UFR de Lettres et Langues, Laboratoire Ligérien de Linguistique de Tours
3, rue des Tanneurs BP 4103
37041 TOURS CEDEX 1
06.82.26.60.00
lll@univ-tours.fr

Rapport de projet de recherche thématique au Laboratoire Ligérien de Linguistique

Réalisation d'une base de données dictionnairique

Maître d'ouvrage :
Jean-Michel FOURNIER
fournier@univ-tours.fr
Université François-Rabelais, Tours

Maître d'oeuvre :
Gwénael BOISSAY
gwenael.boissay@gmail.com

Version du 5 juin 2010

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Réalisation de la base de données

1.1 Étude des fichiers XML

Les fichiers XML étaient fournis tels quels, sans fichiers DTD ou schémas XSD, nécessaires à la description de la structure du contenu du fichier XML. J'ai donc utilisé le logiciel **Altova XMLSpy** en version d'essai de 30 jours, afin de générer (plus ou moins efficacement) les fichiers DTD et schémas XSD correspondants aux fichiers XML. Les balises ont été extraites d'après ces fichiers.
Pour faciliter le travail d'étude des fichiers (notamment en termes de lisibilité et d'extraction d'informations) au client et moi-même, j'ai réalisé un logiciel en Java, nommé **ParseurXML** *(cf chapitre sur le logiciel d'extraction d'informations des fichiers XML)*, permettant de générer un fichier texte contenant les informations des fichiers XML mais plus lisibles pour un humain.

1.1.1 EPD

Liste des balises :
- **epd** : balise racine du fichier
- **entry** : balise indiquant une nouvelle "entrée" dans le dictionnaire
- **prongrp** : balise de regroupement
- **pron** : indique une prononciation anglaise ou américaine (attribut reg={UK,US})
- **pronidx** : indique une variante de prononciation
- **inflection** : indique une inflection de l'entrée
- **second_sense** : même graphie, mais sens différent
- **stress_shift** : insistance sur la prononciation
- **compound** : balise de regroupement
- **headword** : mot du dictionnaire (caractérisant une nouvelle entrée)
- **inf** : contient le mot de l'inflection
- **cm** : contient du texte
- **comment** : un commentaire
- **usage** : l'usage particulier fait du mot
- **capvar** : variante du mot avec une majuscule
- **var** : variante graphique du mot
- **pos** : la catégorie (verbe, nom...)
- **gloss** : une note spéciale
- **i** : balise de mise en forme italique
- **b** : balise de mise en forme gras
- **rb** : balise de mise en forme italique+gras
- **sp** : balise de mise en forme exposant
- **note** : une note quelconque

Hiérarchie des balises :

parent	enfant(s)
entry	headword prongrp inflection var capvar gloss pos note
prongrp	comment pron pronidx
stress_shift	comment cm note
compound	comment cm note

parent	enfant(s)
usage	i, b, sp pron
second_sense	var capvar inf gloss pos prongrp note
inflection	inf prongrp

TAB. 1.1 – Hiérarchie des balises EPD

1.1.2 LPD

Liste des balises :

- **sup** : balise de mise en forme exposant
- **sub** : balise de mise en forme indice
- **lpdfont** : balise de mise en forme impression
- **i** : balise de mise en forme italique
- **WORD** : apparaît en principe après DESC, représente une variante du HWD
- **WARNING** : signifie "incorrect" si elle apparaît dans une PronCodes, ou signifie "unexpected for this spelling" dans une Entry ou une Derivative
- **VARPRON** : variante de prononciation anglaise
- **TEXT** : du texte quelconque
- **REFTYPE** : type de référence (ex : see also...)
- **REFHWD** : le mot sur lequel porte la référence
- **PronCodes** : balise de regroupement
- **Para** : contenu mis en forme de paragraphe à l'impression
- **PRONR** : désigne soit du contenu, soit une prononciation dans une certaine langue
- **PRON** : prononciation anglaise
- **PICCAL** : texte sans importance
- **LPD** : balise racine du document
- **LABEL** : langue spécifique
- **Head** : balise de regroupement
- **HWD** : le mot caractérisant la nouvelle entrée
- **ForeignPron** : balise de regroupement pour les prononciations dans une langue spécifique
- **Entry** : balise indiquant une nouvelle "entrée" dans le dictionnaire
- **EQUIV** : une équivalence

- **Derivative** : balise de regroupement indiquant un dérivé (ou inflexion)
- **DESC** : peut contenir une description, ou une catégorie, ou un sens, ou une langue
- **DERIV** : le mot dérivé
- **Crossref** : balise de regroupement
- **CompoundSection** : balise de regroupement
- **Compound** : indique un composé
- **COMP** : le composé en question
- **AMEVARPRON** : variante de prononciation américaine
- **AMEPRON** : prononciation américaine
- **ALPHASORT** : clé de tri pour l'impression (sans importance)

Hiérarchie des balises :

parent	enfant(s)
Entry	Head PICCAL CompoundSection Crossref ForeignPron WORD WARNING EQUIV TEXT Para DESC PronCodes Derivative
CompoundSection	Compound
Head	HWD
PronCodes	AMEPRON VARPRON AMEVARPRON PRONR PRON
ForeignPron	PronCodes LABEL WORD DESC

parent	enfant(s)
Derivative	PronCodes WORD DESC DERIV ForeignPron EQUIV WARNING
PRON/AMEPRON	TEXT VARPRON i, sup, sub
VARPRON/AMEVARPRON/PRONR	TEXT WARNING i, sup, sub
Compound	PronCodes COMP ForeignPron DESC WORD
Para	PronCodes DESC WORD Derivative
Crossref	REFTYPE REFHWD

Tab. 1.2 – Hiérarchie des balises LPD

1.1.3 Macquarie

Les balises suivantes étaient présentes dans le schéma XSD fourni mais pas dans le fichier XML : *schunk*, *rehead*, *nlabel*, *uncount*, *appendix*, *col*, *use*, *uselabel*, *utext*, *synonym*, *syn*, *syntext*, *resyn*, *opposite*, *opp*, *colloc*, *citations*, *year*, *txt*, *cit*, *auth*, *src*, *place*, *date*, *page*, *info*, *thes*, *geog*, *blockquote*, *br*, *xr*, *vtext*, *engnote*, *u*, *rtext*, *pro*, *lnk*, *slabel*, *ichunk*, *jchunk*, *mfchunk*, *screens*, *word*, *scrhead*, *edithistory*, *correction*, *hi*, *zzajc*, *oldinftype*, *oldinf*, *nplabel*.
Les balises *flags* et *sortkey* ont été jugées inutiles et enlevées physiquement du dictionnaire pour le rendre plus léger. Les balises *ozip*, *oztext*, *ozsrc*, *ozdate*, *inflection*, *inf*, <*prn type="say"*>, <*sterm lemma="{esph|espv}"*>, <*sterm lemma="si" number="pl"*> sont présentes mais ne sont pas traitées.

Liste des balises :

bigmac	cw	aka	refprefix	bn
record	headmod	name	seetext	ron
body	sterm	ip	see	cftext
chunk	abbrev	ozip	head	variants
p	symbol	oztext	cf	sup
def	ginfo	ozsrc	reftext	sub
subdef	scinfo	ozdate	ref	bynames
esp	sc	pron	var	goto
bynametype	t	entety	varprefix	usage
prn	defprefix	inf	byname	lang
inflection	subdefprefix	pos	ety	wrdety
dtext	label	runon	phrety	link
sublabel	fordef	subheadprefix	fordeftype	taboo
context	subhead	table	i	nickname
gramnote	esubhead	tr	b	td

Tab. 1.3 – Liste des balises MCQ

Hiérarchie des balises :

1.2 Choix d'un SGBD

1.2.1 Critères

Plusieurs critères sont déterminants dans le choix du Système de Gestion de Base de Données Relationnel :

- performances
- compatibilité Windows/MacOS/Linux
- disponibilité d'outils d'administration
- capacité de stockage importante
- supporter les standards SQL92 ou plus
- accessible à l'utilisateur lambda
- coût pécunier minimal

1.2.2 Systèmes étudiés

Afin de satisfaire le critère pécunier, seuls des SGBD gratuits seront étudiés.

1.2.2.1 Firebird

+ Points forts	- Points faibles
Windows, Linux, MacOSX	absence d'outils d'administration
pilote JDBC	capacité limitée

1.2.2.2 SQLiteJDBC

+ Points forts	- Points faibles
Windows, Linux, MacOSX	maturité insuffisante
pilote JDBC	capacité limitée
	absence d'outils d'administration

1.2.2.3 One$DB

+ Points forts	- Points faibles
Windows, Linux, MacOSX	absence d'outils d'administration
pilote JDBC	dernière version de 2006

1.2.2.4 H2 database

+ Points forts	- Points faibles
Windows, Linux, MacOSX	outils d'administration sommaires
mode serveur, embarqué ou les deux	

1.2.2.5 HSQLDB

+ Points forts	- Points faibles
Windows, Linux, MacOSX	outils d'administration sommaires
standards SQL92, sql server 2008	8Go maxi
utilisée dans Open Office	

1.2.2.6 Apache Derby / JavaDB

+ Points forts	- Points faibles
Windows, Linux, MacOSX	absence d'outils d'administration
facilement embarquable	absence de fonction MERGE
implémentée en Java	
empreinte faible ($\tilde{2}$.5Mo)	

1.2.2.7 PostgreSQL

+ Points forts	- Points faibles
Windows, Linux, MacOSX	occupation mémoire plus importante
pilote JDBC	moins accessible à l'utilisateur lambda
clustering, backup...	mode serveur uniquement

1.2.2.8 MySQL

+ Points forts	- Points faibles
Windows, Linux, MacOSX	mode serveur uniquement
simplicité d'utilisation	
outils d'administration PhpMyAdmin	
installation facile (XAMPP...)	

