# Documents structurés
## XML, SGML, HTML

***Yves MARCOUX***
***<GRDS> - EBSI***
***Université de Montréal***

Cours donné à l'Université de Toulon et du Var
Automne 1999

---

Dans ce cours, nous parlerons de documents structurés; des principes généraux de cette approche, mais aussi plus spécifiquement de XML, SGML et HTML, qui sont trois formats normalisés de documents structurés, ainsi que de certaines autres normes associées à ces trois normes de base.

Il existe maintenant une telle quantité de ressources accessibles, en particulier des introductions, portant sur XML et sur les documents structurés, que notre but sera ici de faire une présentation originale de sujets choisis, et non de reproduire (ou traduire) bêtement ce qu'on peut trouver facilement ailleurs.

### Les références incontournables

**LA** référence par excellence pour tout ce qui touche XML, SGML et les documents structurés en général est un site maintenu par Robin Cover: <http://www.oasis-open.org/cover/>. Une section de ce site est consacrée aux introductions à XML; il s'agit de <http://www.oasis-open.org/cover/xmlIntro.html>. On en retrouve une bonne vingtaine! Une autre section est consacrée aux livres sur XML; il s'agit de <http://www.oasis-open.org/cover/xml.html#xmlBooks>.

La norme XML elle-même se retrouve à <http://www.w3.org/TR/1998/REC-xml-19980210.html>. Une version annotée de commentaires personnels d'un des éditeurs de la norme, Tim Bray, se retrouve à <http://www.xml.com/axml/axml.html>.

### Autres références

Une version française est proposée à <http://babel.alis.com/web_ml/xml/REC-xml.fr.html>. Mentionnons encore la "foire aux questions" (FAQ) XML, <http://www.ucc.ie/xml/>, et un de nos propres articles, une introduction très générale aux concepts de base des documents structurés, <http://mapageweb.umontreal.ca/marcoux/grds/ico94.htm>.

Les deux ouvrages suivants nous semblent dignes de mention; d'ailleurs, certaines parties de ce cours en sont inspirées:

MICHARD, Alain. *XML: Langage et applications.* Eyrolles, 1998.

MALER, Eve; EL ANDALOUSSI, Jeanne. *Developing SGML DTDs: From Text to Model to Markup.* Prentice Hall PTR, 1996.

## Plan du cours

1. Introduction; les deux principes fondamentaux des documents structurés; syntaxe et sémantique de base des documents structurés; éléments de syntaxe XML.
2. Applications et processeurs XML validants et non validants; Document Type Declaration; Document Type Definition (DTD).
3. Second principe des documents structurés; exemple et contrexemple; documents orientés-données et orientés-information; types d'outils XML; chaîne de traitement type; déroulement d'un projet d'implantation XML; réutilisation de l'information, séparation contenu-traitement, normalisation.
4. Exemples et démonstrations: analyse de besoins, modélisation, rédaction de DTD, saisie, traitement, diffusion.

## Introduction

- SGML = Standard Generalized Markup Language = ISO 8879 (1986)
- XML = eXtensible Markup Language = REC-xml-19980210 du W3C
- HTML 4.0 = HyperText Markup Language = REC-html40-19980424 du W3C
- HTML 4.01: proposed recommendation du W3C
- XHTML 1.0: proposed recommendation du W3C (29 août 1999)
- ISO HTML (en préparation): version ISO de HTML 4.0

ISO n'est pas un acronyme, ce nom est un préfixe grec qui signifie "égal". ISO, fondée en 1947, est la plus importante instance de normalisation au monde et ses normes couvrent pratiquement tous les domaines de l'activité industrielle. Au sein d'ISO, le plus haut statut de normalisation est celui de norme internationale, statut attesté par l'attribution d'un numéro de norme ISO (par exemple, ISO 8879). ISO vend ses normes, qui ont la réputation (justifiée!) d'être chères.

Le W3C (World Wide Web Consortium) a été fondé en octobre 1994 pour élaborer des normes et émettre des directives visant la meilleure utilisation possible du Web dans la société. Ses principales créations incluent les dernières versions de HTML, les Cascading Style Sheets (CSS), la Platform for Privacy Preferences (P3P) et XML. Au niveau du W3C, le plus haut statut de normalisation est la "recommendation". Ce statut est précédé de celui de "proposed recommendation", lui-même précédé du statut de "working draft". Le W3C rend disponibles ses normes gratuitement sur le Web.

## Les deux principes fondamentaux des documents structurés

L'approche des documents structurés est basée sur deux principes:

1) Un document numérique est un fichier texte (en format texte, e.g., ASCII), auquel on superpose des conventions additionnelles qui permettent de représenter une structure hiérarchique (en arbre).

2) La structure hiérarchique du document numérique doit correspondre le mieux et le plus explicitement possible à la nature et à la structure de "l'information" qui doit être véhiculée par le document.

On appelle "document structuré" tout document numérique (donc, concrètement, un fichier, ou une séquence d'octets stockable sur un support quelconque) créé dans le respect de ces deux principes.

## Généralités syntaxiques sur les documents structurés

Le premier principe est facile à définir précisément et à mettre en oeuvre. On note qu'il se subdivise en deux: un jeu de caractères et des conventions additionnelles pour représenter une structure hiérarchique.

### Jeux de caractères

Dans une approche de documents structurés, on adopte tout d'abord un jeu de caractères sous-jacent; celui du fichier texte qui "soutient" le document structuré. Comme d'habitude, un jeu de caractères est constitué d'un **répertoire** (de caractères), d'une **numérotation** (de caractères) et d'un **encodage** (de caractères).

La solution la plus courante est l'une ou l'autre extension de l'ASCII (American Standard Code for Information Interchange), dont la plus populaire est l'ISO-8859-1 (aussi appelé "ISO Latin-1"). Cette norme fait partie d'une série de 15 normes ISO, chacune visant un bassin linguistique spécifique. L'Unicode (ISO-10646-1) propose maintenant une solution vraiment multilingue; nous y reviendrons plus loin dans le contexte précis de XML. L'important à se rappeler est qu'avant l'Unicode, le problème des jeux de caractères était très aigü et a exigé le développement de solutions le plus souvent incompatibles deux à deux.

Malgré tous les problèmes historiques, politiques et culturels, il demeure qu'il est techniquement facile d'établir un jeu de caractères, qu'il soit normalisé ou non.

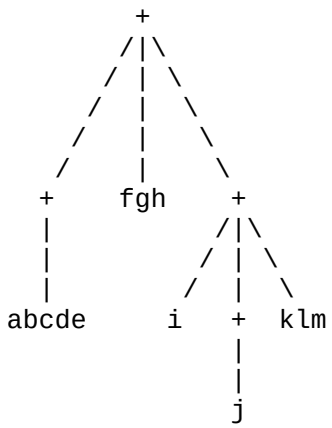
Référence sur les jeux de caractères: <http://www.hut.fi/u/jkorpela/chars.html>. Référence spécifiquement sur les ISO 8859: <http://www.isoc.org:8080/codage/iso8859/jeuxiso.fr.htm>.

### Structure hiérarchique

Pour imposer une structure hiérarchique par dessus un format texte, plusieurs solutions sont possibles. Quelle serait la plus simple? Probablement un langage de parenthésage, analogue aux langages d'expressions utilisés pratiquement partout en informatique. Ainsi, on pourrait "réserver" certains caractères du jeu de caractères sous-jacent, par exemple "(" et ")", et s'en servir pour délimiter des segments de texte et les regrouper hiérarchiquement. Alors, le document numérique suivante:

"((abcde)fgh(i(j)klm))"

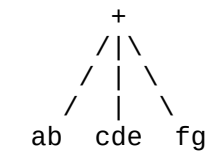
représenterait en fait la structure suivante:



Tout document numérique (lire: chaîne de caractères) ne représenterait bien sûr pas une telle structure. Par exemple "((a)" ou ")ab(d)" ne peuvent être interprétées comme une structure hiérarchique selon nos conventions. On a donc intérêt à définir une classe de documents numériques "bien formés", i.e., qui ont un sens en vertu de notre convention de représentation des structures hiérarchiques. Il s'agit d'un premier type de restriction syntaxique du format qu'on est en train de superposer au format texte.

Comment se définirait un document numérique "bien formé" (*well-formed*), avec ces conventions? Par la conjonction des conditions suivantes (ce n'est qu'une possibilité, mais c'est une des plus simples):
a) Contient le même nombre d'occurrences de "(" que de ")".
c) Tout préfixe propre non-vide du document contient plus d'occurrences de "(" que de ")".

Question: Comment représenter?




Réponse (la question était piège, car on n'avait pas précisé cette convention): Soit ce n'est pas possible, parce qu'on veut que "()" représente un sous-arbre unitaire (racine seulement), soit "()" représente un sous-arbre vide, et alors on peut le représenter par: (ab()cde()fg); mais alors, on ne peut pas représenter de sous-arbre unitaire. Morale de cette question: même pour les questions très anodines, il faut faire attention aux détails avec les documents structurés.

Déjà, on voit poindre le problème à l'horizon: comme on a réservé deux caractères du jeu sous-jacent, comment peut-on les représenter si jamais ils doivent survenir dans le document lui-même (et non en exprimer la structure)? C'est le problème universel de "l'échappement" des caractères spéciaux. Les documents structurés n'y échappent pas (sans jeu de mots)! Nous y reviendrons.

Une approche comme ci-dessus permet bel et bien de structurer un document textuel hiérarchiquement, mais elle permet difficilement de rendre explicite la *nature* de l'information véhiculée par le document. Ainsi, si une telle structure devait représenter un chèque ou une facture, il faudrait "savoir à l'avance" où, dans la structure, chaque élément d'information (numéro de produit, montant, etc.) se retrouve; rien de cela ne serait explicite. Bien sûr, on pourrait ajouter des conventions pour pallier cette lacune, mais rien dans notre formalisme n'encourage particulièrement à le faire.

Quoi qu'il en soit, l'approche la plus courante en documents structurés est légèrement différente. Elle est destinée à "encourager" l'explicitation de la nature de l'information dans les documents numériques. C'est la technique du "balisage", qui est commune à SGML, XML et HTML, et que nous décrivons maintenant.

En comparaison avec l'approche du parenthésage, le balisage permet un nombre illimité de types de parenthèses différents. Et chaque type de parenthèse doit être utilisé pour un type bien précis d'information. Avant d'arriver au balisage proprement dit, illustrons cette idée encore avec un langage de parenthésage. Supposons qu'on veuille représenter une fiche rudimentaire de carnet d'adresses contenant nom, adresse et numéro de téléphone. Nous utilisons quatre types de

parenthèses, mettons ( ), [ ], { } et < >. Puis, on établit la convention selon laquelle chaque type de parenthèses délimite une partie du document qui correspond à l'un des quatre types d'information suivants: nom, adresse, numéro de téléphone et fiche au complet. Par exemple, on pourrait établir la correspondance suivante:

```
( ):  nom
[ ]:  adresse
< >:  numéro de téléphone
{ }:  fiche au complet
```

Ainsi, une fiche serait représentée comme suit:

```
{(Dubois, Céline)[1 rue du Fort, Québec, QC]<+1.514.343.7750>}
```

ou, plus lisiblement:

```
{
 (Dubois, Céline)
 [1 rue du Fort, Québec, QC]
 <+1.514.343.7750>
}
```

Remarquons tout de suite que ces conventions nous offrent beaucoup de souplesse en elles-mêmes. Par exemple, il ne peut y avoir d'ambiguïté entre adresse et numéro de téléphone, dans le cas où on n'a qu'un des deux:

```
{
 (Dubois, Céline)
 <+1.514.343.7750>
}
```

Ou encore, si on a deux adresses:

```
{
 (Dubois, Céline)
 [1 rue du Fort, Québec, QC]
 [2 Place Ville-Marie, suite 305, Québec, QC]
 <+1.514.343.7750>
}
```

Essentiellement, c'est exactement l'approche du balisage, à la différence que, dans le balisage, les types de parenthèses sont créés à volonté, grâce aux conventions suivantes:

- N'importe quel identificateur, flanqué des caractères "<" et ">", peut servir de "parenthèse" ouvrante.
- Le même identificateur, flanqué à gauche des caractères "</" et à droite de ">", sert de "parenthèse" fermante correspondante.

