XML

1. Syntaxe de XML

XML (eXtended Markup Language) est comme HTML un langage à balises. Dans le cours consacré à HTML , nous avons vu qu'une balise est un marqueur utilisé pour délimiter une portion de texte. Par exemple les balises <u> et </u> ci-dessous délimitent la portion de texte vous dans la phrase Je vous souhaite une bonne année :

```
Je vous souhaite <u>une bonne année</u>.
```

Une page HTML est destinée à être affichée dans la fenêtre d'un navigateur : une balise HTML définit soit un composant graphique (<input>, <textarea>), soit une mise en page (<blockquote>,
), soit un paragraphe qui contient du texte (<p>, ,), soit un style de texte (<u>, <h3>).

Une balise XML est moins éclectique car elle structure des données. Par exemple la balise <espece> suivante structure des informations sur le dauphin :

```
<espece>
<nom latin="Tursiops truncatus">Dauphin</nom>
<mesures>
  <description>Collectées en captivité</description>
  <variable>Taille</variable>
  <valeur>3 m</valeur>
  <variable>Temps plongée</variable>
  <valeur>1 h</valeur>
</mesures>
<habitat>
  <description>Aquatique</description>
  <biotope>Océan</biotope>
  <biotope>Estuaire</biotope>
  <region>Pacifique</region>
  <region>Atlantique</region>
</habitat>

<nourriture proportion="80">
  <type>poisson</type>
</nourriture>
<nourriture proportion="20" preference="haute">
  <type>seiche</type>
</nourriture>
</espece>
```

Vous remarquez le nom des balises est libre en xml c'est vous qui les définissez. Toutes les balises ci-dessus sont des balises doubles <x>...</x>. Cependant les données du dauphin pourraient être structurées différemment de sorte que, par exemple, les balises qui définissent la nourriture du dauphin soient des balises simples <nourriture/> :

```
<nourriture type="poisson" proportion="80"/>
<nourriture type="seiche"  proportion="20" preference="haute"/>
```

XML, et d'ailleurs aussi HTML, ont les exigences syntaxiques suivantes :

- Les valeurs d'attribut d'une balise sont placées entre "".
- Une balise simple ou double est toujours fermée.
- Deux balises doubles ne s'entrelacent pas.

Le code XML ci-dessous est syntaxiquement incorrect :

```
<espece>
        <nom latin=Tursiops truncatus>Dauphin</nom>
        <mesures>
                <description>Collectées en captivité</description>
                <variable>Taille</variable>
                <valeur>3 m
        </mesures>
                </valeur>
        <nourriture type="seiche" proportion="20" preference="haute">
</espece>
```

Pour les raisons suivantes :

1) La balise <nom> a un attribut latin de valeur non comprise entre "".

2) Les balises <valeur> et <nourriture> ne sont pas fermées.

3) Les balises <mesures> et <valeur> sont entrelacées.

2. Processeur XML

Nous plaçons maintenant le code XML initial de la balise <espece> dans un fichier dauphin.xml qui peut dès lors être chargé dans un processeur XML tel qu'un navigateur. Lorsque l'utilisateur donne à un processeur XML un document XML à traiter, ce document doit préciser :

- Qu'il est un document XML et en quelle version de XML il est rédigé.
- En quel jeu de caractères il est rédigé (encodage).
- Qu'il se suffit ou non à lui-même, et donc s'il est ou non lié à un autre document XML.

L'en-tête ci-dessous, placé au début du fichier dauphin.xml, suffit à le compléter :

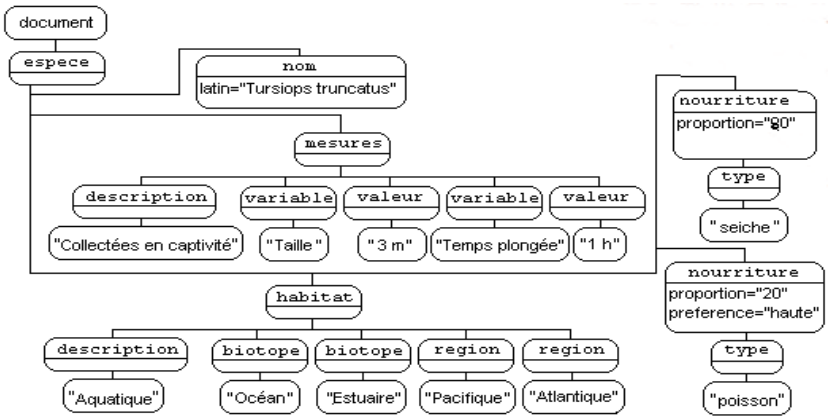
```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

Si nous chargeons maintenant dauphin.xml dans un navigateur, comme aucune feuille de style ne lui est liée, il apparaît sous une forme brute qui met en évidence la structure de données :