1.2.3 Choix final

Dans cette partie sont expliquées les raisons de mon choix qui s'est porté sur **MySQL**.

Deux types de SGBD pourraient répondre aux besoins : JavaDB ou Mysql.
C'est dans une perspective d'avenir que le choix est aussi déterminant, c'est pourquoi le client, M. Fournier,

m'a fait part d'une possibilité d'utilisation et d'extension de la base de données dictionnairique par d'autres laboratoires associés. Il faut donc un système qui puisse être utilisé en local sur une machine, ou à distance installé sur un serveur, et qui puisse supporter un nombre éventuellement important de connexions, et qui soit robuste en termes de performances, d'accès concurrentiels, de sauvegarde, voire de clustering (répartition sur plusieurs serveurs) dans le cas où la base deviendrait d'une taille très importante avec de multiples accès.
Or Mysql est un outil fiable, très couramment utilisé, simple d'utilisation, possédant un outil d'administration inégalable de mon point de vue à savoir Phpmyadmin, et facilement installable grâce à des outils tels que XAMPP, UwAmp, Wampserver2, LAMP, MAMP...

Conclusion : le SGBD retenu sera donc **MySQL**, l'installeur **XAMPP** et l'outil d'administration **PhpMyAdmin**.

1.3 Modélisation des tables

Une règle importante à ne pas oublier est qu'il n'existe **pas** d'ordre dans une base de données. Or, les dictionnaires sont un contenu documentaire ordonné. Aussi, pour respecter la restitution dans l'ordre des informations, des colonnes "ordre" seront utilisées dans les tables. Voir l'annexe B pour un pseudo modèle Merise de la base.

1.3.1 Tables EPD

1.3.1.1 Table des graphies

⇒ *Table contenant les graphies et leurs variantes.*

Champ	Type	Description
id_headword	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
mot	varchar(100)	graphie primaire
var1	varchar(100)	variante graphique
...	varchar(100)	variantes graphiques
var5	varchar(100)	variante graphique
note	text	utilisé pour ajouter des informations supplémentaires

TAB. 1.4 – table headword_epd

1.3.1.2 Table des flexions

⇒ *Table contenant les graphies des flexions.*

Champ	Type	Description
id_inf	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
mot	varchar(100)	graphie de la flexion
id_headword	int(11)	clé étrangère issue du id_headword de la table headword_epd ⇒ utilisée pour retrouver le mot d'origine dont est issu le dérivé

TAB. 1.5 – table flexions_epd

1.3.1.3 Table des composés

⇒ *Table contenant les composés associés à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de id_headword de la table headword_epd ⇒ récupère le contenu des balises compound associées au id_headword
texte	text	le texte associé aux balises compound
id_compose	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.6 – table composes_epd

1.3.1.4 Table des catégories d'un mot

⇒ *Table contenant les catégories associées à un mot.*

Champ	Type	Description
id_liste_categories	int(11)	clé étrangère, issue du id_categorie de la table liste_categories ⇒ utilisée pour retrouver la catégorie d'un mot
id_mot	int(11)	clé étrangère, issue de id_headword ou id_inf ⇒ l'id du mot dont on souhaite retrouver la catégorie
ordre	int(11)	l'ordre des catégories (un mot peut être d'abord un nom, puis un verbe etc...)
id_categorie	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.7 – table categories_epd

1.3.1.5 Table notes

⇒ *Table contenant une note associée à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de id_headword de la table headword_epd ⇒ récupère le texte associé au id_headword
texte	text	le texte associé aux balises note, usage ou stress_shift
id_notes	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.8 – table notes_epd

1.3.1.6 Table Sens

⇒ *Table contenant un sens associée à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de l'id_headword de la table headword_epd si l'origine=headword de l'id_inf de la table inf_epd si l'origine=derive ⇒ récupère le texte associé au id_origine
texte	text	le texte associé aux balises comment ou gloss
origine	varchar(10)	égale à headword si on a récupéré un gloss/stress_shift/comment d'un headword, ou égale à derive si on a récupéré un comment d'un dérivé (balise comment de prongrp d'une inflection)
id_sens	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.9 – table sens_epd

1.3.1.7 Tables de décomposition syllabique de prononciation

⇒ *Tables contenant la décomposition syllabique pour les prononciations des tables headword_epd et flexion_epd.*

Remarque : la colonne *dictionnaire* des 3 tables est inutile puisqu'elles sont séparées, cependant, en vue d'une éventuelle unification de ces tables, elle serait nécessaire, c'est pourquoi elle reste présente.

Champ	Type	Description
id_headword	int(11)	clé étrangère issue du id_headword de la table headword_epd ⇒ utilisée pour retrouver le mot d'origine non décomposé
ordre	int(11)	entier utilisé pour conserver l'ordre d'affichage tel que dans le fichier XML d'origine
dictionnaire	varchar(10)	indicatif du dictionnaire (par défaut : EPD)
region	varchar(50)	UK, US, AU, UknRP, Fr...
schema	varchar(20)	schéma accentuel associé à la décomposition syllabique
nb_syllabes	int(11)	le nombre de syllabes de la décomposition
type_prononciation	varchar(15)	primaire (primary), variante (var) ou variante accentuelle (varacc)
syllabe1	varchar(25)	la 1ère syllabe de la décomposition
...	varchar(25)	...
syllabe20	varchar(25)	la dernière syllabe de la décomposition
note	text	utilisé pour ajouter des informations supplémentaires
id_decomposition	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
schema_phonologique	varchar(20)	schéma accentuel phonologique associé à la décomposition syllabique
catdef	varchar(50)	utilisé lors de la fusion de 2 entrées identiques

TAB. 1.10 – table syllabesheadword_epd

Champ	Type	Description
id_inf	int(11)	clé étrangère issue du id_inf de la table flexions_epd ⇒ utilisée pour retrouver le dérivé d'origine non-décomposé
ordre	int(11)	entier utilisé pour conserver l'ordre d'affichage tel que dans le fichier XML d'origine
dictionnaire	varchar(10)	indicatif du dictionnaire (par défaut : EPD)
region	varchar(50)	UK, US, AU, UknRP, Fr...
schema	varchar(20)	schéma accentuel associé à la décomposition syllabique
nb_syllabes	int(11)	le nombre de syllabes de la décomposition
type_prononciation	varchar(15)	primaire (primary), variante (var) ou variante accentuelle (varacc)
syllabe1	varchar(25)	la 1ère syllabe de la décomposition
...	varchar(25)	...
syllabe20	varchar(25)	la dernière syllabe de la décomposition
note	text	utilisé pour ajouter des informations supplémentaires
id_decomposition	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
schema_phonologique	varchar(20)	schéma accentuel phonologique associé à la décomposition syllabique

TAB. 1.11 – table syllabesflexions_epd

1.3.2 Tables LPD

1.3.2.1 Table des graphies

⇒ *Table contenant les graphies et leurs variantes.*

Champ	Type	Description
id_headword	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
mot	varchar(100)	graphie primaire
var1	varchar(100)	variante graphique
...	varchar(100)	variantes graphiques
var5	varchar(100)	variante graphique
note	text	utilisé pour ajouter des informations supplémentaires

TAB. 1.12 – table headword_lpd

1.3.2.2 Table des flexions

⇒ *Table contenant les graphies des flexions.*

Champ	Type	Description
id_inf	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
mot	varchar(100)	graphie de la flexion
id_headword	int(11)	clé étrangère issue du id_headword de la table headword_lpd ⇒ utilisée pour retrouver le mot d'origine dont est issu le dérivé

TAB. 1.13 – table flexions_lpd

1.3.2.3 Table des composés

⇒ *Table contenant les composés associés à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de id_headword de la table headword_lpd ⇒ récupère le contenu des balises Compound associées au id_headword
texte	text	le texte associé aux balises Compound
id_compose	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.14 – table composes_lpd

1.3.2.4 Table des catégories d'un mot

⇒ *Table contenant les catégories associées à un mot.*

Champ	Type	Description
id_liste_categories	int(11)	clé étrangère, issue du id_categorie de la table liste_categories ⇒ utilisée pour retrouver la catégorie d'un mot
id_mot	int(11)	clé étrangère, issue de id_headword ou id_inf ⇒ l'id du mot dont on souhaite retrouver la catégorie
ordre	int(11)	l'ordre des catégories (un mot peut être d'abord un nom, puis un verbe etc...
flag	varchar(50)	colonne à supprimer à posteriori pour les cas spéciaux (ex : family name)
id_categorie	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.15 – table categories_lpd

1.3.2.5 Table notes

⇒ *Table contenant une note associée à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de id_headword de la table headword_lpd ⇒ récupère le texte associé au id_headword
texte	text	le texte associé aux balises EQUIV, DESC...
id_notes	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.16 – table notes_lpd

1.3.2.6 Table Sens

⇒ *Table contenant un sens associée à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de l'id_headword de la table headword_lpd si l'origine=headword de l'id_inf de la table inf_lpd si l'origine=derive ⇒ récupère le texte associé au id_origine
texte	text	le texte associé aux balises DESC
origine	varchar(10)	égale à headword si on a récupéré un DESC d'un headword, ou égale à derive si on a récupéré un DESC d'un dérivé
id_sens	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.17 – table sens_lpd

