RAPPORT PROJET XML

Réalisé par :

FARAH OMAR Abdoulfatah
EL GHARNAJI Yasir

Master 1 Informatique de Marseille

Département Informatique et Interactions
Faculté des Sciences

Année Universitaire : 2017 - 2018

Sommaire

Sommaire	**2**
MAKEFILE	**3**
XML	**4**
DTD	**5**
XSD	**6**
XSL	**7**
XQUERY	**8**
JAVA	**10**
TIDY	**11**

1. MAKEFILE

Notre makefile se compose de la façon suivante :

dtd	Vérifier la validité du document XML à l'aide de la DTD avec la commande **xmllint --valid**.
NoEnt	Permet de formatter notre document xml en y enlevant les entités en utilisant toujours la commande **xmllint --valid** avec l'option **--noent**.
xsd	Vérifier la validité du document XML à l'aide du Schéma avec la commande **xmllint --valid --schema** .
web	Permet de créer le répertoire www dans lequel toutes les pages HTML se trouvent en vérifiant si il n'existe pas (**mkdir -p**). Permet d'exécuter la commande **xsltproc** et de générer le site WEB dans le répertoire www (www/index.html est la page d'accueil) .
tidyXHTML	Permet de vérifier la validité des pages web générées en exécutant la commande **tidy**. Exécute un script en Bash, qui permet d'appliquer la commande tidy à tous les fichiers .html du répertoire www.
xqReq	Permet d'exécuter la requête XQuery qui produit une page XHTML (dans le repertoire www) qui liste les enseignants avec, pour chacun, les enseignements qu'il assure et les parcours dans lesquels il intervient
javaNomUE	Exécuter un code Java qui va lire le fichier master.xml et en extraire les noms des UE. Le résultat est donné sur la sortie standard sous la forme d'un document XML **(nommé nomUE.xml)** codé en mémoire (arbre DOM) et sérialisé.
all	Execute dans cet ordre : dtd -> NoEnt -> xsd -> web -> tidyXHTML -> xqReq -> javaNomUE

2.XML

Nous avons eu la structure de XML suivante :

```
<master>
      <intervenants>
            <intervenant>
            Contient toutes les informations sur un intervenant.
            </intervenant>
      </intervenants>
      <enseignements>
           <unite>
            Contient toutes les informations sur un UE.
            </unite>
      <enseignements>
      <blocs>
            <bloc>
               Bloc contenant un ID et une liste de références vers un UE.
            </bloc>
      </blocs>
      <semestres>
            <semestre>
              Contenant un titre et une référence vers un ou des block(s).
            </semestre>
      </semestres>
      <specialites>
            <specialite>
              Contenant tous les informations de différents spécialités.
            </specialite>
      </specialites>
     <parcours>
            <parcour>
                  Contient toutes les informations de chaque parcour.
            </parcour>
      </parcours>
</master>
```

3.DTD

Organigramme simplifié de la DTD :

