G. Hunault
Licence 2 Informatique

Angers, mai 2018

Technologie XML

1. Structuration en XHTML

Ecrire le fragment de texte XHTML 1.0 valide au sens de la grammaire *Strict* qui correspond aux indications suivantes. L'élément X1 a pour attribut X2 de valeur A1. Il contient trois éléments auto-fermants nommés X3, X4 et X5. X3 a pour attribut X6 de valeur A2, X4 a pour attribut X7 de valeur A3 et X5 a pour attribut X8 de valeur A4.

Pour mémoire, les noms des éléments et des attributs doivent être écrits en minuscules, mais pas leur valeur.

Si on remplace X1 par div, X2 par class, X3, X4 et X5 par input, X6, X7 et X8 par name, A1 par visible, A2 par modeZ, A3 par modeC et A4 par modeG, quel est le code XHTML correspondant ?

2. Compréhension du vocabulaire

On veut définir, dans une DTD interne, une entité paramètre interne nommée lang, obligatoire, qui peut prendre uniquement les valeurs C, C++, R ou Python avec Python comme valeur par défaut.

Donner la ou les lignes de texte **XML** correspondant à cette définition.

3. Ecriture de DTD externe

Le site `https://www.uniprot.org` fournit un accès à des bases de données de ressources pour des protèines. C'est un site majeur de référence pour la communauté bioinformatique. Nous avons extrait de ce site des informations pour deux protéines, nommées respectivement A3AHG5 et Q96273.

Une protéine est définie par un ou plusieurs numéros d'accession (au moins un), un nom unique. De plus elle est associée à un organisme, dont le nom peut être courant ou scientifique.

Voici un exemple de fichier XML comportant ces deux protéines, nommé deuxproteines.xml :

```
<uniprotMinimal>

  <protein>
      <accession>A3AHG5</accession>
      <accession>K4F957</accession>
      <accession>P83196</accession>
      <accession>Q10M52</accession>
      <name>LEA17_ORYSJ</name>
      <organism>
       <name type="scientific">Oryza sativa subsp. japonica</name>
       <name type="common">Rice</name>
      </organism>
  </protein>

  <protein>
      <accession>Q96273</accession>
      <name>LEA18_ARATH</name>
      <organism>
         <name type="scientific">Arabidopsis thaliana</name>
         <name type="common">Mouse-ear cress</name>
      </organism>
  </protein>

</uniprotMinimal>
```

Donner pour ce fichier deuxproteines.xml une grammaire DTD « minimale et raisonnable » qui permet de décrire les protéines.

4. Un peu de XPATH *via* XMLLINT

Donner les instructions de xmllint exécutées en mode *shell* qui permettent de répondre aux questions suivantes que l'on se pose sur un fichier de protéines structuré comme dans la section précédente.

Attention : il ne s'agit évidemment pas du fichier deuxproteines.xml précédent mais d'un fichier structuré de la même façon. On ne doit donc pas fournir les réponses aux questions mais donner le code xmllint qui fournirait les réponses.

- quel est le nom de la première protéine du fichier ?

- combien y a-t-il de protéines en tout décrites dans le fichier ?

- combien de protéines ont plus d'un numéro d'accession ?

- quel est le nom scientifique de la ou des protéines dont le nom commun d'organisme est Rice ?

- quel est le nom scientifique de la ou des protéines dont le nom commun d'organisme contient le mot cotton ?

5. Transformations XSL (1)

Donner le contenu d'un fichier XSL qui effectue une transformation du fichier deuxproteines.xml de la question 3 et qui produit le fichier XML suivant.

On essaiera de respecter le formatage des lignes, mais ce n'est pas grave s'il manque des espaces ou s'il y en a en trop.

```
<?xml version="1.0"?>
<listeProtéines>
   <protéine numéro="1">LEA17_ORYSJ</protéine>
   <protéine numéro="2">LEA18_ARATH</protéine>
</listeProtéines>
```

Si vous n'arrivez pas à tout reproduire, ce n'est pas grave. Essayez de produire ce que vous pouvez. Il vaut mieux fourir un code XSL plus court mais correct et fonctionnel qu'un code long mais incorrect.

