Elles ne savent pas comment ces moyens se comporteront et elles ne peuvent prédire les résultats qu'elles obtiendront.

En ergonomie, on parlera spécifiquement de compatibilité lorsqu'un dispositif de signalisation présente des liens avec un dispositif de commande ou d'entrée tel que la réponse des opérateurs entre le signal et la commande s'effectue avec le maximum de rapidité et de précision.

Illustration :

La notion de métaphore - du bureau par exemple - répond à ce principe de compatibilité puisque l'on tente de reproduire sur l'écran des objets, un espace d'informations et d'actions présentant des analogies avec le bureau réel. Les objets y sont manipulés directement et on observe le résultat de leurs actions.

Par l'usage de la souris, on "prend" l'objet, on l'ouvre, on le déplace, on le jette,... comme par analogie avec ce que l'on ferait dans le monde réel.

C'est la logique de l'outil que l'on applique sur un objet pour le transformer et qui conditionne la vie quotidienne en interaction avec le monde physique.

La présentation des informations sur écran selon la logique du WYSIWYG satisfait également à ce principe de comptabilité en évitant de devoir connaître, retenir et utiliser des systèmes intermédiaires de codage (comme c'était le cas avec les traitements de texte des générations antérieures, les logiciels de composition en imprimerie,...).

Si, dans un traitement de texte, il faut plusieurs secondes pour observer les effets d'une correction mineure (changement d'une lettre dans un mot, insertion d'un signe, passage à la ligne,....), le temps d'effet de l'action est non compatible avec celui du monde réel et a pour effet de perturber l'utilisateur.

Pour satisfaire au principe de compatibilité, les bordereaux et formulaires à compléter doivent se présenter sous le même aspect, avec les mêmes champs aux mêmes endroits que leur forme

papier habituelle. Les rubriques complétées doivent être enregistrées instantanément même si on ne remplit pas l'entièreté du formulaire en une fois, ce qui n'exclut pas la possibilité de les corriger. Cet aspect est particulièrement important pour les services publics dans la mesure où les citoyens sont familiarisés avec des formulaires qu'ils connaissent déjà. Ainsi aujourd'hui en France on peut capter, remplir et transmettre sa déclaration fiscale directement par Internet. Celle-ci se présente à l'utilisateur exactement comme le support papier correspondant (y compris les couleurs, les polices de caractères, la structure des pages,..). Les zones peuvent être remplies dans un ordre indifférent. On admet même qu'elle soit incomplète.
La composition d'une page Web directement par codage HTML va à l'encontre du principe de compatibilité puisque, dans ce cas, on n'agit pas directement sur les caractéristiques de l'objet telles qu'elles se présentent dans le monde réel. De plus les instructions HTML se présentent selon un codage hermétique et arbitraire, généralement sans lien avec l'effet.

Place dans le processus de développement :

Niveau de la conception globale de l'application et du site Web et niveau du recueil des informations préalables et de la spécification des besoins.
Pour la compatibilité, c'est au niveau de l'étape de l'analyse fonctionnelle que des méthodes adaptées doivent être utilisées pour cerner les caractéristiques pertinentes des informations, des actions à entreprendre et des effets habituels observables. Les méthodes classiques en analyse fonctionnelle ne prennent pas assez en compte la dynamique du réel des utilisateurs. L'analyse ergonomique de l'activité des utilisateurs, du champ informationnel les concernant dans la tâche, de l'observation en temps réel de leurs actions (qui différent des données du point de vue informatique) est une méthode adéquate pour optimiser la compatibilité.

Implémentation optimale :

Se faire assister par un spécialiste - ergonome - des questions relatives à la compatibilité et des méthodes d'analyse et d'observation des utilisateurs.
L'ergonome aidera à dégager les éléments cognitifs fondamentaux et pertinents auxquels les applications informatiques doivent répondre.

Informations complémentaires :

Certains auteurs appellent également " compatibilité ", les liens existants entre les caractéristiques des utilisateurs (mémoire, perceptions, habitudes, compétence, âge, attentes,...). Ces caractéristiques sont à la base même de la démarche en ergonomie et ne peuvent être ramenées à un principe en soi car omniprésentes dans toutes les recommandations et principes développés.

Exemple négatif :

Dans cette représentation d'un hall de gare où le chemin à parcourir, pour se diriger vers un endroit donné de la station, est affiché à la demande sur l'écran d'une borne interactive (PIT), on constate que les déplacements représentés sont exactement contraires aux mouvements naturels. Les pas qui se déplacent sur l'écran (et qui comportent donc un référentiel fort), tournent sur l'écran à gauche, lorsqu'il faut aller vers la droite,... Il y a donc non compatibilité entre la projection de soi sur le marcheur représenté sur l'écran et les directions à prendre lorsqu'on effectuera réellement le chemin.

Recommandations associées :

Homogénéité, cohérence interne {P2}
Utilisabilité, opérativité, utilité, facilité d'utilisation {P6}

Mots Clés :

compatibilité - cohérence - représentation des informations - feed-back - codes - conventions - habitudes - charge mentale - conception globale - spécification des besoins - analyse de l'existant - analyse fonctionnelle - analyse de l'activité

Références :

Spérandio J.-C., (1984), L'ergonomie du travail mental. Masson, Paris, 130p.