1.3.2.7 Tables de décomposition syllabique de prononciation

⇒ *Tables contenant la décomposition syllabique pour les prononciations des tables headword_lpd et flexions_lpd.*

Champ	Type	Description
id_headword	int(11)	clé étrangère issue du id_headword de la table headword_lpd ⇒ utilisée pour retrouver le mot d'origine non décomposé
ordre	int(11)	entier utilisé pour conserver l'ordre d'affichage tel que dans le fichier XML d'origine
dictionnaire	varchar(10)	indicatif du dictionnaire (par défaut : LPD)
region	varchar(50)	UK, US, AU, UknRP, Fr...
schema	varchar(20)	schéma accentuel associé à la décomposition syllabique
nb_syllabes	int(11)	le nombre de syllabes de la décomposition
type_prononciation	varchar(15)	primaire (primary), variante (var) ou variante accentuelle (varacc)
syllabe1	varchar(25)	la 1ère syllabe de la décomposition
...	varchar(25)	...
syllabe20	varchar(25)	la dernière syllabe de la décomposition
note	text	utilisé pour ajouter des informations supplémentaires
id_decomposition	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
schema_phonologique	varchar(20)	schéma accentuel phonologique associé à la décomposition syllabique
catdef	varchar(50)	utilisé lors de la fusion de 2 entrées identiques

TAB. 1.18 – table syllabesheadword_lpd

Champ	Type	Description
id_inf	int(11)	clé étrangère issue du id_inf de la table flexions_lpd ⇒ utilisée pour retrouver le dérivé d'origine non-décomposé
ordre	int(11)	entier utilisé pour conserver l'ordre d'affichage tel que dans le fichier XML d'origine
dictionnaire	varchar(10)	indicatif du dictionnaire (par défaut : LPD)
region	varchar(50)	UK, US, AU, UknRP, Fr...
schema	varchar(20)	schéma accentuel associé à la décomposition syllabique
nb_syllabes	int(11)	le nombre de syllabes de la décomposition
type_prononciation	varchar(15)	primaire (primary), variante (var) ou variante accentuelle (varacc)
syllabe1	varchar(25)	la 1ère syllabe de la décomposition
...	varchar(25)	...
syllabe20	varchar(25)	la dernière syllabe de la décomposition
note	text	utilisé pour ajouter des informations supplémentaires
id_decomposition	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
schema_phonologique	varchar(20)	schéma accentuel phonologique associé à la décomposition syllabique

TAB. 1.19 – table syllabesflexions_lpd

1.3.3 Tables MCQ

1.3.3.1 Table des graphies

⇒ *Table contenant les graphies et leurs variantes.*

Champ	Type	Description
id_headword	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
mot	varchar(100)	graphie primaire
var1	varchar(100)	variante graphique
...	varchar(100)	variantes graphiques
var5	varchar(100)	variante graphique
notes	text	utilisé pour ajouter des informations supplémentaires

TAB. 1.20 – table headword_mcq

1.3.3.2 Table des flexions

⇒ *Table contenant les graphies des flexions.*

Champ	Type	Description
id_inf	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
mot	varchar(100)	graphie de la flexion
id_headword	int(11)	clé étrangère issue du id_headword de la table headword_mcq ⇒ utilisée pour retrouver le mot d'origine dont est issu le dérivé

TAB. 1.21 – table flexions_mcq

1.3.3.3 Table de Sens

⇒ *Table contenant les indications de sens (définitions, obsolète, rare...).*

Champ	Type	Description
id_headword	int(11)	clé étrangère de la table headword_mcq
obsolete_rare	varchar(20)	{Obsolete or Rare, Rare or Obsolete, Obsolete, Rare}
label	text	le contenu de la balise LABEL
sens	text	le contenu des balises def/subdef
num_def	int(11)	le numéro de la définition (e.g. ordre)
id_cat	int(11)	clé étrangère de la table liste_categories
id_sens	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.22 – table sens_mcq

1.3.3.4 Table étymologie

⇒ *Table contenant les indications d'étymologie.*
Construite à priori, la colonne id_lien_dict contiendra d'abord l'id_headword de la table headword_mcq. A posteriori, il faudra remplacer cet id_headword par l'id_lien_dict correspondant à l'id_MCQ égal à id_headword dans liens_dict.

Champ	Type	Description
id_headword	int(11)	clé étrangère de la table headword_mcq (puis de liens_dict)
date	varchar(15)	colonne à NULL par défaut (remplie à posteriori)
langue	varchar(25)	la langue pour l'étymologie (voir balise lang)
contenu	text	le contenu associé à l'étymologie
flag	int(11)	signale un cas à examiner
id_etymo	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.23 – table etymo_mcq

1.3.3.5 Table des composés

⇒ *Table contenant les composés associés à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de id_headword de la table headword_mcq ⇒ récupère le contenu de certaines balises sterm associées au id_headword
texte	text	le texte associé aux balises Compound
id_compose	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.24 – table composes_mcq

1.3.3.6 Table des catégories d'un mot

⇒ *Table contenant les catégories associées à un mot.*

Champ	Type	Description
id_liste_categories	int(11)	clé étrangère, issue du id_categorie de la table liste_categories ⇒ utilisée pour retrouver la catégorie d'un mot
id_mot	int(11)	clé étrangère, issue de id_headword ou id_inf ⇒ l'id du mot dont on souhaite retrouver la catégorie
ordre	int(11)	l'ordre des catégories (un mot peut être d'abord un nom, puis un verbe etc...)
id_categorie	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.25 – table categories_mcq

1.3.3.7 Table notes

⇒ *Table contenant une note associée à une graphie.*

Champ	Type	Description
id_headword	int(11)	clé étrangère, issue de id_headword de la table headword_mcq ⇒ récupère le texte associé au id_headword
texte	text	le texte associé...
id_notes	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.26 – table notes_mcq

1.3.3.8 Tables de décomposition syllabique de prononciation

⇒ *Tables contenant la décomposition syllabique pour les prononciations des tables headword_mcq et flexions_mcq.*

Champ	Type	Description
id_headword	int(11)	clé étrangère issue du id_headword de la table headword_mcq ⇒ utilisée pour retrouver le mot d'origine non décomposé
ordre	int(11)	entier utilisé pour conserver l'ordre d'affichage tel que dans le fichier XML d'origine
dictionnaire	varchar(10)	indicatif du dictionnaire (par défaut : MCQ)
region	varchar(50)	UK, US, AU, UknRP, Fr...
schema	varchar(20)	schéma accentuel associé à la décomposition syllabique
nb_syllabes	int(11)	le nombre de syllabes de la décomposition
type_prononciation	varchar(15)	primaire (primary), variante (var) ou variante accentuelle (varacc)
syllabe1	varchar(25)	la 1ère syllabe de la décomposition
...	varchar(25)	...
syllabe20	varchar(25)	la dernière syllabe de la décomposition
note	text	utilisé pour ajouter des informations supplémentaires
id_decomposition	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
schema_phonologique	varchar(20)	schéma accentuel phonologique associé à la décomposition syllabique
catdef	varchar(50)	utilisé lors de la fusion de 2 entrées identiques

TAB. 1.27 – table syllabesheadword_mcq

Champ	Type	Description
id_inf	int(11)	clé étrangère issue du id_inf de la table flexions_mcq ⇒ utilisée pour retrouver le dérivé d'origine non-décomposé
ordre	int(11)	entier utilisé pour conserver l'ordre d'affichage tel que dans le fichier XML d'origine
dictionnaire	varchar(10)	indicatif du dictionnaire (par défaut : MCQ)
region	varchar(50)	UK, US, AU, UknRP, Fr...
schema	varchar(20)	schéma accentuel associé à la décomposition syllabique
nb_syllabes	int(11)	le nombre de syllabes de la décomposition
type_prononciation	varchar(15)	primaire (primary), variante (var) ou variante accentuelle (varacc)
syllabe1	varchar(25)	la 1ère syllabe de la décomposition
...	varchar(25)	...
syllabe20	varchar(25)	la dernière syllabe de la décomposition
note	text	utilisé pour ajouter des informations supplémentaires
id_decomposition	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
schema_phonologique	varchar(20)	schéma accentuel phonologique associé à la décomposition syllabique

TAB. 1.28 – table syllabesflexions_mcq

1.3.4 Tables communes

1.3.4.1 Table liste des catégories grammaticales

⇒ *Table contenant la liste des catégories grammaticales.*

Champ	Type	Description
id_categorie	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
nom	varchar(50)	nom de la catégorie (v,adv,...)