6. Transformations XSL (2)

Donner le contenu d'un fichier XSL qui effectue une transformation du fichier deuxproteines.xml de la question 3 et qui produit le fichier XHTML suivant. On pourra utiliser les *templates* et autres sous-programmes de stdWeb2.xsl vus en cours et en TP.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" [...]
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr"     [...]
<head>
<meta http-equiv="Content-type" content="text/html;      [...]
<link rel="stylesheet" type="text/css" href="std.css"    [...]
<title>uniprot</title>
</head>
<body class="beige_jpg">
<blockquote>
<table>
   <tr>
     <th>Nom protéine</th><th>Nombre de numéro d'accession</th>
   </tr>
   <tr><td>LEA17_ORYSJ</td><td>4</td></tr>
   <tr><td>LEA18_ARATH</td><td>1</td></tr>
</table>
</blockquote>
</body>
</html>
```

7. Discussion

Essayez de répondre à la question suivante :

> *Peut-on se passer des transformations XSL quand on sait bien programmer l'analyse de fichiers-texte, disons en C, Perl, Pyhton ou PHP ?*

Votre réponse devra mettre en évidence votre culture naissante ainsi que votre recul et votre esprit de synthèse en matière de modélisation et de traitement d'informations structurées.

Votre réponse devra faire 10 lignes au minimum, sans limite de maximum. On utilisera au moins 3 mots de 4 syllabes ou plus pour transmettre « un contenu rédactionnel fort ».

ESQUISSE DE SOLUTION

1. Structuration en XHTML

Voici la structuration formelle du document. Attention, le nom des éléments et des attributs doit être en minuscule (mais pas leur contenu).

```
<x1 x2="A1">
    <x3 x6="A2" />
    <x4 x7="A3" />
    <x5 x8="A4" />
</x1>
```

On obtient alors, en XHTML :

```
<div class='visible'>
  <input name='modeZ'  />
  <input name='modeC'  />
  <input name='modeG'  />
</div>
```

2. Compréhension du vocabulaire

La ligne demandée, à intégrer à une DTD, est :

```
<!ENTITY % lang "(C|C++|R|Python) 'Python' ">
```

3. Ecriture de DTD externe

Une grammaire DTD, concise, cohérente et minimale pour l'exemple fourni doit ressembler à :

```
<!ELEMENT uniprotMinimal (protein)+                    >
<!ELEMENT protein        (accession+,name,organism)    >
<!ELEMENT accession      (#PCDATA)                     >
<!ELEMENT organism       (name)+                       >
<!ELEMENT name           (#PCDATA)                     >

<!ATTLIST name type NMTOKEN #IMPLIED>

<!-- sans plus d'information, ceci est aussi acceptable : -->

<!ATTLIST name type CDATA #IMPLIED>
```

Il est difficile ici de bien prendre en compte les éléments `<name>` car ils sont de deux natures différentes. Une meilleure structuration serait d'utiliser des éléments `<proteinName>` et `<organismName>`.

4. Un peu de XPATH

Voici les expressions XPATH demandées.

```
ls //protein[1]/name

xpath count(//protein)

xpath count(//protein//accession[2])

ls //protein/organism[./name/@type="common"
     and ./name="Rice"]/name[@type="scientific"]

ls //protein/organism/name[@type="common"
     and contains(.,"Rice")]/../name[@type="scientific"]
```

5. Transformations XSL (1)

Voici une transformation possible : à partir de la racine, on produit un élément **listeProtéines** et on déclenche une règle sur tous les éléments **protein**. Pour chacun de ces éléments, la règle crée un élément **protéine** avec le bon contenu et un attribut **numéro**.

Les éléments **xsl :text** utilisés servent juste à la mise en forme.

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

<!-- règle principale -->

<xsl:template match="/">

<listeProtéines>&#10;
  <xsl:apply-templates select="//protein" />
<xsl:text>
</xsl:text>
</listeProtéines>&#10;

</xsl:template>

<!-- règle pour les itms de liste -->

<xsl:template match="protein">

<xsl:text>
   </xsl:text>
<xsl:element name="protéine">
<xsl:attribute name="numéro">
<xsl:value-of select="position()" />
</xsl:attribute>
<xsl:value-of select="name" />
</xsl:element>

</xsl:template>

</xsl:stylesheet>
```

6. Transformations XSL (2)

Voici une transformation possible qui profite des règles de `stdWeb2.xsl` :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="stdWeb2.xsl" />
<xsl:output method="xml" />

<xsl:template match="/">

<xsl:call-template name="debutPage">
  <xsl:with-param name="leTitre">uniprot</xsl:with-param>
</xsl:call-template>

<table>
  <tr><th>Nom protéine</th><th>Nombre de numéro d'accession</th></tr>
  <xsl:apply-templates select=".//protein" />
  <xsl:text>
</xsl:text>
</table><xsl:text>
</xsl:text>

<xsl:call-template name="finPage"   />
</xsl:template>

<xsl:template match="protein">
<xsl:text>
   </xsl:text>
<tr>
   <td><xsl:value-of select="name" /></td>
   <td><xsl:value-of select="count(.//accession)" /></td>
</tr>
</xsl:template>

</xsl:stylesheet>
```