Annexe 2 : Programme Java

```
import java.io.*;
import java.util.*;

class TexteToXML
{
  public static void main(String[] args)
  {
   if (args.length != 1)
     {
      System.err.println("Usage : java TexteToXML  NomDeFiche > NomDeFiche.xml ");
      System.exit(1);
     }
   else
     {
      try
        {
         byte buffer[] = new byte [20480];
         FileInputStream fis = new FileInputStream(args[0]);
         FileOutputStream fos = new FileOutputStream("outputFile");
         int c;

         int bytes = fis.read(buffer,0,20480);
         String str = new String(buffer);
         StringTokenizer st = new StringTokenizer(str);
         /*entête du document Xml*/
         System.out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>");
         System.out.println("<SafirDocument>");

         String word = st.nextToken();
         if (word.equals("Fiche"))
           {
            System.out.println("<TFicheNumero>");
            System.out.println("<Fnumber>"+word+" "+st.nextToken() \n
                              +" "+st.nextToken()+" "+ \n
                              st.nextToken()+"</Fnumber>");
            System.out.println("</TFicheNumero>");
           }
         //System.out.println("okkkkkkkkk"+word+" koooooo");
         String tempDeg = st.nextToken();
         if (tempDeg.equals("Degré :"))
           {
            System.out.println("<TDegre>");
            System.out.println("<TitreDegre>"+tempDeg+ "</TitreDegre>");
            //  }else
            System.out.println("<TypeDegre>");
            while  ( !tempDeg.equals("Titre :"))
              {

               tempDeg=st.nextToken();
               if (!tempDeg.equals("Titre :"))
                 {
                  System.out.print(tempDeg+" ");
                 }
               // System.out.println("tmpppp "+tempDeg+" mpmpmp");
              }
```

```
System.out.println(" ");
System.out.println("</TypeDegre>");
System.out.println("</TDegre>");



if (tempDeg.equals("Titre :"))
   {
    System.out.println("<TDocument>");
    System.out.println("<TitreDoc>"+tempDeg+ "</TitreDoc>");
    // System.out.println("</TDocument>");
   }
System.out.println("<Titre>");
while ( !tempDeg.equals("Description"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Description"))
       {
        System.out.print(tempDeg+" ");
       }
    // System.out.println("tmpppp "+tempDeg+" mpmpmp");
   }
System.out.println("</Titre>");
System.out.println("</TDocument>");

// Traitement de "Description courte :"
if (tempDeg.equals("Description"))
   {
    System.out.println("<TDescriptionCourte>");
    System.out.println("\t<TitreDescriptionCourte>");
    System.out.print("\t\t"+tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    System.out.println("\n\t</TitreDescriptionCourte>");
   }

//Traitement des paragraphes de "Description courte :"
System.out.println("\t<PragDC>");

while ( !tempDeg.equals("Groupe"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Groupe"))
       {
        System.out.print(tempDeg+" ");
        // accum = accum + " "+ tempDeg;
       }
    // System.out.println("tmpppp "+tempDeg+" mpmpmp");
   }
//System.out.print("\t\t"+ accum+" ");
System.out.println("\n\t</PragDC>");
System.out.println("</TDescriptionCourte>");

// Traitement de "Groupe concerné :"
if (tempDeg.equals("Groupe"))
   {
    System.out.println("<TGroupeConcerne>");
```

```
    System.out.println("\t<TitreGroupeConcerne>");
    //System.out.print(tempDeg+" ");
    System.out.print(tempDeg);
    tempDeg=st.nextToken();
    System.out.print(" "+tempDeg+" ");
    System.out.println("\n\t</TitreGroupeConcerne>");
   }


System.out.println("\t<GroupeConcerne>");
tempDeg=st.nextToken();
System.out.println(tempDeg);
System.out.println("\n\t</GroupeConcerne>");

// Traitement du paragraphe du  "Groupe concerné :"
System.out.println("\t<ParaGroupeConcerne>");
while  ( !tempDeg.equals("Priorité :"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Priorité :"))
       {
        System.out.print(tempDeg+" ");
        // accum = accum + " "+ tempDeg;
       }
    // System.out.println("tmpppp "+tempDeg+" mpmpmp");
   }
System.out.println("\n\t</ParaGroupeConcerne>");
System.out.println("</TGroupeConcerne>");

// Traitement de "Priorité :"
if (tempDeg.equals("Priorité :") && st.hasMoreTokens() )
   {
    System.out.println("<TPriorite>");
    System.out.println("\t<TitrePriorite>");
    System.out.println(tempDeg+"\n\t</TitrePriorite>");

   }
System.out.println("\t<ValeurPriorite>");
while  ( !tempDeg.equals("Description"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Description"))
       {
        System.out.print(tempDeg+" ");
        // accum = accum + " "+ tempDeg;
       }
    // System.out.println("tmpppp "+tempDeg+" mpmpmp");
   }
System.out.println("\n\t</ValeurPriorite>");
System.out.println("</TPriorite>");

// Traitement de "Description détaillée :"

if (tempDeg.equals("Description") && st.hasMoreTokens() )
   {
    System.out.println("<TDescriptionDetaillee>");
    System.out.println("\t<TitreDescriptionDetaillee>");
    System.out.print(tempDeg+" ");
```

```
    tempDeg=st.nextToken();
    System.out.println(tempDeg +"\n\t<TitreDescriptionDetaillee>");
   }

// Traitement des paragraphe de "Description détaillée :"
System.out.println("<ParaDescriptionDetaillee>");
while ( !tempDeg.equals("Rapport"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Rapport"))
       {
        System.out.print(tempDeg+" ");

       }

   }

System.out.println("\n\t<ParaDescriptionDetaillee>");
System.out.println("</TDescriptionDetaillee>");

// Traitement de "Rapport avec l'ergonomie :"
if (tempDeg.equals("Rapport") && st.hasMoreTokens() )
   {
    System.out.println("<TRapportAvecErgonomie>");
    System.out.println("\t<TitreRapportAvecErgonomie>");
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
                                System.out.println(tempDeg
                                +"\n\t<TitreRapportAvecErgonomie>");
    /* System.out.println("<TRapportAvecErgonomie>");
      System.out.println("<TitreRapportAvecErgonomie>");
    */
   }

System.out.println("<ParaRapportAvecErgonomie>");
while ( !tempDeg.equals("Illustration :"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Illustration :"))
       {
        System.out.print(tempDeg+" ");

       }

   }

System.out.println("\n\t<ParaDescriptionDetaillee>");
System.out.println("</TRapportAvecErgonomie>");

//Traitement de "Illustration"

if (tempDeg.equals("Illustration :") && st.hasMoreTokens() )
   {
    System.out.println("<TIllustration>");
    System.out.println("\t<TitreIllustration>");
    System.out.print(tempDeg+" ");
```

```
    System.out.println("\n\t</TitreIllustration>");

  }

//traitement des paragraphes Illustration
System.out.println("\n\t<ParaIllustration>");
while ( !tempDeg.equals("Place"))
  {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Place"))
      {
       System.out.print(tempDeg+" ");

      }

  }

System.out.println("\n\t</ParaIllustration>");
System.out.println("</TIllustration>");

//Traitement du titre "Place dans le processus de développement :"

if (tempDeg.equals("Place") && st.hasMoreTokens() )
  {
    System.out.println("<TPlaceDansProcessusDDeveloppement>");

                                System.out.println("\t<TitrePlaceDansProcessusDDeveloppement>"
                        );
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");

                        System.out.println("\n\t</TitrePlaceDansProcessusDDeveloppement>");

  }

//traitement des paragraphes "Place dans le processus de développement :"

System.out.println("<ParaPlaceDansProcessusDDeveloppement>");
while ( !tempDeg.equals("Implémentation"))
  {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Implémentation"))
      {
       System.out.print(tempDeg+" ");

      }

  }
```

```
System.out.println("\n\t</ParaPlaceDansProcessusDDeveloppement>");
System.out.println("</TPlaceDansProcessusDDeveloppement>");


//Traitement du titre "Implémentation optimale :"
if (tempDeg.equals("Implémentation") && st.hasMoreTokens() )
   {
    System.out.println("<TImplementationOptimale>");
    System.out.println("\t<TitreImplementationOptimale>");
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    System.out.println("\n\t</TitreIllustration>");
   }

//traitement des paragraphes du "Implémentation optimale :"
System.out.println("<ParaImplementationOptimale>");
while ( !tempDeg.equals("Informations"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Informations"))
       {
        System.out.print(tempDeg+" ");

       }

   }

System.out.println("\n\t</ParaImplementationOptimale>");
System.out.println("</TImplementationOptimale>");


//Traitement du "Information complémentaire :"
if (tempDeg.equals("Implémentation") && st.hasMoreTokens() )
   {
    System.out.println("<TInformationComplementaire>");
    System.out.println("\t<TitreInformationComplementaire>");
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    System.out.println("\n\t</TitreInformationComplementaire>");
   }

//traitement des paragraphes du "Information complémentaire :"
//si le sous paragraphes "Exemples :" existe

// Si le sous paragraphes "Exemples :" n'existe pas
System.out.println("<ParaInformationComplementaire>");
while ( !tempDeg.equals("Recommandations"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Recommandations"))
       {
        System.out.print(tempDeg+" ");

       }
   }
```

```
System.out.println("\n\t</ParaInformationComplementaire>");
System.out.println("</TInformationComplementaire>");

//Traitement du titre "Recommandations associées :"
if (tempDeg.equals("Recommandations") && st.hasMoreTokens() )
   {
    System.out.println("<TRecommandationsAssociees>");
    System.out.println("\t<TitreRecommandationsAssociees>");
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    System.out.println("\n\t</TitreRecommandationsAssociees>");
   }

// Traitement des paragraphes "Recommandations associées :"
System.out.println("<ParaRecommandationsAssociees>");
while ( !tempDeg.equals("Mots"))
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Mots"))
       {
        System.out.print(tempDeg+" ");
       }
   }
System.out.println("\n\t</ParaRecommandationsAssociees>");
System.out.println("</TRecommandationsAssociees>");

// Traitement du titre "Mots Clés"
if (tempDeg.equals("Mots") && st.hasMoreTokens() )
   {
    System.out.println("<TMotsCles>");
    System.out.println("\t<TitreMotsCles>");
    System.out.print(tempDeg+" ");
    tempDeg=st.nextToken();
    System.out.print(tempDeg+" ");
    System.out.println("\n\t</TitreMotsCles>");
   }
// Traitement des paragraphes "Mots Clés"
System.out.println("<ParaMotsCles>");
while ( !tempDeg.equals("Rendre")&& st.hasMoreTokens() )
   {

    tempDeg=st.nextToken();
    if (!tempDeg.equals("Rendre"))
       {
        System.out.print(tempDeg+" ");
       }
   }
System.out.println("\n\t</ParaMotsCles>");
System.out.println("</TMotsCles>");


// Traitement du titre "Références :"
if (tempDeg.equals("Rendre") && st.hasMoreTokens() )
   {
    System.out.println("<TReferences>");
    System.out.println("\t<TitreReferences>");
    System.out.print(tempDeg+" ");
    System.out.println("\n\t</TitreReferences>");
```

```
            }
          // Traitement des paragraphes "Références :"
          System.out.println("<ParaReferences>");
          while ( st.hasMoreTokens())
            {

             tempDeg=st.nextToken();
             if (st.hasMoreTokens())
               {
                System.out.print(tempDeg+" ");
               }
            }
          System.out.println("\n\t</ParaReferences>");
          System.out.println("</TReferences>");

         }

      }   catch(IOException e)
       {
         System.err.println("FileTest: " +e);
       }
    System.out.println("</SafirDocument>");
   }


 }
}
```

