Université de Montréal

Représentation OWL de la ressource lexicale LVF et son utilisation dans le traitement automatique de la langue

Par :

Radia ABDI

Département d'informatique et de recherche opérationnelle

Faculté des arts et sciences

Mémoire présenté à la faculté des arts et sciences
en vue d'obtention du grade de Maître ès (M.Sc)
en informatique

Septembre, 2013

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé :

Représentation OWL de la ressource lexicale LVF et son utilisation dans le traitement automatique de la langue

Présenté par :

Radia ABDI

A été évalué par un jury composé des personnes suivantes :

Mémoire accepté le :

À mes chers parents,

Remerciements

J’aimerais remercier toutes les personnes qui m’ont aidée durant cette expérience de recherche et d’écriture de ce mémoire :

Guy Lapalme, mon directeur de recherche, pour sa disponibilité, ses critiques et ses conseils toujours opportun ainsi que pour sa grande gentillesse. Sa relecture rigoureuse de ce mémoire et ses remarques pertinentes m’ont permis d’améliorer grandement la qualité de ce travail.

Toute l’équipe de laboratoire RALI, pour leur respect et leur gentillesse ainsi que leur aide précieuse. Merci à Jamel pour ses conseils et sa bonne humeur revigorante, À Fabrizio aussi, pour ses critiques constructives, sa patience et sa disponibilité.

Tous mes enseignants, chacun de son domaine, pour leur contribution à ma formation durant mes études universitaires.

Membres de jury pour l’honneur qu’ils ont fait en participant à ce jury et d’avoir bien voulu apporter leurs jugements sur ce modeste travail.

Et enfin, ma famille, mon père et ma mère qui ont su m’encourager jour après jour et qui ont toujours cru en moi. Sans leur soutien constant, tant affectif que matériel, je n’aurai jamais pu accomplir mes études supérieures.

Résumé

Le dictionnaire LVF (Les Verbes Français) de J. Dubois et F. Dubois-Charlier représente une des ressources lexicales les plus importantes dans la langue française qui est caractérisée par une description sémantique et syntaxique très pertinente. Le LVF a été mis disponible sous un format XML pour rendre l'accès aux informations plus commode pour les applications informatiques telles que les applications de traitement automatique de la langue française. Avec l'émergence du web sémantique et la diffusion rapide de ses technologies et standards tels que XML, RDF/RDFS et OWL, il serait intéressant de représenter LVF en un langage plus formalisé afin de mieux l'exploiter par les applications du traitement automatique de la langue ou du web sémantique. Nous en présentons dans ce mémoire une version ontologique OWL en détaillant le processus de transformation de la version XML à OWL et nous en démontrons son utilisation dans le domaine du traitement automatique de la langue avec une application d'annotation sémantique développée dans GATE.

Mots-clés : LVF, Les Verbes Français, peuplement d'ontologies, ressource lexicale, web sémantique, extraction d'information, OWL.

Abstract

The LVF dictionary (Les Verbes Français) by J. Dubois and F. Dubois-Charlier is one of the most important lexical resources in the French language, which is characterized by a highly relevant semantic and syntactic description. The LVF has been available in an XML format to make access to information more convenient for computer applications such as NLP applications for French language. With the emergence of the Semantic Web and the rapid diffusion of its technologies and standards such as XML, RDF/RDFS and OWL, it would be interesting to represent LVF in a more formalized format for a better and more sophisticated usage by natural language processing and semantic web applications. We present in this paper an OWL ontology version of LVF by demonstrating the mapping process between the data model elements of the XML version and OWL. We give account about its use in the field of natural language processing by presenting an application of semantic annotation developed in GATE.

Key words: LVF, Les Verbes Français, Ontology population, lexical resource, semantic web, information extraction, OWL.

TABLE DES MATIÈRES

TABLE DES FIGURES

Chapitre 1 Introduction

Dans ce chapitre, nous présentons dans un premier temps le contexte et la problématique de notre travail. Nous allons par la suite expliquer le travail qui a été effectué, de l'ontologisation et la représentation OWL des données de « Les verbes français » (LVF) **[20]** jusqu'à son intégration dans une application d'annotation sémantique. La dernière partie s'intéresse à la vision du web sémantique et à son rôle dans la structuration et la modélisation des données d'une manière formelle. Nous présentons brièvement l'objectif du web sémantique et sa position par rapport au web actuel ainsi que les différents langages et formalismes proposés par le W3C.

1.1. Contexte et problématique

Des ressources lexicales riches et disponibles en accès libre en langue anglaise ont facilité le développement des recherches en traitement automatique de cette langue, tels que WordNet, VerbNet, FrameNet …etc. Il existe malheureusement peu d'équivalents en français disponibles en accès libre ce qui a retardé la recherche et les travaux dans le traitement automatique de la langue française.

L'ouvrage « Les Verbes Français » (LVF), réalisée par Jean Dubois et Françoise Dubois-Charlier est une ressource lexicale disponible pour le français, qui a été conçue dans une optique de fournir une description linguistique des verbes français. À cause de quelques problèmes de diffusion et de distribution du LVF, il n'a malheureusement pas été exploité par les chercheurs et les linguistes qui en ignoraient même l'existence. Les seuls travaux qui ont été faits jusqu'ici ont consisté à rendre le LVF plus accessible en termes d'encodage et de format de données : Denis Le Pesant a créé une version LVF sous format Excel[1] pour faciliter son exploitation manuelle mais ce mode d'accès ne s'est pas avéré pratique pour les applications informatiques; Guy Lapalme en a alors proposé une version XML[2] qui facilite l'exploitation de cette ressource par les applications de traitement automatique de la langue.

Ces dernières années, il y a eu un regain de la notion d'ontologies, et cela sous l'impulsion du web sémantique. En effet, la recherche sur le web, étant devenue une activité à haute valeur

[1] http://www.modyco.fr/index.php?option=com_content&view=article&id=1764&Itemid=19&lang=fr

[2] http://rali.iro.umontreal.ca/Dubois/

ajoutée, a poussé un développement rapide de modèles et langages standards (XML, RDF, OWL) et d'outils permettant d'expliciter la sémantique des données issues du web et de raisonner sur ces données en utilisant notamment les notions d'ontologies.

Les caractéristiques des standards du web sémantique de partage, de réutilisation, d'interopérabilité et d'inférence font de ces standards des langages de haut niveau. La représentation du LVF en format XML, considéré comme un des standards de base du web sémantique, nous a incitée à développer une représentation du LVF en d'autres standards plus sophistiqués et plus puissants du web sémantique, en l'occurrence le standard OWL pour la représentation d'ontologies.

L'intérêt de l'application des ontologies aux activités du traitement automatique de la langue a été démontré par de nombreux travaux de recherche académiques plus particulièrement dans le traitement automatique de la langue anglaise. Par exemple, des versions OWL de WordNET ou FrameNet développées dans un but de désambiguïsation du sens des mots ou d'intégration dans une autre ontologie (Mapping) **[1][2].**.

Dans ce travail, nous nous intéressons à transformer LVF en une ontologie OWL et à l'utiliser dans le domaine du traitement automatique de la langue, notamment en annotation sémantique. L'enjeu de ce travail est de montrer l'apport du web sémantique aux applications du traitement automatique de la langue et vice-versa et à quel point la représentation ontologique du LVF peut être bénéfique pour une meilleure exploitation et utilisation dans les différentes applications du TAL et du web sémantique.

1.2. Contribution

LVF mérite d'être exploité de plus en plus dans les travaux de recherche en traitement automatique de la langue française. Dans cette optique, notre principale contribution est l'ontologisation et la mise en œuvre d'une version OWL du LVF ce qui permet de fournir une ressource de qualité supérieure aux développeurs d'applications ainsi que d'ouvrir de nouveaux horizons pour l'utilisation du LVF, que ce soit pour les applications du web sémantique ou pour des applications du traitement automatique de la langue.

L'ontologie LVF que nous avons développée offre une description complète de tous les aspects et concepts de la ressource, son modèle de données est généré automatiquement à partir de ses fichiers XML disponibles sur le site du laboratoire RALI[3].

Nous avons par la suite utilisé cette ontologie dans une application d'annotation sémantique développée dans la plateforme GATE[4] (General Architecture for Text Engineering). L'annotation sémantique est une application très importante que ce soit dans le domaine du traitement automatique de la langue ou dans le web sémantique. Elle consiste à étiqueter les mots avec des liens qui pointent vers une description sémantique dans le but de fournir une identité unique représentant parfaitement le sens du mot annoté. Dans le cas du LVF, nous nous sommes intéressés à l'annotation des verbes à l'aide des concepts de l'ontologie LVF plus précisément à l'aide des entrées des verbes qui représentent les instances de la classe « Entrée » de l'ontologie LVF et qui définissent la description syntactico-sémantique de chaque verbe. Les entrées d'un verbe dans LVF sont numérotées, par exemple : `acheter01`, `acheter02`, `acheter03` …etc. Chaque entrée est définie par un ensemble d'informations comme le schème syntaxique, l'opérateur sémantique, le sens, le domaine, la classe syntactico-sémantique, la conjugaison …etc.

En effet, l'annotation de chaque verbe par son entrée correspondante nécessite la détermination automatique du schème syntaxique ou sémantique du verbe. Cependant, dans cette application, nous nous sommes basés sur une approche d'annotation semi-automatique qui consiste à extraire les verbes et à proposer une liste d'entrées possibles pour chaque verbe. L'utilisateur doit par la suite choisir l'entrée correspondante à l'aide du sens et du schème syntaxique proposés dans la liste.

L'annotation sémantique des verbes français joue un rôle essentiel dans la désambiguïsation du sens des verbes ce qui va servir dans plusieurs applications qui rencontrent des problèmes d'ambiguïté telles que la traduction automatique, la recherche d'informations …etc.

[3] http://rali.iro.umontreal.ca/rali/?q=fr/node/1237

[4] http://gate.ac.uk/sale/tao/split.html

1.3. Concept du web sémantique

1.3.1. Introduction et définition

Ces dernières années ont vu la montée en puissance de deux visions différentes du web, celle du web actuel (Web 2.0) et celle du web sémantique. Le web 2.0 met beaucoup plus l'accent sur l'échange et le partage de connaissances ainsi que l'ouverture et la collaboration des utilisateurs pour la production des informations. Cependant, il ne dispose pas d'outils pour décrire et structurer ses ressources de manière satisfaisante afin de permettre un accès pertinent à l'information, par exemple les liens entre les pages web n'ont aucune signification exploitable par les machines. Le web tel que nous le connaissons aujourd'hui ne considère que des documents textuels ou multimédia, conçus pour être interprétés uniquement par des utilisateurs, liés par des hyperliens non typés qui ne peuvent pas être identifiés sémantiquement par des agents logiciels.

La problématique d'un web interprétable que par les humains a été posée d'où l'initiative du web sémantique qui vise à résoudre l'interprétation des données par les agents logiciels. Tim Berner-Lee a proposé d'étendre le web actuel vers un web plus intelligent où l'information est rendue lisible non seulement par les utilisateurs, mais par les machines aussi : *"The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation"* **[3]**

1.3.2. Les standards du web sémantique

Le web sémantique répond aux problèmes et limitations du web actuel : (i) aucune sémantique n'est attribuée au contenu web, (ii) les métadonnées utilisées sont non structurées et limitées dans leur usage, (iii) absence de modèle de représentation de connaissances et de données publiées sur le web ce qui rend le processus de raisonnement et d'inférence presque impossible.

Les solutions proposées dans le cadre du web sémantique répondent à ces problèmes. Nous présentons les plus essentielles :

- Mettre en œuvre des formalismes et des langages standardisés de représentation des données et de structuration des connaissances. Ces langages permettent de représenter et de modéliser la sémantique des ressources web.

- Rendre disponibles des ressources conceptuelles représentées par ces langages modélisant les domaines des connaissances et facilitant leur accès et leur partage : les ontologies
- Définir les métadonnées plus explicitement à l'aide des langages définis formellement.

Le W3C[5] (World Wide Web Consortium) ainsi que d'autres chercheurs ont beaucoup travaillé sur la concrétisation de la vision du web sémantique et ont proposé plusieurs langages et standards de représentation de connaissances. On se retrouve donc avec une série de formalismes, organisés sous forme d'une pile du web sémantique qui regroupe l'ensemble des technologies les plus importantes du web sémantique (voir la figure 1.1). La base de cette pile représente le modèle commun de représentation des ressources web; URI/IRI. Chaque ressource dans le web est définie par un URI – Uniform Ressource Identifier **[4]**- qui permet de l'identifier d'une manière universelle et unique. On trouve aussi dans la pile des langages de structuration de données tel que XML, d'autres pour la description de la sémantique des données comme RDF et puis des langages de modélisation d'ontologies comme RDFS et OWL. À ceux-ci viennent s'ajouter des notions de logique formelle, de preuve et de confiance qui reprennent certains principes de l'intelligence artificielle. En effet, plus on monte dans la pile plus les langages deviennent plus expressifs et riches.

[5] http://www.w3.org/standards/semanticweb/

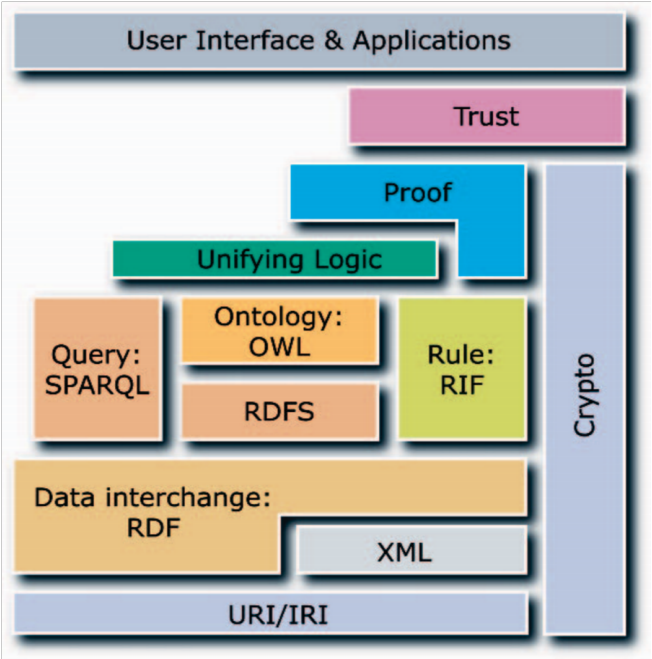


Figure 1.1 *Pile du Web Sémantique*[6]

1.3.2.1. XML (eXtensible Markup Language)

XML est le langage de base permettant de représenter un document d'une manière arborescente en utilisant un système de balisage, on parle beaucoup plus du XML comme un méta-langage.

Ce langage a été développé pour faciliter la publication, l'échange et le partage de données à travers le web sans se soucier des systèmes informatiques. Des standards comme XPath[7] ou XQuery[8] ont été développés afin de parcourir et d'interroger les documents XML. **[5]**

XML permet une manipulation syntaxique mais non sémantique des documents d'où l'intérêt de développer d'autres langages manipulant de la sémantique.

Le W3C a proposé le langage XSL[9] (eXtensible StyleSheet Language) qui est lui-même défini avec le formalisme XML. Il permet de transformer les documents XML à l'aide de feuilles de style en d'autres types de documents tels que HTML, RTF, PDF, Tex …etc.

[6] http://www.w3.org/2001/sw/
[7] http://www.w3.org/TR/xpath/
[8] http://www.w3.org/TR/xquery/

1.3.2.2. RDF (Resource Description Framework)

RDF[10] est une recommandation du W3C développé pour décrire les ressources du web ainsi que leurs relations. Le but de RDF est de fournir une description formelle des données afin qu'elles soient interprétables par les machines. Pour ce faire, RDF utilise une structure bien définie qui décrit toute expression à l'aide de triplets, chacun composé d'un sujet, un prédicat et un objet ou ({ressource, propriété, valeur}).

Un ensemble de triplets peut être représenté par un graphe illustré par un diagramme composé de nœuds et d'arcs orientés dans lequel chaque triplet est représenté par un lien nœud-arc-nœud. Un sujet ou une ressource est une entité d'information référencée par un identificateur qui doit être un URI. La propriété ou le prédicat est la relation ou l'attribut qui décrit la relation entre deux ressources. L'objet est la valeur de la ressource qui est en relation avec le sujet, l'objet peut être un URI, un littéral ou une variable.