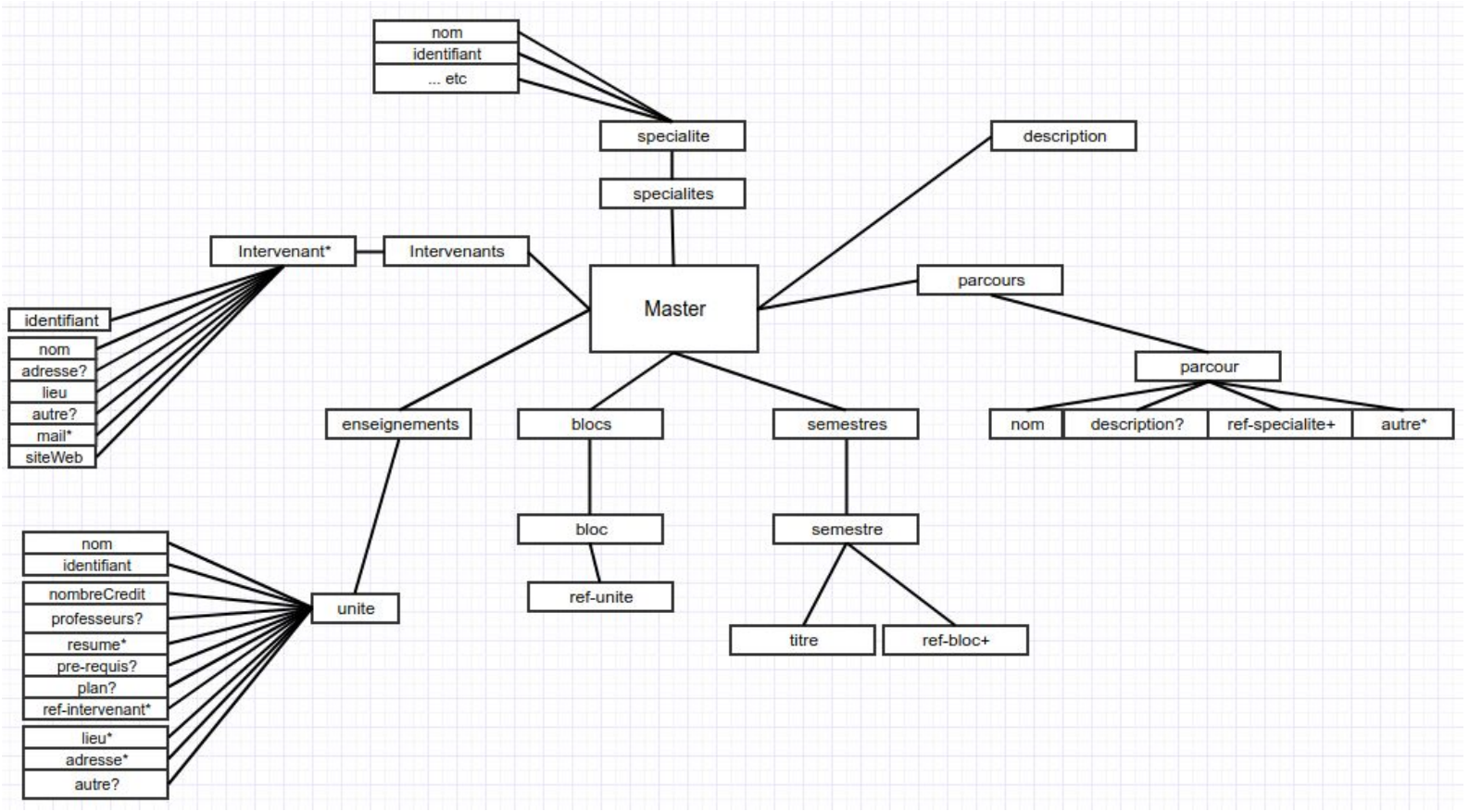

Figure 1 : Schéma de l'organigramme des éléments de la DTD

Les identifiants :

Les éléments : **Intervenant**, **Unite**, **Spécialité, Parcour**, **Semestre** et **Blocs** ont un **@id** en attribut.

4.XSD

Les indexes :

- INTERVENANT
- UNITE
- SPECIALITE
- SEMESTRE
- BLOC

Les restrictions ajoutées :

- Restriction à deux semestres dans une spécialité
- Une adresse peut contenir maximum 100 caractères et pas d'espaces.
- Un nom peut contenir de 2 à 80 caractères en prenant soin de supprimer les espaces de début de fin de chaîne.
- Un titre peut contenir de 1 à 15 caractères sans espaces.
- Les crédits doivent être des entiers positifs.
- Un email doit contenir une adresse email comme valeur (sous la forme blabla@nomdedomaine).
- Un site web doit commencer par **http://**
- Un numéro de téléphone doit matcher notre regex(Accepte les +33 et les numéro séparés d'un espace ou non)

5.XSL

Notre document XSL permet de générer notre site WEB et il se compose de la façon suivante :

`<xsl:result-document href="www/parcours/unites.html">` Permet de créer une page Web qui liste toutes les UEs.
`<xsl:result-document href="www/parcours/intervenants.html">` Permet de créer une page Web qui liste tous les Intervenants.
`<xsl:for-each select="//intervenant">` Permet de générer une page Web par intervenant.
`<xsl:for-each select="//specialite">` Permet de produire une page Web par spécialité.
`<xsl:for-each select="//unite">` Permet de créer une page Web par UE.
`<xsl:for-each select="//parcour">` Permet de procréer une page Web par parcours du Site.
`<xsl:template name="menu">` Permet de faire un menu dynamique à chaque page en prenant en argument le lien de la page courante
`<xsl:template name="genererSpecialite">` Permet de fabriquer toutes les spécialités
`<xsl:template match="ref-intervenant">` Permet de concevoir toutes les références des intervenants
`<xsl:template match="intervenant">` Permet de remplir le contenu de chaque page Intervenant.
`<xsl:template match="enseignement">` Permet de charge le contenu de chaque page Unite.
`<xsl:template name="genererSemestre">` Permet de générer les semestres .
`<xsl:template name="afficherNomEnseignement">` `Fonction permettant d'afficher le nom d'un UE en fonction de son ID.`
`<xsl:template match="specialite">` `Permet de générer la page Spécialité.`