Annexe 3 : Type de document généré par XMLSPY


```
<Word-Document>
<Titre1>
        <p>Fiche Numéro 103</p>
</Titre1>
<Titre2>
        <p>Degré :</p>
</Titre2>
<Normal>
        <p>Principe</p>
</Normal>
<Titre2>
        <p>Titre :</p>
</Titre2>
<Intitulé>
        <p>Compatibilité, cohérence externe {P3}</p>
</Intitulé>
<Titre2>
        <p>Description courte :</p>
</Titre2>
<Normal>
        <p>Faire en sorte que les applications et interfaces concernant des objets présents dans
        le « monde réel » les représentent, selon les cas et les possibilités, d'une manière
        identique, proche ou analogique, tant sur le plan du contenu informationnel que sur
        celui des actions et de leurs effets attendus et observables. Ceci impose que les
        interfaces et les techniques de dialogue soient cohérentes avec les autres outils dont se
        sert l'utilisateur. Il importe donc d'éviter une opposition entre les règles de
        fonctionnement du monde réel et les règles d'utilisation de l'application.</p>
</Normal>
<Titre2>
        <p>Groupe concerné :</p>
</Titre2>
<Normal>
        <p>Tous </p>
        <p>En ce qui concerne les utilisateurs handicapés, la compatibilité doit s'évaluer
        d'une manière spécifique pour chaque groupe d'utilisateur. Elle ne sera pas du même
        ordre pour des usagers à vision correcte ou non-voyants. Le respect de ce principe est
        capital pour favoriser l'accessibilité au plus grand nombre.</p>
</Normal>
<Titre2>
        <p>Priorité :</p>
</Titre2>
<Normal>
        <p>1</p>
</Normal>
<Titre2>
        <p>Description détaillée :</p>
</Titre2>
```

<Normal>

La compatibilité désigne la méthode d'intégration des différents éléments du système tels que les utilisateurs, les tâches à exécuter, les informations reçues ou envoyées, l'environnement,...

Elle porte sur trois aspects :

La présentation et la représentation des informations et des données sur les interfaces utilisateurs : aussi proches que possible de leur représentation dans le monde réel non informatique (structure, formats, labels, nomenclature, disposition relative, interprétation, couleurs, formes, respect des codes habituels et conventions les concernant,...)

Les actions à entreprendre dans l'activité à l'écran : aussi logiques et proches que possible de celles que l'on appliquerait sur les objets physiques appartenant au monde réel (comme compléter les rubriques d'un formulaire dans l'ordre souhaité par l'utilisateur, éviter d'utiliser des instructions arbitraires intermédiaires,...)

Les effets des actions : instantanés (et non différés) permettant à l'utilisateur d'évaluer le résultat de ce qu'il a entrepris, de l'accepter ou de le corriger éventuellement (effets visibles et non différés des actions entreprises, réglages directs sur les objets à régler ou sur les instruments de réglage,...)

</Normal>

<Titre2>

Rapport avec l'ergonomie :

</Titre2>

<Normal>

L'utilisateur de systèmes informatiques à une représentation opérative du monde réel. Il connaît bien les actions à exécuter sur les objets du monde réel ayant leurs caractéristiques propres. Si chaque élément d'information correspondant aux objets du monde réel doit être recodé ou transformé pour être identifié et compris dans le système informatique, il en résulte une charge mentale de travail importante (alors que la puissance des systèmes informatiques permet de prendre ces recodifications intermédiaires en charge d'une manière transparente). Ces retraductions et réinterprétations sont souvent perçues par l'utilisateur final comme arbitraires et fastidieuses par rapport à la tâche. Outre la fatigue et le stress qu'elles causent, elles engendrent également d'importants risques d'erreurs.

En fait, tout vient de ce que le monde virtuel de l'informatique est une réduction du monde réel, réduction obtenue par application d'une vision du fonctionnement de ce monde au travers d'une logique pouvant être informatisée. Cette logique informatique est elle-même soumise à ses propres contraintes et, de ce point de vue, elle cherchera à influer sur ce qui se passe dans le monde réel, comme par exemple, au travers de ce principe d'économie qui fait qu'on « ne peut plus faire tout ce qu'on faisait avant qu'on n'utilise l'ordinateur ». Cette recherche relative d'économie entraîne une représentation caricaturale du fonctionnement du monde réel dans le monde informatique en présentant à l'utilisateur une dynamique des cas et de leur traitement simplifiée.

On parle également d'activités expertes pour désigner la compétence d'un usager dans son domaine d'activité du réel. L'analyse de l'activité vise à provoquer des observations permettant de dégager les éléments pertinents du travail. Une action dans le monde réel comporte, dans la majorité des cas, un effet directement observable. Tout homme est habitué à ces mécanismes qui constituent les fondements mêmes des actions quotidiennes. De même à une action est associé un temps "naturel" d'action. Si le temps de réaction du système informatique est habituellement sans lien avec ce

temps naturel, surtout pour des opérations simples et élémentaires, il perturbera l'utilisateur.

Les personnes âgées préfèrent d'avantage que les personnes jeunes ce qui leur est familier. La technologie leur sera plus acceptable si les éléments de l'interface utilisateur leur sont déjà familiers. On se servira alors de correspondants dans le monde réel (tels ceux présents sur les panneaux de commande d'un téléviseur, d'une chaîne hi-fi, d'un téléphone, d'une voiture, d'un ATM,...). Les personnes âgées seront également plus à l'aise si elles peuvent anticiper le fonctionnement d'un dispositif sur base de leur expérience (il fonctionnera de la même manière que celle connue et sans surprise). Le non-respect de ce dernier facteur psychologique constitue souvent une raison pour laquelle les personnes âgées refusent d'utiliser des moyens informatiques. Elles ne savent pas comment ces moyens se comporteront et elles ne peuvent prédire les résultats qu'elles obtiendront.

En ergonomie, on parlera spécifiquement de compatibilité lorsqu'un dispositif de signalisation présente des liens avec un dispositif de commande ou d'entrée tel que la réponse des opérateurs entre le signal et la commande s'effectue avec le maximum de rapidité et de précision.

</Normal>

<Titre2>

Illustration :

</Titre2>

<Normal>

La notion de métaphore - du bureau par exemple - répond à ce principe de compatibilité puisque l'on tente de reproduire sur l'écran des objets, un espace d'informations et d'actions présentant des analogies avec le bureau réel. Les objets y sont manipulés directement et on observe le résultat de leurs actions.

Par l'usage de la souris, on "prend" l'objet, on l'ouvre, on le déplace, on le jette,... comme par analogie avec ce que l'on ferait dans le monde réel.

C'est la logique de l'outil que l'on applique sur un objet pour le transformer et qui conditionne la vie quotidienne en interaction avec le monde physique.

La présentation des informations sur écran selon la logique du WYSIWYG satisfait également à ce principe de comptabilité en évitant de devoir connaître, retenir et utiliser des systèmes intermédiaires de codage (comme c'était le cas avec les traitements de texte des générations antérieures, les logiciels de composition en imprimerie,...).

Si, dans un traitement de texte, il faut plusieurs secondes pour observer les effets d'une correction mineure (changement d'une lettre dans un mot, insertion d'un signe, passage à la ligne,....), le temps d'effet de l'action est non compatible avec celui du monde réel et a pour effet de perturber l'utilisateur.