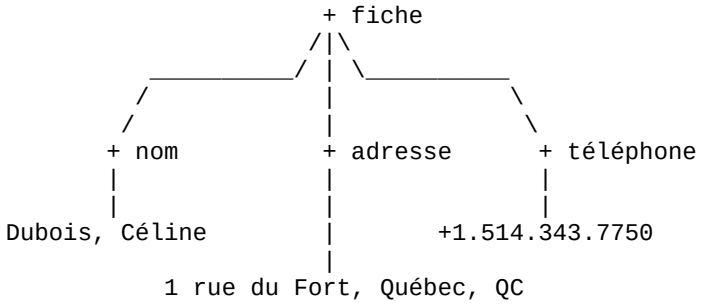
Comme il n'est plus juste de parler de parenthèses, on parle alors de *balises*. Une balise est toute la chaîne de caractères allant du "<" au ">", incluant ces caractères. Un identificateur qui figure dans une ou plusieurs balises est appelé un *identificateur générique*. Une balise qui commence par "</" est appelée balise *fermante* et une balise qui ne commence que par "<" est appelée *ouvrante*. On distingue parfois (mais pas souvent) entre la balise elle-même (la chaîne de caractères dans l'abstrait) et une occurrence de cette balise dans un document numérique.

Ainsi, pour représenter nos fiches de carnet d'adresses en utilisant le balisage, nous pourrions choisir d'utiliser les identificateurs génériques `fiche`, `nom`, `adresse` et `téléphone` et les balises correspondantes. Nos fiches auraient alors l'allure suivante:

```
<fiche>
 <nom>Dubois, Céline</nom>
 <adresse>1 rue du Fort, Québec, QC</adresse>
 <téléphone>+1.514.343.7750</téléphone>
</fiche>
```

En fait, ce document (en supposant un jeu de caractères approprié) est un document XML tout à fait légal (nous ne disons pas "valide", car ce mot a une signification technique très précise en XML).

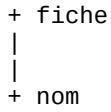
La façon classique de représenter graphiquement un tel document est d'associer les identificateurs génériques aux noeuds internes de l'arbre, qui est l'endroit le plus naturel pour indiquer le type de "parenthèses" qui donnent lieu à un sous-arbre. Ainsi, le dernier exemple ci-dessus est représenté comme suit:




Par rapport à la question de représentation des sous-arbres unitaires ou vides, tous les langages basés sur le balisage (donc, entre autres, XML, SGML et HTML) interprètent une balise ouvrante suivie immédiatement d'une balise fermante correspondante comme un arbre unitaire (et non vide). Ainsi, dans notre exemple, il serait possible d'avoir la fiche suivante:

```
<fiche>
 <nom></nom>
</fiche>
```

et elle correspondrait à l'arbre suivant:




Comment se définit maintenant le concept de document numérique "bien-formé"? Il y a encore une fois plusieurs définitions possibles, mais une des plus simples est comme suit. On utilise une définition auxiliaire:

Un document est dit "quasi-bien-formé" si:
a) Le document contient le même nombre d'occurrences de "<" que de ">"; et

b) Tout préfixe du document contient 0 ou 1 occurrences de plus de "<" que de ">"; et
c) Tout préfixe propre du document se terminant par ">" contient plus du double d'occurrences de "<" que de "</".

Ensuite, on définit:

Un document est "bien-formé" ssi:
a) Il est quasi-bien-formé; et
b) Pour tout sous-document contigu non-vide quasi-bien-formé, il existe un identificateur *w* tel que ("<" *w* ">") est préfixe du sous-document et ("</" *w* ">") en est suffixe.

NOTE: pour déterminer le "quasi-bien-formé" d'un document, il suffit de balayer le document en maintenant deux compteurs. Pour le "bien-formé", il faut une pile où les identificateurs génériques sont empilés lorsqu'on rencontre un balise ouvrante et dépilés lorsqu'on rencontre une balise fermante. (En fait, le travail d'un automate à pile *déterministe*.)

À quelques détails près (notamment, la présence de spécifications d'attribut dans les balises ouvrantes, sur laquelle nous reviendrons), cette définition du "bien-formé" est celle que l'on retrouve dans XML.

Donc, en résumé, dans tout langage de balisage (ce qui inclut SGML, XML et, théoriquement, HTML), on ne s'intéresse qu'aux documents bien-formés, car c'est seulement à eux qu'on peut associer une structure hiérachique.

Pour ce qui est de la partie syntaxique et structurelle des documents structurés (premier principe), les généralités sont terminées, et il est maintenant temps de passer à un format spécifique. Allons-y avec XML, puisqu'il est le "vaisseau amiral" des formats de documents structurés.

## Spécificités syntaxiques de XML

### Unicode comme jeu de caractères

Le répertoire et la numérotation des caractères de tout document XML sont obligatoirement ceux de la norme Unicode (entièrement synchronisée avec la norme ISO-10646-1). Ceci est fixé par la norme XML. L'encodage, cependant, peut varier. Un seul encodage est obligatoire (i.e., doit être reconnu par toute application XML), c'est l'UTF 8. Les autres encodages, par exemple l'ISO-8859-1 (vu comme encodage d'un sous-ensemble d'Unicode), sont facultatifs, en ce sens qu'une application peut refuser de traiter un document qui n'est pas encodé en UTF-8.

Les différentes extensions de l'ASCII ont en général des numérotations à 256 positions, ce qui en limite l'utilité comme jeux multilingues. Unicode est basé sur une numérotation à 65536 positions, ce qui permet d'inclure les caractères de la plupart des langues vivantes non seulement européennes, mais également asiatiques et autres.

L'encodage UTF-8 est intéressant, car il attribue comme code aux 128 premiers caractères de la numérotation Unicode, le code sur un octet correspondant au numéro du caractère. Comme les 128 premiers numéros d'Unicode coïncident avec l'ASCII pur sur 8 bits (ISO-646, aussi appelé "US-ASCII"), cela a pour effet de rentre compatible à UTF-8 tout fichier US-ASCII existant, sans aucune modification.

Voici quelques données sur Unicode, provenant du site du consortium Unicode <http://www.unicode.org>, site à consulter pour plus d'information sur cette norme, dont la troisième version sera publiée sous peu:

| Catégorie | V 2.1 | V 3.0 |
|---|---|---|
| Alphabetics, Symbols | 6511 | 10236 |
| CJK Ideographs | 21204 | 27786 |
| Hangul Syllables | 11172 | 11172 |
| Total assigned characters | 38887 | 49194 |
| Private Use | 6400 | 6400 |
| Surrogates | 2048 | 2048 |
| Controls | 65 | 65 |
| Not Characters | 2 | 2 |
| Total assigned 16-bit code values | 47402 | 57709 |
| Unassigned 16-bit code values | 18134 | 7827 |

Pour plus d'information sur UTF-8, voir <ftp://ftp.isi.edu/in-notes/rfc2279.txt>.

### Commentaires

Les commentaires sont des bouts du document qui ne sont simplement PAS interprétés pour établir la structure hiérarchique correspondant au document.

```
<!-- Ceci est un exemple de commentaire -->
```

Par simplicité, les commentaires ne peuvent JAMAIS contenir deux tirets de suite! (Même s'il suffirait en fait d'interdire la chaîne "`-->`").

### Échappement de base: références d'entités prédéfinies et sections CDATA

Les "références d'entités prédéfinies": pour le problème d'échappement des caractères spéciaux, évoqués plus tôt: `<` `&` (d'autres s'ajouteront plus tard). Convient pour des segments contenant quelques occurrences seulement des caractères spéciaux.

Une technique plus puissante: les sections "CDATA":

```
<![CDATA[
Ce bout peut contenir n'importe quel texte.
Ainsi, je peux donner des exemples de XML:
<livre>, </livre>, <nom>, etc.  Une section CDATA
se termine toujours par...
]]>
```

Par simplicité, la chaîne "`]]>`" n'est *jamais* permise dans le texte du document, sauf pour terminer une section CDATA. Donc, nous avons aussi besoin de l'entité `>`, si on veut mettre la chaîne "`]]>`" comme texte dans notre document. On écrirait alors "`]]>`".

ATTENTION: Le `>` ne peut PAS être utilisé à l'intérieur d'une section CDATA pour y inclure la chaîne "`]]>`" ! Pour inclure cette chaîne dans le contenu textuel d'un document, il faut avoir recours à autre chose qu'une simple section CDATA. On pourrait par exemple utiliser DEUX sections CDATA consécutives (mais il y a d'autres solutions):

```
<noteDeLauteur>
Voici le début de ma note...
<![CDATA[qui contient deux sections CDATA permettant d'inclure
la chaîne défendue: ]]]]><![CDATA[>.  Et voilà, c'est fait!]]>
Et je poursuis et termine ma note!
</noteDeLauteur>
```

## Les références d'entités caractères

Exemples: `&#329; &#x01b2;`

Permettent de référer à n'importe quel caractère Unicode dans un document, même si l'encodage utilisé ne le permet pas (par exemple, ISO-8859-1). La forme `&#nnnnn;` réfère au caractère dont le numéro Unicode est donné par `nnnnn` en décimal, alors que dans la forme `&#xnnnn;` le numéro du caractère est donné en hexadécimal.

## Les références d'entités générales

On peut aussi définir des entités dites "générales" qui peuvent entre autres être utilisées comme abréviations ou comme "variables" pour une terminologie non encore stabilisée ou sujette à changement. Nous verrons plus tard comment on définit de telles entités générales, mais disons tout de suite qu'une entité générale est identifiée par un nom, qu'elle possède un "texte de remplacement" et qu'on y réfère avec la syntaxe:

```
&nom;
```

Ainsi, si l'entité générale nommée `ebsi` possède comme texte de remplacement la chaîne "`École de bibliothéconomie et des sciences de l'information de l'Université de Montréal`", alors le passage:

```
Les études à l'&ebsi; sont régies par le règlement pédagogique de l'Université.
```

est exactement équivalent à:

```
Les études à l'École de bibliothéconomie et des sciences de l'information de
l'Université de Montréal sont régies par le règlement pédagogique de
l'Université.
```

## Digression terminologique

Le vocable "balisage" désigne la technique générale d'utilisation de balises pour représenter une structure hiérarchique dans un document texte, mais c'est aussi le nom donné à l'ensemble des occurrences de balises et d'autres artifices d'échappement (incluant les commentaires) qui se retrouvent dans un document numérique. On distingue ainsi le balisage du document de son contenu textuel (qui serait la concaténation de tout ce qui n'est pas du balisage, *après* remplacement des références d'entités et élimination des commentaires).