TAB. 1.29 – table liste_categories

1.3.4.2 Table de lien entre les dictionnaires

⇒ *Table contenant les liens entre les mots de chaque dictionnaire.*

Champ	Type	Description
id_lien_dict	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
id_epd	int(11)	clé étrangère id_headword de la table headword_epd
id_lpd	int(11)	clé étrangère id_headword de la table headword_lpd
id_mcq	int(11)	clé étrangère id_headword de la table headword_mcq
mot	varchar(100)	graphie primaire
var1	varchar(100)	variante graphique
...	varchar(100)	variantes graphiques
var5	varchar(100)	variante graphique
flag_lien	int(11)	indique qu'il y a un doublon au sein d'un même dictionnaire (ex : agape)
flag_cat	int(11)	indique qu'une catégorie présente dans l'un ne l'est pas dans les autres

TAB. 1.30 – table liens_dict

1.3.4.3 Table de catégories grammaticales pour un lien

⇒ *Table contenant les catégories grammaticales d'un lien entre les dictionnaires.*

Champ	Type	Description
id_lien_dict	int(11)	clé étrangère de la table liens_dict
id_liste_categories	int(11)	clé étrangère de la table liste_categories
ordre	int(11)	ordre des catégories pour un même id_lien_dict
id_catégorie	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée

TAB. 1.31 – table categories_lien

1.3.4.4 Table lexique

⇒ *Table symbolisant "vient de" (ex : displacement vient de displace). Utilisée pour symboliser le lien de composition lexical existant entre plusieurs mots (ex : préfixe+base+suffixe).*

Champ	Type	Description
id_lien_lex	int(11)	clé primaire, unique, NOT NULL, auto-incrementée
id_lien_dict_mot_etudie	int(11)	clé étrangère de la table liens_dict ⇒ le mot étudié (ex : displacement)
id_lien_dict_mot_referent1	int(11)	clé étrangère de la table liens_dict ⇒ le mot de référence (ex : displace pour un préfixe)
id_lien_dict_mot_referent2	int(11)	clé étrangère de la table liens_dict ⇒ le 2nd mot de référence (pour une base)
id_type_lexical	int(11)	type du lien lexical ⇒ {suff, pref, comp, panique, savant, rien}
note	text	utilisé pour ajouter des informations supplémentaires
id_statut	int(11)	clé étrangère de la table liste_statuts
id_construction	int(11)	clé étrangère de la table types_construction

Tab. 1.32 – table lexique

1.3.4.5 Table générale des types de construction lexicaux.

⇒ *Table contenant les types de construction pour la table lexique : juxtaposition, substitution, fusion, a, e, i, o.*

Champ	Type	Description
id_construction	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
nom	varchar(50)	{juxtaposition, substitution, fusion, a, e, i, o}

Tab. 1.33 – table types_construction

1.3.4.6 Table générale de liste des statuts

⇒ *Table contenant les statuts pour la table lexique : séparable, inséparable, discutable,savant ou rien.*

Champ	Type	Description
id_statut	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
statut	varchar(15)	{séparable, inséparable, discutable,savant, null}

Tab. 1.34 – table liste_statuts

1.3.4.7 Table générale de liste des types lexicaux

⇒ *Liste des types :suffixé, prefixé, composé, panique ou rien.*

Champ	Type	Description
id_type_lexical	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
type_lexical	varchar(15)	{suffixé, prefixé, composé, panique, null}

TAB. 1.35 – table liste_types_lexicaux

1.3.4.8 Table générale usage_lien

⇒ *Table contenant les usages faits d'un mot de la table liens_dict.*

Champ	Type	Description
id_lien_dict	int(11)	clé étrangère de la table liens_dict
id_usage	int(11)	clé étrangère de la table liste_usages

TAB. 1.36 – table usage_lien

1.3.4.9 Table générale liste des usages

⇒ *Les usages faits d'un mot : savant, général, discutable ou rien.*

Champ	Type	Description
id_usage	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
usage	varchar(15)	{savant, général, discutable, null}

TAB. 1.37 – table liste_usages

1.3.4.10 Table des types de morphologie

⇒ *Table contenant les différents types de morphologie* ∈{*prefixe, base_libre, base_libre_combinee, base_liee, base_liee_savante, base_liee_autre, suffixe_lexical, suffixe_grammatical, voyelle connective*}.

Champ	Type	Description
id_type_morph	int(11)	identifiant du type
type_morph	varchar(50)	le type de morphologie

TAB. 1.38 – table types_morphologie

1.3.4.11 Table de décomposition morphologique

⇒ *Table contenant la décomposition morphologique d'un mot.*

Champ	Type	Description
id_morph	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
id_lien_dict	int(11)	clé étrangère de la table liens_dict
m1	varchar(15)	1ère morphème ou son type_morph
...	varchar(15)	...
m20	varchar(15)	dernière morphème ou son type_morph
note	text	utilisé pour ajouter des informations supplémentaires

TAB. 1.39 – table decomposition_morphologique

1.3.4.12 Table de labellisation morphologique

⇒ *Table contenant la labellisation de la décomposition morphologique d'un mot.*
Par exemple, unassertive est décomposé en "un as sert ive" dont les éléments sont labellisés respectivement "prefixe prefixe base_liee suffixe".

Champ	Type	Description
id_label_morph	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
id_morph	int(11)	clé étrangère de la table decomposition_morphologique
m1	int(11)	1ère morphème : son type_morph
...	int(11)	...
m20	int(11)	dernière morphème : son type_morph
note	text	utilisé pour ajouter des informations supplémentaires

TAB. 1.40 – table labellisation_morphologique

1.3.4.13 Table de décomposition graphique

⇒ *Table contenant la décomposition graphique d'un mot.*

Champ	Type	Description
id_decomp_graph	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
id_lien_dict	int(11)	clé étrangère de la table liens_dict
syllabe1	varchar(10)	1ère syllabe de l'éclatement
...	varchar(10)	...
syllabe20	varchar(10)	dernière syllabe de l'éclatement
nb_syllabes	int(11)	nombre de syllabes de l'éclatement
note	text	utilisé pour ajouter des informations supplémentaires

TAB. 1.41 – table decomposition_graphique

1.3.4.14 Table de liste des allomorphes

⇒ *Table contenant la liste des allomorphes (prefixe,suffixe,base), avec éventuellement un lien de parenté (ex : col est parent à co-).*

Champ	Type	Description
id_allo	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
type	varchar(25)	le type de l'allomorphe (pref,base,suff)
contenu	varchar(50)	l'allomorphe
actif	double	l'activité en %
id_morph_parent	int(11)	clé étrangère table liste_morphemes_prefsuffbase

TAB. 1.42 – table liste_allomorphes

1.3.4.15 Table des catégories de préfixe/base

⇒ *Table contenant les catégories d'un morphème de type préfixe ou base.*

Champ	Type	Description
id_categorie_elt	int(11)	clé étrangère de la table liste_categories
type	varchar(15)	le type de morphème (préfixe,base)
id_morph_prefbase	int(11)	clé étrangère d'un morphème

TAB. 1.43 – table categories_prefbase

1.3.4.16 Table des catégories de suffixe

⇒ *Table contenant les catégories de départ et d'arrivée d'un morphème de type suffixe.*

Champ	Type	Description
id_categorie_depart	int(11)	clé étrangère de la table liste_categories
id_categorie_arrivee	int(11)	clé étrangère de la table liste_categories
id_morph_suffixe	int(11)	clé étrangère d'un morphème

TAB. 1.44 – table categories_suffixe

1.3.4.17 Table de liste des morphèmes

⇒ *Table contenant la liste des morphèmes (prefixe,suffixe,base).*

Champ	Type	Description
id_morph_prefsuffbase	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
type	varchar(15)	le type de morphème (prefixe,base,suffixe)
origine	varchar(25)	origine linguistique
contenu	varchar(50)	le morphème

TAB. 1.45 – table liste_morphemes_prefsuffbase

1.3.4.18 Table des sources de fréquence

⇒ *Table contenant la liste des sources de fréquence : COCAE, COCAE sport, BNC...*

Champ	Type	Description
id_source_freq	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
source	varchar(50)	l'intitulé de la source (ex : COCAE)

TAB. 1.46 – table source_frequence

1.3.4.19 Table des fréquences

⇒ *Table contenant la liste des sources de fréquence : COCAE, COCAE sport, BNC...*

Champ	Type	Description
id_freq	int(11)	clé primaire, unique, NOT NULL, auto-incrémentée
id_lien_dict	int(11)	clé étrangère de la table liens_dict
id_source_freq	int(11)	clé étrangère de la table source_frequence
frequence	double	la fréquence du mot (générale : COCAE, ou spéciale : COCAE sport)
note	text	utilisé pour ajouter des informations supplémentaires

TAB. 1.47 – table frequence

CHAPITRE 2

Logiciel d'extraction d'informations des fichiers XML

Comme expliqué précédemment, pour faciliter le travail d'étude des fichiers (notamment en termes de lisibilité et d'extraction d'informations) au client et moi-même, j'ai réalisé un logiciel en Java, nommé **ParseurXML**, permettant de générer un fichier texte contenant les informations des fichiers XML mais plus lisibles pour un humain, et permettant surtout de valider l'extraction des bonnes données. Ce logiciel sert donc au final à extraire les données des fichiers XML pour les convertir en fichiers CVS que l'on peut importer dans Mysql. C'est donc un outil pour le développeur.

2.1 Choix techniques

2.1.1 Parsage

Pour le parsage des fichiers XML, j'ai choisi d'utiliser l'API **JDOM**. Cette API possède un point fort qui peut aussi en faire son point faible : le document XML est parcouru et chargé en mémoire sous forme d'objet. Ainsi, lors des traitements, il est assez simple d'accéder à une balise XML ainsi qu'à son contenu et à ses attributs. Cependant, dans notre cas, nous avons des fichiers XML de taille assez importante (ex : 60Mo), ainsi il est conseillé de démarrer le programme dans Netbeans en allouant une taille de mémoire utilisable par la JVM d'au-moins 2Go.

2.1.2 Exportation en fichiers CSV

Les fichiers CSV sont des fichiers textes classiques dont les données sont séparées par un symbole tel qu'une virgule ou un point-virgule qui sont souvent employés. Ce type de fichier peut être importé dans OpenOffice Calc pour être lu sous forme de tableur.
Ici, tout symbole du type espace, virgule, point-virgule etc... est déjà utilisé dans les données (texte), j'ai donc choisi d'utiliser le caractère tabulation comme séparateur des données.
Par ailleurs j'ai découpé la création des fichiers en plusieurs parties car dans sa version actuelle (3.2) OpenOffice n'ouvre pas de fichiers CSV de plus de 65535 lignes.
Lors de l'ouverture des fichiers il faut préciser l'encodage **UTF8** et le symbole séparateur **tabulation**.