Un document RDF est codé en machine grâce à la syntaxe XML. En voici une représentation RDF/XML ainsi que sa représentation en graphe RDF dans la figure 1.2.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cd="http://www.recshop.fake/cd#">
    <rdf:Description
        rdf:about="http://www.recshop.fake/cd/Empire-Burlesque">
        <cd:artist>Bob Dylan</cd:artist>
        <cd:country>USA</cd:country>
        <cd:company>Columbia</cd:company>
        <cd:price>10.90</cd:price>
        <cd:year>1985</cd:year>
    </rdf:Description>
</rdf:RDF>
```

[9] http://www.w3.org/Style/XSL/

[10] http://www.w3.org/RDF/

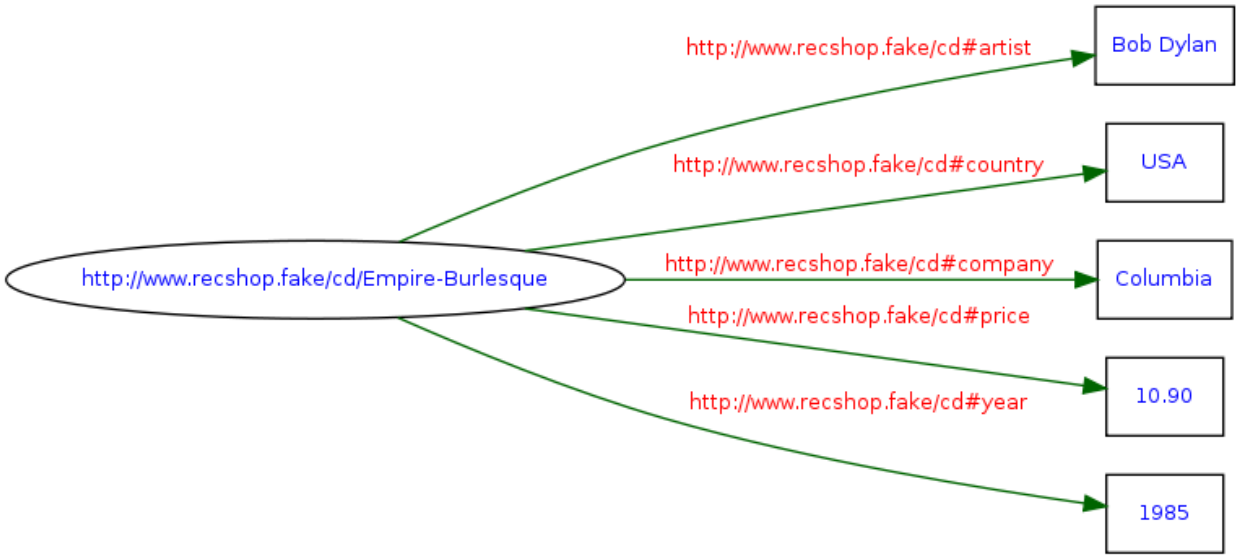


Figure 1.2 *Graphe RDF*

La figure 1.2 montre un extrait d'un document RDF[11] qui définit les informations les CDs de musique. Le sujet et les propriétés sont définis par des URIs et les objets représentent des littéraux de type chaîne de caractères ou entier.

RDFS (RDF schéma) étend le langage RDF pour définir le modèle/schéma des déclarations RDF. Il fournit un mécanisme de spécification de classes et sous-classes (**rdfs:Class**, **rdfs:SubClassOf**) dont les ressources seront des instances de type sujet ou objet. Il permet aussi la formulation des contraintes sur les propriétés entre les instances des classes (**rdfs:range**, **rdfs:domain**).

1.3.2.3. OWL (Web Ontology Language)

La nécessité de modéliser des connaissances dans des domaines complexes a fait émerger quelques limites de RDF/RDFS. RDFS ne permet pas d'exprimer toutes les contraintes pour la modélisation de tous les domaines. Il offre un vocabulaire très simple limité à une hiérarchie de classes et de propriétés ce qui limite les possibilités de modélisation de certains domaines complexes. Le langage OWL recommandé par le W3C, enrichit RDFS en définissant un vocabulaire plus complet pour la description d'ontologies complexes. Le vocabulaire OWL définit de nouvelles notions qui spécifient les classes et le type des propriétés telles que :

[11] Exemple tiré du http://www.w3schools.com/rdf/rdf_example.asp

l'équivalence des classes, l'équivalence des propriétés (relations), la symétrie et la transitivité des relations, la notion de cardinalité de classe.

OWL est basé essentiellement sur le formalisme des logiques des descriptions **[6]**. Avec l'aide de « reasoners » qui traitent la logique de description, il devient possible de se doter d'une capacité d'inférence et de raisonnement déductif sur les concepts de l'ontologie. OWL est reconnu comme le langage le plus utilisé pour la description des ontologies.

1.4. Conclusion

Nous avons présenté dans ce chapitre la problématique et le contexte général de notre travail. Nous nous intéressons, dans le cadre de cette maîtrise, à développer une version ontologique en standard OWL de la ressource LVF et à démontrer par la suite son utilisation dans une application d'annotation sémantique. Nous avons aussi présenté le contexte général ainsi que quelques notions importantes du web sémantique tels que XML, RDF et OWL. Dans la suite des chapitres, nous allons élaborer notre démarche de transformation automatique du LVF de XML à OWL et les détails sur l'application d'annotation sémantique développée dans GATE.

La structure du mémoire est organisée comme telle : au deuxième chapitre, nous démontrerons ce que le web sémantique peut apporter au traitement automatique de la langue et vice-versa avec une étude comparative entre les deux domaines. Dans le troisième chapitre, nous aborderons la notion d'annotation sémantique, son utilisation dans le web sémantique ainsi que les langages utilisés pour l'encodage des annotations. Dans le quatrième chapitre, nous présenterons la ressource LVF en détaillant sa structure et l'organisation des différents types de classes ainsi que les détails d'une entrée LVF. Dans le cinquième chapitre, nous élaborerons le processus d'ontologisation du LVF, la conception ainsi que la démarche suivie pour la création de l'ontologie LVF. Nous détaillerons aussi les étapes de la transformation automatique des fichiers XML du LVF en format OWL. Dans le sixième chapitre, nous présenterons l'application d'annotation sémantique que nous avons développée dans GATE. Ce chapitre est divisé en deux parties, la première partie sera consacrée pour la présentation de l'architecture GATE (General Architecture for Text Engineering). Nous donnerons un aperçu général sur son fonctionnement, ses différents outils et plugins les plus utilisés. La deuxième partie présentera l'application d'annotation LVF et son intégration dans GATE et donnera quelques exemples de résultats d'annotations de verbes à l'aide des entrées de LVF.

Chapitre 2 Apports mutuels entre le TAL et le web sémantique

Le web sémantique et le traitement automatique de la langue sont deux paradigmes différents mais avec quelques points communs ou une vision plus ou moins similaire. Dans ce chapitre, nous en identifierons les points de friction et de contact ainsi que leurs apports mutuels.

2.1. Deux paradigmes disjoints ?

Le TAL et le WS partagent la même problématique, à savoir que le langage naturel ne présente pas de signification pour les machines alors que la plupart des informations électroniques disponibles sont sous forme non structurées et non-formelles.

Hypothèse du TAL :

> Le langage naturel n'est pas interprétable par une machine et le langage des machines est difficile à maîtriser par les êtres humains, on essaye donc de concevoir des systèmes informatiques qui comprennent le langage humain.

Hypothèse du WS :

> Le contenu actuel du Web est en grande majorité du langage naturel non interprétable par les machines, il est donc nécessaire d'ajouter de la structure aux informations du Web pour qu'elles soient interprétables par les machines.

Pour le WS, le langage naturel dans le web est dépourvu de sémantique interprétable par les machines. Le but du web sémantique est faire en sorte que les machines puissent interpréter les documents et les données, mais non nécessairement la parole et les écrits humains. Par contre, l'ambition du TAL à long terme est de comprendre le langage naturel en l'occurrence la parole et l'écrit. En effet, c'est le point qui représente la différence entre les deux domaines qui partent des propositions tout à fait compatibles, mais proposent des solutions et des méthodologies différentes. De plus le web sémantique se limite aux données du Web alors que le TAL traite le langage naturel dans n'importe quel contexte. Les deux paradigmes partagent une vision ambitieuse de l'univers informationnel, une vision à long terme peut-être même utopique.

2.2. TAL → WS

Nous présentons maintenant comment les applications du traitement automatique de la langue peuvent résoudre certains problèmes ou limitations du web sémantique.

Un des buts du Web sémantique est la construction de métadonnées (annotations) et d'ontologies à l'échelle mondiale du web, alors que cette tâche reste toujours problématique sur le plan économique, organisationnel et cognitif.

La construction d'ontologies à l'échelle mondiale représente un des problèmes majeurs du web sémantique pour lequel la concrétisation de cette vision officiellement devient utopique. De ce fait, la construction d'ontologies à partir de textes fait l'objet d'étude depuis plusieurs années dans le domaine d'ingénierie d'ontologies. Les outils du TAL permettent d'extraire des textes des éléments de connaissances à représenter dans une ontologie et facilitent de plus en plus la construction et le peuplement automatique d'ontologies.

L'assistance à la récupération des métadonnées **[7]** ou à la préparation d'ontologies **[8] [9]** implique le recours à une famille de technologies linguistiques, regroupées sous le terme ***d'extraction d'informations,*** qui désignent des tâches plus ou moins complexes telles que :

- **Identification de termes (extraction de termes)** : identifier des groupes de mots (mot ou syntagmes nominaux, verbaux, adjectivaux) présentant certains patrons syntaxiques (par exemple *SYNTEX* **[10]** *:* analyseur syntaxique) en étudiant leur distribution dans un corpus (par exemple *UPERY* **[11]** *:* effectue des calculs sur la proximité distributionnelle entre les syntagmes et termes). Cette extraction et analyse terminologique peuvent aider à identifier les concepts de l'ontologie.
- **Découverte de relations sémantiques** : l'étude des propriétés distributionnelles des termes, c.-à-d. leur apparition dans des contextes similaires, aider à définir des relations entre concepts et le développement d'ontologies. (Yakwa et Cameleon)
- **Repérage d'entités nommées** : reconnaître des entités telles que les noms d'entreprises, de lieux, des noms propres, des dates …etc.
- **Remplissage d'un formulaire** (Template) : trouver les caractéristiques d'un objet donné par exemple, pour un produit, trouver son nom, la société qui le fabrique, son prix ….etc.

La construction d'une ontologie se fait en plusieurs phases. L'identification de termes et le repérage d'entités nommées sont particulièrement utiles pour la phase initiale. Les tâches de découverte de relations aident à la définition de concepts de l'ontologie et de leurs relations. La tâche de remplissage de formulaires est utilisée pour documenter les attributs des objets de l'ontologie. L'ensemble de ces techniques fournit en quelque sorte une procédure pour la création semi-automatique d'ontologies.

Le **TAL** aidera à la transformation du web actuel en une base de connaissances structurées d'une manière semi-automatique. Des procédés sont déjà par exemple fortement utilisés pour repérer de manière automatique la ou les **thématiques d'un texte**. **Le marquage de nom de personnes ou de lieux**, appelée ***extraction d'entités nommées*** **[12]**, en est un exemple. Il est également possible d'y adjoindre des techniques plus proches des considérations de l'analyse sémantique afin de préciser la thématique et même d'extraire du sens. Des techniques, telles que l'analyse de cooccurrence ou la lemmatisation, peuvent alors être employées pour résoudre le contexte et lever les ambiguïtés de sens. Il est également possible d'utiliser des ressources linguistiques propres à la langue dans laquelle l'information est décrite tels des ontologies ou thésaurus.

Cette approche présente toutefois l'inconvénient d'être « risquée » dans le sens où ces techniques ne sont pas fiables à 100%. Il n'est ainsi pas rare, même dans un contexte plutôt favorable, de commettre de 30 à 40% d'erreurs. C'est pourquoi dans la plupart des projets utilisant ces techniques, on préfère garder un document « intermédiaire » dans lequel on peut voir le texte de base ainsi que les connaissances mises en évidence par les méthodes de TAL. Ce document permet de conserver un œil critique sur le résultat final. De même, dans le web sémantique, il conviendra de laisser dans tous les cas un accès aux documents originaux d'où a été extraite l'information afin de pouvoir vérifier dans un second temps l'adéquation entre la connaissance extraite et l'information réelle.

Les techniques de TAL doivent donc avant tout être vues non pas comme un moyen de construire un web sémantique de manière totalement automatisée mais plutôt comme un moyen de nous aider à extraire la « connaissance utile » des pages déjà présentes sur le web d'aujourd'hui afin de les adapter au web sémantique.

2.3. WS → TAL

Dans ce qui suit, nous allons voir le rôle que peut jouer les technologies du web sémantique dans le traitement automatique de la langue.

2.3.1. Les standards W3C

L'interopérabilité joue un grand rôle dans la conception du Web sémantique. Le WS répond aux problèmes d'interopérabilité par la définition de standards permettant d'offrir des langages formels pour toutes les applications du web sémantique et qui, proposés par le W3C, commencent à se diffuser assez largement. Ces standards, décrits dans le premier chapitre, permettent de répondre aux besoins d'interopérabilité et de réutilisation qui se font sentir dans la communauté du traitement automatique de la langue. XML s'y répand rapidement comme format d'encodage et d'échange de données par contre RDF et OWL sont moins fréquemment utilisés.

Le TAL produit des corpus annotés et des lexiques pour lesquels il est important de disposer de métadonnées fiables et normalisées. De ce fait, RDF peut être utilisé pour encoder les annotations linguistiques et enregistrer les descriptions de ces corpus d'une manière plus formelle et normalisée.

Les dictionnaires électroniques constituent des ressources cruciales pour de nombreux systèmes de TAL. La normalisation de ces ressources joue un rôle critique pour le secteur vu la complexité et le coût de leur utilisation et maintenance. Elle fait l'objet de nombreux travaux actuels tel que la définition de schéma RDF ou même d'ontologie au format OWL pour l'encodage des dictionnaires électroniques **[13].**

La constitution de corpus étiquetés est souvent un obstacle pour de nombreuses applications de TAL qui reposent sur des méthodes d'apprentissage. Dans certains cas, l'utilisation des métadonnées produites pour des applications du WS pourrait être utile pour étiqueter des corpus. Ainsi, les corpus seront accompagnés par des métadonnées normalisées et détaillées ce qui rendra l'apprentissage plus efficace notamment la classification automatique **[14] [15]**.

2.3.2. Les ontologies

Les ressources lexicales (WordNET, FrameNET, VerbNET ..etc), les thésaurus (MeSH) et les réseaux sémantiques jouent un rôle important dans la performance des applications TAL. Cependant, leur intégration demande des investissements coûteux. Les ontologies, encodées dans un format normalisé et formel, représentent un facteur déterminant de la qualité des résultats et de la viabilité économique des applications de TAL. Donc, la représentation ontologique de ces ressources va améliorer les performances et la productivité de leur utilisation par différentes applications. On voit s'amorcer un cercle vertueux où les systèmes de TAL participent à la construction et au peuplement des ontologies pour le WS, mais les exploitent aussi pour améliorer leurs résultats et affiner leurs analyses.

2.4. Conclusion

La maturité opérationnelle du web sémantique dépend de la capacité des promoteurs de ce domaine à modéliser et à construire une grande masse de données formalisées (ontologies et métadonnées). Le coût d'un tel projet est un réel obstacle que ce soit du côté économique ou cognitif. Le TAL peut contribuer à la réduction de ces coûts et ainsi favoriser le développement des applications du WS.

Les difficultés du TAL sont dues en partie à un manque d'interopérabilité des différents outils et à une modularité insuffisante des systèmes. Les technologies formelles du WS apportent un début de réponse à ces problèmes en offrant un cadre normatif et des logiciels libres (open source). Le partenariat entre les acteurs du WS et les acteurs du domaine linguistique peuvent améliorer l'avancement des deux projets et amener à un web doté à la fois de sémantique formelle et de sémantique linguistique.

Dans le prochain chapitre, on verra le rôle de l'annotation sémantique et son application dans le web sémantique.

Chapitre 3 Annotation sémantique

Dans ce chapitre, nous allons détailler le processus d'annotation sémantique qui représente l'un des aspects applicatifs du web sémantique. Nous présentons des exemples d'utilisation d'ontologies ainsi que des standards du web sémantique tels que XML, RDF et RDFS pour la définition du modèle d'annotation ainsi leur encodage.

3.1. Quelques Définitions

En général, le terme annotation réfère à une note, une critique, une remarque ou encore un commentaire qui accompagne un texte afin d'y apporter plus de précision et d'explication. Autrement dit, l'annotation est l'action d'apposer une note sur une partie de document ou de texte de document.

Dans le domaine de la documentation et des bibliothèques, l'annotation de ressources documentaires est une vieille tradition. La Digital Library Federation (DLF), une association qui regroupe une quinzaine de bibliothèques américaines, a défini trois types d'annotations qui peuvent s'appliquer sur les ressources documentaires d'une bibliothèque numérique **[16]**:

1. *Annotation administrative,* représente les informations associées à la création et à la maintenance de la ressource numérique telles que « qui, quoi, où et comment ». Le langage Dublin Core représente le standard ISO le plus utilisé pour l'annotation des ressources numériques avec des descripteurs tels que : l'auteur, titre, éditeur, date de publication, langue, description … etc.
2. *Annotation structurelle,* relie les ressources en vue d'une représentation logique d'une ressource.
3. *Annotation descriptive,* décrit une ressource vis-à-vis de son contenu en attribuant des étiquettes sémantiques au contenu textuel de la ressource.

Dans le cadre du web sémantique, une annotation descriptive est le plus souvent appelée annotation sémantique. L'objectif des annotations sémantiques est d'exprimer la sémantique du contenu d'une ressource afin d'en améliorer la compréhension, la recherche par des agents logiciels et la réutilisation par les utilisateurs. On peut définir l'annotation sémantique comme **une représentation formelle des étiquettes sémantiques attachées au contenu**

textuel d'une ressource, exprimées à l'aide de concepts, relations et instances décrits par un des standards formels du web sémantique tels que les ontologies (OWL), RDF, XML.

3.2. Annotation et le web sémantique :

L'objectif du web sémantique est de décrire le contenu du web avec des annotations sémantiques et normalisées. Les ressources sont annotées à partir des connaissances disponibles dans une ou plusieurs ontologies. Ces annotations regroupées en entrepôts de métadonnées deviennent utiles pour des agents de recherche d'information utilisant ou non des moteurs d'inférence permettant de déduire de nouvelles connaissances.

Les résultats satisfaisants avec l'annotation sémantique dépendent de la disponibilité d'ontologies, de l'automatisation du processus d'annotation et du passage à l'échelle. Le problème d'annotation sémantique touche à plusieurs domaines de recherche complémentaires tels que l'intelligence artificielle, l'ingénierie des connaissances et le traitement automatique de la langue.

Dans la section suivante, nous montrerons comment les ontologies peuvent être utilisées pour annoter sémantiquement un document. Ensuite, nous décrirons les langages d'annotation disponibles.

3.2.1. Annotation sémantique à partir d'ontologie

Une ontologie est une conceptualisation d'un domaine de connaissance spécifique riche et complexe, elle permet la mise en place de mécanismes d'inférence et de raisonnement pour la découverte et l'exploitation de la connaissance.

Les annotations sémantiques sont modélisées par les ontologies et consistent en de simples pointeurs, aussi appelés descripteurs, vers les termes de l'ontologie c'est-à-dire concepts, relations et instances. Ces descripteurs fonctionnent comme des entrées d'index améliorant les résultats des moteurs de recherche.

Une même ressource peut être annotée par plusieurs ontologies en même temps. Les annotations sémantiques créées à partir d'une ontologie améliorent la recherche d'information grâce aux mécanismes d'inférence et de raisonnement, mais elles peuvent aussi être combinées avec l'enrichissement d'une ontologie. En effet, on peut peupler une

ontologie à partir des annotations sémantiques d'un document afin d'étendre encore plus l'ontologie pour couvrir plus de termes et de connaissances.

Francis Coppola naît le 7 avril 1939 à Detroit, dans le Michigan. Il est le deuxième des trois enfants de Carmine et Italia Coppola. Son père, originaire de New York, est chef d'orchestre. Francis fera appel à lui pour composer la musique du « Parrain », en 1972. Sa mère, elle, est la fille du célèbre compositeur napolitain Francesco Pennino, auteur de l'Opéra « Senza Mamma », dont un extrait figure dans « Le Parrain II ». Comédienne, elle joue dans plusieurs films de Vittorio De Sica, avant d'embrasser la carrière de « mamma ». Francis a de qui tenir![…]

Figure 3.1 *Extrait de l'article « le Clan Coppola » paru dans le magazine « ELLE »* ***[17]***

Dans l'exemple de la figure 3.1, on veut annoter ce texte avec les concepts d'une ontologie qui modélise le domaine de la « Presse People ». On suppose que l'ontologie contient des concepts tels que « Personnalité » et « Œuvre artistique », des propriétés comme « lieu de naissance », « date de naissance », « date de création », « profession », etc. et des relations du genre « estCrééPar(Œuvre artistique, Personalité) », « aLienParentéAvec(Personalité, Personalité) ». Par conséquent, le texte serait annoté soit par les instances de concepts. Par exemple, Francis Coppola représente une instance du concept « Personnalité » et Senza Mamma est une instance du concept « Œuvre artistique », soit par les valeurs de propriétés. Par exemple, « Détroit» peut être la valeur de l'attribut « Lieu de naissance » qui peut être attaché à l'annotation de l'instance « Francis Coppola », soit par les instances de relations comme la relation « estCrééPar » qui peut relier deux instances par exemple « Francis Coppola » et « le Parrain ».

3.2.2. Langages de l'annotation sémantique

Dans cette partie, nous nous concentrons sur les langages de description qui peuvent représenter les annotations. Plusieurs langages et métalangages permettent l'encodage des annotations sémantiques. En effet, le web sémantique offre une panoplie de métalangages standardisés tels que **XML**, **RDF(S)** et **OWL**, détaillés dans les chapitres précédents, qui peuvent encoder les annotations. En plus des standards du web sémantique, d'autres langages développés par la communauté de recherche peuvent aussi être pris en considération. Nous allons en présenter quelque-uns.

3.2.2.1. HTML-A

C'est une extension d'HTML, proposée par l'initiative (KA) **[16]**, qui permet d'insérer des annotations sémantiques dans les pages web grâce à la balise <A>. Cette approche ne spécifie pas le langage d'implémentation de l'ontologie de référence ce qui fait que les agents logiciels du web devraient savoir interpréter cette extension correctement.

Dans la figure 3.2, le concept et l'instance de l'ontologie sont déclarés à l'en-tête HTML, «Personnalité:FFCoppola». Toutes les annotations insérées dans le corps du HTML font référence à l'instance «FFCoppola».