`<xsl:template name="hyperlink">`
`Permet de remplacer automatiquement toutes les adresses « http:// »`
`par des liens cliquables`

Pour notre feuille de style XSL, on a utilisé XSLT version 2.0 et pour le tester, on a utilisé le processeur saxon avec cette commande :

alias saxon="java -jar saxon9/saxon9he.jar"
saxon -xsl:xsl/master.xsl master.xml -o:www/index.html

6.XQUERY

Exécute une requête qui affiche la liste de tous les enseignants ainsi que la liste de toutes les UEs qu'il assure et les parcours dans lesquels il intervient. N'ayant malheureusement pas trouvé la référence de tous les enseignants sur les UE et les parcours du site internet du Master de Luminy, nous avons manuellement entrées des références vers des enseignants aléatoire afin de tester notre requête en se basant sur les données-master 2017/2018.

Algorithme :

Pour chaque nom d'intervenant dans <Intervenant>
  On ordonne par intervenant
  On retourne la liste des noms d'intervenants
  Pour chaqu'une des matières dans <unité>

    On récupère la ref-intervenant de chaque UE.

    On ordonne par id
    Si l'id de l'intervenant est égal à celui de la ref-intervenant alors

      on retourne le nom de cette UE.

    Sinon
      On ne fait rien
    FinSi
  FinPour

  Pour chaqu'une des spécialités dans <specialité>

    On récupère la ref-intervenant de chaque spécialité.

    On ordonne par id
    Si l'id de l'intervenant est égal à celui de la ref-intervenant alors

      on retourne le nom de cette spécialité.

    Sinon
      On ne fait rien
    FinSi
  FinPour
FinPour

Afin d'appliquer cet algorithme, nous utilisons la technologie XPath 2.0.
Nous faisons appel à :

- Des Boucles FOR
- Des Conditions SI
- Des Déclarations de Variables locales : let $variable
- La fonction **fn:distinct-values** qui permet de ne sélectionner que des valeurs distinctes
- la fonction **fn:compare** qui permet de comparer deux à deux des valeurs contenues dans des variables
- La fonction **string** qui permet de caster le contenu d'une variable en string.

Nous utilisons des déclarations d'option afin de créer un document xHTML valide :

```
declare namespace saxon="http://saxon.sf.net/";
 declare option saxon:output "method=xml";


 declare option saxon:output "doctype-public=-//W3C//DTD XHTML 1.0 Strict//EN";


 declare option saxon:output "doctype-system=http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd";


 declare option saxon:output "omit-xml-declaration=no";

  declare option saxon:output
"indent=yes";
```

Nous n'avons pas de soucis, tout fonctionne correctement dans cette partie.

7.JAVA

L’algorithme qu’on a utilisé permet de parcourir tous les éléments du document master.xml et de retenir l'élément unité et puis il affiche le nom de cette unité . Pour cela on a ajouté des conditions pour vérifier si on est bien sur élément unité afin d’afficher son contenu .
Le résultat de cet algorithme est donné sur la sortie standard sous forme d'un document XML (nommé **nomUE.xml**) codé en mémoire (arbre DOM) avec notre fonction **buildTreeNom()** et sérialisé par notre fonction **encoding()** .

Algorithme

```
noeudUnite = false
noeudNom = false
        Tant qu'il existe des noeuds
                si noeud courant == <enseignement>  alors
                     noeudUnite = true
                si noeud courant == <nom> && noeudUnite == true alors
                        noeudNom = true
                si noeud noeudUnite == true && noeudNom == true alors

                        charger noeud dans notre arbre XML


                noeudUnite = false
                noeudNom = false
```

8. TIDY

Afin de pouvoir exécuter la commande **tidy** sur l'ensemble de nos fichiers HTML, nous avons décidé de créer un script bash qui explore chaque fichier du répertoire WWW et y applique notre commande.
Voici le code de notre Script :

```
#/!bin/bash
compt=0
for FILE in `find ../www/ -name "*.html" `;
do
    execPath=$PWD/${FILE}
    executionTidy="tidy -q --show-warnings false -utf8 $execPath"
    $executionTidy
compt=$((compt+1))
done
compt=$((compt+1))
echo 'Fichiers tidy check : ' $compt > nbFichiersTidy.txt
```