Le mot "élément" désigne aussi bien un noeud dans l'arbre représenté par le document numérique, que la partie (nécessairement bien formée!) du document qui correspond à un tel noeud (incluant les balises ouvrante et fermante).

Attention: un élément n'est pas une balise, ni une occurrence de balise, ni un identificateur générique!

Comme nous verrons, il peut y avoir des choses avant la première balise ouvrante dans un document numérique. Tout ce qu'il y a AVANT la première balise ouvrante s'appelle le *prologue* du document. Tout ce qui est compris depuis la première balise ouvrante jusqu'à sa balise fermante correspondante (obligatoirement la dernière du document numérique), inclusivement, est appelé *l'instance de document*.

## Attributs

Pour une raison historique (principalement de compatibilité avec SGML), XML offre la possibilité d'inclure dans une balise ouvrante une *liste de spécifications d'attribut*. Une liste de spécifications d'attribut est en fait la linéarisation (i.e., l'expression sous forme d'une chaîne de caractères) d'un ensemble de paires "nom-valeur", chaque paire ayant la forme:

*_nom* ="*valeur*"

ou bien

*_nom* ='*valeur*'

(simples ou doubles guillemets), où le signe "_" représente un espace obligatoire (pour séparer une paire de la précédente ou de l'identificateur générique).

Un exemple de liste de spécifications d'attribut:

```
 type="interne" niveau='2' sécurité="secret"
```

Un exemple de balise ouvrante munie de cette liste de spécification d'attribut:

```
<contact type="interne" niveau='2' sécurité="secret">
```

Un exemple d'élément complet avec une balise ouvrante comportant une liste de spécifications d'attribut:

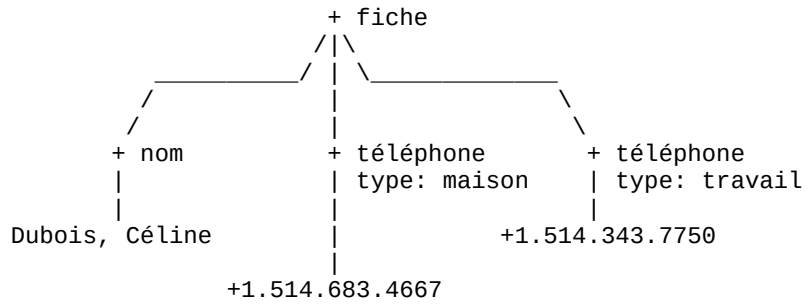
```
<ref type='thèse'>
Roy, Luc.  XML dans la vraie vie.  Thèse de doctorat (en préparation).
Université de Montréal, 2002.
</ref>
```

Conceptuellement, un attribut est une information que l'on désire apposer "à côté" d'un noeud interne de l'arbre du document, plutôt que **sous** ce noeud, à l'instar de ses sous-éléments. En théorie, un attribut devrait préciser la nature de l'information contenue dans l'élément, en complément de son identificateur générique. Il peut s'agir d'une façon d'indiquer une sous-catégorie, par exemple, par rapport à une catégorie générale correspondand à l'identificateur générique. Plus sur cette question lorsque nous parlerons du second principe des documents structurés.

Pour nos fiches, on pourrait par exemple utiliser un attribut pour l'élément téléphone, qui indiquerait s'il s'agit d'un numéro de téléphone au travail ou à la maison. Voici comment pourrait se lire une fiche après l'introduction d'un tel attribut:

```
<fiche>
 <nom>Dubois, Céline</nom>
 <téléphone type='maison'>+1.514.683.4667</téléphone>
 <téléphone type='travail'>+1.514.343.7750</téléphone>
</fiche>
```

Et voici de quoi aurait l'air l'arbre correspondant à ce document:




Notez que, avec l'introduction des attributs, il nous faut introduire aussi deux nouvelles entités prédéfinies pour échapper respectivement le guillement simple et double: `&apos;` et `&quot;`.

### Balise d'élément vide

Il peut arriver que le contenu d'un élément soit vide. Dans ce cas, on peut juxtaposer l'une à côté de l'autre les balises ouvrante et fermante de l'élément:

```
<marqueur></marqueur>
```

Mais XML offre la possibilité d'abréger cette succession de deux balises en une seule, d'un type appelé "balise d'élément vide":

```
<marqueur/>
```

Cet exemple est rigoureusement équivalent à `<marqueur></marqueur>`. La balise d'élément vide évite la répétition d'identificateur générique qu'entraîne l'usage des balises ouvrante et fermante. Notez qu'une balise d'élément vide peut, comme une balise ouvrante, comporter une liste de spécifications d'attribut:

```
<marqueur numéro="4" />
```

### Déclaration XML (facultative)

Permet de rendre le document XML autodescriptif et également de spécifier un encodage différent de l'Unicode UTF-8. Elle doit se trouver en tout début de document. Exemples:

```
<?xml version="1.0"?>
<?xml version="1.0" encoding="ISO-10646-UTF-8"?>
<?xml version="1.0" encoding="ISO-8859-1"?>
```

En l'absence d'une déclaration XML précisant un encodage (encoding), toute application XML est tenue de supposer qu'un document XML utilise l'encodage ISO-10646-UTF-8.

## Déclaration de type de document (facultative)

La déclaration de type de document sert à plusieurs choses. Entre autres, elle permet la définition d'entités générales. Il existe plusieurs formes de déclaration de type de document; nous en verrons deux, dont voici la première:

```
<!DOCTYPE nom [
Déclarations de balisage
]>
```

Le `nom` donné doit obligatoirement coïncider avec l'identificateur générique de l'élément de plus haut niveau de l'instance de document (c'est-à-dire l'identificateur générique de la première balise ouvrante du document).

Ce qu'il y a entre les [ ] s'appelle le "sous-ensemble interne de déclarations". Il contient des "déclarations de balisage", dont nous ne verrons dans un premier temps que les déclarations d'entités générales et les déclarations d'entités paramètres. Notez que si on veut définir des entités de façon à pouvoir les utiliser dans un document, il **faut absolument** que celui-ci comporte une déclaration de type de documents (c'est le seul endroit où on peut définir des entités).

- Déclaration d'entité générale interne:

```
<!ENTITY nom "Texte de remplacement">
```

Exemples archétypaux: abréviation d'un long bout de texte, ou variable pour terminologie encore mouvante.

- Déclaration d'entité générale externe parsée:

```
<!ENTITY nom SYSTEM "Un URL">
```

Comme une entité générale interne, mais le texte de remplacement est dans un fichier (il y a donc indirection). Exemple archétypal: document modularisé en plusieurs fichiers ("sous-documents"), par exemple, une thèse en plusieurs chapitres. (Notez que les balises ouvrante et fermante de l'élément de plus haut niveau *doivent* obligatoirement être dans le document maître, et non dans les entités externes contenant les premier et dernier chapitres, car le texte de remplacement d'une entité doit toujours être bien formé.)

```
<!DOCTYPE thèse [
<!ENTITY chap1 SYSTEM "chapitre1.xml">
<!ENTITY chap2 SYSTEM "chapitre2.xml">
<!ENTITY chap3 SYSTEM "chapitre3.xml">
]>
<thèse>
&chap1;
&chap2;
&chap3;
</thèse>
```

- Définition d'entité paramètre externe:

```
<!ENTITY % nom SYSTEM "Un URL">
```

Comme une entité générale externe, mais peut être référencée dans le sous-ensemble interne de déclarations même. On y réfère avec le `%` plutôt que le `&`, donc: `%nom;` et non `&nom;`. Exemple archétypal: déclarations modularisées de jeux d'entités pour caractères "spéciaux".

Supposons que le fichier `entités.xml` contienne les déclarations suivantes:

```
<!ENTITY Agrave  "&#192;" >
<!ENTITY Aacute  "&#193;" >
<!ENTITY Acirc   "&#194;" >
<!ENTITY Atilde  "&#195;" >
```

Alors, on pourrait utiliser ces définitions à partir d'un document XML de cette façon:

```
<!DOCTYPE mondoc [
<!ENTITY % entit SYSTEM "entités.xml">
%entit;
]>
<mondoc>
&Agrave; partir de maintenant, je peux écrire CH&Acirc;TEAU sans problème.
</mondoc>
```

C'est comme si les définitions d'entités figuraient directement dans le sous-ensemble interne de déclarations. Plusieurs jeux "standard" d'entités (noms normalisés) ont été définis pour différents sous-ensembles d'Unicode. Voir par exemple <http://www.altheim.com/specs/charents.html>.

### Déclaration de texte

Les entités externes (paramètres ou générales parsées) peuvent commencer par une "déclaration de texte" pour préciser un encodage particulier. Une déclaration de texte a la même forme qu'une déclaration XML, sauf que l'information de version est optionnelle. Cette déclaration doit être au tout début du fichier. Exemple:

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

## Notion de "bien-formé" nouvelle mouture

Avec toutes les possibilités syntaxiques offertes par XML, dont nous n'avons vu que quelques-unes, notre définition de "bien-formé" en prend un coup! Ne vous en faites pas, nous ne définirons pas en détail cette nouvelle notion, mais la norme XML, elle, le fait! Mentionnons simplement deux conditions de ce "bien-formé" nouvelle mouture, qui concernent les attributs:

- Dans une liste de spécifications d'attributs, aucun nom d'attribut ne peut apparaître plus d'une fois.
- La valeur d'un attribut (la chaîne entre guillements) ne peut comporter, directement ou indirectement (par référence d'entité générale), le caractère "`<`", autrement que par référence (directe ou indirecte) à l'entité `<`.

Ainsi: `<livre type="roman" type='novel'>` n'est pas une balise ouvrante bien-formée, ni `<livre type="<roman>">`. Par contre, `<livre type="<roman>">` est bien formée.

## Documents valides, DTD

Avant de voir quelques autres types de déclarations de balisage et une autre forme de déclaration de type de document, il nous faut introduire la notion de *document valide*. L'introduction de cette notion est motivée par le fait que, dans le contexte d'une application particulière, on peut vouloir

restreindre notre univers documentaire d'intérêt à un sous-ensemble propre de l'ensemble des documents bien formés. Ainsi, dans notre petit exemple de fiches de carnet d'adresses, on pourrait vouloir ne pas considérer comme intéressantes les fiches qui ont plus d'un nom. Il se pourrait qu'on veuille les considérer comme non significatives, au même titre que les documents malformés.

XML permet de définir de telles restrictions *au-delà du bien-formé* par le truchement d'un ensemble de déclarations de balisage qu'on appelle une DTD (pour *Document Type Definition*). Une DTD est un ensemble de déclarations de balisage comportant: des déclarations d'entités (dont nous avons déjà vu quelques formes), des déclarations de listes d'attributs et des déclarations de type d'élément. Une déclaration de liste d'attributs permet de restreindre quels attributs on peut utiliser avec un type d'élément donné. Une déclaration de type d'élément permet de restreindre ce qu'on peut retrouver comme contenu à l'intérieur d'un type d'élément donné (i.e., entre les balises ouvrante et fermante).