```
−<espece>
    <nom latin="Tursiops truncatus">Dauphin</nom>
  −<mesures>
      <description>Collectées en captivité</description>
      <variable>Taille</variable>
      <valeur>3 m</valeur>
      <variable>Temps plongée</variable>
      <valeur>1 h</valeur>
    </mesures>
  −<habitat>
      <description>Aquatique</description>
      <biotope>Océan</biotope>
      <biotope>Estuaire</biotope>
      <region>Pacifique</region>
      <region>Atlantique</region>
    </habitat>
  −<nourriture proportion="80">
      <type>poisson</type>
    </nourriture>
  −<nourriture proportion="20" preference="haute">
      <type>seiche</type>
    </nourriture>
  </espece>
```

3. Arborescence XML

En interne, le navigateur traduit une page XML en un arbre (une arborescence) dans lequel chaque balise devient un nœud balise et chaque donnée un nœud texte. Comme chaque nœud appartient à l'arbre, on dit aussi qu'il est un élément de l'arbre. La page dauphin.xml produit l'arborescence suivante :



La racine de l'arborescence est un élément XML qui représente la page XML elle-même. Un chemin est un parcours de l'arborescence depuis un élément jusqu'à un ou plusieurs autres, et qui progresse de père en fils. Par exemple espece/habitat/biotope est un chemin. Un chemin absolu part de la racine, qui est notée /. Par exemple, les deux chemins suivants sont absolus :

/espece/mesures/variable
/espece/mesures/valeur

Un chemin peut être considéré comme la notation d'un ensemble d'éléments :

- Les deux chemins ci-dessus notent respectivement les ensembles d'éléments <variable> et <valeur> contenus dans l'élément <mesures> (deux éléments chacun).
- Les chemins /espece/mesures/representation et /espece/mesures/representation notent chacun un ensemble auquel appartient un élément <representation>, mais non le même.

Grammaire XML

La structuration d’une page XML est à la charge du concepteur de la page et son écriture (son remplissage avec des données) à la charge de son rédacteur. Les mêmes informations peuvent évidemment être structurées de plusieurs manières, ainsi dans le fichier dauphin.xml :

- Les différentes nourritures pourraient être rassemblées sous une même balise :

```
<nourritures>
  <nourriture type="poisson" proportion="80"/>
  <nourriture type="seiche"  proportion="20" preference="haute"/>
</nourritures>
```

- <valeur> devrait être une sous-balise de <variable> ou un attribut de <variable> :

```
<variable nom="Taille">
  <valeur>3 m</valeur>
</variable>
```

ou :