2.2 Aperçu du logiciel

Lorsque la quantité de mémoire devient trop faible, afin d'éviter une exception de du type stack overflow, l'indicateur vert passe au rouge et un message apparaît invitant l'utilisateur à relancer l'application.

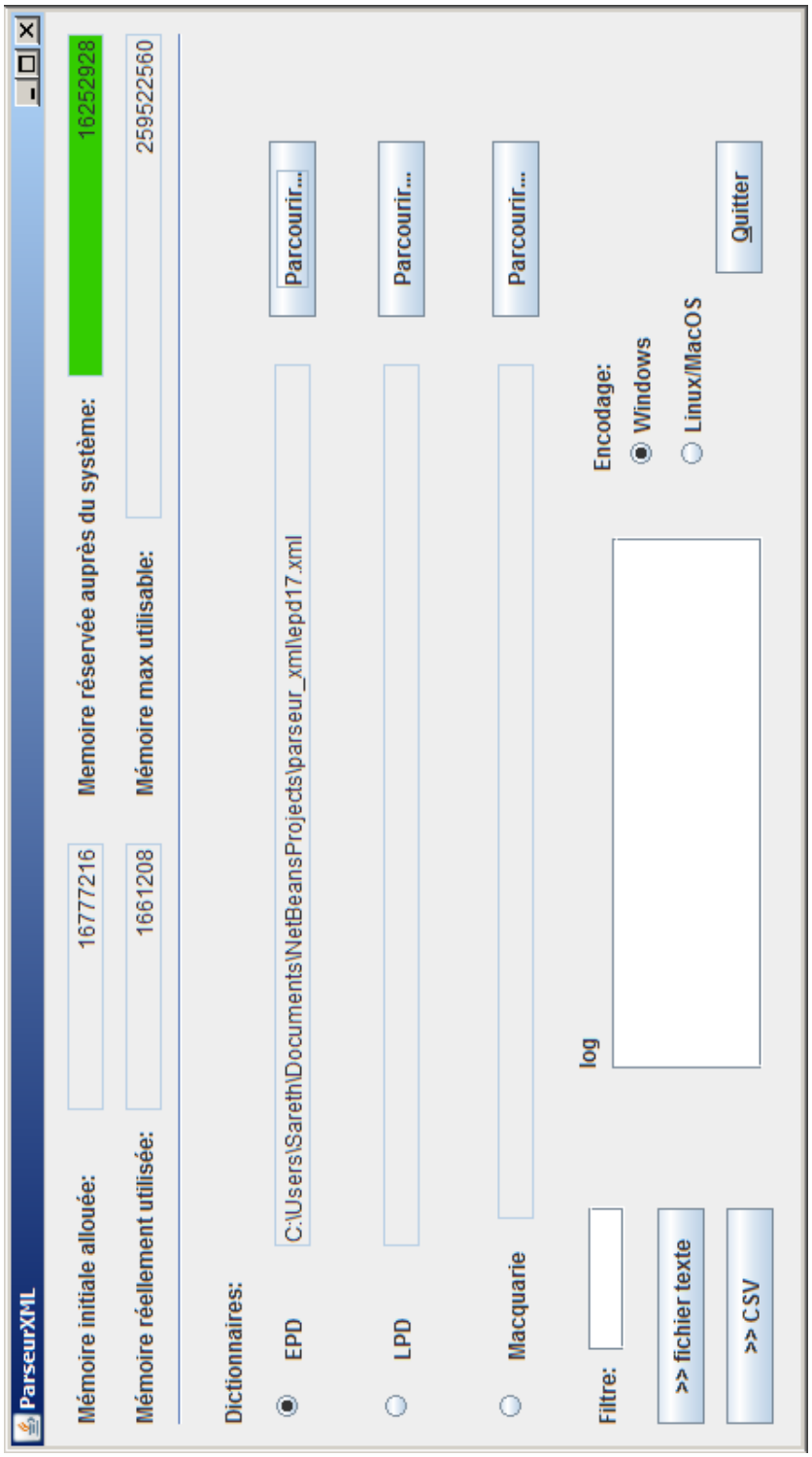


FIG. 2.1 – Aperçu de l'interface

2.3 Détails techniques

Le logiciel n'est en principe destiné qu'à n'être utilisé que par le développeur, afin de créer les fichiers CSV.

Il y a cinq packages dans le projet Netbeans :
- **app** : contient les classes associées au traitement des documents XML
- **epd**,**lpd**,**mcq** : contient les classes de "stockage" des données XML sous forme d'objets, ainsi que les classes de remplissage des fichiers CSV
- **ui** : contient les fichiers de l'interface graphique

Les classes XXXprocessor.java sont celles qui permettent d'exporter en fichier texte plus lisible les données d'un fichier XML.

Les classes XXXxmlToTable.java sont celles qui permettent d'extraire, de traiter et de sauvegarder les données issues des fichiers XML, et de les exporter dans des fichiers CSV destinés à être importés dans la base, par exemple en utilisant phpmyadmin (cf annexe A).

La méthode employée pour le parsage de EPD n'est pas la même que pour LPD ou MCQ. En effet, son schéma XSD étant assez léger, j'ai suivi scrupuleusement ce schéma pour construire la structure de parsage (ex : telle balise vient après telle autre, et tant de fois exactement).

Cependant, je n'ai pas appliqué cette méthode pour les autres dictionnaires (LPD et MCQ) car cela devenait beaucoup trop fastidieux et peu intelligent. J'ai donc opté pour une démarche plus générale, en recensant pour chaque balise, quelles étaient les balises enfants susceptibles d'apparaître, quel que soient leur nombre ou leur ordre d'apparition (à quelques exceptions près).
Ainsi, pour chaque balise, on effectue un parcours de ses éventuels enfants, et éventuels attributs, et on effectue le traitement en conséquence.

Remarque : il est conseillé de consulter l'annexe sur la marche à suivre pour ajouter un nouveau dictionnaire (démarche utilisée pour ajouter les 3 dictionnaires).

2.4 Modifications post-création

Des modifications post-création sont à appliquer à la base pour éliminer des données inutiles (ex : virgules en trop)

```
- - – trouver les 1, 2... dans sens_epd et les éliminer
DELETE FROM ‘sens_epd‘ WHERE texte REGEXP (’[0-9]’)

- - –nettoyer label=’,’ dans sens_mcq
UPDATE ‘sens_mcq‘ SET label=” WHERE label =’,’

- - – nettoyer sens vides mcq
DELETE FROM ‘sens_mcq‘ WHERE ‘obsolete_rare‘=” AND ‘label‘=” AND ‘sens‘=”
```

- - – remplacer UK par AU pour MCQ UPDATE ‘syllabesheadword_mcq‘ SET region=’AU’ WHERE region=’UK’ UPDATE ‘syllabesflexions_mcq‘ SET region=’AU’ WHERE region=’UK’

CHAPITRE 3

Logiciel de création de liens

3.1 Présentation

La table liens_dict sert à relier entre eux les dictionnaires présents dans la base (EPD, LPD, MCQ). Pour cela, il est nécessaire de créer les liens pour chaque mot, de rassembler leurs catégories provenant de chacun des dictionnaires et d'automatiser l'éclatement graphique.

Ce logiciel est lui-aussi destiné à n'être utilisé que par le développeur pour créer les fichiers CSV de remplissage de la base.

Remarques importantes :
- la base doit avoir été remplie au préalable avec les fichiers CSV du logiciel **ParseurXML**
- le caractère séparateur employé dans ces fichiers CSV n'est plus le caractère tabulation mais la virgule (cf annexe A)

Les fichiers créés sont au nombre de trois :
- liens_dict.csv (table *liens_dict*)
- categories_liens.csv (table *categories_lien*)
- decomposition_graphique.csv (table *decomposition_graphique*)

La création du premier fichier consiste à parcourir les mots de EPD et pour chaque mot, de compter le nombre d'occurrences de ce même mot dans les autres dictionnaires, et de créer le lien entre les dictionnaires possédant ce mot.
Si un mot est présent plus d'une fois dans un dictionnaire, **flag_lien** est mis à 1.
Si le dictionnaire suivant ne possède pas au-moins toutes les catégories du mot du dictionnaire en cours, alors **flag_cat** est mis à 1.

La création du second fichier consiste à parcourir les catégories du mot en cours dans chaque dictionnaire, puis à les rassembler toutes pour les affecter au lien correspondant dans *liens_dict*, en ayant unicité des catégories pour chaque mot.

La création du troisième fichier consiste à parcourir la table *liens_dict* (remplie au préalable avec le 1er fichier créé), et pour chaque lien, créer une ligne dans le fichier CSV représentant une entrée dans la table *decomposition_graphique*, avec une décomposition graphique automatisée avec des règles simples, qui fonctionnent pour la plupart des mots.

3.2 Détails techniques

Pour ce logiciel j'ai choisi d'utiliser Hibernate pour interagir avec la base de données.
Pour créer les classes entités, et les divers fichiers de configuration Hibernate, j'ai suivi le tutoriel disponible à l'adresse suivante http://netbeans.org/kb/docs/java/hibernate-java-se.html
Les packages "entity" et "util" sont donc créés et remplis à l'aide de la procédure d'automatisation de

création des fichiers décrite dans le tutoriel.