```
<html>
<head><Title>Le Clan coppola</Title>
<A ONTO="Personnalité:FFCoppola"/>
</head>
<body>
Francis Coppola naît le <A ONTO="Personnalité[dateNaissance=body]">7 avril 1939</A> à <A
ONTO="Personnalité[lieuNaissance=body]">Detroit</A>, dans le <A
ONTO="Personnalité[lieuNaissance=body]">Michigan</A>.
</body>
</html>
```

Figure 3.2 *Exemple d'annotation en HTML-A*

3.2.2.2. SHOE (Simple HTML Ontology Extension)

SHOE a le même principe que HTML-A qui est de permettre aux agents logiciels du web d'interpréter sémantiquement les pages web et d'améliorer les mécanismes de recherche d'informations. Cette approche est similaire à l'approche **HTML-A** sauf qu'au lieu d'utiliser l'attribut **ONTO** dans l'élément **A**, **SHOE** propose d'utiliser un ensemble d'éléments prédéfinis tels **INSTANCE**, **RELATION**, **CATEGORY**, etc**.** Ces informations doivent être interprétables par des agents web **[18]**.

SHOE permet de :

- Créer et définir de nouvelles ontologies de simple hiérarchie.
- Déclarer les relations entre les classes, appelées **CATEGORY,** et leurs attributs
- Classifier les entités selon le schème « is-a »
- Décrire un ensemble de règles d'inférence simplifiées.

L'exemple de la figure 3.3 montre l'utilisation de SHOE pour l'annotation d'une page web. On remarque avec SHOE que les concepts de l'ontologie de référence sont déclarés directement dans le code HTML ainsi que l'URL de l'ontologie contrairement à HTML-A.

```
<html>
<head><Title>Le Clan coppola</Title></head>
<body>
Francis Coppola naît le 7 avril 1939 à Detroit, dans le Michigan.

<INSTANCE KEY="FFCoppola">
<USE-ONTOLOGY ID="People-Ontology" URL="http://www.elle.com/SHOE/people.html"
VERSION="1.0" PREFIX="people">
<CATEGORY NAME="people.FFCoppola">
<RELATION NAME="people.dateNaissance">
        <ARG POS=1 VALUE="FFCoppola">
        <ARG POS=2 VALUE="7 avril 1939">
</RELATION>
<RELATION NAME="people.lieuNaissance">
        <ARG POS=1 VALUE="FFCoppola">
        <ARG POS=2 VALUE="Detroit">
</RELATION>
</INSTANCE>

</body>
</html>
```

Figure 3.3 *Exemple d'annotation en SHOE.*

SHOE et HTML ont été abandonnés au profit de langages plus formels et recommandés par le W3C.

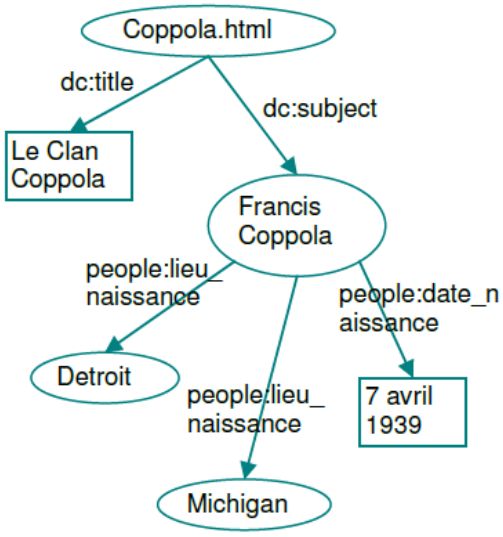
3.2.2.3. Langages du web sémantique

On présente dans ce qui suit quelques exemples d'annotations avec les standards du web sémantique décrits au premier chapitre.

- **Annotation avec RDF/RDFS**

RDF répond aux besoins de base de la plupart des outils d'annotation. La représentation XML du RDF permet aux agents logiciels de manipuler très facilement les annotations de manière interopérable; la modélisation du RDF sous forme de graphe sémantique apporte une flexibilité ainsi qu'une capacité de réutilisation des énoncés d'un document RDF afin de les rendre plus complexes et riches. RDF ne fournit pas un mécanisme de description d'hiérarchie de classes ou de types, mais par contre il définit les relations entre sujet et objet avec des triplets « prédicat (sujet, objet) ». Il est donc plus ou moins limité en expressivité

d'énoncés complexes. L'exemple de la figure 3.4 donne une idée sur l'utilisation du RDF pour l'exemple de Francis Coppola.




```
<xml "version="1.0" encoding="UTF-8">
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:people="http://www.mondeca.com/onto#people">

<rdf:Description
   rdf:about="http://www.elle.fr/Coppola.html">
   <dc:title>Le Clan Coppola</dc:title>
   <dc:subject rdf:resource="#Francis Coppola"/>
</rdf:Description

<rdf:Description rdf:about="Francis Coppola">
   <people:date_naissance>7 avril 1939
     </people:date_naissance>
   <people:lieu_naissance rdf:resource="#Detroit/>
   <people:lieu_naissance rdf:resource="#Michigan/>
</rdf:Description>

</rdf:RDF>
</xml>
```

Figure 3.4 *Annotation sémantique en RDF*

Une solution pour améliorer l'expressivité de RDF est RDFS (RDF Schéma) qui permet de décrire des relations au niveau des classes et propriétés. Par conséquent, RDFS introduit les primitives de bases de la modélisation ontologique pour le web sémantique.

```
<xml "version="1.0" encoding="UTF-8" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description rdf:about="http://www.elle.fr/Coppola.html">
   <rdf:type rdf:about="#Article"/>
   <rdfs:label>Le Clan Coppola</rdfs:label>
   <dc:subject rdf:resourceRef="#FFCoppola"/>
</rdf:Description>

<rdfs:Class rdf:ID="#Lieu">
<rdfs:Class rdf:ID="#Personne">
<rdfs:Class rdf:ID="#Personnalité">
   <rdfs:subClassOf rdf:resource ="#Personne"/>
</rdfs:Class>

<rdf:Property rdf:about="lieu_naissance">
   <rdfs:domain rdf:resource="#Personne"/>
   <rdfs:range rdf:resource="#Lieu"/>
</rdf:Property>
<rdf:Property rdf:about="date_naissance">
   <rdfs:domain rdf:resource="#Personne"/>
   <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<Personnalité rdf:about="FFCoppola">
   <rdfs:label>Francis Coppola</rdfs:label>
   <lieu_naissance rdf:resource="#Detroit"/>
   <lieu_naissance rdf:resource="#Michigan"/>
   <date_naissance>7 avril 1939</date_naissance>
</Personnalité>
</xml>
```

Figure 3.5 *Annotation avec RDFS*

- **Annotation avec OWL**

L'annotation avec OWL peut être très intéressante car c'est le langage le plus expressif des langages ontologiques pour le web sémantique. En pratique, il est conçu comme une extension à RDF/RDFS car il réutilise presque toutes les constructions de RDF/RDFS et rajoute d'autres composantes telles que les propriétés objet, les propriétés type de données, un mécanisme pour le raisonnement, l'héritage des propriétés ainsi que l'expression de différentes contraintes sur les propriétés comme la cardinalité et les caractéristiques des propriétés.

OWL s'appuie plus fortement encore sur la théorie *des logiques de description* ce qui permet aux machines d'effectuer des raisonnements automatisés sur les inférences sur la base de connaissance.

De ce fait, encoder les annotations avec le langage OWL permet une meilleure expressivité et une meilleure exploitation des capacités d'inférence et de raisonnement logique.

```
<xml "version="1.0" encoding="UTF-8" xmlns:owl="http://www.w3.org/2002/07/owl#">
<rdf:RDF>
<owl:Ontology rdf:about="http://www.mondeca.com/onto#people">
   <dc:title>People</dc:title>
</owl:Ontology>

<owl:Class rdf:ID="Lieu">
<owl:Class rdf:ID="Personne">
<ow:Class rdf:ID="Personnalité">
   <rdfs:subClassOf rdf:resource ="#Personne"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="lieu_naissance">
   <rdfs:domain rdf:resource="#Personne"/>
   <rdfs:range rdf:resource="#Lieu"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="date_naissance">
   <rdfs:domain rdf:resource="#Personne"/>
   <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</owl:DatatypeProperty>

<owlThing rdf:ID="FFCoppola">
   <rdf:type rdf :resource="#Personnalité"/>
   <rdfs:label>Francis Coppola</rdfs:label>
   <lieu_naissance rdf:resource="#Detroit"/>
   <lieu_naissance rdf:resource="#Michigan"/>
   <date_naissance>7 avril 1939</date_naissance>
</Personnalité>

<rdf:Description rdf:about="http://www.elle.fr/Coppola.html">
   <rdf:type rdf:about="#Article"/>
   <rdfs:label>Le Clan Coppola</rdfs:label>
   <dc:subject rdf:resourceRef="#FFCoppola"/>
</rdf:Description>