On associe une DTD à un document XML par la déclaration de type de document. Les déclarations qui constituent une DTD peuvent se retrouver entièrement dans le sous-ensemble interne de déclarations du document, ou encore, une partie ou la totalité d'entre elles peut se retrouver dans une entité externe (fichier), que l'on peut spécifier avec une seconde forme de déclaration de type de documents:

```
<!DOCTYPE nom SYSTEM "Un URL" [
Déclarations de balisage
]>
```

Le sous-ensemble interne de déclarations est encore ce qui est compris entre le [ et le ]. S'il est vide (c'est à dire que l'entièreté des déclarations constituant la DTD se retrouve dans l'entité externe identifiée par l'URL), on peut omettre le [ et le ]. Si le sous-ensemble interne de déclarations est non vide, les déclarations qu'il contient sont traitées AVANT les déclarations en provenance de l'entité externe.

Un document bien-formé qui, par surcroît, possède une déclaration de type de document **et** est conforme aux contraintes exprimées par la DTD correspondant à cette déclaration de type de document est dit *valide* par rapport à cette DTD. Un document sans déclaration de type de document ne peut PAS être valide (mais il peut cependant être bien formé). En revanche, un document avec déclaration de type de document peut très bien être bien formé (et il peut additionnellement être valide).

## Déclaration de liste d'attributs

Voici un exemple de déclaration de liste d'attributs:

```
<!ATTLIST genID attr1 CDATA "défaut"
                attr2 CDATA #IMPLIED
                attr3 CDATA #REQUIRED >
```

## Déclaration de type d'élément

Voici un exemple de déclaration de type d'élément:

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

Les contraintes de contenu du type d'éléments sont exprimées via un modèle de contenu, utilisant les constructeurs ",", "|" et les modificateurs d'occurrence "?", "*" et "+".

Il est à noter que le genre de contraintes de contenu que l'on peut imposer via une déclaration de type d'élément ne permet pas de contrôler le *contenu textuel* d'un type d'élément. Par exemple, il est impossible, via une DTD, d'exiger que le contenu d'un élément, mettons "`noSérie`", soit exclusivement numérique, ou composé d'exactement 8 caractères, ou les deux. Actuellement, de telles contraintes ne sont pas imposables dans le cadre de XML lui-même; il faut les mettre en force au niveau de l'*application* XML. Le W3C reconnaît qu'il serait extrêmement intéressant de pouvoir spécifier de telles contraintes au niveau de XML lui-même (entre autres parce que cette spécification serait alors normalisée), et s'apprête à proposer un formalisme, plus expressif que celui des DTD, qui rendra la chose possible: il s'agit de ce qu'on appelle les "schémas XML". En plus des contraintes de type DTD, un schéma XML permet d'imposer des contraintes de contenu textuel des éléments, analogues aux types (primitifs) de données que l'on retrouve dans les bases de données classiques ou dans les langages de programmation.

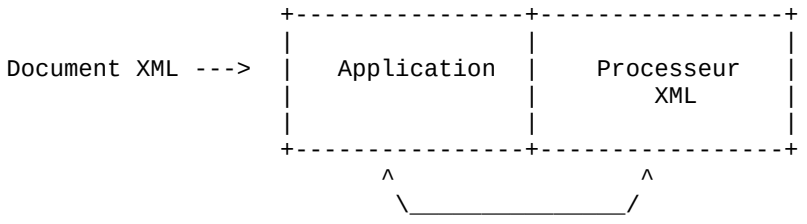
Pour plus d'information sur les schémas XML, voir <http://www.w3.org/TR/xmlschema-1/>.

## Applications validantes et non validantes

XML offre la possibilité de développer soit des applications qui travaillent exclusivement sur des documents valides, soit des applications qui travaillent sur des documents bien-formés. Le premier type d'application s'appelle "application validante" et le second "application non validante". La possibilité d'applications non validantes est nouvelle dans XML par rapport à SGML, qui forçait les applications à n'accepter que des documents valides par rapport à une ou plusieurs DTD.

Notez qu'une application non validante n'est pas obligée d'accepter n'importe quel document bien formé; seulement, elle doit elle-même effectuer les vérifications qui lui permettent de déterminer si un document est acceptable pour elle ou non. En comparaison, une application validante peut prendre pour acquis que tout vice de validité aura été détecté AVANT même que le document ne lui parvienne pour traitement. Une application validante "délègue" donc une partie des vérifications préalables des documents à un "processeur XML validant", qui filtre en quelque sorte les documents parvenant à l'application. Celle-ci peut bien sûr effectuer des validations additionnelles, comme par exemple sur le contenu textuel de certains éléments.

La norme XML définit très précisément ce qui doit être vérifié par les applications non validantes et validantes. Ces définitions s'appuient sur l'architecture type d'une application XML, que voici:




C'est l'application qui reçoit le document XML, et elle le passe au processeur XML, qui peut être validant ou non, selon que l'application est validante ou non. Le processeur effectue alors certains traitements sur le document (par exemple, le remplacement des références d'entités par leur texte de remplacement) et signale ensuite certains "événements" à l'application (par exemple, la rencontre d'une balise ouvrante), laquelle réagit comme elle veut à ces événements.

Notons que ceci est une architecture type de référence, qui permet de décrire le comportement des applications validantes et non validantes. Mais il serait tout à fait possible d'écrire une seule et même application qui utilise à la fois un processeur XML validant et un processeur XML non validant. Ainsi, cette application pourrait effectuer certains traitements sur les documents valides et d'autres traitements sur les documents bien formés mais non valides.

**Un des principes fondamentaux de XML est que c'est l'application (validante ou non) et l'application seulement qui détermine la "signification" des identificateurs génériques et la façon dont les éléments correspondants sont traités. XML ne met, ni ne permet de mettre, aucune restriction sur ce point.**

C'est à la conception même de l'application qu'on décide si celle-ci utilisera un processeur validant ou non validant. Le document XML lui-même n'y peut rien. Même s'il possède une déclaration de type de document, il ne peut pas obliger l'application à le valider.

La norme XML précise de façon très détaillée le traitement qui doit être effectué par un processeur XML, qu'il soit validant ou non. Signalons entre autres qu'un processeur XML, **même non-validant**, doit obligatoirement faire un certain traitement minimal de la déclaration de type de document, lorsque celle-ci est présente. Ainsi, par exemple, toute entité générale définie dans le sous-ensemble interne de déclarations doit être reconnue par un processeur XML, même non validant. Également, tout processeur XML, même non validant, s'affichant comme conforme à XML ne peut PAS passer sous silence les erreurs de bien-formé des documents traités; il a l'**obligation** de signaler ces erreurs à l'application.

## Digression sur HTML et XML

HTML (HyperText Markup Language), le langage natif des documents Web, développé originellement à la fin des années 1980 par Tim Berners-Lee, l'actuel directeur du W3C, est depuis quelques versions déjà, présenté comme une DTD SGML (puis, plus récemment, XML). Par contre, la plupart des applications HTML ne se comportent même pas comme une application non validante, en ce sens qu'elles tolèrent sans signaler d'erreur à peu près tous les écarts imaginables à la validité HTML, et même au simple bien-formé. Même si l'on peut dire que le HTML 4.0 est bel et bien une DTD SGML, il n'en demeure pas moins que les applications HTML ne sont pas des applications SGML. Un nouvel effort du W3C, baptisé XHTML, vise à définir très précisément le HTML (sur la base du HTML 4.01) comme application XML en bonne et due forme. Aujourd'hui, toute initiative d'envergure de production de documents numériques basés sur HTML devrait probablement s'appuyer sur XHTML pour s'assurer la plus grande disponibilité possible d'outils et de fonctionnalités.

Avec passablement de retard par rapport au W3C, ISO travaille actuellement à normaliser sa propre version de HTML, qui sera, à quelques détails près, identique au HTML 4.0 du W3C. Cette initiative nous semble pour le moins mystérieuse, mais vise probablement à démontrer qu'ISO continue à jouer un rôle dans la normalisation, même dans les domaines couverts par le W3C.

## Documents orientés-données versus orientés-information

Il est utile à ce point de distinguer entre "documents orientés-données" et "documents orientés-information" (notez qu'il s'agit de deux noms originaux, un peu arbitraires, mais tout de même illustratifs). Les premiers (orientés-données) sont analogues à des fiches. C'est le genre de document qui se prête à la saisie par formulaire (numérique ou traditionnel). La structure de tels documents est très rigide et ne permet presque aucune variation. L'archétype des documents

orientés-données est la ligne d'une table de base de données relationnelle. Un autre exemple est la fiche de base de données textuelles, qui n'est pas en première forme normale (pouvant avoir des "occurrences multiples"), et dont la structure, bien qu'un peu plus souple qu'en relationnel, est tout de même passablement rigide.

Les seconds (orientés-information) ont une structure plus variable; la signification de leur contenu est plus floue, elle n'est pas entièrement déterminée par un "schéma" rigide. Typiquement, les documents orientés-information sont majoritairement textuels, ils servent le plus souvent à produire des documents pour consommation humaine (soit sur papier, soit sur écran) et comportent des passages importants en langue naturelle, typiquement des phrases complètes, avec ponctuation, etc. En gros, c'est les documents que l'on crée habituellement avec un logiciel de traitement de texte (Word, WordPerfect, etc.). Cela inclut tous les textes littéraires. D'autres exemples sont des catalogues de magasins, des encyclopédies, des manuels de documentation technique, dictionnaires, plans, notes de service, devis, contrats, procès-verbaux, appels d'offres, procéduriers, etc.

Beaucoup des utilisations "à la mode" de XML sont en fait pour des documents orientés-données. Ce n'est pas que ce soit condamnable, loin de là, car certains bénéfices comme la normalisation, la lisibilité humaine et l'explicitation de la structure logique de l'information sont obtenus. Mais ce n'est pas dans ces contextes que les pleins bénéfices des documents structurés sont les plus évidents.

En effet, les documents sur lesquels les entreprises ont de tout temps dépensé (et souvent perdu!) des sommes faramineuses sont d'abord et avant tout les documents orientés-information et c'est pour faciliter la gestion et l'exploitation efficaces des documents de ce type que les documents structurés ont été inventés.

Pour cette raison, nos exemples à partir de maintenant seront donnés avec des documents orientés-information. L'exemple jouet de fiches de carnet d'adresses que nous avons utilisé plus haut serait plutôt un exemple de documents orientés-donnés.

Entre les documents orientés-données et ceux orientés-information, il tend à se dégager de nos jours un type de documents (ou en tout cas d'utilisations de XML) que l'on pourrait qualifier d'*orientés-formules*. Mentionnons particulièrement le CML (Chemical Markup Language) pour les formules chimiques et le MathML (Mathematical Markup Language), pour les formules mathématiques et scientifiques en général. Il est difficile de parler ici de documents proprement dits, car ces langages sont surtout utilisés pour représenter des fragments très spécialisés de documents, fragments dont la structure est très variable. Disons que, même si les documents structurés n'ont pas été inventés spécifiquement pour les documents orientés-formules, ces derniers semblent bénéficier considérablement de l'approche documents structurés, surtout normalisés.

## Le second principe des documents structurés

Nous avons terminé notre survol des éléments de la syntaxe de XML que nous voulions couvrir. Nous sommes maintenant presque prêts à passer aux implications pratiques du second principe des documents structurés qui, rappelons-le, s'énonce comme suit:

> *La structure hiérarchique du document numérique doit correspondre le mieux et le plus explicitement possible à la nature et à la structure de "l'information" qui doit être véhiculée par le document.*

Ce principe est plus difficile à cerner que le premier, entre autres parce qu'il fait intervenir le mot "information", que personne ne veut, ou ne peut, définir! Chose sûre, il a à voir avec les "usagers" d'un éventuel système de traitement de documents structurés. Le respect de ce principe a donc des ramifications jusque dans la méthodologie de conception des systèmes d'information, l'analyse des besoins et la prise en compte des facteurs humains.