Pour satisfaire au principe de compatibilité, les bordereaux et formulaires à compléter doivent se présenter sous le même aspect, avec les mêmes champs aux mêmes endroits que leur forme papier habituelle. Les rubriques complétées doivent être enregistrées instantanément même si on ne remplit pas l'entièreté du formulaire en une fois, ce qui n'exclut pas la possibilité de les corriger. Cet aspect est particulièrement important pour les services publics dans la mesure où les citoyens sont familiarisés avec des formulaires qu'ils connaissent déjà. Ainsi aujourd'hui en France on peut capter, remplir et transmettre sa déclaration fiscale directement par Internet. Celle-ci se présente à l'utilisateur exactement comme le support papier correspondant (y compris les couleurs, les polices de caractères, la structure des

pages,..). Les zones peuvent être remplies dans un ordre indifférent. On admet même qu'elle soit incomplète.

La composition d'une page Web directement par codage HTML va à l'encontre du principe de compatibilité puisque, dans ce cas, on n'agit pas directement sur les caractéristiques de l'objet telles qu'elles se présentent dans le monde réel. De plus les instructions HTML se présentent selon un codage hermétique et arbitraire, généralement sans lien avec l'effet.

</Normal>

<Titre2>

Place dans le processus de développement :

</Titre2>

<Normal>

Niveau de la conception globale de l'application et du site Web et niveau du recueil des informations préalables et de la spécification des besoins.

Pour la compatibilité, c'est au niveau de l'étape de l'analyse fonctionnelle que des méthodes adaptées doivent être utilisées pour cerner les caractéristiques pertinentes des informations, des actions à entreprendre et des effets habituels observables. Les méthodes classiques en analyse fonctionnelle ne prennent pas assez en compte la dynamique du réel des utilisateurs. L'analyse ergonomique de l'activité des utilisateurs, du champ informationnel les concernant dans la tâche, de l'observation en temps réel de leurs actions (qui différent des données du point de vue informatique) est une méthode adéquate pour optimiser la compatibilité.

</Normal>

<Titre2>

Implémentation optimale :

</Titre2>

<Normal>

Se faire assister par un spécialiste - ergonome - des questions relatives à la compatibilité et des méthodes d'analyse et d'observation des utilisateurs.

L'ergonome aidera à dégager les éléments cognitifs fondamentaux et pertinents auxquels les applications informatiques doivent répondre.

</Normal>

<Titre2>

Informations complémentaires :

</Titre2>

<Normal>

Certains auteurs appellent également « compatibilité », les liens existants entre les caractéristiques des utilisateurs (mémoire, perceptions, habitudes, compétence, âge, attentes,...). Ces caractéristiques sont à la base même de la démarche en ergonomie et ne peuvent être ramenées à un principe en soi car omniprésentes dans toutes les recommandations et principes développés.

</Normal>

<Titre2>

Exemple négatif :

</Titre2>

<Normal>

Dans cette représentation d'un hall de gare où le chemin à parcourir, pour se diriger vers un endroit donné de la station, est affiché à la demande sur l'écran d'une borne interactive (PIT), on constate que les déplacements représentés sont exactement contraires aux mouvements naturels. Les pas qui se déplacent sur l'écran (et qui

comportent donc un référentiel fort), tournent sur l'écran à gauche, lorsqu'il faut aller vers la droite,... Il y a donc non compatibilité entre la projection de soi sur le marcheur représenté sur l'écran et les directions à prendre lorsqu'on effectuera réellement le chemin.
</Normal>
<Titre2>
Recommandations associées :
</Titre2>
<Normal>
Homogénéité, cohérence interne {P2} Utilisabilité, opérativité, utilité, facilité d'utilisation {P6}
</Normal>
<Titre2>
Mots Clés :
</Titre2>
<Normal>
compatibilité - cohérence - représentation des informations - feed-back - codes - conventions - habitudes - charge mentale - conception globale - spécification des besoins - analyse de l'existant - analyse fonctionnelle - analyse de l'activité
</Normal>
<Titre2>
Références :
</Titre2>
<Normal>
Spérandio J.-C., (1984), L'ergonomie du travail mental. Masson, Paris, 130p.
</Word-Document>