</rdf:RDF>
</xml>
```

Figure 3.6 *Annotation avec OWL*

3.3. Conclusion

Dans ce chapitre, nous avons décrit d'une manière générale l'annotation sémantique en mettant l'accent sur l'annotation à partir d'ontologies ainsi que sur les langages qui permettent l'encodage des annotations tels que RDF et OWL. Dans le chapitre suivant, nous présenterons la ressource lexicale LVF, sa structure et son organisation.

Chapitre 4 La ressource LVF (Les Verbes Français)

Les Verbes Français (LVF) est une base de données numérique réalisée par J. Dubois et F. Dubois-Charlier dont le but est de classifier les verbes selon leur syntaxe et leurs interprétations sémantiques.

Les auteurs ont utilisé les méthodes classiques de la grammaire distributionnelle et transformationnelle pour élaborer ce dictionnaire. 12310 verbes y sont représentés. Certains verbes ayant plusieurs entrées, 290 verbes ont 10 entrées ou plus, par exemple le verbe *passer* est représenté par 61 entrées (un cas extrême du LVF).

Les objectifs et les méthodes utilisés pour la réalisation du LVF ont été présentés par les auteurs dans **[19]** de la façon suivante :

a. « la **classification syntaxique des verbes français** repose sur l'hypothèse qu'il y a **adéquation** entre les schèmes syntaxiques de la langue et l'interprétation sémantique qu'en font les locuteurs de cette langue (…).
b. Le **schème syntaxique** est défini, d'une part, **par la nature des constituants de la phrase**, leurs propriétés et leurs relations, d'autre part, **par les mots du lexique** qui entrent dans les types de constituants définis (…).
c. L'objet est donc de faire l'inventaire le plus complet possible de ces schèmes **syntaxiques,** selon les méthodes classiques de la grammaire distributionnelle et transformationnelle, et selon les différences des paradigmes lexicaux constatées entre les schèmes syntaxiques (…).
d. Une fois l'inventaire réalisé, on a établi une classification visant à tenir compte de **l'adéquation entre la syntaxe et l'interprétation sémantique**, elle-même réalisée par un schème sémantico-syntaxique donné sous la rubrique « opérateur » (...) »

Cette base de données est une ressource lexicale très riche et novatrice. Elle a été diffusée par D. Le Pesant, qui en a créé une version sous format Excel pour en faciliter l'exploitation à l'aide des fonctions de recherche et de tri. Cependant le mode d'accès Excel n'est pas très pratique pour les applications informatiques. G. Lapalme a alors proposé une version de cette ressource sous le format XML qui explicite certains codages et facilite le traitement des données. XML est un

standard très commun et très utilisé pour la structuration des données, mais aussi très efficace pour l'extraction d'informations, l'interrogation des données et les transformations automatiques en pages web. Donc, une version HTML a été générée à partir du fichier XML en appliquant une transformation automatique en pages web consultables. Dans ce qui suit, nous allons détailler la structure et l'organisation de cette ressource.

4.1. Détails d'une entrée LVF

Une forme verbale dans LVF a plusieurs emplois et donne donc lieu à plusieurs entrées représentées chacune par un numéro (01, 02, 03, etc.). Par exemple, le verbe `chercher` présente 10 entrées ou emplois différents (**chercher 01, …, chercher 10**) correspondant à des schèmes syntaxiques différents **[20]**. Une entrée est définie par un schème syntaxique et un opérateur codé pour faciliter le traitement automatique, mais aussi par d'autres informations linguistiques telles que le sens, des exemples de phrases, le lexique, la conjugaison …etc.

4.1.1. Les schèmes syntaxiques

Un schème syntaxique est une suite de caractères alphanumériques codés qui indiquent la nature du verbe (transitif direct, indirect, intransitif, pronominal), le type du constituant sujet et objet (humain, animal ou chose, complétive) et aussi la nature des compléments (locatif, prépositionnel, instrumental, à modalité, etc.). Le schème syntaxique représente une notation qui correspond au type et à la nature du sujet et compléments. Chaque entrée d'un verbe est représentée sous la rubrique « Schème ».

Les lettres qui indiquent le type de verbe : A (intransitif), N (transitif indirect), T (transitif direct), P (pronominale) sont présentés dans la figure 4.1.

Code	Type	Construction
A	Intransitif	Sujet + Circonstant
N	Intransitif indirect	Sujet + Complément Prépositionnel
T	Transitif	Sujet + Objet direct + CompPrep + Circonstant
P	Pronominal	Sujet + Objet direct + CompPrep + Circonstant

Figure 4.1 *Codage du type du verbe*

Le sujet correspond au premier caractère après A, N, T, P, l'objet correspond au deuxième caractère après T et P, le complément prépositionnel et circonstant correspondent au deuxième caractère pour N et A et le troisième et quatrième caractère pour T et P. Ainsi, dans T1901, nous avons T : transitif direct, le premier caractère 1 : sujet humain, le deuxième caractère 9 : objet humain ou chose, le troisième et quatrième caractère 0 et 1 : complément prépositionnel et circonstant.

La figure 4.2 présente la notation pour la nature du sujet et des compléments.

1	Humain	Qn
2	Animal	an(imal)
3	Chose	Qc
4	Complétive ou chose	Q, Inf, qc
5	Complétive ou inf	Q/D + Inf
7	Pluriel humain	qn+pl
8	Pluriel chose	qc+pl
9	Humain ou chose	qn, qc

Figure 4.2 *Codage de la nature du sujet et des compléments*

Exemple : croire01 [T1400] : *on croit que tu dis la vérité*

La figure 4.3 présente la notation pour les prépositions.

a	*à*	**d**	*contre*	**i**	*de*	**l**	*auprès*	**n**	*divers mvts*
b	*de*	**e**	*par*	**j**	*dans*	**m**	*devant*	**q**	*pour*
c	*avec*	**g**	*sur, vers*	**k**	*pour*				

Figure 4.3 *Codage des prépositions*

Exemple : **avertir01** [T11b0] : *on avertit Pierre de mon arrivée*

La figure 4.4 présente le codage du type du complément.

1	Locatif (on l'on est)	Bivouaquer qp
2	Locatif de destination	Accourir qp
3	Locatif d'origine	Décamper de qp
4	Double locatif	Conduire de qp à qp
5	Temps	Durer deux heures, persévérer le temps
6	Modalité (manière, mesure, quantité)	Aller bien, chausser du 36, manger bcp
7	Cause	Mourir d'un cancer
8	Instrumental, moyen	Montrer par un geste, blesser avec une arme

Figure 4.4 *Codage du type de complément*

Exemple : **conduire 04** [T3140] : *ces empreintes conduisent l'enquêteur au voleur.*

Le code T3140 indique que le sujet est une chose (*ces empreintes*) par le code 3, l'objet est un humain (*l'enquêteur*) par le code 1 et le type de complément est double locatif, c'est-à-dire conduire de qp à qp, par le code 4. Le code 0 indique qu'il n'y a pas de préposition à coder.

4.1.2. Les opérateurs

On attribue des propriétés sémantiques à chaque verbe dans LVF par le biais de la rubrique « opérateur » qui est une étiquette interprétative du sens et de l'emploi du verbe. Les opérateurs constituent les entités fondamentales de chaque classe et ils interprètent sémantiquement les schèmes syntaxiques. Ils codent la synthèse des propriétés syntaxiques (déjà représentées par les schèmes de construction syntaxique) et leur éventuel élément d'invariance sémantique qui se manifeste entre les verbes qui ont des propriétés syntaxiques communes.

Comparons par exemple les trois opérateurs suivants :

- loq AV (parler avec : *discuter avc qn*)
- loq AV qn D/SR qc (parler avec qn de/sur qc : *bavarder avec qn d'une question*)
- loq A (parler à qn : *causer à qn)*

Le radical « loq » représente une constante sémantique, mais aussi le fait que les verbes qui la possèdent sont à complément nominal ce qui démontre l'adéquation entre la construction syntaxique et l'interprétation sémantique par cette rubrique « opérateur ». Quant aux suffixes (AV, D, SR, qn, qc), ils correspondent à des propriétés de sous-catégorisation et de sélection lexicale.

Les opérateurs sont composés d'éléments primaires et secondaires. L'élément primaire est une sorte de radical et l'élément secondaire représente les suffixes qui accompagnent le radical. Par exemple, « *dic ordre A qn D nég* » est un opérateur complexe constitué de 6 éléments, dont « disc » qui veut dire « dire » en latin, représente l'élément primaire. Les suffixes « ordre A qn D nég » sont les éléments secondaires et permettent une compréhension maximale de l'opérateur.

4.2. Structure du LVF

Dans la section précédente, on a donné un aperçu des entrées LVF ainsi que certaines notions de base qui représentent ces entrées et qui forment les principes de construction du LVF tels que les schèmes syntaxiques et les opérateurs. Dans ce qui suit, nous allons présenter la hiérarchie des classes du LVF qui représente un regroupement des entrées selon leur construction syntaxique et leur interprétation sémantique.

4.2.1. Hiérarchie de classes

Le LVF organise les verbes français selon des classes sémantico-syntaxiques, qui sont des classes sémantiques définies par la syntaxe. Le principe de la classification repose sur l'adéquation entre les schèmes syntaxiques et la sélection distributionnelle dans la construction, et l'interprétation sémantique. Les arguments syntaxiques sont la base du classement sémantique des verbes. Les schèmes sont regroupés en classes et sous-classes : 248 sous-classes syntaxiques, 54 classes sémantico-syntaxiques et 14 classes génériques.

Les classes sont présentées selon une hiérarchie de trois niveaux :

4.2.1.1. Classes génériques

Les classes génériques se retrouvent au niveau le plus général de la classification du LVF. Elles sont codées au moyen d'une lettre majuscule, telle « C » pour « communication ». Les différentes classes sont construites par des regroupements successifs d'opérateurs semblables. Chaque classe générique est définie par un ensemble d'opérateurs. Par exemple la classe générique F « verbes d'agression et de contact » a été définie en regroupant 1727 verbes ayant un opérateur comportant les radicaux *ict* (frapper) et *tact* (toucher). Le LVF comporte les 14 classes génériques suivantes :

C	Communication
D	Don, privation
E	Entrée, sortie
F	Frapper, toucher
H	États physiques et comportements
L	Locatif
M	Mouvement sur place

N	Munir, démunir
P	Verbes psychologiques
R	Réalisation, mise en état
S	Saisir, serrer, posséder
T	Transformation, changement
U	Union, réunion
X	Verbe auxiliaires

4.2.1.2. Classes sémantico-syntaxique

Les classes génériques sont subdivisées en 54 classes sémantico-syntaxiques codées par une lettre et un chiffre telle que **C1**. La lettre représente le code de la classe générique et le chiffre représente le numéro de la classe. Il y a en tout quatre classes sémantico-syntaxiques qui représentent la plupart des classes génériques :

i. La première classe représente un verbe en construction intransitive, transitive indirecte ou pronominale avec un sujet humain, ou une construction transitive avec un objet humain. L'emploi de cette classe n'est pas figuré (animé ou pas).
ii. La seconde classe se distingue de la première par un emploi figuré.
iii. La troisième se distingue de la première par un actant non animé en sujet ou objet selon la construction.
iv. La quatrième se distingue de la troisième par un emploi figuré **[19]**

Ainsi, la classe F « frapper, toucher » qui se base sur les opérateurs *ict* (frapper) et *tact* (toucher), se décline en quatre sous-classes selon l'objet est *quelqu'un* (sens concret F1 et sens figuré F2) ou *quelque chose* (sens concret F3, sens figuré F4). Il est à noter qu'il y a des classes génériques qui échappent à cette répartition canonique telle qu'elle est définie dans les points précédents, sont les classes 'C', 'D', 'P', et 'X'.

4.2.1.3. Sous-classes syntaxiques

Les sous-classes syntaxiques, au nombre de 248 en tout, ont un but de regrouper les opérateurs semblables selon les schèmes syntaxiques. Chaque sous-classe syntaxique est définie par un sous-ensemble de la classe des opérateurs de la classe générique qui l'englobe. Dans une sous-

classe syntaxique, on regroupe les entrées qui ont les mêmes schèmes syntaxiques et des opérateurs sémantiques similaires. Une sous-classe syntaxique est codée par une lettre en minuscule telle que C1**a**.

La figure 4.5 montre un exemple de répartition de classes sémantico-syntaxiques et sous-classes syntaxique de la classe générique F. On remarque que la répartition des sous-classes syntaxiques ne suit pas de principe car le nombre de sous-classes varie pour chaque classe sémantico-syntaxiques.

Classe F1 (716 entrées); frapper ou toucher qn

F1a (177 entrées); frapper qn, un animal; se battre
F1b (141 entrées) frapper à mort, tuer; se tuer
F1c (90 entrées) attaquer qn
F1d (165 entrées) blesser qn, son corps
F1e (66 entrées) lutter, frapper, buter contre qn/qc
F1f (77 entrées) toucher qn, le corps de qn

Classe F2 (311 entrées); figuré de F1

F2a (99entrées); frapper à mal, à mort, vaincre, ruiner
F2b (88 entrées) attaquer ou toucher qn
F2c (39 entrées) frapper d'une peine ou accuser qn
F2d (73 entrées) s'opposer à qn/qc, lutter contre
F2e (12 entrées) butter contre, frapper sur

Classe F3 (483 entrées); frapper ou toucher qc

F3a (147 entrées); frapper qc
F3b (144 entrées) abîmer qc
F3c (151 entrées) détruire qc ou être détruit
F3d (41 entrées) toucher qc ou toucher à qc

Classe F4 (217 entrées); figuré de F3

F4a (19 entrées); s'opposer ou s'attaquer à qn/qc
F4b (188 entrées) supprimer, détruire, abîmer qc
F4c (151 entrées) toucher à qc

Figure 4.5 *Structure de la classe générique F*

4.3. Types d'accès au LVF

4.3.1. Version XML

La version XML de la ressource LVF a été proposée par G. Lapalme dans le but de rendre l'accès à cette ressource plus commode et facile pour les applications informatiques. XML, étant un format universel de structuration et d'organisation des données, permet d'interroger et d'extraire plus facilement les informations à l'aide du langage de requête XPath ce qui facilitera l'interrogation des entrées LVF, mais aussi d'effectuer des transformations automatiques en

format HTML, RDF, OWL à l'aide des feuilles de style XSLT **[20]**. L'organisation des données LVF en XML est présentée dans la figure 4.6.