```
<variable nom="Taille" valeur="3 m"/>
```

- Les balises <region> et <biotope> pourraient être groupées dans des balises <regions> et <biotopes>.

Toute structure est toujours discutable, ainsi celle du fichier dauphin.xml en vaut bien une autre. Cependant elle n’est pour l’instant qu’implicite : rien n’empêche son rédacteur de placer une balise <region> dans une balise <mesures> et donc au mauvais endroit. Il faut que le concepteur définisse formellement cette structure, qui d’ailleurs sera aussi celle d’autres fichiers XML contenant la même nature d’information, tels que lion.xml, loriot.xml ou encore varan.xml…

Exercice 1. Structuration d'informations 1

XML permet de structurer une information. Il est donc nécessaire, avant d'envisager d'utiliser ce format, de se familiariser avec cette structuration.

Le paragraphe suivant contient de l'information "en vrac". Réorganisez-la de manière à mettre en évidence sa structure logique, sans forcément passer par une mise en forme XML.

Une bouteille d'eau Cristaline de 150 cl contient par litre 71 mg d'ions positifs calcium, et 5,5 mg d'ions positifs magnésium. On y trouve également des ions négatifs comme des chlorures à 20 mg par litre et des nitrates avec 1 mg par litre. Elle est recueillie à **St-Cyr la Source**, dans le département du Loiret. Son

code barre est 3274080005003 et son pH est de 7,45. Comme la bouteille est sale, quelques autres matériaux comme du fer s'y trouvent en suspension. Une seconde bouteille d'eau Cristaline a été, elle, recueillie à la source d'**Aurèle** dans les Alpes Maritimes. La concentration en ions calcium est de 98 mg/l, et en ions magnésium de 4 mg/l. Il y a 3,6 mg/l d'ions chlorure et 2 mg/l de nitrates, pour un pH de 7,4. Le code barre de cette bouteille de 50 cl est 3268840001008. Une bouteille de même contenance est de marque Volvic, et a été puisée à... **Volvic**, bien connu pour ses sources donnant un pH neutre de 7. Elle comprend 11,5 mg/l d'ions calcium, 8,0 mg/l d'ions magnésium, 13,5 mg/l d'ions chlorures et 6,3 mg/l d'ions nitrates. Elle contient également des particules de silice. Son code barre est 3057640117008.

PS : Volvic est dans le Puy-de-Dôme...

Exercice 2. Structuration d'informations 2

Cet exercice est du même type que l'exercice précédent. Il s'agit de structurer, sous la forme d'un fichier XML, le texte suivant :

Il existe diverses variétés de nuages. La plupart de ceux dont nous allons parler ne produit aucun "hydrométéore", sauf le cumulonimbus, qui est accompagné d'averses (parfois sous la forme de neige, de grésil ou de grêle).

L'altocumulus et le cirrocumulus partagent les mêmes "espèces" : lenticularis, stratiformis, castellanus et flocus. On retrouve ces deux espèces également chez le cirrus, ainsi que les espèces spissatus, uncinus et fibratus. Les espèces stratiformis, lenticularis et castellanus sont quant à elles partagées également avec les strato-cumulus.

Ces derniers peuvent se traîner au ras du sol et monter à 2000m, mais certains nuages ont une altitude minimale à peine plus élevée, puisqu'elle n'est que de 200m pour les cumulus, et de 300m pour les cumulonimbus. Il est vrai que ces derniers compensent en montant jusqu'à une altitude maximale de 18000m, soit plus haut encore que les cirrus, qui plafonnent à 12000m. L'altitude minimale de ces derniers coïncide avec la fin de la présence possible des altocumulus, à 6000m. Et c'est autour de cette zone, entre 5000 et 7000m, que se trouvent les cirrocumulus. L'altitude minimale des altocumulus est de 2000m, soit quatre fois moins que l'altitude maximale des cumulus.

Ces pauvres cumulus ne sont pas favorisés en nom d'espèces, puisqu'ils se trouvent affligés de noms tels que fractus, mediocris, humilis et congestus... alors que les cumulonimbus ont des espèces aux noms plus... capillaires tels que calvus, capillatus. Les très gros cumulonimbus sont appelés mammatus.

1. Grammaire

La structure d'une feuille XML est formellement déclarée au moyen d'une grammaire, le document DTD auquel il est lié. Dans cette grammaire, les balises de la feuille XML sont décrites au moyen de la balise <!ELEMENT> comme simples (vides donc EMPTY) ou doubles (qui possèdent un contenu composé d'un texte (#PCDATA) ou d'autres balises) :

```
<!-- Les balises <nom>, <variable>, <valeur>, <biotope>, <region>,
    <type>, <description> sont des balises doubles qui contiennent
    du texte -->
<!ELEMENT nom (#PCDATA)>
<!ELEMENT variable (#PCDATA)>
<!ELEMENT valeur (#PCDATA)>
<!ELEMENT biotope (#PCDATA)>
<!ELEMENT region (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT description (#PCDATA)>

<!-- Une nourriture a un type -->
<!ELEMENT nourriture (type)>
```

Si un élément père peut contenir 0, 1 ou plusieurs fois un élément fils, l'élément fils doit être quantifié par le symbole * :

```
<!-- Une espèce animale = un nom, des mesures, un habitat et une liste
    de nourritures -->
<!ELEMENT espece (nom, mesures, habitat, nourriture*)>

<!-- Un habitat est une description suivie des biotopes que l'espèce
    animale habite et des régions de son aire de répartition
<!ELEMENT habitat (description, biotope*, region*)>
```

Si un élément père contient au moins une fois un élément fils, l'élément fils doit être quantifié par le symbole + :

```
<!-- Les mesures contiennent au moins une variable et sa valeur -->
<!ELEMENT mesures (description, (variable, valeur)+)>
```

Chaque attribut d'un élément doit être déclaré au moyen d'une balise <! ATTLIST> comme obligatoire (REQUIRED) ou optionnel (IMPLIED) :

```
<!ATTLIST nom       latin CDATA #REQUIRED>
<!ATTLIST nourriture proportion CDATA #IMPLIED>
<!ATTLIST nourriture preference CDATA #IMPLIED>
```

La grammaire peut être incluse dans la page XML, mais elle est plus généralement stockée dans un fichier indépendant .dtd. Si nous plaçons la grammaire ci-dessus dans le fichier espece.dtd, alors dauphin.xml mais aussi lion.xml, loriot.xml... pourront l'importer au moyen d'une balise <!DOCTYPE>. L'en-tête du fichier dauphin.xml devient :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>

<!-- La grammaire espece.dtd s'applique à la balise <espece> -->
<!DOCTYPE espece SYSTEM "espece.dtd">
```

Au moyen d’un valideur XML, nous pouvons maintenant vérifier si le fichier XML dauphin.xml est conforme à sa grammaire. Vous trouverez, par exemple un valideur XML disponible sur http://validator.w3.org/.