## Historique

Comme nous avons dit, les documents structurés ont été inventés pour faciliter la gestion de documents orientés-information. L'histoire des documents structurés nous renseigne sur le pourquoi du second principe. Penchons-nous donc un peu sur cette histoire, qui commence avec un problème de production de textes juridiques.

En effet, Charles F. Goldfarb, créateur de SGML, était juriste à l'emploi de IBM à la fin des années soixante, et était confronté au problème de production de textes juridiques. Le problème était une dépendance des documents numériques envers des équipements spécifiques de photocomposition. En effet, les différentes opérations de mise en page étaient directement codées dans les documents numériques via des codes exclusifs à un équipement particulier. Un changement d'équipement (ou de fournisseur de service) entraînait une nécessité de ressaisir les textes en intégrant de nouveaux codes compatibles avec le nouvel équipement.

Une des approches étudiées fut le "codage générique", qui prévoit l'insertion de codes dans les documents numériques, non pas en fonction d'équipement spécifique, mais des logiciels utilisés. Ainsi, les opérations de mise en page sont toujours codées, mais au moins ce codage demeure le même tant qu'on utilise le même logiciel. Le logiciel s'occupe de traduire les codes génériques en codes spécifiques à l'équipement utilisé au moment de la restitution physique du document. On n'a donc plus de dépendance vis-à-vis de l'équipement, mais il subsiste tout de même une dépendance face au logiciel, qui ne se montre cependant que lorsqu'on essaie d'utiliser les documents numériques à d'autres fins que celles pour lesquelles ils ont tout d'abord été saisies.

Essentiellement, le codage générique (mais qu'on appelle maintenant "balisage générique") est encore le genre de balisage utilisé par les logiciels de traitement de texte (lorsqu'on n'utilise pas les facilités de feuilles de styles). Il faut noter que le balisage générique ne cause pas seulement des problèmes lorsqu'on change de logiciel. En effet, même sans changer de logiciel, certaines opérations intervenant dans la réutilisation de l'information contenue dans les documents sont ardues. Supposons par exemple que l'on désire changer le style typographique d'une série de documents (il s'agit donc bien de réutilisation). En général, un tel changement impliquera l'identification de la fonction de chaque segment du document (par exemple: titre, sous-titre, légende de figure, etc.) de façon à pouvoir lui attribuer les nouveaux attributs typographiques. Or, avec le balisage générique, cette identification est nécessairement fragile, puisqu'elle s'appuie sur des attributs typographiques plutôt que sur un marquage explicite. Une telle identification ne peut être faite automatiquement **que si** un protocole typographique très strict a été respecté intégralement lors de la saisie des documents, **et si** ce protocole typographique ne comporte aucune "collision" stylistique, c'est-à-dire si chaque fonction possible (titre, sous-titre, etc.) est doté d'une typographie distinctive qui lui est exclusive. Par exemple, si l'on veut changer automatiquement la typographie des titres dans un document, et que les titres sont centrés et en caractères gras, il faut que **rien d'autre** que les titres ne soit centré et en caractères gras dans le document.

Après plusieurs essais et un produit commercial d'IBM appelé GML, Goldfarb et son équipe arrivèrent à la conclusion que seul un balisage *descriptif*, indépendant non seulement du matériel, mais également des logiciels, pouvait permettre une véritable réutilisation de l'information. (Notons anecdotiquement que le sigle GML signifie Generalized Markup Language, mais coïncide également avec les initiales des trois auteurs d'un article décrivant ce système: Goldfarb, Mosher et Lorie.) Avec un balisage descriptif, les balises utilisées n'ont plus aucun lien avec les opérations de traitement des documents, mais traduisent exclusivement la nature ou la fonction intrinsèque du **contenu** des différents segments du document. Ainsi, un titre est balisé "TITRE", une légende de figure est balisée "LÉGENDE", etc.

## Normalisation

Le balisage descriptif n'est en fait rien d'autre que le second principe des documents structurés formulés en termes de balisage. Comme nous venons de voir, il a été imaginé et est directement orienté vers la **réutilisation** de l'information contenue dans les documents, bienfait fondamental que les documents structurés peuvent apporter. Notons que la normalisation joue directement dans le même sens, car si un format est normalisé, il sera beaucoup plus facilement transférable, et donc réutilisable, sur des plateformes géographiquement, culturellement, et même temporellement, éloignées. La normalisation est donc un prolongement tout à fait naturel du second principe puisque, comme lui, elle favorise la réutilisabilité de l'information documentaire.

## Exemple et contrexemple

Voyons maintenant un exemple et un contrexemple de la mise en application du second principe. Supposons qu'on veuille produire des mémos pour la communication entre employés dans une société. Traditionnellement, on utilisera un logiciel de traitement de texte pour les produire, on les imprimera et on en distribuera des photocopies. Sur papier, un tel mémo se présenterait par exemple comme suit:

**MÉMORANDUM**

**De:** Julia Royer
**À:** Jean Picard
Émilie Dugré

**Sujet:** Invitation

Veuillez noter que la prochaine réunion du conseil d'administration se tiendra le 27 septembre 2000.

SVP, avisez-moi dans les plus brefs délais si vous ne pouvez pas y assister.

Avec un logiciel de traitement de texte, mettons Word, le document numérique n'est pas directement visualisable. Cependant, le format RTF (Rich Text Format), qui est fonctionnellement équivalent au format Word, est, lui, visualisable directement, puisque basé sur le format texte US-ASCII. Si on examine un extrait du document RTF correspondant au mémo ci-dessus, cela donne à peu près ce qui suit:

```
\adjustright \fs20\lang3084\cgrid \b\f2\fs38\cf1\cgrid0
MÉMORANDUM}{\f2\fs38\cf1\cgrid0
\par }\pard \nowidctlpar\widctlpar\adjustright {\f2\fs38\cf1\cgrid0
\par }{\b\f2\fs38\cf1\cgrid0 De:}{\f2\fs38\cf1\cgrid0  Julia Royer
\par }{\b\f2\fs38\cf1\cgrid0 À:  }{\f2\fs38\cf1\cgrid0 Jean Picard
\par     Émilie Dugré \par \par }{\b\f2\fs38\cf1\cgrid0 Sujet: }
{\f2\fs38\cf1\cgrid0 Invitation
\par \par }\pard nowidctlpar\widctlpar\brdrt\brdrs\brdrw10\brsp20
\par }\pard \nowidctlpar\widctlpar\adjustright {\f2\fs38\cf1\cgrid0
Veuillez noter que la prochaine réunion du conseil d'administration
se tiendra le 27 septembre 2000.
\par \par SVP, avisez-moi dans les plus brefs délais si vous ne pouvez pas y assister.}{ \par }}
```

Comme on peut voir, le genre de "balises" que l'on retrouve dans ce document numérique est directement orienté vers le formatage du document. En effet, on peut repérer une instruction "\adjustright" qui signifie d'aligner quelque-chose à droite, les "\b" signifient probablement de mettre en gras, etc. Rien dans ces balises n'indique directement la nature de l'information, seulement son formatage.

Pour être complètement honnête avec Word et les logiciels de traitement de texte, il faut dire que l'extrait RTF ci-dessus correspond au cas où on n'a pas utilisé les facilités de feuilles de styles du logiciel. Si une feuille de styles est utilisée (de façon judicieuse), alors certaines des balises peuvent correspondre à la nature de l'information plutôt qu'à son formatage. D'ailleurs, l'utilisation de feuilles de styles dans des logiciels de traitement de texte constitue une approche privilégiée pour la création de documents structurés (via conversion) dans les situations où il est impossible que les auteurs des documents travaillent directement dans un format de documents structurés (XML ou SGML). Nous reviendrons sur ce point plus tard.

Il va sans dire que le document RTF dont un extrait est présenté ci-dessus constitue le contrexemple de l'application du second principe. Voyons maintenant l'exemple, c'est-à-dire le même mémo, mais codé en XML, dans le respect du second principe:

```
<mémo>
<auteur> Julia Royer </auteur>
<destinataires>
  <nom> Jean Picard </nom>
  <nom> Émilie Dugré </nom>
</destinataires>
<sujet> Invitation </sujet>
<corps>
  <par> Veuillez noter que la prochaine réunion du conseil
        d'administration se tiendra le 27 septembre 2000. </par>
  <par> SVP, avisez-moi dans les plus brefs délais si vous ne
        pouvez pas y assister. </par>
</corps>
</mémo>
```

Ici, il est clair que les balises traduisent la nature et la structure de l'information véhiculée par le document, et ce, explicitement. Il n'est pas difficile d'imaginer que la réutilisation d'un tel document à d'autres fins que celles pour lesquelles il a été originellement créé, serait beaucoup plus aisée qu'avec le document RTF précédent. Si vous vous demandez comment on pourrait bien réutiliser un document aussi anondin, voici un exemple: on pourrait vouloir constituer un répertoire de tous les mémos émis dans l'entreprise pendant l'année. Avec les mémos XML, on

peut en toute certitude extraire automatiquement les auteurs et les sujets des mémos; par contre, avec le RTF, cette opération est plutôt fragile, car, par exemple, pour identifier l'auteur du mémo, il faut soit se fier sur la présence du mot "De:" (mais d'autres formulations sont aussi possibles) ou bien sur le fait qu'il figure en premier dans le document (mais est-ce vraiment toujours le cas?).

### Liens entre le premier et le second principe des documents structurés

Maintenant que nous voilà convaincus du bien-fondé du second principe des documents structurés, soulignons trois caractéristiques des documents structurés qui, du moins *a priori*, encouragent l'application de ce principe:

D'abord, le simple fait que le type de structure adopté soit hiérarchique est un bon point de départ, car il faut reconnaître que beaucoup de documents orientés-information réels ont une structure naturelle hiérarchique (par exemple, division en parties, chapitres, sections, etc.). Parfois, la structure naturelle d'un document dépasse le simple hiérarchique, et il faudra donc permettre la représentation de structures hypertextuelles, mais disons que le hiérarchique est un bon point de départ.

Autre caractéristique favorable au respect du second principe: l'obligation de préciser un identificateur générique dans la balise ouvrante de chaque élément. Cette obligation fournit une "belle occasion" de rendre explicite et transparent le type d'information contenu dans chaque élément. Encore faudra-t-il la saisir!

Troisième point: le fait de pouvoir spécifier des contraintes de validité sur les documents numériques. En effet, une des façons de s'assurer que la structure d'un document correspond le mieux possible à la structure de l'information à véhiculer est parfois d'interdire certaines structures qui ne voudraient rien dire en termes informationnels.

### Le second principe pour les documents orientés-données

Comme nous avons dit, l'évolution ayant mené à l'invention des documents structurés est d'abord et avant tout liée à la production de documents orientés-information. Il est évident que les défis de modélisation sont beaucoup plus grands pour ceux-ci que pour les documents orientés-données.

Notons deux choses:

1) Qui peut le plus peut le moins (ici, on suppose que XML dispose du formalisme des schémas qui permettent d'exprimer des contraintes de contenu textuel, et pas seulement de celui des DTD). Ainsi, il est clair qu'un formalisme permettant de modéliser des documents orientés-information permettra également de modéliser des documents orientés-données.