```
<verbes date="2011-01-12" version="1.1" nb="12308">
   <verbe mot="abaisser" nb="9" id="abaisser">
      <entree>
         <mot no="1">abaisser 01</mot>
         <domaine nom="locatif, lieux">LOC</domaine>
         <classe generique="transformation, changement" semantico-
syntaxique="non-animé propre"
                 construction-syntaxique="c">T3c</classe>
         <operateur>r/d bas qc</operateur>
         <sens>baisser</sens>
         <phrases>
            <phrase>On <lexie-ref>a~</lexie-ref> le rideau de fer,le
store.</phrase>
            <phrase>Le rideau du magasin s'<lexie-ref>a~</lexie-
ref>.</phrase>
         </phrases>
         <conjugaison groupe="1" sous-groupe="baisser, pleurer, etc."
                      auxiliaire="avoir (sauf si pronominal ou entrée en
être)">1bZ</conjugaison>
         <construction>
           <scheme type="transitif direct" sujet="humain" objet="chose"
                   circonstant="instrumental, moyen">T1308</scheme>
           <scheme type="pronominal" sujet="chose"
circonstant="instrumental, moyen">P3008</scheme>
         </construction>
         <derivation der-able="positif seul (parfois avec modification
du radical)"
                     der-ment="formation directe sur la conjugaison"
                     der-eur="eur, ion">1-- -1 --RA --</derivation>
         <nom>-I</nom>
         <lexique desc="dictionnaire de base" nbmots="15000">2</lexique>
      </entree>
      <entree>
      ⋮
     </entree>
     <entree>
      ⋮
     </entree>
  </verbe>
```

Figure 4.6 *Extrait du fichier XML*

La figure 4.6 présente un petit fragment du fichier XML du LVF illustrant la structure d'une entrée LVF. Cette structure de représentation des entrées correspond à la représentation du tableur du fichier Excel de la ressource LVF. Dans le cadre de ce travail, on utilisera la version XML du LVF pour la transformation automatique en format OWL.

4.3.2. Version site web

Le site web[12] du LVF a été généré automatiquement à partir de son fichier XML. Il permet de naviguer plus facilement dans le contenu du LVF et d'organiser et de mieux afficher le contenu des classes sémantico-syntaxiques. Il est possible de consulter le contenu du LVF par classe. La figure 4.7 illustre la navigation dans la hiérarchie des classes du LVF qui permet de sélectionner le contenu d'une classe générique ou sémantico-syntaxique par un simple clic. Elle permet aussi d'accéder aux entrées d'une sous-classe syntaxique en cliquant sur le lien. La figure 4.8 montre le résultat affiché du contenu de la sous-classe syntaxique **D1a**. Les entrées soulignées sont regroupées selon leurs schèmes syntaxiques et leurs opérateurs et permettent d'accéder directement à la description des entrées dans la liste alphabétique.

[12] Le site web LVF : *http://rali.iro.umontreal.ca/Dubois/*

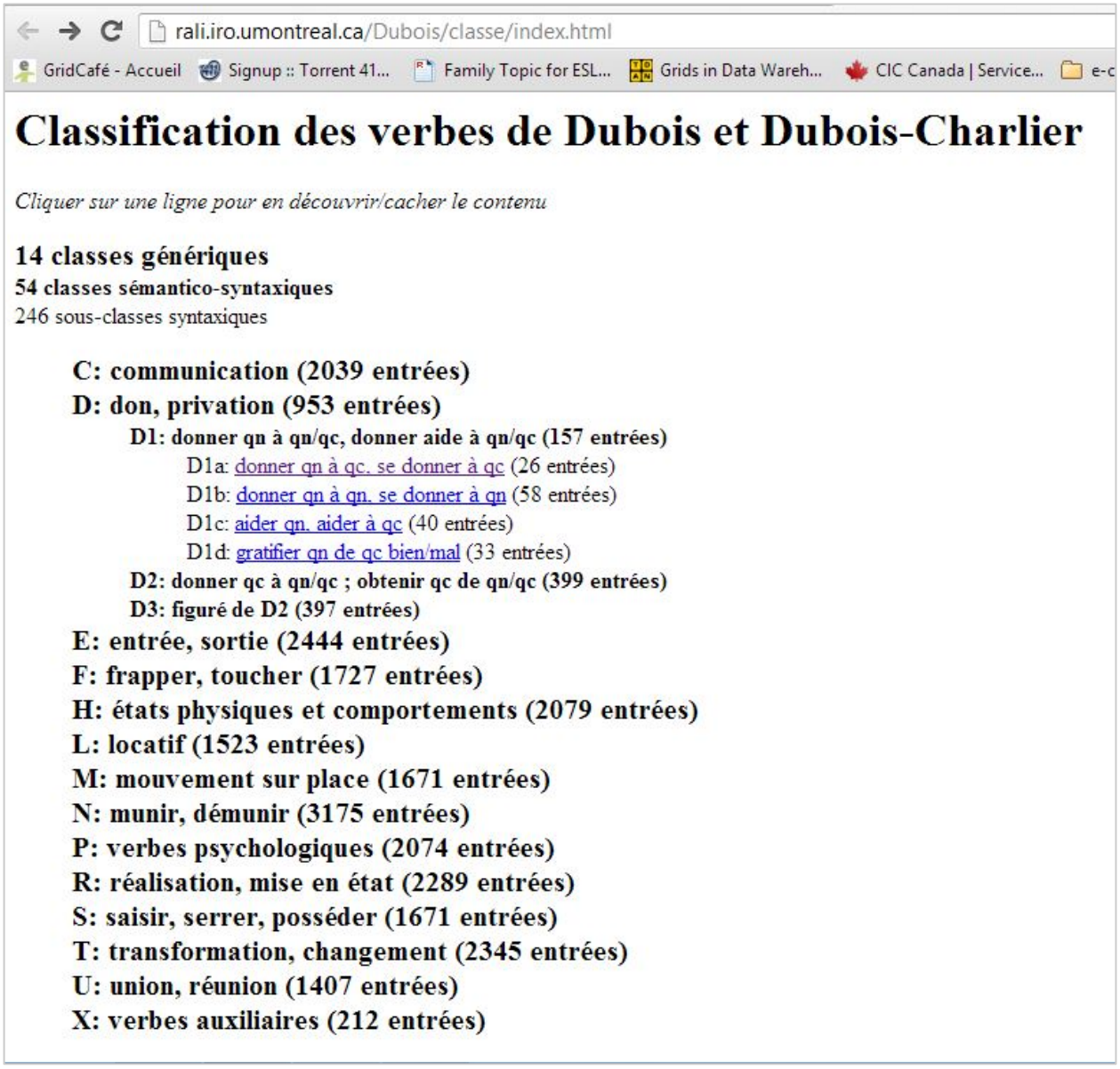
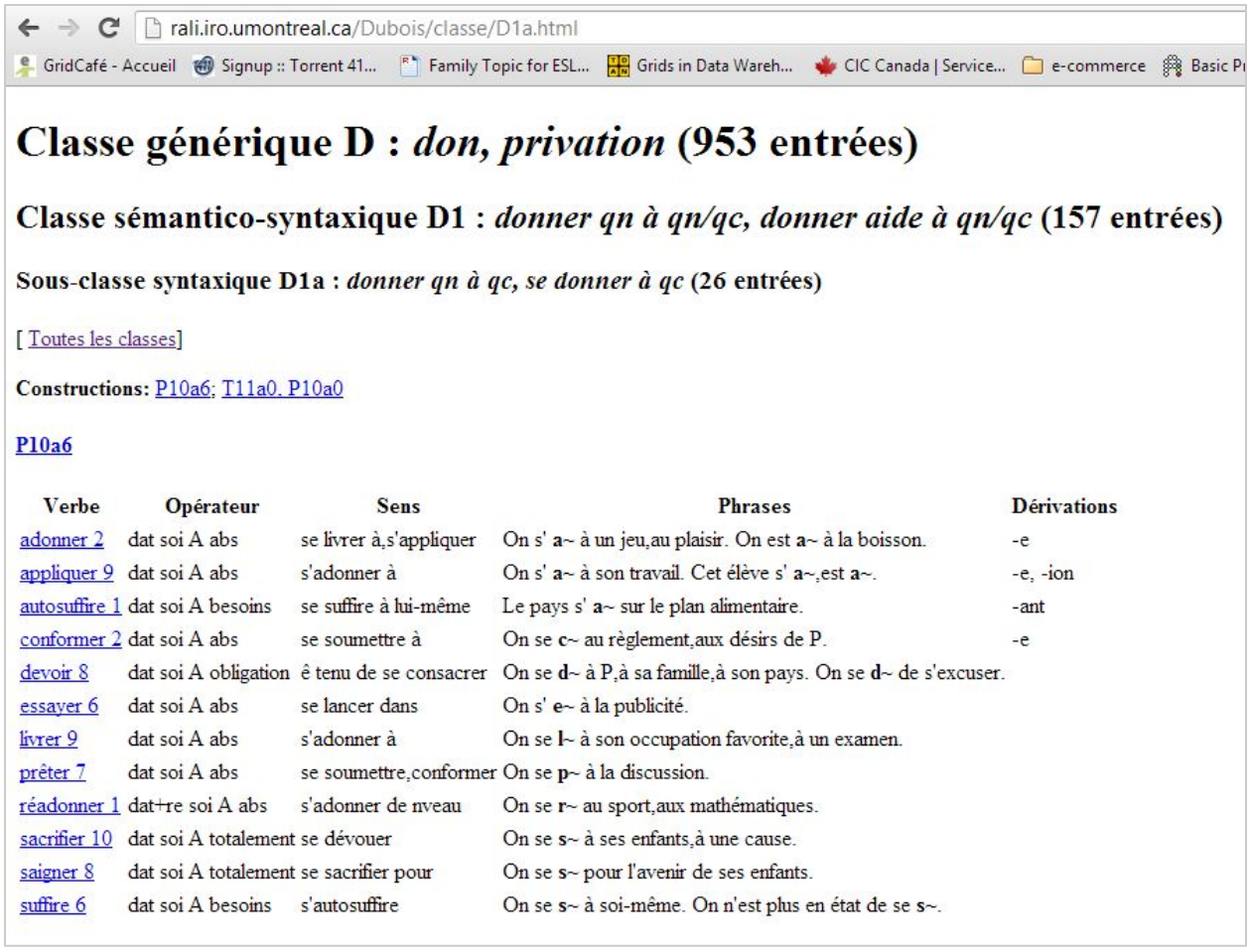


Figure 4.7 *Hiérarchie de classes dans le site web LVF*



rali.iro.umontreal.ca/Dubois/classe/D1a.html

GridCafé - Accueil | Signup :: Torrent 41... | Family Topic for ESL... | Grids in Data Wareh... | CIC Canada | Service... | e-commerce | Basic Pr

Classe générique D : *don, privation* (953 entrées)

Classe sémantico-syntaxique D1 : *donner qn à qn/qc, donner aide à qn/qc* (157 entrées)

Sous-classe syntaxique D1a : *donner qn à qc, se donner à qc* (26 entrées)

[Toutes les classes]

Constructions: P10a6; T11a0, P10a0

P10a6

Verbe	Opérateur	Sens	Phrases	Dérivations
adonner 2	dat soi A abs	se livrer à,s'appliquer	On s' **a~** à un jeu,au plaisir. On est **a~** à la boisson.	-e
appliquer 9	dat soi A abs	s'adonner à	On s' **a~** à son travail. Cet élève s' **a~**,est **a~**.	-e, -ion
autosuffire 1	dat soi A besoins	se suffire à lui-même	Le pays s' **a~** sur le plan alimentaire.	-ant
conformer 2	dat soi A abs	se soumettre à	On se **c~** au règlement,aux désirs de P.	-e
devoir 8	dat soi A obligation	ê tenu de se consacrer	On se **d~** à P,à sa famille,à son pays. On se **d~** de s'excuser.	
essayer 6	dat soi A abs	se lancer dans	On s' **e~** à la publicité.	
livrer 9	dat soi A abs	s'adonner à	On se **l~** à son occupation favorite,à un examen.	
prêter 7	dat soi A abs	se soumettre,conformer	On se **p~** à la discussion.	
réadonner 1	dat+re soi A abs	s'adonner de nveau	On se **r~** au sport,aux mathématiques.	
sacrifier 10	dat soi A totalement	se dévouer	On se **s~** à ses enfants,à une cause.	
saigner 8	dat soi A totalement	se sacrifier pour	On se **s~** pour l'avenir de ses enfants.	
suffire 6	dat soi A besoins	s'autosuffire	On se **s~** à soi-même. On n'est plus en état de se **s~**.	

Figure 4.8 *Affichage des entrées de la sous-classe D1a*

appliquer		1	PEI	L3b	lc qc SR/CT	faire adhérer à	On **a~** du rouge sur les lèvres. Le vernis s' **a~** mal.	T13g0, P30g0
		2	LOC	L3b	lc qc SR/CT	plaquer,appuyer sur	On **a~** son oreille contre la porte. L'appareil s' **a~** sur le mur.	T13g0, P30g0
		3	SOM	F3a	ict coup,baiser A qn	coller,asséner	On **a~** une gifle à P.	T13a0
		4	DRO	D3b	dat qc A c effet	faire subir à	On **a~** la loi à tous.	T13a0, P30a0
		5	SOC	R4c	m.e.état autre	mettre en acte	On **a~** les ordres sans réfléchir. La théorie ne peut s' **a~**.	T1306, P3000
		6	DRO	D3a	dat peine A qn	infliger	On **a~** à P la peine de vingt ans de réclusion.	T13a0, P30a0
	(s)	7	PSY	P1h	ger.mens A qc/inf	consacrer	On s' **a~** à résoudre ce problème. On **a~** son esprit à ceci.	P10a6, T13a0
		8	TIT	D3b	dat nom A qn	appliquer	On **a~** un surnom au professeur. Ce surnom s' **a~** bien à P.	T13a0, P30a0
	(s)	9	PSY	D1a	dat soi A abs	s'adonner à	On s' **a~** à son travail. Cet élève s' **a~**,est **a~**.	P10a6
		10	LIT	D3b	dat abs A qn	adapter	On **a~** cette remarque à P,à la situation actuelle.	T13a0, P30a0

Figure 4.9 *Affichage des entrées du verbe « appliquer »*

La figure 4.9 montre la description des entrées du verbe « appliquer » en cliquant sur le lien « appliquer 9 » de la figure 4.8. La deuxième colonne indique le numéro de l'entrée, la troisième colonne est une abréviation du domaine générique comme LOC pour localisation, PSY pour psychologie …etc. la quatrième colonne indique la classe sémantico-syntaxique de l'entrée, la

cinquième décrit l'opérateur sémantique, la sixième donne le sens avec des synonymes, la septième décrit quelques exemples de phases et la dernière indique le schème syntaxique.

Il est également possible d'accéder aux informations du LVF en consultant un index alphabétique de verbes ou en tapant une expression régulière. La figure 4.10 montre le résultat de la requête ***a….quer*** c'est-à-dire les verbes qui contiennent un ***a*** et se terminent par ***quer*** et qui contiennent 9 lettres entre le a et la fin de « quer ». Il y a en tout 14 verbes qui respectent cette contrainte et qui sont affichées selon leur ordre alphabétique avec toutes leurs informations respectives.

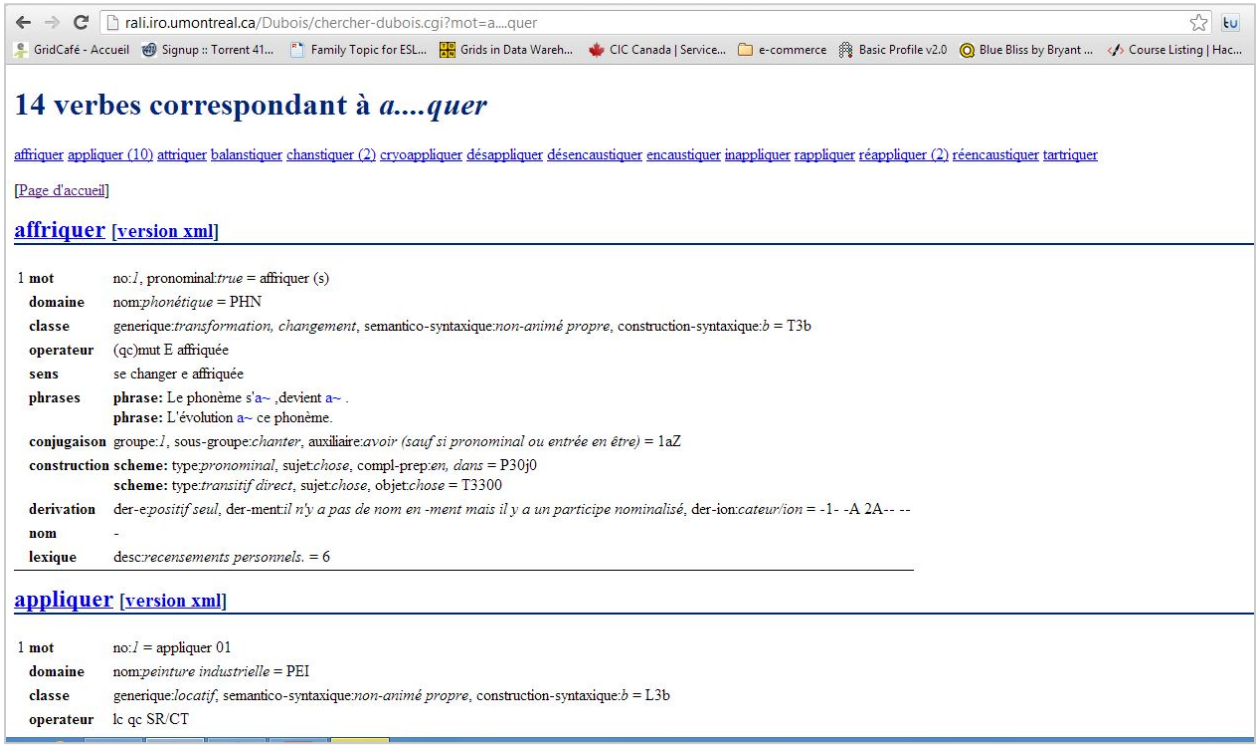


Figure 4.10 *Résultat de recherche par expression régulière*

4.4. Conclusion

LVF (Les Verbes Français) est une ressource lexicale qui classe les verbes français selon leur syntaxe et leur interprétation sémantique. Elle a été diffusée en version Excel et en version XML dans le but de faciliter son exploitation par les applications informatiques. Le LVF repose sur une hiérarchie de classes syntactico-sémantique et classe les verbes selon leur schème syntaxique et sémantique.

Dans ce chapitre, nous avons présenté la hiérarchie des classes du LVF ainsi que la démarche utilisée pour la classification des verbes. Nous avons aussi vu les détails d'une entrée d'un verbe et les informations présentées sur une entrée à l'aide d'exemples. La version HTML permet une consultation en ligne et offre des fonctionnalités de recherche à l'aide d'expressions régulières.

Dans ce qui suit, nous allons présenter notre travail d'ontologisation du LVF; d'abord le modèle de l'ontologie LVF et la transformation automatique à partir des fichiers XML.

Chapitre 5 Transformation OWL du LVF

Les ontologies représentent des ressources de modélisation et de conceptualisation très importantes. L'ontologisation de la ressource lexicale LVF fournit un format relationnel aux données du LVF. OWL, décrit au deuxième chapitre, supporte la représentation d'un domaine de connaissance en classes, propriétés et instances pour l'utilisation dans un environnement distribué comme le web. Il complémente RDF et RDFS en apportant plus de richesse et d'expressivité en ajoutant les restrictions, les contraintes de cardinalité sur les propriétés.

Il y a deux principales motivations pour le développement d'une ontologie OWL du LVF. Premièrement, la représentation du LVF en standard formel du W3C nous permet de répondre aux besoins des applications du web sémantiques telles que l'annotation sémantique et l'extraction d'informations…etc. Cette ontologie permettra aussi d'ouvrir de nouveaux horizons pour différents champs d'applications sémantiques et de traitement automatique de la langue française qui utilisent le LVF. Deuxièmement, les standards W3C du web sémantique représentent des langages sophistiqués qui offrent un niveau de qualité supérieur d'application et d'interopérabilité entre les applications.

Dans ce chapitre, nous allons présenter le processus de transformation du document XML du LVF en une ontologie OWL. Nous allons en premier lieu présenter le schéma conceptuel de l'ontologie LVF, ensuite nous allons décrire la méthode utilisée pour la transformation automatique ainsi que quelques détails sur la conversion avec les feuilles de style XSLT et nous terminons par la présentation du résultat de l'ontologie sur le logiciel Protégé[13].

5.1. Conception de l'ontologie LVF

Une ontologie définit la structure et l'organisation des concepts (classes) et des propriétés du domaine à modéliser. Comme LVF est déjà organisé selon une hiérarchie que ce soit dans la version Excel ou la version XML, nous nous sommes basée sur celle-ci pour définir les différents concepts et propriétés de notre ontologie.

[13] http://protege.stanford.edu/

Il existe plusieurs méthodologies pour développer des ontologies. Ici nous allons suivre les étapes présentées dans **[21]**. Tout au long de ce processus, nous discutons les détails de conception du LVF et les décisions de modélisation que nous avons prises.

5.1.1. Etape 1 : Définition de la portée de l'ontologie

Nous définissons le domaine et la portée de l'ontologie du LVF en répondant aux questions suivantes :

- Quel domaine va couvrir l'ontologie LVF? dans quel but utiliserons-nous cette ontologie?
- À quels types de question l'ontologie devra-elle fournir des réponses?
- Qui va utiliser cette ontologie ?

L'ontologie LVF va représenter une ressource lexicale ou un dictionnaire linguistique des verbes français. Principalement, l'ontologie va être utilisée dans le domaine linguistique ou le traitement automatique de la langue française. Le but principal de l'ontologie LVF est d'apporter plus de sémantique et de sens aux textes français traités automatiquement par des agents logiciels dans différentes applications telle que l'annotation sémantique. Cela permet aussi d'apporter des solutions pour les applications web qui veulent exploiter la ressource LVF étant donné que le standard utilisé pour la représentation des ontologies est un standard web explicite défini par la W3C.

L'ontologie LVF fournit des réponses à différentes requêtes concernant les verbes français, par exemple, leur utilisation et leur emploi dans la langue, leur sens, leur construction syntaxique ...etc. Voici quelques questions auxquelles peut répondre l'ontologie LVF :

- Quels sont les différents emplois d'un verbe dans la langue?
- Quel est le sens de chaque entrée? Quelle est sa construction syntaxique?
- Quel est la nature d'un verbe (transitif direct/indirect, intransitif, pronominal) ?
- Quels sont les verbes ayant les mêmes caractéristiques syntaxiques ou sémantiques ?

5.1.2. Étape 2 : Énumérer les termes importants

Pour mener à bien la conception de notre ontologie, il est utile d'établir une liste exhaustive de tous les termes qui vont constituer cette ontologie sans se soucier des chevauchements entre les

concepts, ni des relations entre les termes ou tout autre propriété des concepts, ni de déterminer si les concepts sont des classes ou des facettes. Dans le cas du LVF, il est impossible d'énumérer tous les termes car c'est un dictionnaire très riche en vocabulaire, mais il est possible d'énumérer quelques termes importants présentés dans la structuration du dictionnaire (voir la figure 5.1) :

Verbes	Date	auxiliaire
Verbe	Version	der-able
Entrée	Nb	der-ment
Mot	ID	der-eur
Domaine	Nom	desc
Classes	Générique	No
Operateur	Semantico-syntaxique	opérateurs-abréviations
Sens	Construction syntaxique	niveau
Phrases	Groupe	Der-ant
Phrase	Sous-groupe	Der-age
Conjugaison	Type	Der-oir
Dérivation	Circonstant	Der-ure
Construction	Sujet	Der-ion-eur
Schème	Objet	Der-e
Lexique	Nbmots	
Code-schemes	Sous-classe-syntaxique	
Groupes-sémantico-syntaxique	Compléments	
Conjugaison-auxiliaire	Auxiliaires	
Domaines-niveaux	Domaines	
Abbreviations	operateurs	

Figure 5.1 *Liste de quelques termes LVF*

Ces termes ont été récupérés à partir du fichier XML du LVF qui contient tout le vocabulaire de la ressource. Dans la prochaine étape, on définira les classes et la hiérarchie des classes ainsi que leurs propriétés respectives.

5.1.3. Étape 3 : Définir les classes et la hiérarchie taxinomique des classes

La hiérarchie taxinomique des classes est développée par la définition des concepts les plus génériques et se poursuit par la spécialisation des concepts. On pourra par la suite définir une classe pour chaque concept. Par exemple, on peut commencer par créer des classes pour les concepts **`Classes`** et **`Generique`**. Dans ce cas, il est évident que la classe **`Classes`** va être la super-classe de la classe `generique`. Idem pour `Derives` et `Der-able`. La figure 5.2 présente la hiérarchie des classes de l'ontologie LVF :