Le code écrit proprement dit se situe donc dans le package "remplissagetables", et c'est dans la fonction **main** de la classe *RemplirTables* que sont appelées les fonctions de création des fichiers CSV.

Remarque : la création des fichiers est assez longue (30 à 40 minutes).

CHAPITRE 4

Logiciel d'interaction avec la base

Il s'agit du programme le plus important et le plus utilisé par l'utilisateur de la base de données dictionnairique. C'est par ce logiciel que l'utilisateur va pouvoir consulter, créer, supprimer, mettre à jour des données. Il possède des fonctionnalités de recherche avancées, ainsi que des fonctions d'éditions avancées pour l'exportation, ou encore des modules pour des post-traitements, un éditeur SQL.

4.1 Accès aux données

Afin de pouvoir effectuer des recherches, des modifications, etc... le logiciel doit interagir avec la base de données. Le SGBD sélectionné est MySql comme précisé précédemment. Ainsi, les interactions seront des requêtes SQL.

Pour accéder aux tables de la base, je me suis d'abord orienté vers les ORM tels que Hibernate. Cependant, les requêtes SQL n'apparaissent pas en clair lorsqu'elles sont exécutée, or une des fonctionnalité étant le contrôle des actions de l'utilisateur qui modifient la base, cet outil ne donnait pas satisfaction.

J'ai donc opté pour le pattern **DAO** *(Data Access Objects)* qui permet d'associer à une table une classe possédant des attributs privés correspondants aux colonnes de la table. Puis, à cette classe, on associe un objet d'accès aux données qui permet d'effectuer les requêtes dont on a besoin (select, insert into, delete...).
Ainsi, pour chaque table, on a une classe qui représente ses colonnes (ou champs) et une classe qui permet d'effectuer des requêtes.

Les classes d'accès aux données se terminent par "DAO", et utilisent toutes une même classe qui est chargée de la connexion à la base, de l'exécution des requêtes, de la sauvegarde des requêtes de modifications (insert/update/delete...). Toutes les requêtes effectuées dans le programme passent par cette classe nommée **QueryManager**.

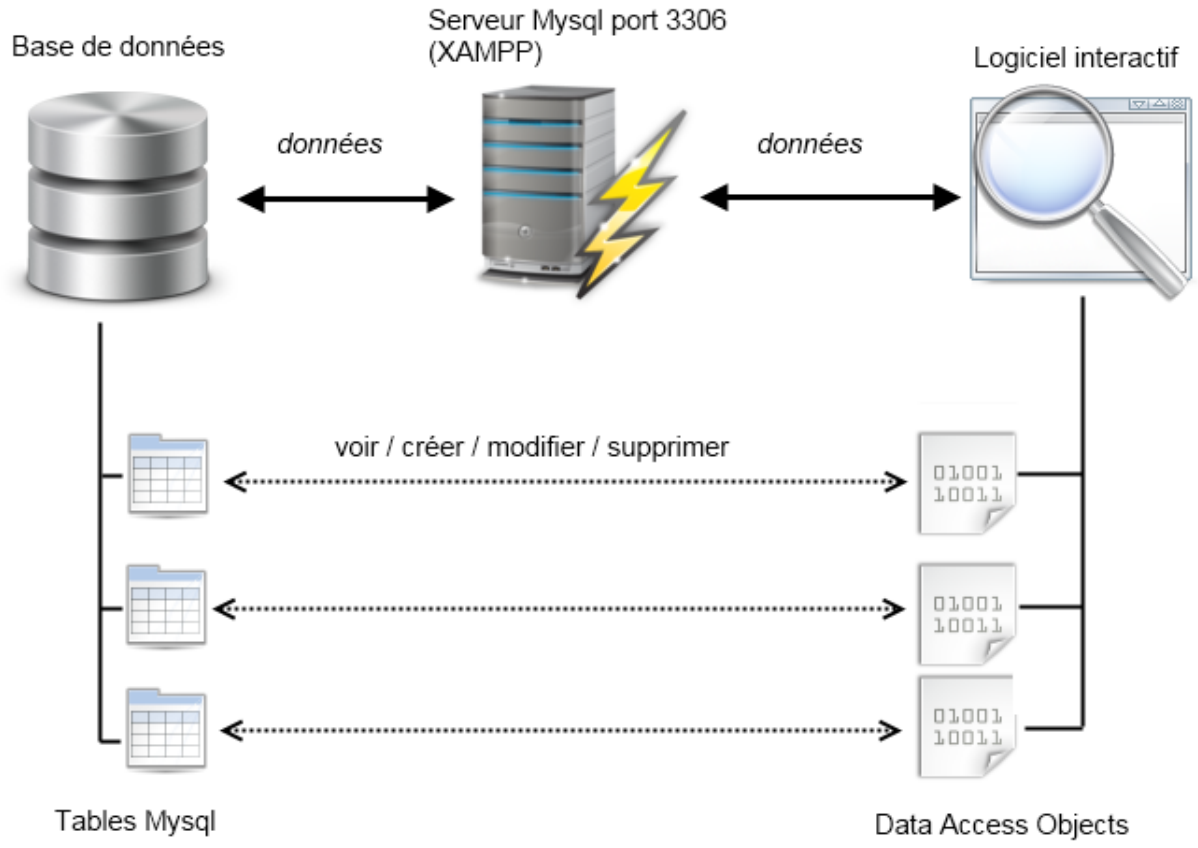


FIG. 4.1 – Modelisation du pattern DAO

4.2 Look And Feel

Le "Look And Feel" est l'apparence que va posséder une application Java. Elle peut être native au système ou encore choisie préalablement.
Je me suis rendu compte qu'utiliser le L&F natif n'était pas une bonne solution, car lorsque je dessinais une interface graphique avec l'utilitaire Matisse de Netbeans (ce dernier utilisant le L&F du système), lors de l'exécution apparaissaient des disparités de positionnement entre autres. Aussi, pour avoir une interface graphique qui soit conviviale et multiplateformes, j'ai choisi d'utiliser le L&F **Nimbus**.
Pour dessiner les interfaces avec cette apparence, il est nécessaire de configurer Netbeans pour qu'il possède lui aussi cette apparence (cf annexe).

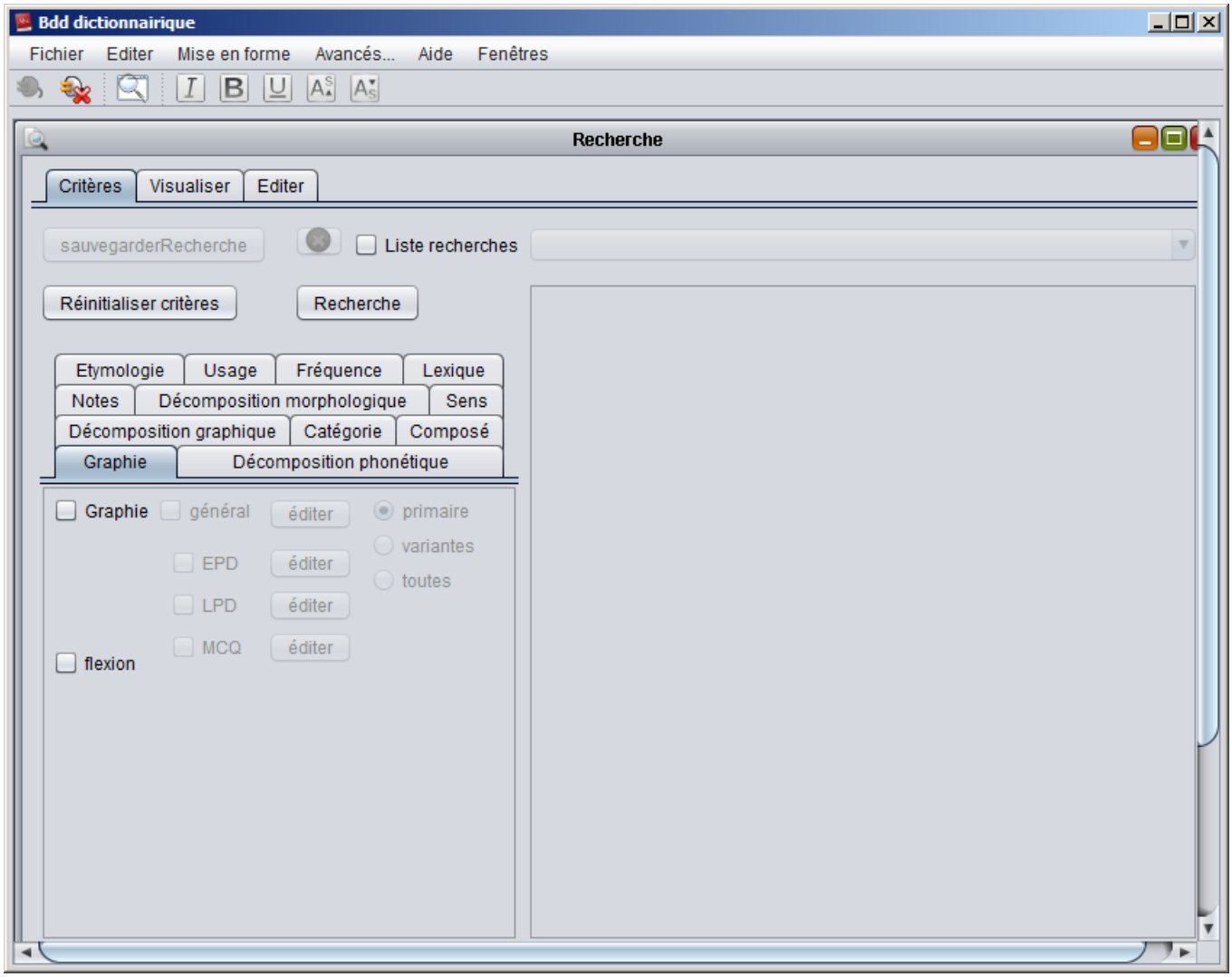


FIG. 4.2 – Logiciel avec l'apparence Nimbus

4.3 Détails techniques

4.3.1 Principe de recherche/édition multi-critères

Dans une requête SQL, on va distinguer 4 types de clauses :