2) Le respect du second principe des documents structurés est d'une grande simplicité dans le cas des documents orientés-données.

Pour illustrer ces deux points, notons que la modélisation d'une structure de table relationnelle par une DTD est un jeu d'enfants.

## Méthodologie adaptée aux documents structurés

On a beau être convaincu du bien-fondé du second principe des documents structurés, pour être en mesure de le mettre en oeuvre dans les contextes les plus variés, il faut posséder une

méthodologie adaptée. Pour bien situer les étapes méthodologiques sur lesquelles le second principe influe, regardons d'abord les façons types dont les documents structurés sont utilisés concrètement.

Nous présenterons donc maintenant:

- Les principaux types d'outils XML.
- Une chaîne de traitement type pour documents XML.
- Le déroulement type d'un projet d'implantation XML.

## Types d'outils XML

- Éditeurs structurés (SGML: Author/Editor; XML: Microsoft XML Notepad, SoftQuad X-Metal).
- Processeurs XML: XP (gratuit, non validant), OmniMark (gratuit, validant). Utilisables souvent via une API, particulièrement l'API normalisée gratuite SAX (Simple API for XML), disponible en Java. Pour information sur XP: <http://www.jclark.com/xml/xp/index.html>. Pour information sur SAX: <http://www.megginson.com/SAX/index.html>. Pour information sur OmniMark: <http://www.omnimark.com/develop/index.html>.
- Navigateurs Web capables de XML (incluant support des feuilles de styles CSS ou XSL): Microsoft Internet Explorer 5.0 ou plus récent. Pour SGML: Interleaf Panorama (originellement publié par SoftQuad).
- Publication assistée par ordinateur (PAO): Adobe FrameMaker+SGML (inclut support XML dans la dernière version).
- Convertisseurs généraux: OmniMark (gratuit), voir coordonnées ci-dessus.

Des répertoires d'outils sont pointés par le site de Robin Cover (voir référence au début du présent document). Un répertoire en français d'outils se trouve à <http://mapageweb.umontreal.ca/marcoux/grds/outils-xml/>.

### Feuilles de styles

Quelques mots sur les fonctionnalités de feuilles de styles: il s'agit d'associer aux identificateurs génériques des attributs de présentation (par exemple, police de caractères, graisse, etc.). Les attributs de présentation peuvent dépendre de: l'identificateur générique de l'élément, sa "lignée ancestrale" (la série de parents jusqu'à la racine du document), la présence ou l'absence d'un attribut dans la balise d'ouverture, la valeur d'un attribut.

Un formalisme de feuille de styles normalisé pour les documents SGML est DSSSL (Document Style Semantics and Specification Language). Il s'agit d'une norme ISO. Mais DSSSL a été adoptée en 1996, soit 10 ans après SGML. Sur l'entrefait, plusieurs éditeurs de produits SGML ont adopté des mécanismes de feuilles de styles non normalisés officiellement (et moins expressifs que DSSSL). Un de ces formalismes est le langage "ViewPort" (du nom d'un produit de la société suédoise Synex). C'est le langage de feuilles de styles utilisé dans Panorama et plusieurs autres produits.

Le formalisme des CSS (Cascading Style Sheets) du W3C, développé pour HTML est applicable aussi à XML, et est reconnu par les principaux navigateurs Web. XSL, dont une partie (XSLT) est actuellement au statut de proposition de recommandation au sein du W3C, est un langage de feuilles de styles expressément développé pour XML. Une des particularités de XSL est qu'il permet le réarrangement des éléments dans la sortie formatée. Ainsi, même si un document

présente un auteur et un titre dans cet ordre, on pourrait inverser ces deux éléments pour présentation avec une feuille de styles XSL. Cette possibilité, en apparence anodine, est absente de la plupart des autres formalismes de feuilles de styles (elle est par contre aussi présente dans DSSSL, qui a beaucoup inspiré le développement de XSL).

## Chaîne de traitement type

Auteurs -> Documents XML -> Entrepôt de documents -> Indexation robotisée -> Outils de recherche -> Requêtes de recherche -> Lecteurs/utilisateurs des documents

Par ailleurs, certains auteurs peuvent aussi produire des documents en logiciel de traitement de texte, qui sont ensuite convertis (une "up-conversion", puisqu'on augmente le contenu sémantique du document) automatiquement ou semi-automatiquement en XML. Dans un tel scénario, on étudiera attentivement la possibilité de fournir aux auteurs (et de les obliger à utiliser!) une feuille de styles qui permet le codage d'une certaine information sémantique dans les documents directement par les auteurs. Cette information est récupérée et exploitée par le programme de conversion. OmniMark (avec ou sans l'application gratuite "rtf2xml", qui effectue une première conversion grossière du RTF vers XML) est un outil de choix très puissant pour de telles conversions. (Pour plus d'information sur rtf2xml, voir <http://www.xmeta.com/omlette/rtf2xml/>.)

L'entrepôt de documents pourrait être une base de données en bonne et due forme, mais avec l'existence de robots-indexeurs capables de XML, on peut dans bien des cas se contenter d'un système de fichier, que l'on recherche à l'aide d'un outil de recherche externe.

Par ailleurs, l'entrepôt de documents est aussi directement exploité par des applications d'extraction et/ou conversion (par exemple, développés avec OmniMark ou un produit SAX), qui produisent automatiquement des produits dérivés, comme des sites Web, des bases de données, des index, etc.

Dans le cas d'une application de documents orientés-données, l'entrepôt de documents pourra être une simple base de données traditionnelle, par exemple, relationnelle.

## Déroulement type d'un projet XML

Au début, un projet XML ne se distingue pas d'un autre projet d'informatique documentaire. En effet, au moment de l'étude des besoins ou de faisabilité, on ne devrait pas savoir encore qu'il s'agit d'un projet XML, cette technologie n'étant qu'une des ressources disponibles pour répondre aux besoins en train d'être identifiés. Rappelons cependant que pour tout projet documentaire, l'étude de faisabilité devrait toujours comporter un répertoriage de toute la matière première documentaire disponible dans l'entreprise en périphérie des documents déjà identifiés dans la portée du projet. L'idée est de ne pas manquer de "belles occasions" d'inclure dès le départ des documents que l'on voudra de toutes façons inclure plus tard. Il faut également se faire une image claire de l'utilisation des différents documents (qui les utilise, pourquoi, comment?). Il faut, comme toujours, établir un plan et un calendrier de réalisation du projet.

Après l'étude de faisabilité, l'aval de l'administration et l'attribution des budgets nécessaires, il faudra comme toujours constituer une équipe pour piloter le projet, laquelle devrait inclure des membres de tous les groupes impliqués (particulièrement, des utilisateurs).

Pendant ou peu après l'étude de faisabilité, on aura identifié que les documents structurés constituent la meilleure approche. Il aura fallu alors comparer avec d'autres technologies, comme

les BD traditionnelles (relationnelles), textuelles, orientées-objet, et l'hypertexte classique (e.g., sites Web en XHTML ou HTML). Nous supposerons ici dans ce scénario type que les documents structurés ont été identifiés comme la meilleure approche.

Il faut alors procéder à l'analyse documentaire des documents répertoriés préalablement et déterminer exactement la portée du nouveau système à mettre en place (quels documents il traitera). Suite à cette analyse, on procédera à la modélisation documentaire proprement dite, à la rédaction des DTD, puis aux autres étapes classiques, comme le choix des outils (matériels, logiciels) et, le cas échéant, le développement d'applications sur mesure (dans le cas des documents structurés, ce sera le plus souvent des applications de conversion). Puis, on procédera aux tests itératifs avec des usagers et à l'implantation du nouveau système.

## Modélisation de documents orientés-information

L'étape sur laquelle influe le plus le second principe est la conception de la ou des DTD du système (ou des schémas XML, si l'on se projette dans le futur). C'est cette étape que l'on appelle la modélisation documentaire. Au cours de cette étape, le second principe tirera du côté d'une espèce d'idéalisation de l'information et de son utilisation dans le monde. Par contre, des considérations pratiques tireront du côté de la réalité, et des concessions devront en général être faites en regard du second principe. Rappelons le but du respect du second principe au cours de cette étape: assurer la plus grande indépendance possible de notre information par rapport aux outils disponibles pour la traiter, de façon à en maximiser la réutilisabilité. L'important n'est pas tant de le respecter à tout prix, mais d'être conscient des licenses que l'on prend par rapport à lui et des implications que ces licenses ont sur les pleins bénéfices potentiels des documents structurés.

Voici d'abord quelques règles générales permettant de respecter le plus possible le second principe lors de la modélisation documentaire.

D'abord, la **règle d'or** des documents structurés, qui est en fait la quintessence du concept de balisage descriptif:

> *Les balises (incluant les identificateurs génériques) et les attributs (incluant leur nom) doivent être utilisés de telle façon qu'ils indiquent le plus clairement possible, dans une langue naturelle pertinente, le type de l'information qui constitue le contenu des éléments et des attributs.*

Cette règle peut paraître une évidence, mais ce n'est pas le cas: par exemple, le format MARC (Machine Readable Cataloguing), utilisé pour l'échange de données catalographiques entre bibliothèques, évite délibérément les balises en langue naturelle, leur préférant des étiquettes numériques de façon à ne privilégier aucune langue par rapport aux autres. En général, cependant, l'usage de balises en langue naturelle est préférable, la question principale étant alors le choix de la langue.

Dans l'exemple de fiches de carnet d'adresses que nous avons présenté précédemment, moyennant quelques hypothèses raisonnables, nous avons respecté cette règle.

Ensuite, une autre règle générale (originale) pourrait être énoncée comme suit:

> *La profondeur du balisage doit être déterminée en fonction des besoins des applications courantes de traitement des documents, cependant, la nature du balisage doit être établie de façon complètement indépendante des applications courantes.*

C'est cette règle qui nous assurera que le balisage adopté n'est pas biaisé en faveur d'une application par rapport aux autres.

### Types de balisages descriptifs

Tout en restant en accord avec le second principe, il est possible d'adopter plusieurs approches (non mutuellement exclusives) au balisage descriptif. Alain Michard répertorie sous le vocable de "règle" un certain nombre de recommandations quant au balisage. Nous citons:

- Inclusion de métadonnées descriptives dans le document numérique. Il faut ici mentionner le RDF (Resource Description Framework), métamodèle extrêmement général de métadonnées développé par le W3C et basé sur XML, et vers lequel se réorientent à peu près tous les efforts majeurs de normalisation des métadonnées, comme le Dublin Core, GILS, EAD (Encoded Archival Description), etc. Pour plus d'information sur les métadonnées et RDF, consulter <http://metadata.net/> et <http://www.w3.org/Metadata/>.
- Marquage métatypographique.
- Indépendance physique (ne pas traduire l'apparence par le balisage).
- Marquage sémantique.

Le ou les bons choix seront dictés par les besoins particuliers du milieu d'implantation.

### Méthodologies de modélisation

Les types de balisage descriptif répertoriés par Michard ne constituent en fait que certaines possibilités, les plus répandues. Une autre approche, lors de la modélisation, est d'utiliser une DTD toute faite, de préférence normalisée par un groupe ayant des intérêts similaires à ceux du milieu d'implantation (e.g., DocBook, TEI, ISO-12083, etc.). Mais la seule approche qui garantisse les bénéfices des documents structurés est de modéliser sur mesure, en fonctions des besoins spécifiques du milieu d'implantation.