```
Verbe
Entree
Phrase
Scheme
Dérivation
Nom
Lexiques
Operateur
Sens
Domaines-niveaux
      Domaines
      Niveaux
Operateurs-abreviations
      Abreviations
      Operateurs
```

```
Classes
      Generique
      Groupes-semantico-
      syntaxique
      Semantico-syntaxique
      Sous-classe-syntaxique
Codes-schemes
      Circonstants
      Complements
      Sujets
      Types
Conjugaisons-auxiliaires
      Auxiliaires
      Conjugaisons
Derives
      Der-able
      Der-age
      Der-ant
      Der-e
      Der-ion-eur
      Der-ment
      Der-oir
```

Figure 5.2 *Hiérarchie des classes du LVF*

Nous expliquerons dans la deuxième section le processus d'extraction de ces classes ainsi que quelques détails de conception, pour l'instant nous présentons le résultat final de la génération des classes.

5.1.4. Définir les propriétés des classes (attributs)

Après avoir défini la structure des classes, nous devons décrire la structure interne des concepts en définissant pour chaque classe l'ensemble de ses propriétés ou attributs. Toutes les sous-classes héritent des propriétés de leurs super-classes. Le tableau de la figure 5.3 liste les propriétés avec leurs classes correspondantes.

Property	**Domain**	**Range**
Has-classe	Entree	Sous-classe-syntaxique
Has-conjugaison	Entree	Conjugaison
Has-construction	Entree	Construction
Has-derivation	Entree	Derivation
Has-domaine	Entree	Domaine
Has-entree	Verbe	Entree
Has-lexique	Entree	Lexiques
Has-nom	Entree	Nom
Has-operateur	Entree	Operateur
Has-phrase	Entree	Phrases
Has-semantico-syntaxique	Generique	Semantico-syntaxique
Has-sens	Entree	Sens
Has-sous-classe-syntaxique	Semantico-syntaxique	Sous-classe-syntaxique

Figure 5.3 *Liste des propriétés objet (ObjectProperties)*

Chaque propriété définit une relation sémantique entre deux classes. Par exemple, la propriété `Has-classe` définit une relation entre la classe `entree` et la classe `sous-classe-syntaxique` qui veut dire que chaque instance de la classe `entree` a une classe syntaxique. Outre les propriétés identifiées entre les classes, on définit les attributs de chaque classe. La

figure 5.4 indique tous les attributs des classes présentées ci-haut ainsi que leurs facettes qui décrivent la valeur du type, les valeurs autorisées, la cardinalité, et d'autres caractéristiques de valeurs que les attributs peuvent avoir.

Property	**Domain**	**Range**
autre-forme	entree	xs:string
auxiliaire	conjugaison	xs:string
ciconstant	scheme	xs:string
code	conjugaisons, niveaux, circonstants, domaines, auxiliaires, sujets, compléments, types, der-~	xs :NMTOKEN, xs :integer, xs :NCName
Compl-prep	scheme	xs:string
complement	entree	xs:string
date	*verbe*	xs:date
def	operateurs, abbreviations	xs:string
defectif	conjugaison	xs:token
Der-able	derivation	xs:string
Der-age	derivation	xs:string
Der-ant	derivation	xs:string
Der-e	derivation	xs:string
Der-eur	derivation	xs:string
Der-ion	Derivation	xs:string
Der-ment	derivation	xs:string
Der-oir	derivation	xs:string
Der-ure	derivation	xs:string
description	circonstants, complements	xs:string
Forme-etre	entree	xs:token
groupe	conjugaison	xs:string
id	verbe	xs:ID
mot	verbe	xs:string
nb	verbe, nom	xs:positiveInteger
nbmots	lexiques	xs:integer
negative	entree	xs:token

no	lexiques, entree	xs:positiveInteger, xs:integer
objet	scheme	xs:string
paronyme	verbe	xs:IDREF
pronominal	entree	xs:token
remplacement	nom	xs:string
sous-groupe	conjugaison	xs:string
sujet	scheme	xs:string
type	scheme	xs:string
version	verbe	xs:decimal
desc	sous-classe-syntaxique, semantico-syntaxique, generique	xs:string

Figure 5.4 *Liste des attributs (DataProperties)*

Toutes les étapes qu'on a vues dans cette section représentent une phase de conception et de modélisation de l'ontologie LVF dans le but de définir et d'explorer le vocabulaire ainsi que le modèle de données du LVF. Dans la section prochaine, on va essayer de générer le schéma conceptuel de l'ontologie en appliquant une transformation automatique à partir des fichiers XML du LVF.

5.2. Transformation automatique de XML à OWL

Plusieurs stratégies pour la transformation de XML en OWL ont été proposées. Certaines approches **[22] [23]** proposent une méthode générique de transformation XML en modèle OWL à partir de XML schéma et des données du fichier XML, d'autres pensent qu'il est impossible de proposer une approche automatique convenable pour une transformation automatique complète de XML vers OWL, car XML ne définit aucune contrainte sémantique. Contrairement à cela, d'autres approches considèrent qu'il y a une sémantique dans les documents XML qui peut être découverte à partir de la structure des documents, en l'occurrence l'approche de Melnik **[24]**.

Même si XML n'est pas censé représenter des informations sémantiques ou représenter de la sémantique entre les données, les balises imbriquées peuvent représenter une relation `is-a` ou `part-of` ou `subType-of`. On peut considérer la structure XML comme relationnelle et se baser sur celle-ci pour obtenir le modèle OWL.

Le modèle de données XML décrit un arbre de nœuds, par contre le modèle OWL peut être représenté à base de triplets RDF sujet-prédicat-objet. Nous exploiterons donc la structure d'arbre XML pour générer la hiérarchie de classes correspondante. Le schéma XML, qui définit la structure de l'arbre XML, nous servira pour générer le modèle de l'ontologie : les classes et propriétés. Le fichier XML contient les données du document ce qui nous permet de générer les instances des classes créées précédemment. On peut diviser le processus de transformation automatique en deux parties :

- Génération automatique du modèle de l'ontologie : la transformation du schéma XML en classes et propriétés OWL représentant le modèle de base de l'ontologie.
- Génération automatique des instances de l'ontologie : le peuplement de l'ontologie à partir du fichier XML

5.2.1. Génération automatique du modèle de l'ontologie :

Le modèle de l'ontologie est généré automatiquement à partir du schéma XML du fichier LVF. Un schéma XML est un fichier qui permet de décrire la structure d'un document XML, plus précisément, il définit les éléments/nœuds et les attributs XML ainsi que leurs types de données, il permet aussi de définir l'ordre d'imbrication des nœuds XML c'est-à-dire quel élément est l'élément parent ou l'élément fils. Un document XML est validé par son schéma XML correspondant dans le but de vérifier la consistance des données dans le document.

Comme le schéma XML définit la structure et les facettes des données d'un fichier XML, on va l'utiliser pour générer automatiquement la structure de notre ontologie. On suppose que le document XML contient une structure relationnelle entre les données et on déterminera la signification et les relations possibles entre les éléments du document XML. Les nœuds du document XML peuvent représenter des classes car ils représentent des concepts dans la ressource LVF tels que Verbe, Entrée, Domaine, Sens, Opérateur …etc. L'imbrication des nœuds peut dans certains cas indiquer la présence d'une relation de type `is-a` ou `part-of` mais dans notre cas, on considère la relation de type `Has-` tel que :

- Un verbe a des entrées

- Une entrée a un domaine, une classe, un opérateur, des phrases, un sens, une construction, un lexique, un nom ...etc.

Les attributs d'un élément XML représentent en général les attributs de la classe correspondant au même élément XML.

Le document XML du LVF définit les données sur les verbes et les entrées. Cependant, Il existe d'autres fichiers XML qui apportent des informations supplémentaires sur les classes, les schèmes, les codes de conjugaison, les codes des opérateurs et de dérivation. Ces fichiers XML décrivent certains détails importants pour la compréhension des codes utilisés dans le LVF tels que les codes des opérateurs, les schèmes syntaxiques, les codes de conjugaisons ainsi que les codes des différentes classes. Les schémas XML de ces fichiers ont été aussi exploités dans le processus de transformation afin de compléter le modèle de données de l'ontologie LVF pour une représentation plus complète du LVF.

Nous avons utilisé une feuille de style XSLT (voir annexe II) pour générer les classes, leur hiérarchie et les propriétés. Cette feuille de style prend en entrée le schéma XML du document LVF pour produire un modèle de l'ontologie LVF écrit en OWL. Le fichier résultant va contenir la définition des classes et de la hiérarchie des classes, la définition des propriétés objets - *ObjectProperties* (les relations) et des propriétés de données - *DataProperties* (les attributs)

Nous présentons dans la figure 5.5 les règles de base utilisées pour la transformation XSD en OWL. Les éléments `<xs:element>` avec un `<xs:complexType>` correspondent à une déclaration `<owl:Class>` en OWL, ainsi, on suppose que tous les nœuds de l'arbre XML du LVF sont des classes sauf quelques cas d'exception. Les éléments XML imbriqués présentent dans certains cas une propriété de type objet (`ObjectProperty`) qui se traduit en XSD par `<xs:complexType>` contenant une séquence `<xs:sequence>` de `<xs:element>` c'est-à-dire que certains nœuds parents peuvent avoir une relation sémantique avec leurs nœuds descendants. Les `<xs:attribute>` représentent des attributs en OWL de type `DataProperty` et donc on les transforme en `<owl:DataProperty>`.

XSD	**OWL**
<xs:element name="**verbe**"> <xs:complexType> Ou <xs:element name="**verbe**" type="texte"/>	<owl:Declaration> <owl:Class IRI="**#verbe**"/> </owl:Declaration>
<xs:element name="entree"> <xs:complexType> <xs:sequence> <xs:element ref="**domaine**"> </xs:sequence> </xs:element>	<owl:Declaration> <owl:ObjectProperty IRI="#Has-**domaine**"/> </owl:Declaration>
<xs:attribute name="**date**" type="xs:date"/> Ou <xs:attribute name="**date**"> <xs:simpleType> Ou <xs:complexType> <xs:simpleContent> <xs:extension base="texte"> <xs:attribute> Ou <xs:complexType> <xs:simpleContent> <xs:restriction base="texte"> <xs:attribute>	<owl:Declaration> <owl:DataProperty IRI="**#date**"/> </owl:Declaration> <owl:DataPropertyRange> <owl:DataProperty IRI="**#date**"/> <owl:Datatype IRI="xs:date"/> </owl:DataPropertyRange> <owl:DataPropertyDomain> <owl:DataProperty IRI="**#date**"/> <owl:Class IRI="#..."/> </owl:DataPropertyDomain>
<xs:complexType> not(@mixed) et not(<xs:complexType>/<xs:attribute>/<xs:simpleType>) et not(<xs:complexType>/<xs:simpleContent>) et count(<xs:complexType>/<xs:sequence>/<xs:element>)=1	<owl:SubClassOf> <owl:Class IRI="#..."/> <owl:Class IRI="#..."/> </owl:SubClassOf>

Figure 5.5 *Règles de transformation XSD à OWL*

Le tableau de la figure 5.6 présente les règles de base de transformation des autres fichiers XSD (classesGeneriques.xsd, operateurs.xsd, scheme.xsd, codeConjugaison.xsd, domaine.xsd, classes.xsd …etc.). Ces règles peuvent parfois changer en fonction de la classe à traiter par exemple, `<owl:DataProperty>` peut être généré à partir d'une autre configuration pas nécessairement à la rencontre d'un élément `<xs:attribute>` mais en règle général c'est le cas.