1. les clauses SELECT
2. les clauses FROM
3. les clauses INNER JOIN ou LEFT JOIN
4. les clauses WHERE

Ainsi, lors de la sélection de plusieurs critères, il faut fabriquer la requête SQL. Cela nécessite d'être attentif aux noms d'alias employés pour les tables. Par exemple, la recherche va retourner des lignes de la table liens_dict systématiquement, c'est pourquoi la requête `SELECT * FROM liens_dict ld` est ajoutée en premier dans la fonction RechercheAvancee.nouvelleRecherche(), et ensuite sont ajoutées les clauses de jointures (INNER JOIN ou LEFT JOIN), et enfin les conditions WHERE, et notamment la condition supplémentaire sur les id_lien_dict lorsque l'utilisateur choisit d'effectuer une recherche sur un résultat précédemment sauvegardé.

Le principe est le même pour l'édition, sauf que les critères d'édition se "rajoutent" aux critères de recherche, ce qui complexifie la chose (en fait, la sélection pour l'édition se fait sur les id_lien_dict trouvés lors d'une recherche, sur le même principe que la recherche sur un résultat sauvegardé).
Tout comme la classe RechercheAvancee gère la construction de la requête pour les critères de recherche, c'est la classe **EditionAvancee** qui est chargée de construire la requête pour les critères d'édition. Mais cette requête est plus difficile à construire, car tous les types de clauses sont nécessaires pour la construire, et surtout il est nécessaire de donner des alias différents pour toutes les colonnes de toutes les tables renvoyées, autrement dit un alias de nom de colonne doit être unique pour toutes les tables, de même que les alias des tables eux-mêmes. En effet, certaines tables n'étant accessible que grâce à un identifiant d'une autre table, si cette table est déjà identifiée par un alias, le second alias utilisé pour invoquer la même table afin d'accéder à une autre table doit être différent, sinon il y aura une erreur SQL stipulant que l'on a déclaré 2 fois le même alias.

4.3.2 Ajouter un critère

Dans la recherche avancée, plusieurs critères peuvent être sélectionnés. Je présente ici la méthode pour ajouter le critère de décomposition graphique. Le même principe sous-jacent est employé pour les critères d'édition de façon quasi-identique.

- créer un nouveau gestionnaire de critère *GestionnaireCriteresDecompositionGraphique.java* dans le package **gestion.criteres**
- l'ajouter comme nouveau membre de *RechercheAvancee.java* du package **mditest**

```
private GestionnaireCriteresDecompositionGraphique gcdg
```

- créer les accesseurs en lecture/écriture et appeler la fonction *updateFenetreRecherche()* dans l'accesseur en écriture

```
public GestionnaireCriteresDecompositionGraphique getGcdg() {
    return gcdg;
}
public void setGcdg(GestionnaireCriteresDecompositionGraphique gcdg) {
    this.gcdg = gcdg;
    updateFenetreRecherche();
}
```

- ajouter "setGcdg(null)" dans la fonction RechercheAvancee.reinitialiser()

- ajouter "&& gcdg==null" dans la fonction RechercheAvancee.aucunCritere() dans le return
- ajouter dans RechercheAvancee.actualiserListeCriteres() :

```
if(gcdg !=null)
    listModel.addElement(gcdg.toString()) ;
```

- ajouter dans RechercheAvancee.nouvelleRecherche() au début :

```
if(gcdg !=null)
{
    sb.append(" ") ;
    sb.append(gcdg.getClauseInnerJoin()) ;
}
```

et "gcdg !=null" dans la condition du "if" suivant la ligne "StringBuilder sb2=new StringBuilder() ;",

```
if(gcm !=null || gcdp !=null || gcdg !=null...)
```

et ajouter dans ce "if" le code :

```
if(gcdg !=null)
{
    if((gcm !=null || gcdp !=null)&& !sb2.toString().isEmpty()&& !gcdg.getClauseWhere().isEmpty())
        sb2.append(" AND ") ;
    sb2.append(gcdg.getClauseWhere()) ;
}
```

- dans *FenetreRechercheGenerale.java* du package **uipackage.consultation**, ajouter un onglet de critère, ainsi qu'une fenêtre *CriteresDecompositionGraphique.java* dans **uipackage.criteres**, et ensuite ajouter dans *FenetreRechercheGenerale.java* le membre :

```
private CriteresDecompositionGraphique criteresDecompGraph
```

et enfin modifier la fonction FenetreRechercheGenerale.reinitCriteres pour y ajouter :

```
criteresDecompGraph=null ;
jcheckbox_decompGraph.setSelected(false) ;
```

Remarque : plus le nombre de critères sélectionnés est élevé, plus le temps de recherche est grand.

CHAPITRE 5

Logiciels de synchronisation des bases

La base d'un utilisateur étant locale, sur son poste de travail, chaque utilisateur possède au départ la version originale de la base. Dès lors, il peut la modifier comme bon lui semble. Si ces changements sont pertinents, et non destinés à une phase d'essai, il peut, avec la supervision du super-utilisateur, appliquer ces modifications également à la base centrale située sur un poste serveur, uniquement destiné au stockage de la base. Deux applications ont été créées pour pouvoir réaliser ces diverses opérations : **MajBddClient** (qui doit être installée sur le poste utilisateur) et **MajBddServeur** (qui doit être installée sur le poste de la base centrale).

5.1 Un mot sur les deux logiciels

5.1.1 MajBddClient

Le logiciel doit être fourni avec le dossier **sqldata** contenant les dossiers **mise_a_jour** et **sauvegarde**, et le fichier *structure_bdd.sql*

Le dossier **mise_a_jour** doit contenir le fichier (de création/mise à jour de la base locale) provenant du serveur, et ce fichier doit être nommé *bddictionnairique.sql* pour être valable.

5.1.2 MajBddServeur

Le logiciel doit être fourni avec le dossier **sqldata** contenant les dossiers **base_originale** et **sauvegarde**.

Le dossier **base_originale** doit contenir le fichier de création de la base originale *bddictionnairique.sql* ainsi que le fichier *structure_bdd.sql*.

5.2 Base d'un utilisateur (ou client)

5.2.1 Principe de sauvegarde des changements en local

Dans le logiciel d'interaction avec la base (ProjetBddictionnairique), lorsque l'utilisateur lambda effectue des opérations impliquant une requête sur la base modifiant les données, par exemple comportant INSERT/DELETE/UPDATE..., la fonction **QueryManager.sauvegarderRequete** doit être appelée, en lui passant en paramètre la syntaxe de la requête sous forme d'un String.

Lorsque l'utilisateur a terminé et quitte le logiciel, soit le fichier **queries.sql** est créé s'il n'existe pas, soit les requêtes sauvegardées y sont écrites, en encodage UTF-8 pour conserver l'intégrité des données. Cette opération s'effectue lors de la fermeture de l'application, par l'intermédiaire de l'événement de fermeture de fenêtre **MDIApplication.formWindowsClosing** qui va invoquer la méthode **QueryManager.printQueriesList** chargée d'écrire les requêtes dans le fichier *queries.sql*

Remarque : le fichier *queries.sql* est stocké soit dans le même répertoire que ProjetBddictionnairique.jar sous Windows/MacOS, soit dans le répertoire utilisateur /home/user/ sous Linux.

Si l'utilisateur lambda souhaite que les modifications qu'il a apporté à sa base le soient aussi sur la base centrale, il doit fournir le fichier *queries.sql* au super-utilisateur qui a accès au poste serveur où est stockée la base centrale, et il doit effacer le fichier de son poste une fois qu'il la donné à la personne concernée.

5.2.2 Mise à jour

L'utilisateur lambda utilise le logiciel MajBddClient soit pour créer la base locale la première fois qu'il a installé le logiciel, soit pour la mettre à jour avec la version de la base centrale provenant du serveur.

Quoi qu'il en soit, le fichier contenant la base se nomme *bddictionnairique.sql* et doit être placé dans le dossier **sqldata/mise_a_jour**.

5.3 Base centrale (ou serveur)

5.3.1 Distribuer la base à un client

Pour distribuer aux utilisateurs classiques la dernière version de la base, le super-utilisateur doit utiliser le logiciel MajBddServeur en cliquant sur le bouton "serveur=¿client", et s'il le souhaite il peut choisir l'option de compression au format zip pour que le fichier résultant prennent moins de place et soit plus facilement transportable. Cette option est d'ailleurs conseillée car la base fait dans les 180Mo à l'écriture de ce rapport, et compressée elle ne fait plus que 23Mo.
Ensuite, il peut donner le fichier à l'utilisateur qui fera l'opération de mise à jour décrite précédemment.