2. Erreur grammaticale

Un document XML est dit bien formé lorsqu’il respecte la syntaxe XML (à chaque balise de début correspond une balise de fin, les valeurs des attributs sont entre " ", deux balises doubles ne s’entrelacent pas...). Il est dit valide lorsqu’il respecte sa grammaire. Ci-dessous nous modifions le fichier dauphin.xml pour le rendre invalide :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE espece SYSTEM "espece.dtd">
<espece>
<nom>Dauphin</nom>
<donnees>
  <donnee>Taille</donnee>
  <valeur>3 m</valeur>
</donnees>
<habitat>
  <description>Aquatique</description>
  <region>Pacifique</region>
  <region>Atlantique</region>
  <biotope>Estuaire</biotope>
  <biotope>Océan</biotope>
</habitat>
</espece>
```

Lors de l’analyse du fichier dauphin.xml par un valideur XML, les erreurs suivantes sont décelées :

Message d’erreur	Signification
Contenu de l’élément <donnees> incomplet	Il manque un élément <description>.
Contenu de l’élément <habitat> invalide : attente de <biotope> après <description>	Les biotopes doivent suivre la description et non pas les régions.
L’attribut obligatoire ’latin’ est manquant	Le nom latin est un attribut obligatoire du nom vernaculaire de l’espèce.

3. Opérateur aléatoire

Pour que les éléments fils d’un élément puissent être disposés dans un ordre quelconque, par exemple pour qu’indifféremment d’abord les biotopes ou les régions puissent prendre place à l’intérieur de la balise <habitat>, notre grammaire emploie l’opérateur | :

```
<!ELEMENT habitat(description, (biotope* | region*))>
```

Ou encore pour que les biotopes et les régions soient indifféremment mêlés :

```
<!ELEMENT habitat(description, (biotope | region)*)>
```

Si dans la grammaire espece.dtd la règle ci-dessus est donnée à <habitat>, la balise suivante est valide :

```
<habitat>
  <description>Aquatique</description>
  <region>Pacifique</region>
  <biotope>Estuaire</biotope>
  <biotope>Océan</biotope>
  <region>Atlantique</region>
</habitat>
```

Espaces de noms

1. Définition

Soit la grammaire contenue dans le fichier œuvres.dtd :

```
<!ELEMENT oeuvres (oeuvre*)>
<!ELEMENT oeuvre (titre_oeuvre, auteur_oeuvre, partie+)>
<!ELEMENT partie (numero, titre, duree)>
```

Nous créons ci-dessous une page XML œuvres.xml conforme à cette grammaire :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE oeuvres SYSTEM "oeuvres.dtd">
<oeuvres>
  <oeuvre>
    <titre_oeuvre>Les quatre saisons</titre_oeuvre>
    <auteur_oeuvre>Vivaldi</auteur_oeuvre>
    <partie><numero>1</numero>
          <titre>Le printemps</titre><duree>9:53</duree></partie>
    <partie><numero>2</numero>
          <titre>L'été</titre><duree>7:43</duree></partie>
    <partie><numero>3</numero>
          <titre>L'automne</titre><duree>5:55</duree></partie>
    <partie><numero>4</numero>
          <titre>L'hiver</titre><duree>8:10</duree></partie>
  </oeuvre>
  <oeuvre>
    <titre_oeuvre>La tétralogie</titre_oeuvre>
    <auteur_oeuvre>Wagner</auteur_oeuvre>
    <partie><numero>1</numero>
          <titre>L'or du Rhin</titre><duree>2:00:00</duree></partie>
    <partie><numero>2</numero>
          <titre>La Valkyrie</titre><duree>3:00:00</duree></partie>
    <partie><numero>3</numero>
          <titre>Siegfried</titre><duree>4:00:00</duree></partie>
    <partie><numero>4</numero><titre>Le crépuscule des dieux</titre>
          <duree>5:00:00</duree></partie>
  </oeuvre>
```

</oeuvres>

L’ensemble des noms portés par les balises du document œuvres.xml et aussi œuvres.dtd est :

{ oeuvres, oeuvre, titre_oeuvre, auteur_oeuvre, partie, numero, titre, duree }

Cet ensemble de noms est l’espace de noms du document. Le nom « compositeur » n’appartient pas à cet ensemble, donc une balise <compositeur> n’a pas sa place dans œuvres.xml.