Annexe 4 : Feuille de style XSL (XML→ XML)

```
<?xml version='1.0' encoding="UTF-8" ?>
<!—cette feuille de style transforme un document XML généré par XMLSPY en un autre document XMl qui respecte le
vocabulaire et les contraintes décritent dans notre DTD -->


<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="xml" encoding="UTF-8" version="1.0" indent="yes" />
    < !—Traitement de l'élément root du document décrit dans l'Annexe 2 -->
    <xsl:template match="Word-Document ">

            <SafirDocument>
                <xsl:apply-templates/>
        </SafirDocument>
    </xsl:template>

<xsl:template match="Titre1/p">
        <TFicheNumero>
                <Fnumber>
                    <xsl:value-of select="."/>
                </Fnumber>
            </TFicheNumero>
</xsl:template>

<xsl:template match="Titre2/p">

<xsl:choose>

<xsl:when test="   text()='Degré :' ">
<TDegre>
        <TitreDegre><xsl:value-of select="../p"/></TitreDegre>
        <xsl:choose>
            <xsl:when test="../following-sibling::Normal[1]">
            <TypeDegre><xsl:value-of select="./../following-sibling::Normal[1]/p"/></TypeDegre>
            </xsl:when>
        </xsl:choose>

    </TDegre>
</xsl:when>


<xsl:when test="   text()='Titre :' ">
<TDocument>
        <TitreDoc><xsl:value-of select="."/></TitreDoc>
        <xsl:choose>
        <xsl:when test="../following-sibling::Intitulé[1]">
        <Titre><xsl:value-of select="./../following-sibling::Intitulé[1]/p"/></Titre>
        </xsl:when>
        </xsl:choose>
    </TDocument>
</xsl:when>
<!--
<!ELEMENT TDescriptionCourte (TitreDescriptionCourte, PragDC*)>
<!ELEMENT TitreDescriptionCourte (#PCDATA)>
<!ELEMENT PragDC (#PCDATA )>
-->
<xsl:when test="  text()='Description courte :' ">
<TDescriptionCourte>
        <TitreDescriptionCourte>  <xsl:value-of select="."/>
        </TitreDescriptionCourte>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1] ">
            <xsl:for-each  select="../following-sibling::Normal[1]/p ">
            <PragDC><xsl:value-of select="."/></PragDC>
            </xsl:for-each>
```

```
            </xsl:when>
        </xsl:choose>
</TDescriptionCourte>
</xsl:when>


<!--
<!ELEMENT TGroupeConcerne (TitreGroupeConcerne, GroupeConcerne*, ParaGroupeConcerne*)>
<!ELEMENT TitreGroupeConcerne (#PCDATA)>
<!ELEMENT GroupeConcerne (#PCDATA)>
<!ELEMENT ParaGroupeConcerne (#PCDATA )>
-->

<xsl:when test="  text()='Groupe concerné :' ">
<TGroupeConcerne>
        <TitreGroupeConcerne>    <xsl:value-of select="."/>
        </TitreGroupeConcerne>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
                    <ParaGroupeConcerne ><xsl:value-of select="."/>
                                        </ParaGroupeConcerne >
            </xsl:for-each>
        </xsl:when>
        <xsl:otherwise>
            <!-- nothing -->
        </xsl:otherwise>
        </xsl:choose>
</TGroupeConcerne>
</xsl:when>

<!--
<!ELEMENT TPriorite (TitrePriorite, ValeurPriorite)>
<!ELEMENT TitrePriorite (#PCDATA)>
<!ELEMENT ValeurPriorite (#PCDATA)>
-->

<xsl:when test="  text()='Priorité :' ">
<TPriorite>
        <TitrePriorite><xsl:value-of select="."/> </TitrePriorite>
        <xsl:choose>

        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
                <ValeurPriorite><xsl:value-of select="."/></ValeurPriorite>
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
    </TPriorite>
</xsl:when>


<!--
<!ELEMENT TDescriptionDetaillee (TitreDescriptionDetaillee, ParaDescriptionDetaillee*)>
<!ELEMENT TitreDescriptionDetaillee (#PCDATA)>
<!ELEMENT ParaDescriptionDetaillee (#PCDATA)>
-->
<xsl:when test="  text()='Description détaillée :' ">
<TDescriptionDetaillee>
        <TitreDescriptionDetaillee><xsl:value-of select="."/></TitreDescriptionDetaillee>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
                <ParaDescriptionDetaillee><xsl:value-of select="."/></ParaDescriptionDetaillee>
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
```

```
</TDescriptionDetaillee>
</xsl:when>

<!--
<!ELEMENT TRapportAvecErgonomie (TitreRapportAvecErgonomie, ParaRapportAvecErgonomie*)>
<!ELEMENT TitreRapportAvecErgonomie (#PCDATA)>
<!ELEMENT ParaRapportAvecErgonomie (#PCDATA)>
-->
<xsl:when test="  text()='Rapport avec l érgonomie :' ">
<TRapportAvecErgonomie>
        <TitreRapportAvecErgonomie><xsl:value-of select="."/></TitreRapportAvecErgonomie>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaRapportAvecErgonomie>     <xsl:value-of select="."/> </ParaRapportAvecErgonomie>
        </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TRapportAvecErgonomie>
</xsl:when>
<!-- deixième cas -->
<xsl:when test="  text()='Rapport avec l érgonomie :' ">
<TRapportAvecErgonomie>
        <TitreRapportAvecErgonomie><xsl:value-of select="."/></TitreRapportAvecErgonomie>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaRapportAvecErgonomie>     <xsl:value-of select="."/> </ParaRapportAvecErgonomie>
        </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TRapportAvecErgonomie>
</xsl:when>

<!-- dexiéme version de ce titre -->
<xsl:when test="  text()='Rapport avec l érgonomie:' ">
<TRapportAvecErgonomie>
        <TitreRapportAvecErgonomie><xsl:value-of select="."/></TitreRapportAvecErgonomie>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaRapportAvecErgonomie>     <xsl:value-of select="."/> </ParaRapportAvecErgonomie>
        </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TRapportAvecErgonomie>
</xsl:when>

<xsl:when test="  text()='Rapport avec l érgonomie :' ">
<TRapportAvecErgonomie>
        <TitreRapportAvecErgonomie><xsl:value-of select="."/></TitreRapportAvecErgonomie>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaRapportAvecErgonomie>     <xsl:value-of select="."/> </ParaRapportAvecErgonomie>
        </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TRapportAvecErgonomie>
</xsl:when>

<!--
<!ELEMENT TIllustration (TitreIllustration, ParaIllustration*)>
```

```
<!ELEMENT TitreIllustration (#PCDATA)>
<!ELEMENT ParaIllustration (#PCDATA)>
-->

<xsl:when test="  text()='Illustration :' ">
<TIllustration>
        <TitreIllustration><xsl:value-of select="."/></TitreIllustration>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
    <ParaIllustration><xsl:value-of select="."/></ParaIllustration>
        </xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TIllustration>
</xsl:when>

<!--
<!ELEMENT TPlaceDansProcessusDDeveloppement (TitrePlaceDansProcessusDDeveloppement,
ParaPlaceDansProcessusDDeveloppement*)>
<!ELEMENT TitrePlaceDansProcessusDDeveloppement (#PCDATA)>
<!ELEMENT ParaPlaceDansProcessusDDeveloppement (#PCDATA) >
-->
<xsl:when test="  text()='Place dans le processus de développement :' ">
<TPlaceDansProcessusDDeveloppement>
        <TitrePlaceDansProcessusDDeveloppement><xsl:value-of select="."/></TitrePlaceDansProcessusDDeveloppement>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaPlaceDansProcessusDDeveloppement><xsl:value-of
select="."/></ParaPlaceDansProcessusDDeveloppement>
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TPlaceDansProcessusDDeveloppement>
</xsl:when>

<!--
<!ELEMENT TImplementationOptimale (TitreImplementationOptimale, ParaImplementationOptimale*)>
<!ELEMENT TitreImplementationOptimale (#PCDATA)>
<!ELEMENT ParaImplementationOptimale (#PCDATA)>
-->

<xsl:when test="  text()='Implémentation optimale :' ">
<TImplementationOptimale>
        <TitreImplementationOptimale> <xsl:value-of select="."/>     </TitreImplementationOptimale>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaImplementationOptimale><xsl:value-of select="."/></ParaImplementationOptimale>
                    </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TImplementationOptimale>
</xsl:when>

<!--
<!ELEMENT TInformationComplementaire (TitreInformationComplementaire, ParaInformationComplementaire*)>
<!ELEMENT TitreInformationComplementaire (#PCDATA)>
<!ELEMENT ParaInformationComplementaire (#PCDATA)>
-->
<xsl:when test="  text()='Information complémentaire :' ">
<TInformationComplementaire>
        <TitreInformationComplementaire> <xsl:value-of select="."/>
        </TitreInformationComplementaire>
```

```
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaInformationComplementaire><xsl:value-of select="."/> </ParaInformationComplementaire>

                                <\xsl:for-each>
        </xsl:when>
        </xsl:choose>


</TInformationComplementaire>
</xsl:when>

<!-- Dexième version "Informations complémentaires :" -->
<xsl:when test="  text()='Informations complémentaires :' ">
<TInformationComplementaire>
        <TitreInformationComplementaire> <xsl:value-of select="."/>
        </TitreInformationComplementaire>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaInformationComplementaire><xsl:value-of select="."/> </ParaInformationComplementaire>

                                <\xsl:for-each>
        </xsl:when>
        </xsl:choose>


</TInformationComplementaire>
</xsl:when>

<!--
<!ELEMENT TExemples (TitreExemples*, ParaExemples*)>
<!ELEMENT TitreExemples (#PCDATA)>
<!ELEMENT ParaExemples (#PCDATA)>
-->
<xsl:when test="  text()='Exemple négatif :' ">
<TExemples >
        <TitreExemples> <xsl:value-of select="."/>
        </TitreExemples>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaExemples ><xsl:value-of select="."/> </ParaExemples >

            <\xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TExemples>
</xsl:when>

<!-- paragraphe Exemples : -->
<xsl:when test="  text()='Exemples :' ">
<TExemples >
        <TitreExemples> <xsl:value-of select="."/>
        </TitreExemples>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaExemples ><xsl:value-of select="."/> </ParaExemples >

            <\xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TExemples>
</xsl:when>
```

```
<!-- Paragraphe Exemple  positif/négatif -->
<xsl:when test="  text()='Exemple négatif :' ">
<TExemples >
        <TitreExemples> <xsl:value-of select="."/>
        </TitreExemples>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaExemples ><xsl:value-of select="."/> </ParaExemples >

            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TExemples>
</xsl:when>

<!-- Dexième version  "Exemple négatif"-->
<xsl:when test="  text()='Exemple négatif' ">
<TExemples >
        <TitreExemples> <xsl:value-of select="."/>
        </TitreExemples>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaExemples ><xsl:value-of select="."/> </ParaExemples >

            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TExemples>
</xsl:when>

<!-- Dexième version  "Exemples négatifs"-->
<xsl:when test="  text()='Exemples négatifs' ">
<TExemples >
        <TitreExemples> <xsl:value-of select="."/>
        </TitreExemples>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaExemples ><xsl:value-of select="."/> </ParaExemples >
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TExemples>
</xsl:when>

<!-- 3ème version  "Exemples négatifs :"-->
<xsl:when test="  text()='Exemples négatifs :' ">
<TExemples >
        <TitreExemples> <xsl:value-of select="."/>
        </TitreExemples>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaExemples ><xsl:value-of select="."/> </ParaExemples >
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TExemples>
</xsl:when>


<!-- Exemples positifs et/ou négatifs : -->
<xsl:when test="  text()='Exemples positifs et/ou négatifs :' ">
<TExemples >
        <TitreExemples> <xsl:value-of select="."/>
```

```
        </TitreExemples>
        <xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaExemples ><xsl:value-of select="."/> </ParaExemples >
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
</TExemples>
</xsl:when>


<!--
<!ELEMENT TRecommandationsAssociees (TitreRecommandationsAssociees, ParaRecommandationsAssociees*)>
<!ELEMENT TitreRecommandationsAssociees (#PCDATA)>
<!ELEMENT ParaRecommandationsAssociees (#PCDATA)>
-->

<xsl:when test="  text()='Recommandations associées :' ">
<TRecommandationsAssociees>
        <TitreRecommandationsAssociees><xsl:value-of select="."/>
</TitreRecommandationsAssociees>
<xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaRecommandationsAssociees ><xsl:value-of select="."/> </ParaRecommandationsAssociees >
                        </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TRecommandationsAssociees>
</xsl:when>


<!-- Dexième version "Recommandations associées:" -->
<xsl:when test="  text()='Recommandations associées:' ">
<TRecommandationsAssociees>
        <TitreRecommandationsAssociees><xsl:value-of select="."/>
</TitreRecommandationsAssociees>
<xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaRecommandationsAssociees ><xsl:value-of select="."/> </ParaRecommandationsAssociees >
                        </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TRecommandationsAssociees>
</xsl:when>

<!--
<!ELEMENT TMotsCles (TitreMotsCles, ParaMotsCles*)>
<!ELEMENT TitreMotsCles (#PCDATA)>
<!ELEMENT ParaMotsCles (#PCDATA)>
-->
<xsl:when test="  text()='Mots Clés'  ">
<TMotsCles>
        <TitreMotsCles><xsl:value-of select="."/>
</TitreMotsCles>
<xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaMotsCles ><xsl:value-of select="."/> </ParaMotsCles >
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>
```

```
</TMotsCles>
</xsl:when>


<!-- Dexième version "Mots Clés :"  -->
<xsl:when test="  text()='Mots Clés :'  ">
<TMotsCles>
        <TitreMotsCles><xsl:value-of select="."/>
</TitreMotsCles>
<xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaMotsCles ><xsl:value-of select="."/> </ParaMotsCles >
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TMotsCles>

</xsl:when>

<!--
<!ELEMENT TReferences (TitreReferences, ParaReferences*)>
<!ELEMENT TitreReferences (#PCDATA)>
<!ELEMENT ParaReferences (#PCDATA)>
-->
<xsl:when test="  text()='Références :' ">
<TReferences>
        <TitreReferences> <xsl:value-of select="."/>
</TitreReferences>
<xsl:choose>
        <xsl:when test="../following-sibling::Normal[1]">
            <xsl:for-each select="../following-sibling::Normal[1]/p">
            <ParaReferences ><xsl:value-of select="."/> </ParaReferences>
            </xsl:for-each>
        </xsl:when>
        </xsl:choose>

</TReferences>
</xsl:when>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

Annexe 5 : Feuille de style XSL (XML → HTML)