XSD	**OWL**

`<xs:element>`
`  <xs:complexType>, not (@mixed)`
	`<owl:Declaration>`
	`  <owl:Class IRI="#{…}"/>`
	`</owl:Declaration>`

`<xs:complexType>`
`  <xs:sequence>`
`  <xs:element>, not(@maxOccurs)`
	`<owl:SubClassOf>`
	`  <owl:Class IRI="#{…}"/>`
	`<owl:Class IRI="#{…}"/>`
	`</owl:SubClassOf>`

`<xs:element>`
`  <xs:complexType>`
`  <xs:attribute>`
`Ou`
`<xs:element>`
`  <xs:complexType>`
`  <xs:sequence>`
`  <xs:element>, not(@maxOccurs)`
	`<owl:Declaration>`
	`<owl:DataProperty`
	`IRI="#{…}"/>`
	`</owl:Declaration>`

Figure 5.6 *Règles de transformation des autres fichiers XSD*

Après la transformation, le résultat illustré dans la figure 5.7[14] montre les principales classes et leurs relations obtenues après avoir appliqué la feuille de style XSLT sur les fichiers XSD.

Dans le schéma, les relations entre deux classes représentent des propriétés objet (`ObjectProperty`) et définissent des triplets, par exemple la classe entrée est en relation avec plusieurs classes comme domaine et opérateur et donc les triplets sont {`entree`, `Has-domaine`, `domaine`} et {`entree`, `Has-opérateur`, `opérateur`}. Les relations entre les classes (Object Properties) ont été définies dans la première section en haut dans le tableau de la figure 5.3.

[14] Figure produite à l'aide de l'outil OntoGraf du logiciel Protégé

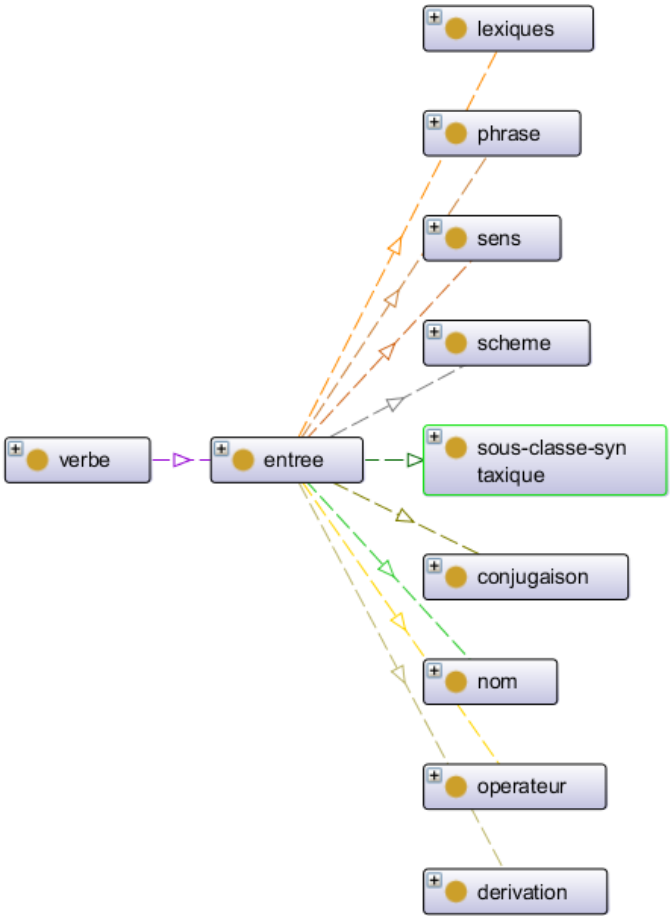


Figure 5.7 *Diagramme des classes de l'ontologie LVF*

La figure 5.8 présente les autres classes de l'ontologie LVF qui ont été extraites à partir des autres fichiers XSD tels que schèmes, opérateurs, classes…etc. Le but de ces classes est de définir explicitement les codes de chaque concept tel que les codes des schèmes, les codes de conjugaison, les codes des opérateurs, des classes …etc.

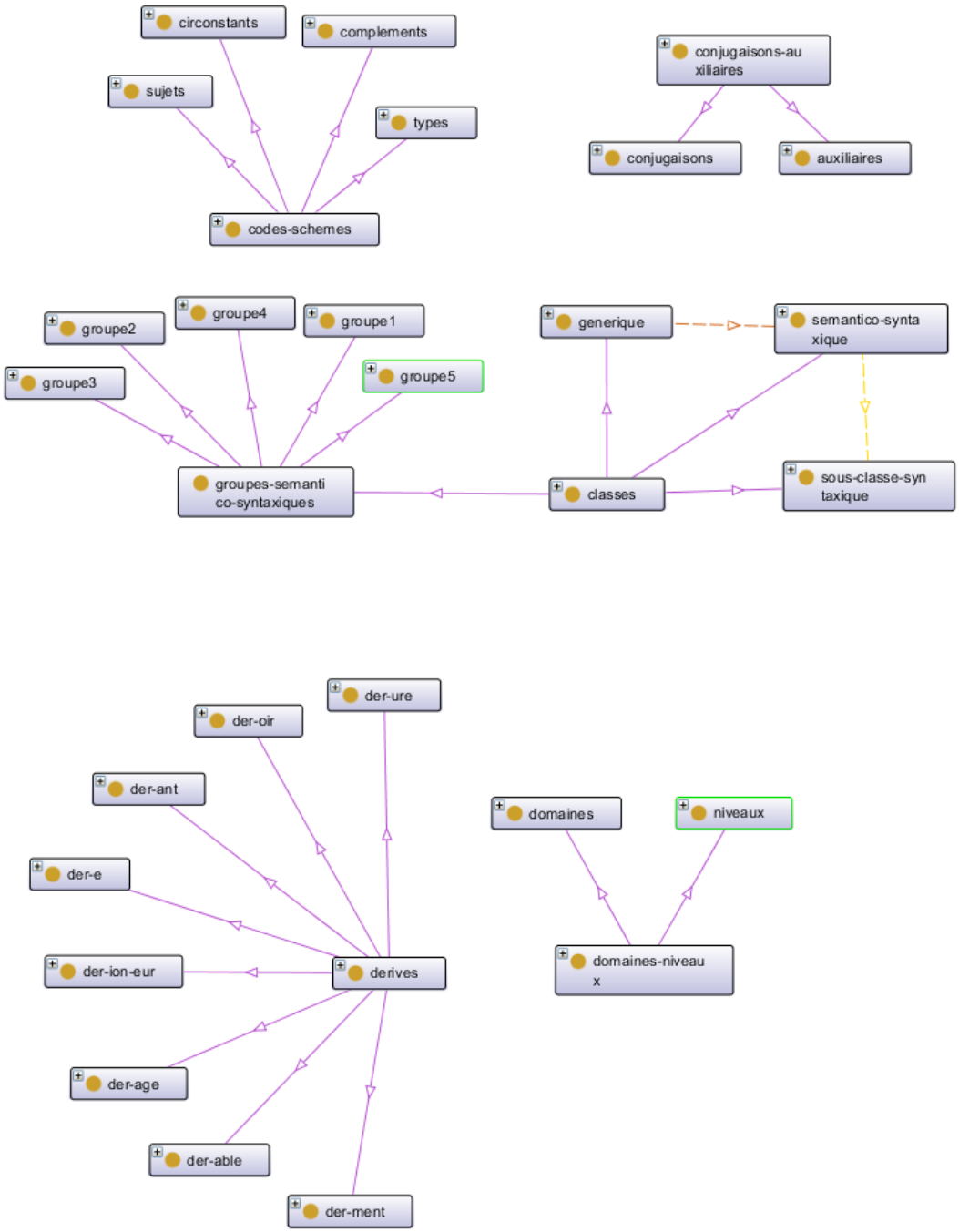


Figure 5.8 *Suite des classes de l'ontologie LVF*

Dans cette section, nous avons présenté le processus de transformation du schéma XML en modèle OWL de l'ontologie. Nous avons défini les classes et leur hiérarchie ainsi que leurs

propriétés. La transformation a été effectuée à l'aide d'une feuille de style XSLT et d'un ensemble de règles de transformation que nous avons détaillées.

La prochaine étape consiste à définir les instances des classes et des propriétés à partir des données du fichier XML. Nous allons voir le processus de transformation automatique XSLT dans la section qui suit.

5.2.2. Génération automatique des instances de l'ontologie

L'étape précédente consistait à générer automatiquement le modèle de l'ontologie, en d'autres termes les classes, leur hiérarchie et les propriétés. Dans cette étape, nous nous intéressons au peuplement de ces classes et propriétés.

Le peuplement de l'ontologie LVF a été effectué à l'aide d'une deuxième feuille de style XSLT qui sert à transformer le document XML d'origine en un document OWL en peuplant l'ontologie avec des instances de classes et à les relier par leurs propriétés et à affecter des valeurs aux attributs.

Dans un fichier XML, les nœuds XML représentent les classes OWL et leur hiérarchie représente les relations entre les classes que nous avons déjà définies à l'aide de la première feuille de style. De ce fait, nous avons parcouru les fichiers XML qui contiennent les instances, en respectant le modèle de classe qui a été généré précédemment.

Contrairement à la première partie, nous n'avons pas établi une liste de règles génériques de transformation XML à OWL. Nous nous sommes basée sur le contenu de chaque fichier XML et nous avons défini les éléments qui représentent des instances. Il n'y a donc pas de règles générales pour la transformation car elles dépendent du contenu de chaque fichier et de ses instances.

Le résultat du peuplement automatique de l'ontologie peut être visualisé dans Protégé, un éditeur d'ontologies qui permet de lire créer et éditer des ontologies dans la plupart des formats : RDF, RDFS et OWL. La figure 5.9 illustre le résultat final de l'ontologie LVF dans Protégé en commençant par l'URI de l'ontologie (http://www.rali.iro.umontreal.ca/Dubois.owl) dans la barre en haut, la hiérarchie des classes dans l'onglet gauche ainsi que les instances et les informations sur leur utilisation dans l'onglet droit par exemple en cliquant sur la classe `Verbe`, on peut visualiser toutes les verbes de l'ontologie LVF.

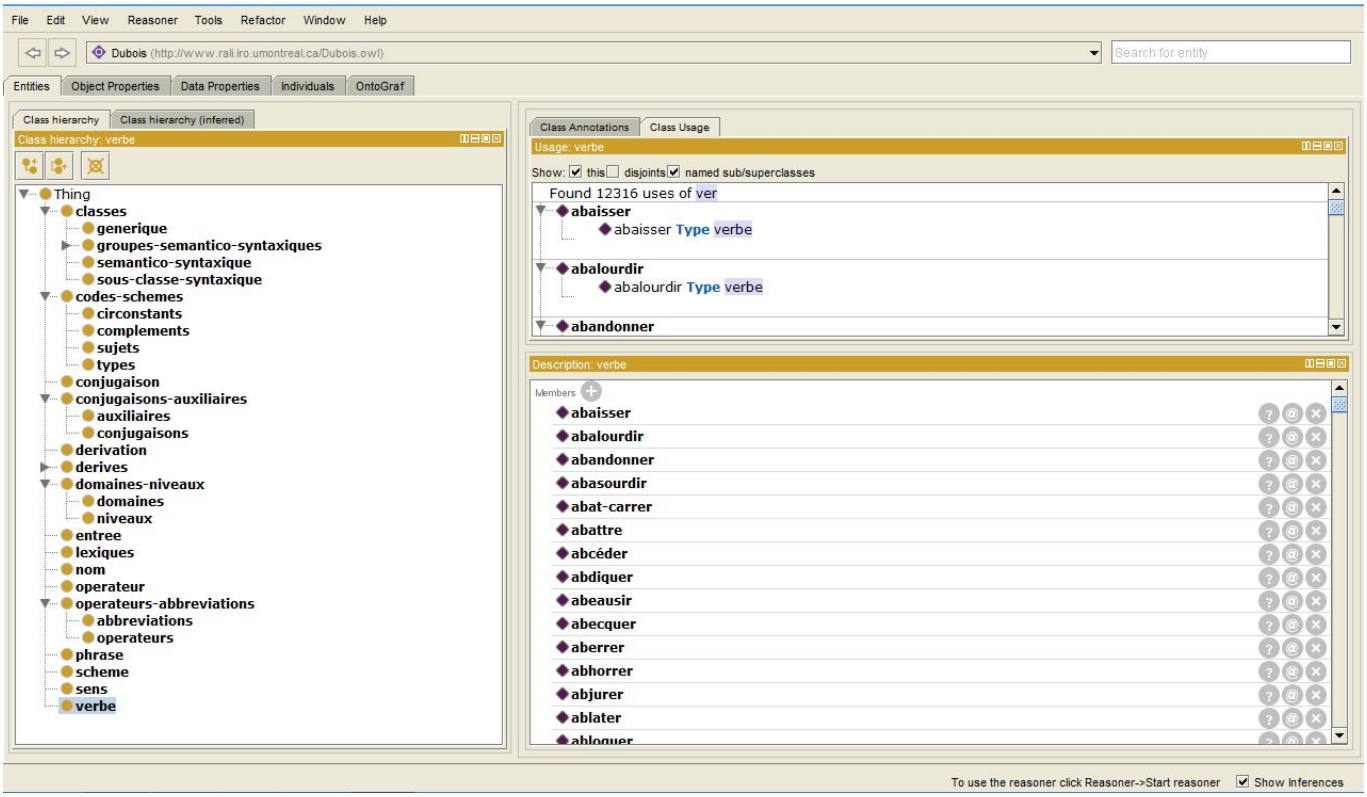


Figure 5.9 *Ontologie LVF dans Protégé*

Difficultés :

La génération automatique du graphe des classes à partir des fichiers XML n'a pas été une tâche triviale. Une des difficultés majeures était de générer un modèle de classe qui respecte le modèle défini au départ dans la phase de conception et le modèle défini par la hiérarchie des documents XML. Donc, l'extraction d'un modèle cohérent et facile à traduire en modèle de classes était la première tâche difficile à résoudre. En général, on a considéré que les nœuds des fichiers XML représentent des classes et leur hiérarchie représente les relations entre les classes sauf que cette logique ne peut pas être appliquée sur l'ensemble des fichiers XML car il y a forcément des nœuds qui ne peuvent pas correspondre à une classe pertinente dans le modèle qu'on veut générer. Le même problème a été rencontré lors de la génération des attributs de chaque classe vu qu'il y a des attributs non pertinents aussi.

Une fois le modèle généré, l'autre difficulté était de définir les instances de chaque classe et d'établir des règles génériques pour l'extraction des instances à partir des fichiers XML en

respectant le modèle de classes généré par les fichiers XML schema. Il fallait définir une manière automatique pour parcourir les fichiers XML afin retrouver le modèle des classes et d'extraire leurs instances.

5.3. Conclusion

La transformation du LVF en format OWL est une étape importante dans ce travail. Nous nous sommes basée sur la structure des fichiers XML du LVF pour créer automatiquement le modèle de l'ontologie et du contenu de ces fichiers pour son peuplement. Cette ontologie regroupe tous les verbes et toutes les informations présentés dans le dictionnaire Dubois et Dubois-Charlier.

La représentation du LVF sous un format normalisé tel que OWL ouvre de nouveaux horizons pour l'utilisation du LVF que ce soit dans le traitement automatique de la langue ou dans le web sémantique. Nous montrons un exemple d'utilisation de l'ontologie LVF dans une application d'annotation sémantique des verbes français décrite dans le chapitre suivant.

Chapitre 6 Annotateur LVF

Partie 1 : Plateforme GATE

Dans cette section, nous allons montrer, à travers une application d'annotation développée sous la plateforme GATE, l'utilisation de la ressource ontologique LVF pour l'annotation d'un texte. Nous introduisons dans un premier temps la plateforme GATE et ses différents modules et fonctionnalités. Ensuite, nous décrivons l'application d'annotation semi-automatique à partir de l'ontologie LVF en vue d'une utilisation pour des applications de traitement automatique de la langue et de la recherche d'informations, par exemple la désambiguisation du sens.

6.1.1. Plateforme GATE

GATE (General Architecture for Text Engineering) est une suite d'outils Java qui a été développée à l'université de Sheffield en 1995 pour le développement d'applications de traitement automatique de la langue. C'est une plateforme d'ingénierie linguistique qui repose sur un ensemble de composants bien définis et réutilisables dans des contextes variés. GATE est défini comme une plateforme flexible qui permet aux utilisateurs d'intégrer leurs propres composants, appelés plugins CREOLE (Collection of REusable Objects for Language Engineering). Les fonctions de base fournies par GATE sont des fonctions de segmentation, d'extraction d'information, d'analyse morpho-syntaxique et d'annotation … etc.

Les composantes de GATE sont divisées en trois types :

- *Processing Resources(PRs)* : ressources de traitement telles que les outils de segmentation, les analyseurs grammaticaux ou morphologiques.
- *Language Resources(LRs)* : ressources lexicales ou linguistiques telles que les documents textuels, les ontologies et les corpus.
- *Visual Resources(VRs)* : ressources visuelles c'est-à-dire les ressources de l'interface graphique (GUI) de GATE.

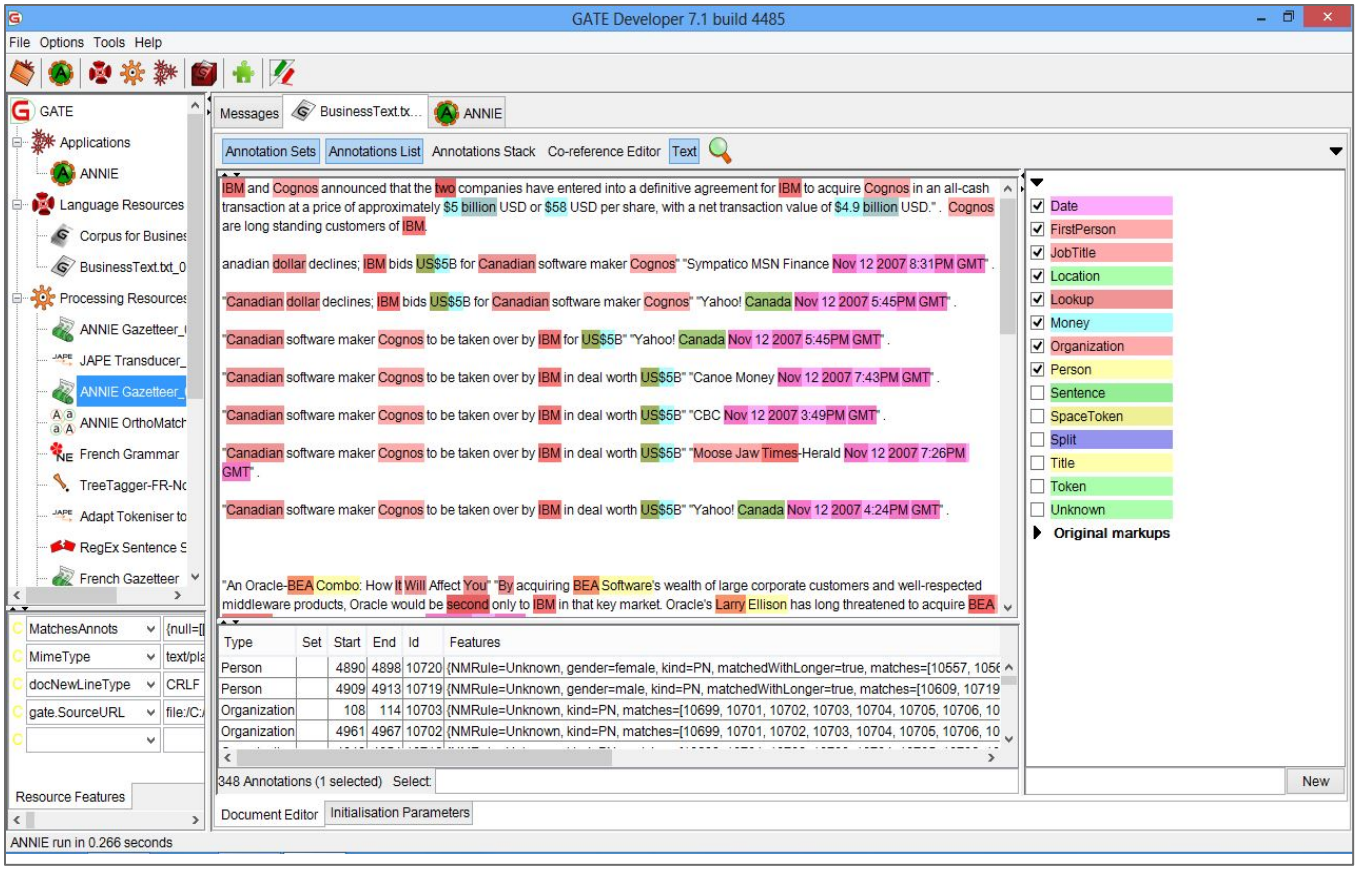


Figure 6.1 *Interface GATE*

La Figure 6.1 montre les différentes parties de l'interface de la plateforme GATE. Le volet gauche est divisé en trois sections : Applications, Language Resources et Processing Resources. La section Application regroupe les différentes applications, chaque application est un pipeline de ressources de traitement à exécuter. La section Processing Resources sert à charger et initialiser des ressources de traitement dans l'interface GATE et la section Language Resources permet de créer des documents et des corpus à traiter.

La partie centrale affiche les documents et les corpus ainsi que les résultats de leurs traitements. Le volet droit représente la liste des annotations générées après le traitement, chaque annotation identifie un mot ou un ensemble de mots dans le document avec une couleur spécifique. La Figure 6.3 montre un exemple de la structure des documents et l'organisation des annotations dans GATE. La structure des documents dans GATE est basée sur l'architecture TIPSTER **[25],** qui organise un texte et ses annotations dans une structure XML.

La partie (1) de la Figure 6.3 définit quelques attributs ou caractéristiques du document, son nom, son créateur, son emplacement ...etc. La partie (2) représente la sérialisation du contenu du texte avec des nœuds. Chaque mot est défini par un nœud de début (`StartNode`) et un nœud de fin (`EndNode`) caractérisés par un Id qui est un numéro qui représente la position du mot dans le texte. La partie (3) du document est la partie la plus importante qui permet de définir les annotations sur le texte. Comme illustré dans la figure, les annotations sont organisées dans un `AnnotationSet` qui regroupe un ensemble d'annotations. Chaque annotation est caractérisée par un ensemble d'attributs tels qu'`Id`, `Type`, `StartNode` et `EndNode`. Un ensemble de caractéristiques (features) est ajouté à chaque annotation ou chaque caractéristique est déterminée par un nom et une valeur. Le tableau de la Figure 6.2 résume la structure des annotations dans GATE et montre un exemple d'annotation d'une simple phrase avec des étiquettes morpho-syntaxiques.

Texte Marie savoure la soupe. |0.....|5 ...|10..|15 ..|20				
Annotations				
Id	Type	StartNode	EndNode	Features
1	Token	0	5	Pos=NAM, kind=word, length=5, orth= upper
2	Token	6	13	Pos=VER, kind=word, length=7, orth=lower
3	Token	14	17	Pos=DET, kind=word, length=2, oth=lower
4	Token	18	22	Pos=NOM, kind=word, length=5, orth=lower
5	Sentence	0	22	

Figure 6.2 *Annotation GATE*