Plusieurs petits textes ont été écrits sur le processus de modélisation documentaire en XML. Quelques-uns sont très orientés structure (syntaxe), négligeant quasi-complètement le second principe, et ne s'appliquant par le fait même directement qu'aux documents orientés-données, sans que cette limitation ne soit mentionnée explicitement.

À notre connaissance, un seul ouvrage traite de façon systématique de tous les aspects de la modélisation documentaire XML, incluant le respect du second principe. Il s'agit du livre de Eve Maler et Jeanne El Andaloussi donné en référence au début du présent texte.

La méthode de Maler et El Andaloussi est basée sur l'identification des "composants sémantiques" que l'on retrouve dans les documents à modéliser. En gros, la méthode est de déterminer la profondeur du balisage en fonction des composants sémantiques identifiés et la nature du balisage en fonction d'une catégorisation plus ou moins fine de ces composants, selon les besoins de traitement des documents.

## Problèmes particuliers (attributs, liens hypertextuels, tableaux)

### Utilisation des attributs

Une bonne règle générale (mais qui comporte des exceptions) est qu'un attribut doit contenir une information qui ne figurerait pas nécessairement directement telle quelle dans le flux du texte si le

document était imprimé (au sens classique, sur papier), mais qui serait tout de même utile au traitement du contenu de l'élément. Notre exemple avec nos fiches est une illustration parfaite de cette règle: l'information contenue dans l'attribut (est-ce que le numéro est au travail ou à la maison?) ne figurerait pas nécessairement telle quelle avant le numéro de téléphone imprimé, mais pourrait servir par exemple à positionner le numéro dans l'une ou l'autre des colonnes d'un tableau (dont l'en-tête, lui, préciserait le type de numéros de téléphone).

## Liens hypertextuels

Le second principe dit que la structure du document numérique doit coller le mieux possible à la nature et à la structure de l'information véhiculée par le document. Dans plusieurs cas, la structure intrinsèque de l'information n'est pas hiérarchique, mais bien arbitraire, donc, hypertextuelle. Comme les documents structurés ne peuvent pas représenter directement des structures non hiérarchiques, on s'appuiera sur une sémantique spéciale de certains éléments (et/ou attributs) pour représenter les liens arbitraires. En vertu du principe fondamental évoqué précédemment, selon lequel la signification des balises est déterminée exclusivement par l'application, cette sémantique relève des applications et est donc hors du domaine de XML et des documents structurés. Autrement dit, il faut définir une application qui interprète certains éléments et/ou attributs comme des liens hypertextuels.

Évidemment, cette solution en soi n'est qu'à demi satisfaisante. En effet, on aimerait bien que la représentation de liens hypertextuels, même si elle dépend des applications, soit normalisée d'une application à l'autre. Ce problème a depuis longtemps été identifié dans le contexte de SGML. La solution proposée (par ISO) a été le développement d'une norme spécifiquement conçue pour la représentation normalisée, en SGML, des documents hypermédias: HyTime (Hypermedia Time-based Document Structuring Language, norme ISO 10744, adoptée en 1997). HyTime est une norme très générale, qui inclut beaucoup plus que la représentation des liens hypertextuels. Une partie de cette norme, portant justement sur la représentation des liens hypertextuels, a été implantée dans certains produits SGML, dont le navigateur Panorama.

HyTime n'est pas applicable directement à des documents XML, mais le W3C est à développer des normes similaires pour XML. Ces normes sont inspirées de HyTime sous certains points de vue. Il s'agit de XLink, XPath et XPointer, qui sont à divers degré de normalisation (voir le site du W3C pour plus de détails). Un aspect intéressant des ces normes est qu'elles permettront la représentation de ce qu'on appelle des "liens indépendants", un concept hérité de HyTime, qui consiste en un lien qui n'est stocké ni dans le document de départ, ni dans le document d'arrivée du lien, mais un troisième document, indépendant des documents liés. Ce concept est intéressant car il permet l'ajout de liens à partir de ou vers des ressources sur lesquelles on ne possède pas de droit d'écriture. Évidemment, la visibilité de tels liens est conditionnée par la visibilité du document contenant les liens, en plus de celle des documents de départ et d'arrivée.

## Tableaux

Il arrive fréquemment que l'on ait à représenter des tableaux dans des documents structurés. *A priori*, on pourrait penser que la structure d'un tableau est porteuse d'une information sémantique riche. Mais, l'analyse montre que, souvent, il ne s'agit que d'un artifice présentationnel n'ayant pas de justification sémantique très forte. Pourtant, la plupart du temps, cette structure doit être représentée dans les documents numériques, de façon à pouvoir reproduire les tableaux formatés à partir des documents numériques.

Ce problème a été identifié dès les premières utilisations de SGML. On a trouvé qu'une des solutions les plus habiles était d'utiliser une "sous-DTD", c'est-à-dire un ensemble de balises,

conçue spécifiquement pour représenter des tableaux. En fait, dans le cadre du premier programme majeur de l'armée américaine préconisant l'usage de SGML, une DTD spécifique a été développée à cette fin. Cette DTD, encore beaucoup utilisée (en version XML) comme sous-DTD pour la représentation de tableaux dans divers projets, s'est vue baptisée du nom du programme lui ayant donné naissance: CALS (qui signifiait à l'origine *Computer-aided Acquisition and Logistics Support*, mais a été renommé *Continuous Acquisition and Lifecycle Support* lorsque les techniques et méthodes du programme original ont été migrées vers des grappes industrielles non militaires).

## Une utilisation concrète: le projet Érudit aux PUM

Il s'agit d'un projet de revues savantes électroniques piloté par les Presses de l'Université de Montréal. Les revues sont diffusées en SGML, PDF, HTML et sur papier à partir d'une seule et même source SGML. Voir le système complet en opération au <http://www.erudit.org/>.

## Un exercice de modélisation

Dans cet exercice, nous modéliserons en XML une partie de l'information provenant de boîtiers de disques compacts musicaux. Nous nous situons dans le contexte hypothétique suivant:

1. On devra minimalement gérer assez d'information pour alimenter le lecteur CD de Windows.
2. On veut être capable de faire du copier-coller au niveau d'une piste, par exemple pour les échanges entre individus.
3. On veut pouvoir visualiser sa collection de CD selon divers critères (artistes, genre, etc.) à partir de l'information conservée.

L'objectif de l'exercice n'est pas d'arriver à une modélisation directement implantable ou la plus fonctionnelle possible, ni la plus propre possible, mais simplement de mettre en évidence le genre de questions qui se posent dans le processus de modélisation documentaire en documents structurés.

### Observations durant l'analyse documentaire

L'univers documentaire à modéliser possède la structure hiérarchique inhérente suivante:

$$\text{LABEL} \Rightarrow \text{COLLECTION} \Rightarrow \text{COFFRET} \Rightarrow \text{DISQUE} \Rightarrow \text{PISTE}$$

Les entités (au sens E/R) COLLECTION et COFFRET ne sont pas présentes dans tous les cas; leur "absence" correspondrait, normalement, à des instances unitaires (c'est-à-dire, à un seul élément).

Parallèlement à cette hiérarchie, on trouve les entités COMPOSITEUR et EXÉCUTANT, qui sont le plus souvent en relation avec COFFRET, mais peuvent à l'occasion être en relation avec COLLECTION, DISQUE ou PISTE. Nous choisissons de ne pas subdiviser les entités COMPOSITEUR et EXÉCUTANT, vue leur importance relativement faible dans les applications recensées. Par contre, nous ne pouvons pas les ignorer totalement (ne serait-ce que parce que le lecteur CD de Windows comporte un champ "Artiste") et modéliserons donc au moins une forme de relation entre ces entités et les entités documentaires retenues.

Une certaine notion de sous-piste semble exister, mais elle est jugée dès le départ trop imprécise et trop peu répandue pour être retenue.

La notion de coffret est suffisamment proche de celle de disque pour qu'on puisse éliminer l'entité COFFRET et la remplacer par un attribut (au sens E/R) de DISQUE qui situe simplement un disque par rapport à un éventuel coffret. Les entités LABEL et COLLECTION, normalement contexte d'occurrence des instances de l'entité DISQUE, ne constituent pas des objets d'intérêt direct pour les applications recensées. Nous en ferons donc des attributs (toujours au sens E/R) de l'entité DISQUE. C'est un genre de compromis courant en informatique documentaire. Le prix à payer est que le problème du *contrôle des identificateurs* se pose alors (voir ci-après).

### Digression sur le problème du contrôle des identificateurs

Le problème du **contrôle des identificateurs** se pose lorsqu'on "déguise" une entité du monde réel en un attribut du modèle. En effet, il n'y a alors ni clé étrangère ni identificateur d'objet disponible pour identifier de façon univoque les instances des entités cachées. Il faut donc établir une procédure permettant, à la saisie, de déterminer une valeur d'attribut qui "identifie" de la meilleure façon possible l'instance d'entité. Plusieurs artifices existent, le plus efficace étant probablement le choix dans une liste de valeurs déjà entrées pour cet attribut (voir ci-dessous). Il est clair, cependant, que peu importe l'artifice utilisé, il n'y a aucune garantie de complétude ou de cohérence des données, et il y a bien sûr redondance.

Nous illustrons le problème avec l'entité COMPOSITEUR, mais il se posera également avec EXÉCUTANT, LABEL, GENRE et COLLECTION.

Voici un exemple concret de ce problème et une illustration de différentes solutions possibles. Supposons qu'on choisit de "cacher" l'entité COMPOSITEUR, en la "déguisant" en attribut de PISTE. Le moment venu de saisir une nouvelle piste, il faudra déterminer la valeur de "l'attribut" COMPOSITEUR. Mais cet attribut n'est ni une clé étrangère ni un identificateur d'objet; simplement une chaîne de caractères! Rien n'empêchera donc, *a priori*, que, pour une certaine piste, on donne "BACH JS" comme valeur, pour une autre, "Bach, Johann Sebastian" et enfin, pour une troisième, "Bach, J.-S." (surtout si ces pistes sont sur des disques différents). Mais en réalité, ces trois valeurs distinctes d'attribut désignent la même instance de COMPOSITEUR.

Deux types d'outils (combinables) s'offrent d'emblée pour diminuer les méfaits d'une telle situation: les protocoles de saisie à l'intention des responsables de la saisie et la validation programmatique. Dans le premier cas, on informe, forme et sensibilise le personnel responsable de la saisie à une façon "normalisée" de déterminer les valeurs d'attribut. Il peut s'agir de texte d'aide en ligne, de directives de saisie lorsque le curseur se trouve dans le champ approprié, de sessions de formation, etc. Dans le second cas, on paramétrise le programme de saisie pour qu'il effectue certaines vérifications lexicales sur les données saisies. Par exemple, on pourrait faire vérifier la présence d'une virgule suivie d'un espace, la présence de majuscules, etc.

Une approche programmatique particulièrement efficace et souvent pertinente est d'offrir, à la saisie, le choix parmi la liste (triée, évidemment) des valeurs déjà entrées pour l'attribut. Ainsi, au moment de saisir le compositeur associé à une piste, le système nous montrerait la liste triée alphabétiquement de tous les identificateurs de compositeur préalablement saisis. Si, par exemple, le compositeur que l'on doit saisir est "Jean-Sébastien Bach" et qu'on voit "Bach, J.-S." dans la liste, on réutilisera la forme déjà présente dans la liste sans la ressaisir, ce qui contribuera à augmenter la cohérence des données. Même si le compositeur recherché ne se retrouve pas dans les données déjà saisies, l'affichage de celles-ci fournit des exemples de la "stylistique" à adopter pour rédiger la nouvelle valeur.