```
<!--?xml version="1.0" ?-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html" version="1.0" encoding="UTF-8"/>
<!-- ================================================ -->
<!--
<!ELEMENT SafirDocument (TFicheNumero, TDegre, TDocument, TDescriptionCourte, TGroupeConcerne, TPriorite,
TDescriptionDetaillee, TRapportAvecErgonomie, TIllustration, TPlaceDansProcessusDDeveloppement,
TImplementationOptimale, TInformationComplementaire, TExemples*, TRecommandationsAssociees, TMotsCles,
TReferences)>
-->
<!-- ================================================ -->

    <xsl:template match="SafirDocument ">
        <html>
            <head>

                <title>La fiche 101 </title>

            </head>
            <body>
             <table border="10" width="98%" name="intitule" cellspacing="0" cellpadding="0" align="center" >
            <tr>
  <td width="17%"> <!-- début table Entête Page -->
  <!-- cadre titre de gauche e haut -->
   <table name="entete" width="100%" border="10" cellpadding="0" bgcolor="orange" cellspacing="0" >
    <tr>
     <td height="28" width="136" class="coulcoco"></td>
     <td height="28" width="19" class="coulcoco"></td>
    </tr>
    <tr>
     <td height="25" width="136" class="coulblanc">
      <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size ="-1"><b>Principe
       {P1} </b></font></div>
     </td>
     <td height="25" width="19" class="coulblanc"> </td>
    </tr>
   </table><!-- fin table Entête Page -->
  </td>

<!-- cadre pour les commandes Imprimrnte, retour arrière ,avant -->
<td width="83%">
  <!--cadre en blanc" -->
   <!-- cadre des commades imprimante retour avant -->
   <table name="menuhaut" width="100%" border="10" cellpadding="0" bgcolor="black" cellspacing="0">
    <tr>
     <td align="left" width="6%" class="coulblanc" height="28"> nbsp;</td>
     <td align="left" width="14%" class="coulblanc" height="28"><a href="index.html" onMouseOver=" " onMouseOut="
"><img name="home1" border="0" src="saph-home-off.gif" width="28" height="28" alt="retour page
d'acceuil"/></a></td>
     <td align="left" width="14%" class="coulblanc" height="28"><a href="fiche100.htm" onMouseOver=" "
onMouseOut=""><img name="avant1" border="0" src="saph-avant-off.gif" width="28" height="28" alt="vers
recommandation précédente"/></a></td>
     <td align="left" width="14%" class="coulblanc" height="28"><a href="fiche102.htm" onMouseOver=""
onMouseOut=""><img name="apres1" border="0" src="saph-apres-off.gif" width="27" height="28" alt="vers
recommandation suivante"/></a></td>
     <td align="left" width="14%" class="coulblanc" height="28"><a href="aideinfo.htm" onMouseOver=""
onMouseOut=""><img name="aide1" border="0" src="saph-aide-off.gif" width="28" height="28" alt="aide sur le
site"/></a></td>
     <td align="left" width="14%" class="coulblanc" height="28"><a href="printinfo.htm" onMouseOver=""
onMouseOut=""><img name="print1" border="0" src="saph-print-off.gif" width="28" height="28" alt="choix
d'impression"/></a></td>
     <td align="left" width="14%" class="coulblanc" height="28"><a href="chercheinfo.htm" onMouseOver=""
onMouseOut=""><img name="cherche1" border="0" src="saph-cherche-off.gif" width="28" height="28" alt="recherche
dans le site"/></a></td>
     <td align="left" width="10%" class="coulblanc" height="28">nbsp;</td>
    </tr>
    <tr>
```

```
        <td align="left" width="6%" class="coulcoco" height="25">nbsp;</td>
        <td align="left" width="14%" class="coulcoco" height="25"><img src="bord-blanc.gif" width="3" height="25"
align="top"/>
         <font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><b><a href="index.html" onMouseOver=" "
onMouseOut="">Accueil</a></b></font></td>
        <td align="left" width="14%" class="coulcoco" height="25"><img src="bord-blanc.gif" width="3" height="25"
align="top"/>
         <font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><b><a href="fiche100.htm" onMouseOver=""
onMouseOut="">Avant</a> </b></font></td>
        <td align="left" width="14%" class="coulcoco" height="25"><img src="bord-blanc.gif" width="3" height="25"
align="top"/>
         <font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><b><a href="fiche102.htm" onMouseOver=""
onMouseOut=""></a></b></font></td>
        <td align="left" width="14%" class="coulcoco" height="25"><img src="bord-blanc.gif" width="3" height="25"
align="top"/>
         <font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><b><a href="aideinfo.htm" onMouseOver=""
onMouseOut="">Aide</a> </b></font></td>
        <td align="left" width="14%" class="coulcoco" height="22"><img src="bord-blanc.gif" width="3" height="25"
align="top"/>
         <font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><b><a href="printinfo.htm" onMouseOver=""
onMouseOut="">Imprimer</a></b></font></td>
        <td align="left" width="14%" class="coulcoco" height="25"><img src="bord-blanc.gif" width="3" height="25"
align="top"/>
         <font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><b><a href="chercheinfo.htm" onMouseOver=""
onMouseOut="">Chercher</a></b></font></td>
        <td align="left" width="10%" class="coulcoco" height="25"> nbsp;</td>
      </tr>
    </table>
</td>
</tr>
            <xsl:apply-templates/>
            </table>
            </body>
        </html>
    </xsl:template>

<!-- ============================================== -->
<!--
<!ELEMENT TFicheNumero (Fnumber)>
<!ELEMENT Fnumber (#PCDATA)>
-->
<!-- ============================================== -->
    <xsl:template match="TFicheNumero">
        <p>
            <b>
                <i>
                    <font face="Verdana, Arial, Helvetica, sans-serif" size="4">
                        <!--<xsl:value-of select="Fnumber"/>-->
                    </font>
                </i>
            </b>
        </p>
    </xsl:template>