2. Espaces de noms Internet

Il existe des espaces de noms partagés par une multitude de documents et non propres à un ou quelques documents. Par exemple, l’espace de nom HTML :

html : { a, b, body, br, div, head, p, table, td, th, tr ... }

Ces espaces de noms ont sur Internet des identifiants uniques, qui sont des URL. Ainsi l’espace de noms HTML 4.0 est identifié par l’URL www.w3.org/TR/REC-html40

Exercice : Écriture d'une DTD avec éléments

Rédiger une DTD pour une bibliographie. Cette bibliographie :

- contient des livres et des articles ;
- les informations nécessaires pour un livre sont :
    - son titre général ;
    - les noms des auteurs ;
    - ses tomes et pour chaque tome, leur nombre de pages ;
    - des informations générales sur son édition comme par exemple le nom de l'éditeur, le lieu d'édition, le lieu d'impression, son numéro ISBN ;
- les informations nécessaires pour un article sont :
    - son titre ;
    - les noms des auteurs ;
    - ses références de publication : nom du journal, numéro des pages, année de publication et numéro du journal
- on réservera aussi un champ optionnel pour un avis personnel.

Tester cette DTD avec un fichier XML que l'on écrira ex-nihilo et validera.

Exercice : Écriture d'une DTD avec attributs

Modifier la DTD précédente...

- ... en ajoutant un attribut optionnel soustitre à l'élément titre ;
- ... en faisant de l'élément tome un élément vide et en lui ajoutant un attribut requis nb_pages et un attribut optionnel soustitre ;
- ... en faisant de l'élément nom_journal un attribut de l'élément journal et en lui donnant comme valeur par défaut *Feuille de Chou* ;
- ... en faisant de l'élément annee un attribut de type énuméré, prenant comme valeurs possibles 2000, 2001, 2002, "avant_2000" et "inconnue" et proposant comme valeur par défaut inconnue.

Utiliser cette DTD pour créer un fichier XML valide.

Feuille de style XSL

1. Définition

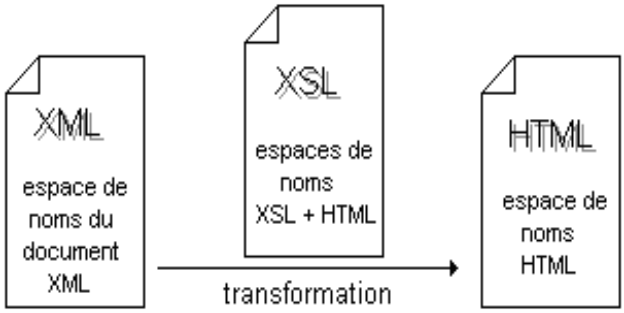
Une feuille de style XSL (eXtensible Stylesheet Language), à la différence de sa cousine CSS qui s'appose sur un document HTML pour modifier sa présentation, s'appose sur un document XML pour effectuer sa transformation en un document qui se réfère à un autre espace de noms, habituellement HTML. Une feuille XSL est elle-même un document XML qui emploie des balises appartenant à l'espace de noms XSL, identifié par l'URL

www.w3.org/TR/WD-xsl :

xls : { template, apply-templates, value-of select, value-of ... }

La balise <stylesheet> ci-dessous, par ses attributs xmlns, indique à une page XSL qu'elle contient des balises de deux espaces de noms : l'espace de noms XSL et l'espace de nom de transformation (dans l'exemple HTML) :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
                xmlns="http://www.w3.org/TR/REC-html40">
```



Après son en-tête, une feuille XSL comporte des balises <template>. Un template, dans son attribut match contient la notation d'un ensemble d'élément et dans son corps du code HTML souvent incrusté de balises XSL. Nous complétons la feuille XSL entamée ci-dessus avec un template dont l'attribut match contient la notation "/" de l'ensemble constitué du seul élément racine :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
            xmlns="http://www.w3.org/TR/REC-html40">

<xsl:template match="/">
<html>
<body>
  Aucune donnée n'a été traitée.
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Nous remarquons que, conformément à la balise <stylesheet>, les balises XSL de la feuille sont préfixées de xsl cependant que les balises HTML ne sont pas préfixées.

2. Attachement d'une feuille XSL

Nous sauvegardons la feuille XSL ci-dessus dans le fichier oeuvres.xsl et nous l'attachons à la page œuvres.xml qui devient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE oeuvres SYSTEM "oeuvres.dtd">
<?xml-stylesheet type="text/xsl" href="oeuvres.xsl"?>
<oeuvres>
...
</oeuvres>
```

Lorsqu'un processeur XML reçoit un document XML auquel une feuille XSL est attachée, il comprend qu'il doit afficher dans sa fenêtre non pas le document XML tel qu'il est mais le résultat HTML de sa transformation par la feuille XSL. Alors, il applique le template <template match="/"> sur la racine du document puis il récupère en retour le code HTML que lui fournit le corps du template.