Bien que, dans un tel cas, on s'approche des fonctionnalités d'une relation avec clé étrangère, il est important de mesurer toute la différence qui existe, malgré tout, entre ces deux types de solution.

### Une solution possible

Voici une solution possible, sous la forme d'une DTD XML commentée.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!ELEMENT discothèque (disque*)>
<!ELEMENT disque (id, infoGénérale, pistes)>
<!ELEMENT pistes (piste+)>
<!ELEMENT infoGénérale (genre, artistes?, durée?, notes?)>
<!ELEMENT id (titre, sousTitre?, label, numéro, collection?, noSéq?)>
<!ELEMENT artistes (compositeur | (exécutant, compositeur?))>
<!ELEMENT piste (titrePiste, durée, artistes?, notes?)>

<!ELEMENT titre (#PCDATA)>
<!ELEMENT sousTitre (#PCDATA | sép)*>
<!ELEMENT label (#PCDATA)>
<!ELEMENT numéro (#PCDATA)>
<!ELEMENT collection (#PCDATA)>
<!ELEMENT noSéq (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT compositeur (#PCDATA)>
<!ELEMENT exécutant (#PCDATA)>
<!ELEMENT titrePiste (#PCDATA | sép)*>
<!ELEMENT durée (#PCDATA)>
<!ELEMENT notes (#PCDATA)>
<!ELEMENT sép EMPTY>
```

**Explications:**

Dans l'analyse documentaire, il fut remarqué que certains titres de pistes étaient complexes, par exemple:

```
II Sanctus
a) Benedictus
b) Osanna
```

Cependant, le tout correspond à une seule piste sur le disque. Il s'agit d'un cas très rare, que nous solutionnons par le compromis structurel suivant: l'élément `titrePiste` peut contenir non seulement du texte, mais également des occurrences de l'élément vide `sép`, qui sert exclusivement de "séparateur" pour imposer une certaine structure. Ainsi, l'exemple ci-dessus serait représenté comme:

```
<titrePiste>Sanctus<sép/>a) Benedictus<sép/>b) Osanna</titrePiste>
```

Minimalement, une application de traitement devra remplacer l'élément vide `<sép/>` par un espace; éventuellement, le traitement pourrait être plus sophistiqué (saut de ligne, etc.).

Nous utilisons la même technique pour le sous-titre du disque (non pas parce que le besoin est criant, mais parce que l'adoption de la même technique que pour les titres de piste ne rajoute aucune complexité à la solution).

L'association avec les artistes (exécutant et/ou compositeur) peut se faire au niveau du disque et/ou de la piste. Ici, toute latitude est laissée aux applications de saisie/importation. La

sémantique est la suivante: toute association au niveau du disque est en force pour toutes les pistes du disque SAUF si une autre association est donnée au niveau de la piste, dans quel cas cette dernière association SUPPLANTE l'association existant au niveau du disque.

Il faut dire un mot de la fonction copier-coller. On a dit qu'elle devait être supportée au niveau de la piste. Avec notre solution, elle est facilement supportée à ce niveau, mais pas nécessairement en une seule invocation. En effet, si l'association avec les artistes est donnée au niveau du disque et que l'on veut copier une seule piste (qui ne possède pas d'association "locale") en préservant cette association, alors il faudra d'une part copier la piste, et ensuite, remonter au niveau du disque et copier l'élément `artistes` à ce niveau et le coller dans la nouvelle piste. Cette opération est directement automatisable, car c'est exactement le même élément XML (`artistes`) qui est utilisé pour contenir l'information, que ce soit au niveau du disque ou de la piste. Par contre, même sans automatisation explicite de la fonction dans une application de saisie/importation, celle-ci demeure facilement réalisable, comme nous venons de l'expliquer. Si elle avait entraîné des manipulations fastidieuses pour réaliser le copier-coller au niveau de la piste, nous n'aurions pas retenu cette possibilité d'association à deux niveaux. Comme elle n'entraîne aucune manipulation fastidieuse, cette solution nous semble acceptable et nous l'adoptons.

Une mention d'artistes peut comporter un compositeur, un exécutant ou les deux (et doit comporter au moins un des deux). Si les deux sont donnés, l'exécutant doit être donné en premier. Une solution "facile" et probablement convenable dans bien des cas, pour les applications n'utilisant qu'un seul nom d'artiste (par exemple, le lecteur CD de Windows), est de prendre le contenu du premier sous-élément de `artistes` comme nom d'artiste. S'il y a un exécutant, c'est lui qui sera pris; sinon, ce sera le nom du compositeur.

Une durée globale peut être spécifiée au niveau du disque, pour les cas où une telle durée *diffère* du total des durées des pistes (parce qu'elle inclut les silences entre les pistes).

L'élément `notes` permet d'inscrire toute information jugée pertinente, à la saisie, et qui ne trouve pas de place naturelle dans les autres éléments de la DTD. Il peut être associé à un disque ou à une piste.

L'élément `id` permet d'identifier un disque par ses titre et sous-titre, et de le relier à un label, une collection et/ou un coffret multi-disque. Il peut contenir les sous-éléments `titre`, `sousTitre`, `label`, `numéro`, `collection`, `noSéq`. Le numéro devrait identifier l'unité achetable (coffret ou disque) à l'intérieur du label. La collection ne sera présente que si l'unité achetable fait partie d'une collection à l'intérieur du label. Le `noSéq` (numéro séquentiel) sera utilisé, lorsque approprié, pour identifier la position séquentielle d'un disque à l'intérieur d'un coffret. Illustrons ces possibilités par trois cas de figure:

- Le disque est simple (il ne s'agit pas d'un coffret multi-disque); il n'appartient pas à une collection. Le titre est "Spring on the G string", il n'y a pas de sous-titre, le label est Astrée et le numéro du disque est 12083. On codera cette information comme suit:

  ```
  <id><titre>Spring on the G string</titre>
  <label>Astrée</label><numéro>12083</numéro></id>
  ```

- Il s'agit d'un coffret de deux disques, label Virgin, numéro de coffret 102103. Le titre global du coffret est "Best of ABBA". Le second disque (représenté ici) s'intitule "The golden years: 1975-1980":

```
<id><titre>Best of ABBA</titre>
<sousTitre>The golden years: 1975-1980<sousTitre>
<label>Virgin</label><numéro>102103</numéro><noSéq>2</noSéq></id>
```

- Le disque est le deuxième d'un coffret de 3 disques, lui-même faisant partie d'une collection intitulée "Great keyboard music of all times". Le titre du coffret est "Orlando Gibbons: a pionneer", le label est Erato, le numéro de coffret 90230. Le disque lui-même s'intitule "Pianoforte performances" et porte le sous-titre "The historical Glenn Gould performances":

```
<id><titre>Orlando Gibbons: a pionneer</titre>
<sousTitre>Pianoforte performances<sép/>The historical Glenn Gould
performances</sousTitre>
<label>Erato</label><numéro>90230</numéro><collection>Great keyboard music
of all times</collection><noSéq>2</noSéq></id>
```

Le "genre" musical est associé à un disque complet (et non à un piste). La description des genres devrait se limiter à des catégories extrêmement générales (par exemple: classique, jazz, western, rock, etc.). Pour éviter l'éparpillement, il y aurait intérêt à adopter une liste fermée de genres, comme on en retrouve dans les boutiques de disques. Le fait de permettre (et d'obliger) un et un seul genre par disque peut paraître un manque de flexibilité, mais présente l'avantage de forcer une classification très générale. Vouloir être très précis pour les genres musicaux serait une pure perte de temps, étant donné le caractère très subjectif et flou de telles classifications. *A priori*, il n'y a pas d'objection à laisser le contenu de `genre` vide (même si l'élément lui-même est obligatoire), ce qui sémantiquement, devrait s'interpréter comme "inclassable" ou "hétérogène".

Rappelons finalement que le problème du contrôle des identificateurs pour les éléments `compositeur`, `exécutant`, `label`, `genre` et `collection` devra faire l'objet de mesures adéquates, tel qu'expliqué ci-dessus.

Voici un exemple complet (et valide) de discothèque comprenant trois disques, dont les deux derniers font partie d'un même coffret. On suppose que la DTD présentée précédemment est stockée dans un fichier nommé `disco.dtd`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE discothèque SYSTEM "disco.dtd">
<discothèque>

 <disque>
  <id>
   <titre>Requiem pour les rois de France</titre>
   <sousTitre>Musique pour les funérailles du roi Henri IV</sousTitre>
   <label>Astrée</label>
   <numéro>87093</numéro>
  </id>
  <infoGénérale>
   <genre>classique</genre>
   <artistes>
    <exécutant>Ensemble Doulce mémoire</exécutant>
    <compositeur>du Caurroy, Eustache</compositeur>
   </artistes>
  </infoGénérale>
  <pistes>
   <piste>
    <titrePiste>Kyrie Eleison</titrePiste>
    <durée>33:40</durée>
   </piste>
   <piste>
    <titrePiste>Agnus Dei</titrePiste>
    <durée>24:53</durée>
   </piste>
  </pistes>
 </disque>

 <disque>
  <id>
   <titre>The best of Youssou N'dour</titre>
   <sousTitre>Set</sousTitre>
   <label>Virgin</label>
   <numéro>12083</numéro>
   <collection>World Music</collection>
   <noSéq>1</noSéq>
  </id>
  <infoGénérale>
   <genre>populaire</genre>
   <artistes>
    <exécutant>Youssou N'dour et l'Étoile de Dakar</exécutant>
   </artistes>
   <notes>
    Ce disque est aussi publié séparément comme Virgin 88075.
   </notes>
  </infoGénérale>
  <pistes>
   <piste>
    <titrePiste>Set</titrePiste>
    <durée>12:40</durée>
   </piste>
   <piste>
    <titrePiste>Toxical wastes</titrePiste>
    <durée>7:53</durée>
   </piste>
   <piste>
    <titrePiste>Please come back</titrePiste>
    <durée>17:32</durée>
   </piste>
  </pistes>
 </disque>
```

```
 <disque>
  <id>
   <titre>The best of Youssou N'dour</titre>
   <sousTitre>Sweet Dakar Suite</sousTitre>
   <label>Virgin</label>
   <numéro>12083</numéro>
   <collection>World Music</collection>
   <noSéq>2</noSéq>
  </id>
  <infoGénérale>
   <genre>populaire</genre>
   <artistes>
    <exécutant>Youssou N'dour et l'Étoile de Dakar</exécutant>
   </artistes>
   <notes>
    Ce disque est aussi publié séparément comme Virgin 89013.
   </notes>
  </infoGénérale>
  <pistes>
   <piste>
    <titrePiste>It's about time</titrePiste>
    <durée>21:04</durée>
   </piste>
   <piste>
    <titrePiste>Seven seconds away</titrePiste>
    <durée>0:07</durée>
   </piste>
   <piste>
    <titrePiste>M'bogo djerish bah</titrePiste>
    <durée>23:17</durée>
   </piste>
  </pistes>
 </disque>

</discothèque>
```