<!-- ============================================== -->
<!--
<!ELEMENT TDegre (TitreDegre, TypeDegre)>
<!ELEMENT TitreDegre (#PCDATA)>
<!ELEMENT TypeDegre (#PCDATA)>
-->
<!-- ============================================== -->
    <xsl:template match="TDegre">
        <p>
            <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                <b>
                <!-- <xsl:value-of select="TitreDegre"/>-->
                </b>
            </font>
            <p align="JUSTIFY">
                <!--<xsl:value-of select="TypeDegre"/>-->
            </p>
```

```
        </p>
    </xsl:template>

<!-- ================================================ -->
<!--
<!ELEMENT TDocument (TitreDoc, Titre)>
<!ELEMENT TitreDoc (#PCDATA)>
<!ELEMENT Titre (#PCDATA)>
-->
<!-- ================================================ -->
    <xsl:template match="TDocument">

    <tr>
   <td>
<table border="10"  width="98%" name="intitule" cellspacing="0" cellpadding="0" align="center" >
      <tr>
        <td width="74%" height="30">
          <div><font face="Verdana, Arial, Helvetica, sans-serif"> <xsl:value-of select="Titre"/></font></div>
        </td>
        <td width="26%" height="30" align="right">
          <div><font face="Verdana, Arial, Helvetica, sans-serif"><font class="coulbleu">
          Priorité 1 </font> </font> </div>
        </td>
      </tr>
     </table>

   </td>
    </tr>


    </xsl:template>

<!-- ================================================ -->
<!--
<!ELEMENT TDescriptionCourte (TitreDescriptionCourte, PragDC*)>
<!ELEMENT TitreDescriptionCourte (#PCDATA)>
<!ELEMENT PragDC (#PCDATA )>
-->
<!-- ================================================ -->

    <xsl:template match="TDescriptionCourte/PragDC">
      <tr>
        <td>
          <div><br><img src="ligne-titre-lg.gif" width="300" height="10" alt="enoncé" name="ligne_tit"/></br></div>
          <div align="justify"><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b>
              <p><xsl:value-of select="."/></p>
          <div><img src="ligne-titre-lg.gif" width="300" height="10" alt=" " name="ligne_tit"/></div>
        </b></font></div>

        </td>
    </tr>
    </xsl:template>

<xsl:template match="TGroupeConcerne/TitreGroupeConcerne">
<tr>
        <td>

<div align="justify"><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b>

        <p> <br><xsl:value-of select="."/></br></p>
        </b></font></div>
</td>

    </tr>

    </xsl:template>

    <xsl:template match="TGroupeConcerne/GroupeConcerne">
<tr>
        <td>
```

```
<div align="justify"><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b>

        <p> <br><xsl:value-of select="."/></br></p>
        </b></font></div>
</td>

    </tr>

    </xsl:template>

<xsl:template match="TGroupeConcerne/ParaGroupeConcerne">
<tr>
        <td>

<div align="justify"><font face="Verdana, Arial, Helvetica, sans-serif" size="-1">
        <p> <xsl:value-of select="."/></p>
    </font></div>
</td>

    </tr>

    <tr>
   <td>

 <table border="0" width="80%" align="center" cellspacing="10" cellpadding="10">

              <p><font face="Verdana, Arial, Helvetica, sans-serif" size="-1" ><b><br>En
               savoir plus...</br></b></font></p>
              <div align="left"><a href="#detaille" onMouseOver="" onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout1" border="0" alt=" "/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#detaille" onMouseOver=""
onMouseOut="">Description
               détaillée </a></b></font></div>
              <div align="left"><a href="#ergonomie" onMouseOver="" onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout2" border="0" alt=" "/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#ergonomie" onMouseOver=""
onMouseOut="">Rapport avec l'ergonomie </a></b></font></div>
              <div align="left"><a href="#illustration" onMouseOver="" onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout3" border="0" alt=" "/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#illustration"
onMouseOver="">Illustration
               </a></b></font></div>
              <div align="left"><a href="#processus" onMouseOver="" onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout4" alt=" " border="0"/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#processus" onMouseOver=""
onMouseOut="">Place
               dans le processus de développement </a></b></font></div>
              <div align="left"><a href="#implementation" onMouseOver="" onMouseOut=""><img src="bout-orange-
off.gif" width="14" height="17" align="top" name="bout5" alt=" " border="0"/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#implementation"
onMouseOver="" onMouseOut="">Implémentation
               optimale </a></b></font> </div>
              <div align="left"><a href="#complement" onMouseOver="" onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout6" border="0" alt=" "/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#complement" onMouseOver=""
onMouseOut="">Informations
               complémentaires </a></b></font> </div>
               <div align="left"><a href="#exemples" onMouseOver="" onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout6" border="0" alt=" "/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#exemples" onMouseOver=""
onMouseOut="">Exemples  </a></b></font> </div>

              <div align="left"><a href="#associe"  onMouseOver=" " onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout8" border="0" alt=" "/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#associe" onMouseOver=" "
onMouseOut=" ">Recommandations
               associées </a></b></font></div>
              <div align="left"><a href="#motscles"  onMouseOver=" " onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout9" border="0" alt=" "/></a>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#motscles" onMouseOver=" "
onMouseOut=" ">Mots  clés </a></b></font></div>
```

```
            <div align="left"><a href="#référencess"  onMouseOver="" onMouseOut=""><img src="bout-orange-off.gif"
width="14" height="17" align="top" name="bout9" border="0" alt=" "/></a>
              <font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><b><a href="#références" onMouseOver=" "
onMouseOut=" ">Références
    </a></b></font></div>

        </table>
        </td>
         </tr>
</xsl:template>

<!-- ================================================ -->
<!--
<!ELEMENT TDescriptionCourte (TitreDescriptionCourte, PragDC*)>
<!ELEMENT TitreDescriptionCourte (#PCDATA)>
<!ELEMENT PragDC (#PCDATA )>
-->
<!-- ================================================ -->


   <xsl:template match="TDescriptionCourte/TitreDescriptionCourte">
       <p>
           <b>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
               <!-- <xsl:value-of select="."/>-->
               </font>
           </b>
       </p>
   </xsl:template>

<!-- ================================================ -->
<!--
<!ELEMENT TPriorite (TitrePriorite, ValeurPriorite)>
<!ELEMENT TitrePriorite (#PCDATA)>
<!ELEMENT ValeurPriorite (#PCDATA)>
-->
<!-- ================================================ -->

       <xsl:template match="TPriorite">
       <p align="JUSTIFY">
           <b>
               <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                   <!--<xsl:value-of select="TitrePriorite"/> -->
               </font>
           </b>
       </p>
       <p align="JUSTIFY">
           <!--<xsl:value-of select="ValeurPriorite"/>-->
       </p>
   </xsl:template>



   <xsl:template match="TDescriptionDetaillee">
   <tr>
           <td>
           <p><a name="detaille"></a><font face="Verdana, Arial, Helvetica, sans-serif" class="coulorange" size="-
1"><b>                <br><xsl:value-of select="TitreDescriptionDetaillee"/></br>
           </b>
           </font>
       </p>

       <xsl:for-each select="./ParaDescriptionDetaillee">
           <p align="JUSTIFY">
                   <font face="Verdana, Arial, Helvetica, sans-serif" size="-1">
                   <xsl:value-of select="."/>
               </font>
           </p>
       </xsl:for-each>
```

```
        </td>
    </tr>
<!-- lien retour début-->
<div width="100%" bgcolor="orange" border="0" cellspacing="0" cellpadding="0">
        <tr valign="top">
          <td height="16">
           <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top"
onMouseOver=" " onMouseOut=""><br>retour
            début </br></a></font><a href="#top" onMouseOver="" onMouseOut=""><img src="fleche-orange.gif"
width="16" height="16" align="absbottom" border="0"/></a></div>
          </td>
        </tr>
       </div>

    </xsl:template>

<xsl:template match="TRapportAvecErgonomie">
        <tr>
        <td>
        <p><a name="ergonomie"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1" ><b>
    <xsl:value-of select="TitreRapportAvecErgonomie"/>
            </b>
                </font>