Dans notre exemple le code est le suivant :

```
<html><body>Aucune donnée n’a été traitée.</body></html>
```

3. Application des templates

Le template <template match="/"> de la page œuvres.xls retourne du code HTML figé et ne se préoccupe nullement des données du document XML. Si maintenant ce template contient une balise <apply-templates/>, comme ci-après :

```
<xsl:template match="/">
<html>
<body>
  <xsl:apply-templates/>
</body>
</html>
</xsl:template>
```

Alors pour chaque élément fils de l’ensemble d’éléments auquel le template <template match="/"> s’applique (donc pour chaque fils de la racine), le processeur XML cherche parmi tous les autres templates de la feuille (que nous allons définir) celui qui peut s’appliquer sur ce fils pour obtenir le code HTML propre à cet élément fils.

4. La feuille de style XSL œuvres.xsl

Au moyen de la feuille oeuvres.xsl et à partir du document œuvres.xml nous voudrions produire le document HTML suivant :

Les quatre saisons de Vivaldi La tét

Partie	Titre	Duree
1	Le printemps	6:21
2	L'été	4:32
3	L'automne	5:15
4	L'hiver	6:10

Partie	
1	L'
2	Le
3	Si
4	Le

L’ensemble des éléments enfants de la racine est le seul élément <oeuvres>. Pour le traiter, nous définissons le template suivant :

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

Ce template est exécuté par le processeur depuis le template <template match="/"> : il traite donc les éléments de l’ensemble noté par /oeuvres. Il se contente de demander l’application d’un template (qui reste à définir) sur les fils des éléments de son ensemble, donc sur l’ensemble noté par

/oeuvres/oeuvre. Le template <template match="oeuvre"> s'applique à cet ensemble, et donc à chaque oeuvre : il retourne un calque dans lequel il écrit le titre et l'auteur de l'œuvre, qui sont deux éléments fils de l'oeuvre. Pour obtenir le texte de ces éléments, il utilise la balise XSL <value-of select> :

```
<xsl:template match="oeuvre">
<div style="font-family:Tahoma; font-size:16pt;" >
  <!-- extrait les textes des fils <titre_oeuvre> et <auteur_oeuvre>
      d'un élément <oeuvre> -->
  <xsl:value-of select="titre_oeuvre"/> de
  <xsl:value-of select="auteur_oeuvre"/>
</div>
</xsl:template>
```

Ce template doit aussi demander l'application du template qui traite les parties d'une œuvre. Il est complété comme ci-dessous :

```
<xsl:template match="oeuvre">
<div style="font-family:Tahoma; font-size:16pt;" >
  <xsl:value-of select="titre_oeuvre"/> de
  <xsl:value-of select="auteur_oeuvre"/>
</div>
<table border="1">
  <tr><th>Partie</th><th>Titre</th><th>Duree</th></tr>
  <xsl:apply-templates/>
</table>
</xsl:template>
```

Le template <template match="partie"> traite les éléments de l'ensemble noté par /oeuvres/oeuvre/partie (donc chaque partie d'une œuvre) :

```
<xsl:template match="partie">
<tr>
  <td><xsl:value-of select="numero" /></td>
  <td><xsl:value-of select="titre" /></td>
  <td><xsl:value-of select="duree" /></td>
</tr>
</xsl:template>
```

Voici notre feuille œuvres.xsl achevée ! Remarquons enfin que, plutôt que de placer des chemins relatifs dans les attributs match des templates, nous aurions pu placer des chemins absolus :

```
<xsl:template match="/oeuvres">
...
</xsl:template>
<xsl:template match="/oeuvres/oeuvre">
...
</xsl:template>
<xsl:template match="/oeuvres/oeuvre/partie">
...
</xsl:template>
```