5.3.2 Appliquer les modifications d'un client

L'utilisateur classique doit avoir effectué l'opération décrite plus haut "sauvegarde des changements en local".
Dès lors, le super-utilisateur utilise le logiciel MajBddServeur pour choisir le fichier *queries.sql* à lire pour effectuer les modifications sur la base centrale.
Pour plus de sécurité, la confirmation du super-utilisateur est requise pour appliquer chaque modification à la base, mais il peut choisir de l'enlever.
De même, pour plus de sécurité, une case à cocher invite à effectuer une sauvegarde de la base centrale avant de la modifier.

CHAPITRE 6

Annexe A

6.1 Importation de fichiers CSV dans phpMyadmin

1. sélectionner une table de la base
2. sélectionner l'onglet importer
3. sélectionner le fichier à importer (bouton Parcourir)
4. jeu de caractères : utf-8
5. cocher "CSV via LOAD DATA"
6. remplir "Champs terminés par" avec le caractère de séparation (tabulation ou virgule)
7. vider "Champs entourés par" et "Caractère spécial"
8. remplir "Lignes terminées par" avec le caractère de fin de ligne \n

6.2 Configurer Netbeans pour l'apparence Nimbus

Rendez-vous à l'adresse suivante : http://wiki.netbeans.org/FaqCustomLaf
Nimbus est le "Look And Feel" (l'apparence) du logiciel d'interaction avec la base.

6.3 Créer une base de données

```
- - création de la base en UTF8 si elle n'existe pas déjà
CREATE DATABASE IF NOT EXISTS bdd_dictionnairique DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci ;
```

6.4 Générer des diagrammes de classes UML dans la Javadoc

(Lorsqu'il n'y a pas de caractères particuliers en UTF8)

1. Télécharger **UmlGraph-5.2.jar** (http://www.umlgraph.org/download.html) et l'ajouter au projet Netbeans dans les bibliothèques
2. Télécharger et installer **Graphviz 2.26.3** (http://www.graphviz.org/)
3. éditer le fichier build.xml et y ajouter la cible suivante, puis lancer la génération de la Javadoc :

```
<target name="javadoc" description="Generates UML-Javadoc">
        <property file="nbproject/project.properties"/>
        <mkdir dir="${dist.javadoc.dir}"/>
        <javadoc source="${javac.source}" packagenames="org.*"
              destdir="${dist.javadoc.dir}"
              private="true" encoding="UTF-8">
            <classpath>
                <path path="${javac.classpath}"/>
```

```
            <path path="${module.classpath}"/>
          </classpath>
          <fileset dir="${src.dir}" excludes="${excludes}"
            includes="${includes}">
            <filename name="**/*.java"/>
          </fileset>
          <doclet name="org.umlgraph.doclet.UmlGraphDoc"
            path="${file.reference.UmlGraph-5.2.jar}">
            <param name="-attributes" />
            <param name="-operations" />
            <param name="-qualify" />
            <param name="-types" />
            <param name="-visibility" />
          </doclet>
        </javadoc>
    </target>
```

6.5 Ajout d'un nouveau dictionnaire

Selon la qualité des données et de la complexité de la structuration des données XML (s'il s'agit effectivement d'un dictionnaire au format XML), le processus d'ajout peut être assez rapide (ex : EPD) ou alors très long (ex : LPD). Pour ajouter un dictionnaire, je propose une méthodologie générale à suivre.

- étude de la base : quel que soit le format de la base (XML, SQL...) il faut avant tout en cerner la structure et la comprendre. Dans le cas d'un fichier XML, j'ai utilisé le logiciel Altova XML Spy en version d'évaluation pour extraire les schémas XSD (ou encore les fichiers DTD), dont j'ai étudié la structure. Ensuite, j'ai créé des classes pour parser ces fichiers et rendre un fichier texte qui soit plus lisible pour le client spécialiste (rôle des classes nommées *XXXprocessor.java* où XXX est le nom du dictionnaire).

- définir l'utilisation des données et modéliser les tables : lorsque le parsage se fait sans problème, et que l'on peut extraire les données comme on veut dans un fichier, le client spécialiste doit choisir ce qu'il veut en faire. C'est donc le rôle du développeur de l'aider en lui proposant des "structures d'accueil" pour ses données, sous forme d'une base de données avec des tables qui vont recueillir et organiser les données. Il est d'ailleurs nécessaire de faire attention à choisir une colonne destinée à ordonner les données, car on rappelle que dans un SGBD les données (ou les lignes d'enregistrement) ne sont pas ordonnées, et cela est important pour des bases documentaires telles que des dictionnaires.

- créer les nouvelles tables : lorsqu'une donnée est définitivement destinée à telle ou telle table, on peut donc créer la table physiquement, par exemple avec phpmyadmin.

- implémenter le transfert des données : dans le cas des dictionnaires XML, cela consiste à parser les fichiers, à créer des objets de stockage (sous forme de classes Java) et ensuite à "sérialiser" en quelque sorte ces données dans des fichiers CSV qui peuvent être importés facilement dans les tables de la base.

- modifier le logiciel d'interaction avec la base : il faut créer les classes DAO associées aux tables, les fenêtres de consultation/modification...

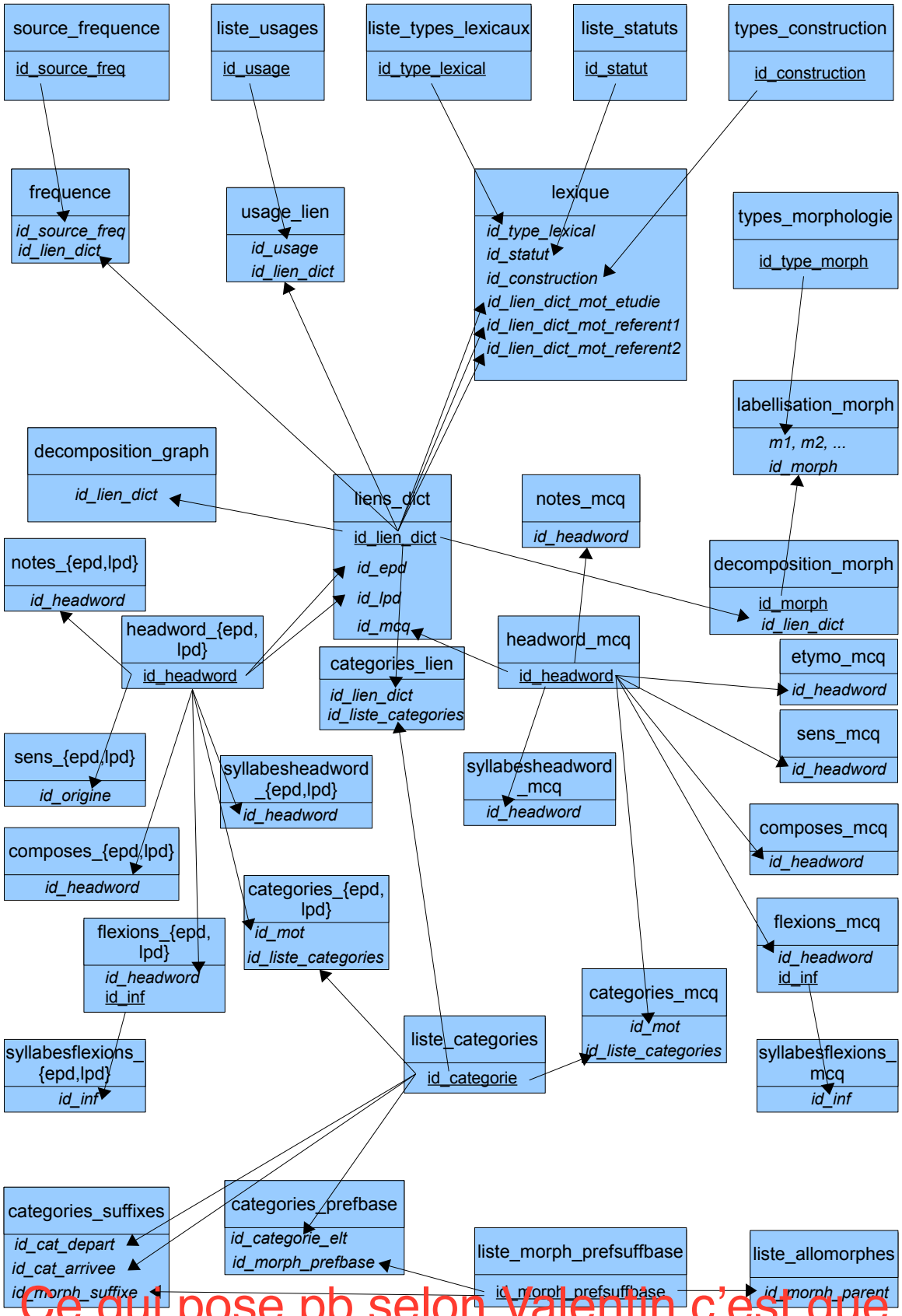
CHAPITRE 7

Annexe B

Le schéma donné ici correspond aux tables présentées dans le chapitre "Réalisation de la base de données". Il est similaire à un MLD dans la méthode Merise. On rappelle que les actions d'insertion/mise à jour/suppression sur les clés primaires possédant des références en clés étrangères sont gérées par programmation et non pas en cascade pour les raisons déjà évoquées de donner le choix à l'utilisateur de laisser ou non des orphelins.

Description du schéma :

- chaque rectangle bleu représente une table
- les clés primaires sont soulignées
- les clés étrangères sont en italiques
- une flèche part d'une clé primaire et pointe vers une clé étrangère dans une autre table y faisant référence.



Ce qui pose pb selon Valentin c'est que la fonction recherche se fait sur des éléments non fiables au lieu d'éléments primaires présents dans les dictionnaires