        </p>
        <xsl:for-each select="./ParaRapportAvecErgonomie">
            <p align="JUSTIFY">
                <font face="Verdana, Arial, Helvetica, sans-serif" size="-1">
                    <xsl:value-of select="."/>
                </font>
            </p>
        </xsl:for-each>
        <div width="100%" border="0" cellspacing="0" cellpadding="0">
          <tr valign="top">
          <td height="16">
           <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
            début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
border="0"/></a></div>
          </td>
        </tr>
       </div>

        </td>
        </tr>
    </xsl:template>


<xsl:template match="TIllustration">
    <tr>
        <td>
        <p align="JUSTIFY">
            <b>
                <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                    <xsl:value-of select="TitreIllustration"/>
                </font>
            </b>
        </p>
        <xsl:for-each select="./ParaIllustration">
            <p align="JUSTIFY">
                <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                    <xsl:value-of select="."/>
                </font>
            </p>
        </xsl:for-each>
        <div width="100%" border="0" cellspacing="0" cellpadding="0">
          <tr valign="top">
          <td height="16">
           <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
            début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
border="0"/></a></div>
            </td>
```

```
        </tr>
       </div>
      </td>
      </tr>
    </xsl:template>



<xsl:template match="TPlaceDansProcessusDDeveloppement">
      <tr>
      <td>
    <p><a name="processus"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1" class="coulorange"><b>
                        <xsl:value-of select="TitrePlaceDansProcessusDDeveloppement"/></b>
              </font>
      </p>
      <xsl:for-each select="./ParaPlaceDansProcessusDDeveloppement">
          <p align="JUSTIFY">
              <font face="Arial" size="2">
                  <xsl:value-of select="."/>
              </font>
          </p>
      </xsl:for-each>
      <div width="100%" border="0" cellspacing="0" cellpadding="0">
        <tr valign="top">
          <td height="16">
           <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
            début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
border="0"/></a></div>
          </td>
        </tr>
       </div>
      </td>
      </tr>

    </xsl:template>

<!-- ================================================ -->
<!--
<!ELEMENT TImplementationOptimale (TitreImplementationOptimale, ParaImplementationOptimale*)>
<!ELEMENT TitreImplementationOptimale (#PCDATA)>
<!ELEMENT ParaImplementationOptimale (#PCDATA)>
-->
<!-- ================================================ -->

    <xsl:template match="TImplementationOptimale">
      <tr>
      <td>
      <p><a name="implementation"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"
class="coulorange"><b>
                        <xsl:value-of select="TitreImplementationOptimale"/></b>
              </font>
      </p>
      <xsl:for-each select="./ParaImplementationOptimale">
          <p align="JUSTIFY">
              <font face="Verdana, Arial, Helvetica, sans-serif" size="-1">
                  <xsl:value-of select="."/>
              </font>
          </p>
      </xsl:for-each>
      <div width="100%" border="0" cellspacing="0" cellpadding="0">
        <tr valign="top">
          <td height="16">
           <div align="right">><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
            début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
border="0"/></a></div>
          </td>
        </tr>
       </div>
      </td>
      </tr>
```

```
    </xsl:template>


<!-- ================================================ -->
<!--
<!ELEMENT TInformationComplementaire (TitreInformationComplementaire, ParaInformationComplementaire*)>
<!ELEMENT TitreInformationComplementaire (#PCDATA)>
<!ELEMENT ParaInformationComplementaire (#PCDATA)>
-->
<!-- ================================================ -->
    <xsl:template match="TInformationComplementaire">
        <tr>
        <td>
        <p><a name="complement"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"
class="coulorange">            <b>
                        <xsl:value-of select="TitreInformationComplementaire"/>
                        </b>
                </font>

        </p>
        <xsl:for-each select="./ParaInformationComplementaire">
            <p align="JUSTIFY">
                <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                    <xsl:value-of select="."/>
                </font>
            </p>
        </xsl:for-each>
        <div width="100%" border="0" cellspacing="0" cellpadding="0">
          <tr valign="top">
           <td height="16">
            <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
             début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
border="0"/></a></div>
           </td>
          </tr>
         </div>
        </td>
        </tr>

    </xsl:template>


<!-- ================================================ -->
<!--
<!ELEMENT TRecommandationsAssociees (TitreRecommandationsAssociees, ParaRecommandationsAssociees*)>
<!ELEMENT TitreRecommandationsAssociees (#PCDATA)>
<!ELEMENT ParaRecommandationsAssociees (#PCDATA)>
-->
<!-- ================================================ -->
    <xsl:template match="TRecommandationsAssociees">
        <tr>
        <td>
        <p><a name="associe"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"
class="coulorange"><b>
                        <xsl:value-of select="TitreRecommandationsAssociees"/>
                        </b>
                </font>

        </p>
        <xsl:for-each select="./ParaRecommandationsAssociees">
            <p align="JUSTIFY">
                <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                    <a href="mwa33.html">
                        <xsl:value-of select="."/>
                    </a>
                </font>
            </p>
        </xsl:for-each>
        <div width="100%" border="0" cellspacing="0" cellpadding="0">
          <tr valign="top">
```

```
                <td height="16">
                 <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
                  début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
 border="0"/></a></div>
                </td>
               </tr>
              </div>
             </td>
             </tr>

        </xsl:template>

    <!-- ================================================= -->
    <!--
    <!ELEMENT TMotsCles (TitreMotsCles, ParaMotsCles*)>
    <!ELEMENT TitreMotsCles (#PCDATA)>
    <!ELEMENT ParaMotsCles (#PCDATA)>
    -->
    <!-- ================================================= -->
        <xsl:template match="TMotsCles">

            <tr>
            <td>
            <p><a name="motscles"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"
 class="coulorange"><b>
                        <xsl:value-of select="TitreMotsCles"/>
                        </b>
                    </font>
            </p>
            <xsl:for-each select="./ParaMotsCles">
                <p align="JUSTIFY">
                    <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                        <xsl:value-of select="."/>
                    </font>
                </p>
            </xsl:for-each>
            <div width="100%" border="0" cellspacing="0" cellpadding="0">
              <tr valign="top">
               <td height="16">
                <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
                 début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
 border="0"/></a></div>
               </td>
              </tr>
             </div>
            </td>
            </tr>

        </xsl:template>

    <!-- ================================================= -->
    <!--
    <!ELEMENT TReferences (TitreReferences, ParaReferences*)>
    <!ELEMENT TitreReferences (#PCDATA)>
    <!ELEMENT ParaReferences (#PCDATA)>
    -->
    <!-- ================================================= -->
        <xsl:template match="TReferences">

            <tr>
            <td>
            <p><a name="références"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"
 class="coulorange"><b>
                        <xsl:value-of select="TitreReferences"/>
                        </b>
                    </font>
            </p>
            <xsl:for-each select="./ParaReferences">
                <p align="JUSTIFY">
                    <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                        <xsl:value-of select="."/>
```

```
                </font>
            </p>
        </xsl:for-each>
        <div width="100%" border="0" cellspacing="0" cellpadding="0">
          <tr valign="top">
           <td height="16">
            <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
             début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
border="0"/></a></div>
           </td>
          </tr>
         </div>
        </td>
        </tr>

    </xsl:template>

<xsl:template match="TExemples">

        <tr>
        <td>
        <p><a name="exemples"></a><font face="Verdana, Arial, Helvetica, sans-serif" size="-1"
class="coulorange"><b>
                        <xsl:value-of select="TitreExemples"/>
                        </b>
                </font>
        </p>
        <xsl:for-each select="./ParaExemples">
            <p align="JUSTIFY">
                <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
                    <xsl:value-of select="."/>
                </font>
            </p>
        </xsl:for-each>
        <div width="100%" border="0" cellspacing="0" cellpadding="0">
          <tr valign="top">
           <td height="16">
            <div align="right"><font face="Verdana, Arial, Helvetica, sans-serif" size="-2"><a href="#top" >retour
             début </a></font><a href="#top" ><img src="fleche-orange.gif" width="16" height="16" align="absbottom"
border="0"/></a></div>
           </td>
          </tr>
         </div>
        </td>
        </tr>
</xsl:template>


    </xsl:stylesheet>
```