Feuille de style XSL avancée

1. Fonction text() d’un élément

Nous concevons maintenant une feuille XSL espece.xls qui puisse s’appliquer à un fichier XML respectant la grammaire espece.dtd développée ci-dessus. Pour l’instant, nous voulons seulement que cette feuille extraie la description de l’habitat et des mesures. Nous choisissons de référencer l’espace de nom XSL http://www.w3.org/1999/ XSL/Transform car, outre les noms XSL, il définit aussi les expressions numériques et une balise <if> :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            xmlns="http://www.w3.org/TR/REC-html40">
<xsl:template match="/">
  <html><body>
  <xsl:apply-templates/>
  </body></html>
</xsl:template>

<xsl:template match="espece">
   <xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

Nous pourrions proposer les templates <template match="mesures"> et <template match= "habitat"> ci-dessous :

```
<xsl:template match="mesures">
   <b>Mesures:</b>
   <div>Description: <i><xsl:value-of select="description"/></i></div>
</xsl:template>

<xsl:template match="habitat">
   <b>Mesures:</b>
   <div>Description: <b><xsl:value-of select="description"/></b></div>
</xsl:template>
```

Mais nous voulons définir un template <xsl:template match="description"> qui traite les descriptions elles-mêmes, aussi nous proposons plutôt :

```
<xsl:template match="mesures">
   <b>Mesures:</b><xsl:apply-templates/>
</xsl:template>

<xsl:template match="habitat">
   <b>Habitat:</b><xsl:apply-templates/>
</xsl:template>

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

Le template <xsl:template match="description"> s’applique sur les éléments

des deux ensembles notés /espece/mesures/description et /espece/habitat /description.

Pour chaque élément traité par le template <template match="description"> la balise XSL <value-of select="text()"/> demande au processeur XML d'appeler sa fonction text() pour en obtenir le texte. Si nous voulons que la description d'une mesure et celle d'un habitat soient différemment traitées, nous écrivons deux templates dont les attributs match contiennent des chemins absolus :

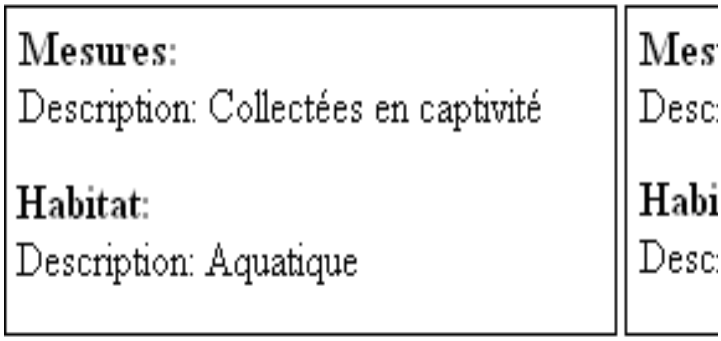
```
<xsl:template match="/espece/habitat/description">
  <div>Description: <i><xsl:value-of select="text()"/></i> </div>
</xsl:template>

<xsl:template match="/espece/mesures/description">
  <div>Description: <b><xsl:value-of select="text()"/></b> </div>
</xsl:template>
```

Maintenant nous ajoutons dans dauphin.xml la ligne d'en-tête :

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

Puis nous chargeons dauphin.xml dans un processeur XML, successivement avec les deux versions de espece.xls. Nous pouvons alors comparer les deux résultats :



2. Application sélective des template

Pour que le template <template match="habitat"> demande que soient traités les éléments fils <biotope> puis les éléments fils <region> des éléments de son ensemble (constitué du seul élément <habitat>), nous le modifions et nous ajoutons deux nouveaux templates <template match="biotope"> et <template match="region"> :

```
<xsl:template match="habitat">
  <b>Habitat:</b><xsl:apply-templates select="description"/>
  <ol>
    <xsl:apply-templates select="biotope"/>
  </ol>
  <ol>
    <xsl:apply-templates select="region"/>
  </ol>
</xsl:template>
```

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

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

Nous obtenons le résultat suivant :

Mesures:
Description: **Collectées en captivité**
Habitat:
Description: *Aquatique*

1. Océan
2. Estuaire

1. Pacifique
2. Atlantique

3. Balise for-each et attribut select

Le template <template match="habitat"> pourrait aussi parcourir explicitement les éléments <biotope> puis les éléments <region> :

```
<xsl:template match="habitat">
  <b>Habitat:</b><xsl:apply-templates select="description"/>
  <ol>
   <xsl:for-each select="biotope">
     <li><xsl:value-of select="text()"/></li>
   </xsl:for-each>
  </ol>
  <ol>
   <xsl:for-each select="region">
     <li><xsl:value-of select="text()"/></li>
   </xsl:for-each>
  </ol>
</xsl:template>
```

Les deux boucles <xsl:for-each >...</xsl:for-each> traitent un à un les éléments des deux sous-ensembles constitués par les fils <biotope> et <region> de l'élément <habitat>. Pour chacun d'entre eux, ces boucles produisent du code HTML : Texte du biotope ou Texte de la région. Les templates <template match="biotope"> et <template match="region"> ne sont dès lors plus nécessaires.

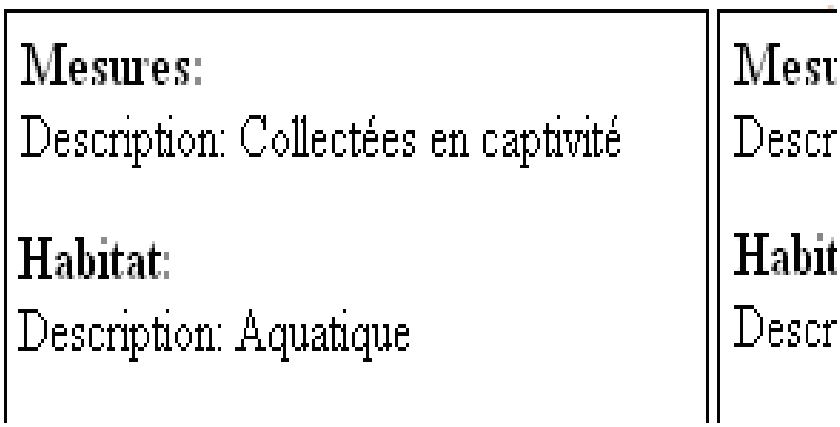
4. Lecture des attributs

Nous modifions maintenant le template <template match="espece"> de la feuille espece.xsl pour qu'il s'applique aussi aussi aux éléments <nourriture> et nous ajoutons le template <template match="nourriture"> :

```
<xsl:template match="espece">
  <table>
    <tr>
     <td><xsl:apply-templates select="mesures"/></td>
     <td><xsl:apply-templates select="habitat"/></td>
     <td><table border="1">
          <xsl:apply-templates select="nourriture"/>
        </table>
     </td>
    </tr>
  </table>
</xsl:template>

<xsl:template match="nourriture">
  <tr>
    <td><xsl:value-of select="type"/></td>
    <td><xsl:value-of select="@proportion"/></td>
  </tr>
</xsl:template>
```

Pour chaque élément <nourriture> auquel il s'applique, le template <template match="nourriture"> retourne un code HTML qui contient la valeur de son attribut proportion ainsi que le texte de son élément fils <type>. Le résultat est maintenant :



5. Balise if et attribut test

Dans un template, il est possible d'employer une balise XSL <if> pour filtrer les éléments. Par exemple, si nous ne nous intéressons qu'aux nourritures dont la proportion est de plus de 70, le template <template match="nourriture"> devient :

```
<xsl:template match="nourriture">
  <xsl:if test="@proportion&gt;=70">
    <tr>
      <td><xsl:value-of select="type"/></td>
      <td><xsl:value-of select="@proportion"/></td>
    </tr>
  </xsl:if>
</xsl:template>
```

Pour éviter toute confusion au processeur XML, les opérateurs de test < et >

doivent être remplacés par < et >.

Nous remarquons maintenant que plutôt qu'écrire ce template, nous aurions pu traiter les nourritures dans le template <template match="espece"> tel que montré ci-dessous :

```
<xsl:template match="espece">
  <table>
    <tr>
     <td><xsl:apply-templates select="mesures"/></td>
     <td><xsl:apply-templates select="habitat"/></td>
     <td><table border="1">
          <xsl:for-each select="nourriture">
           <xsl:if test="@proportion&amp;gt;=70">
            <tr>
              <td><xsl:value-of select="type"/></td>
              <td><xsl:value-of select="@proportion"/></td>
            </tr>
           </xsl:if>
          </xsl:for-each>
         </table>
     </td>
    </tr>
  </table>
```

Pour finaliser la feuille espece.xsl, nous traitons maintenant les éléments <variable> et <valeur> dans le template <template match="mesures"> :

```
<xsl:template match="mesures">
  <b>Mesures:</b><xsl:apply-templates select="description"/>
  <table border="1">
    <tr>
      <td>
        <xsl:for-each select="variable">
          <div><xsl:value-of select="text()"/></div>
        </xsl:for-each>
      </td>
      <td>
        <xsl:for-each select="valeur">
          <div><xsl:value-of select="text()"/></div>
        </xsl:for-each>
      </td>
    </tr>
  </table>
</xsl:template>
```

Le résultat final est le suivant :