```
<GateDocument version="2">
<!-- The document's features--> …… (1)
<GateDocumentFeatures>
<Feature>
  <Name className="java.lang.String">gate.SourceURL</Name>
  <Value
className="java.lang.String">file:/u/abdiradi/Projet/Problématique.docx</V
alue>
</Feature>
<Feature>
  <Name className="java.lang.String">MimeType</Name>
  <Value className="java.lang.String">application/vnd.openxmlformats-
officedocument.wordprocessingml.document</Value>
</Feature>
<Feature>
  <Name className="java.lang.String">CREATOR</Name>
  <Value className="java.lang.String">m</Value>
</Feature>
</GateDocumentFeatures>

<!—The document content area with serialized nodes --> ……
(2)

<TextWithNodes><Node id="0"/>Marie<Node id="2"/> <Node
id="3"/>savoure<Node id="15"/> <Node id="16"/>la<Node id="19"/> <Node
id="20"/>soupe<Node id="21"/> </TextWithNodes>

<!-- The default annotation set --> …… (3)
<AnnotationSet>
<Annotation Id="1" Type="SpaceToken" StartNode="0" EndNode="5">
<Feature>
  <Name className="java.lang.String">length</Name>
  <Value className="java.lang.String">5</Value>
</Feature>
<Feature>
  <Name className="java.lang.String">kind</Name>
  <Value className="java.lang.String">word</Value>
</Feature>
<Feature>
  <Name className="java.lang.String">string</Name>
  <Value className="java.lang.String">Marie</Value>
</Feature>
</Annotation>
<Annotation Id="2" Type="SpaceToken" StartNode="5" EndNode="6">
<Feature>
  <Name className="java.lang.String">length</Name>
  <Value className="java.lang.String">1</Value>
</Feature>
<Feature>
  <Name className="java.lang.String">string</Name>
  <Value className="java.lang.String"> </Value>
</Feature>
<Feature>
  <Name className="java.lang.String">kind</Name>
  <Value className="java.lang.String">space</Value>
</Feature>
```

Figure 6.3 *Structure d'un document GATE*

6.1.1.1. Processing Resources (PRs)

Les ressources de traitement (Processing Resources) sont généralement appliquées aux textes au sein d'une cascade, autrement dit un pipeline ou une chaîne de traitement de différents outils qui s'exécutent d'une manière séquentielle. Dans GATE, les ressources de traitement sont utilisées pour créer et manipuler automatiquement les annotations sur les documents. Le flot d'entrées/sorties entre les différentes ressources de traitement est représenté sous forme d'annotations. Plusieurs ressources de traitement sont déjà intégrées à GATE mais il est possible d'intégrer d'autres outils existants ou même importer les outils de GATE dans d'autres applications.

GATE est une plateforme qui intègre plusieurs plugins et ressources qui peuvent servir dans plusieurs types de traitements de textes ainsi que dans différentes langues. Un des plugins les plus utilisés pour le traitement de l'Anglais est ANNIE (A Nearly-New Information Extractor), un système d'extraction d'informations et d'entités nommées qui comprend un ensemble de ressources de traitement organisées en pipeline. Les ressources utilisées dans ANNIE sont les suivantes :

- *Tokeniser :*

Un tokeniser permet de diviser le texte en entités lexicales simples telles que les mots, les ponctuations, les numéros, les symboles … etc. Les annotations engendrées par le tokeniser sont de type *Token* et *SpaceToken*.

- *Gazetteers*

Le rôle d'un Gazetteer est d'identifier les entités nommées dans le texte. Il utilise un ensemble de listes de noms et d'abréviations tels que les noms de villes, d'organisations, jours de semaine, prénoms … etc. Le Gazetteer va annoter toutes les entités nommées qu'il trouve dans le texte et qui existent dans ses listes avec une annotation de type *Lookup*.

- *Sentence Splitter*

Un Sentence splitter est un transducteur qui permet de segmenter le texte en phrases. Le splitter utilise les sorties du Gazetteer pour faire une bonne segmentation en phrase car les sorties du Gazetteer vont aider le Sentence splitter à mieux distinguer la fin des phrases, par exemple, ne

pas confondre le point de la fin d'une phrase avec un autre point d'un autre type comme un point d'une abréviation quelconque

- *POS tagger*

Un POS tagger est un tagger grammatical pour l'Anglais, basé sur une version du tagger de Brill **[Hepple2000]** qui utilise un lexique et un ensemble de règles (le résultat d'entraînement d'un large corpus extrait du journal Wall Street) pour l'étiquetage de l'anglais. Il permet de fournir des étiquettes grammaticales sur chaque mot. Dans GATE, les sorties du POS tagger sont sous forme d'annotations dont la catégorie est l'étiquette grammaticale.

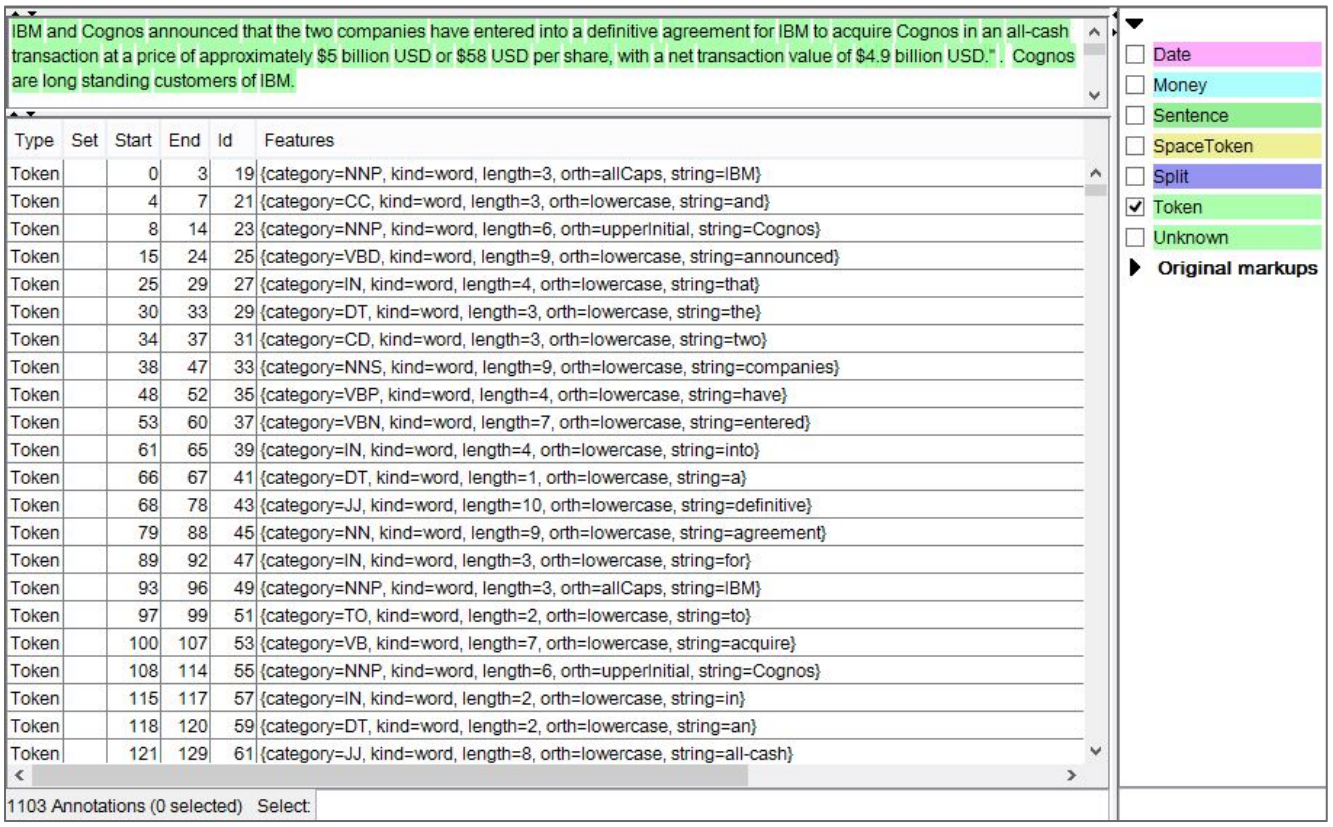


Figure 6.4 *Sorties du POS sur GATE*

La Figure 6.4 montre le résultat de l'étiquetage grammatical dans l'interface de GATE. Nous constatons que le tagger va ajouter aux annotations de type Token la caractéristique (feature) Category qui contiendra l'étiquette grammaticale du mot concerné.

- *Semantic tagger (NE Transducer)*

Le but du Semantic tagger est de manipuler les annotations générées par les phases précédentes, afin de produire des annotations en sortie. Il est appelé aussi NE transducer (Transducteur d'entités nommées)

Le Gazetteer fournit des annotations de type Lookup pour toute entité nommée rencontrée dans le texte, le Semantic tagger va générer des annotations à l'aide des grammaires JAPE (défini plus bas) qui vont analyser les annotations Lookup et en produire des annotations plus sémantiques qui précisent les entités nommées de type Lookup (par exemple : Personne, Organisation, Pays, Date … etc.)

Le tagger sémantique est basé sur le langage JAPE (Java Annotation Pattern Engine), qui est un langage dérivé de CPSL- Common Pattern Specification Language- et qui permet de définir des expressions régulières sur les annotations dans les documents afin d'effectuer des traitements spécifiques.

- *Langage JAPE*

Le langage JAPE (Java Annotation Patterns Engine) est un langage qui permet de décrire des expressions régulières sur des annotations GATE. Les expressions régulières sont généralement appliquées sur des chaines de caractères ou des séquences d'éléments mais dans JAPE, elles sont appliquées pour des structures de données plus complexes, en l'occurrence le graphe des annotations GATE.

Une grammaire JAPE est présentée par un nombre de phases, chacune consistant en un ensemble de règles de pattern/action. De ce fait, une grammaire est divisée en deux types de règles, les règles LHS (Left-Hand-Side) qui décrivent les expressions régulières autrement dit, les motifs ou patrons sur les annotations (Annotation Pattern), et les règles RHS (Right-Hand-Side) qui consistent à décrire le traitement à appliquer sur les annotations matchées en LHS. Un exemple de grammaire JAPE est affiché dans la Figure 6.5.

Dans l'exemple, on voit que la partie LHS est celle qui précède le '-- >' et la partie RHS est celle qui le suit. On cherche à matcher le texte avec une annotation de type *Lookup* avec la valeur de *feature* égale à « *firstname* », un *Token* dont le string est égal à « *and* » et un autre *Token* dont

le premier caractère est en majuscule. La partie RHS associe une annotation 'Person' au texte correpondant.

```
Phase: Name
Input: Token Lookup
Options: control= appelt debug = false
Rule: PersonAndPerson
Priority: 30
//Anne and Kenton
({Lookup.majorType == firstname}):person1
({Token.string == "and"})
({Token.orth == upperInitial}):person2
-->
:person1.Person = {rule = "PersonAndPerson"},
:person2.Person = {rule = "PersonAndPerson"}
```

Figure 6.5 *Exemple de grammaire JAPE*

6.1.1.2. Les ontologies dans GATE

Les ontologies dans GATE sont considérées comme des ressources linguistiques (Language Resources). GATE fournit une API pour la modélisation et la manipulation des ontologies mais il offre aussi des plugins pour les ontologies tels que Ontology plugin, Ontology_Tools et Ontology_BDM_computation. Ces plugins ont été créés pour fournir une interface graphique ainsi que pour faciliter la manipulation des ontologies dans l'interface de GATE.

- *Ontology/Ontology_Tools plugins*

Le plugin Ontology permet aux utilisateurs de GATE de créer, d'importer une ontologie sur l'interface graphique. OWLIM Ontology Language Resource représente une des ressources du plugin Ontology. Il permet de créer, charger, sauvegarder et mettre à jour les ontologies dans GATE. Afin de charger une ontologie dans un répertoire OWLIM, l'utilisateur doit fournir quelques paramètres de configuration tels que l'URL de l'ontologie, son espace de noms, le format de l'ontologie (RDF/XML, N3, NTriples et turtle), les URLs des ontologies externes à importer ainsi que leurs espaces de noms …etc.

Le plugin Ontology_Tools implémente les outils de manipulation des ontologies tels qu'Ontology Editor/Viewer ainsi que l'outil d'annotation ontologique manuelle OAT.

L'outil Ontology Editor/Viewer est utilisé pour naviguer et visualiser les différentes ressources d'une ontologie. C'est un outil qui permet aussi de mettre à jour l'ontologie en utilisant des fonctions d'ajout/suppression ou modification de classes, instances, propriété ou restriction.

L'outil d'annotation OAT (Ontology Annotation Tool) permet aux utilisateurs d'annoter manuellement un texte au regard d'une ou de plusieurs ontologies chargées dans GATE. L'outil OAT annote un texte avec les ressources ontologiques telles que les classes, les instances et propriétés. La Figure 6.6 montre un exemple d'annotation à partir d'une ontologie avec l'outil OAT. Toutes les annotations en couleurs affichées sur le texte correspondent à une plusieurs ressources de l'ontologie.

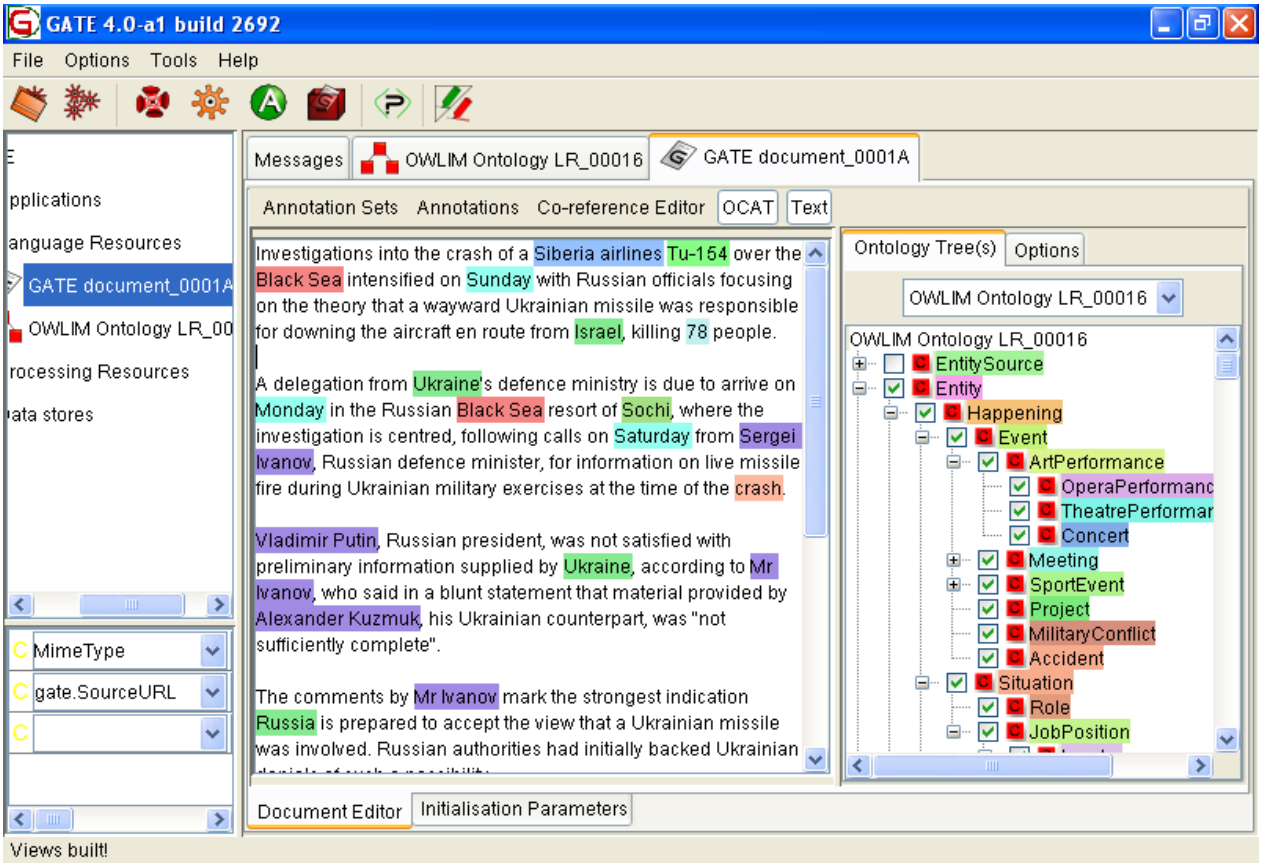


Figure 6.6 *Outil d'annotation OAT*

Partie 2: Outil d'annotation sémantique à partir du LVF

6.2.1 Introduction

Dans cette section, nous allons montrer les étapes à suivre pour l'annotation semi-automatique des verbes à partir de l'ontologie LVF. Le processus d'annotation se divise en deux parties. La première partie est le prétraitement du texte (tokenisation, étiquetage grammatical...etc), ceci est fait grâce aux modules intégrés dans la plateforme GATE plus précisément à l'aide du plugin destiné au traitement automatique du Français Lang_French ainsi que le Tagger Framework. La deuxième partie représente le processus d'annotation semi-automatique à partir de l'ontologie.

6.2.2 Présentation de l'application d'annotation

6.2.2.1 Prétraitement

Avant de procéder à l'annotation sémantique, il est indispensable de passer par l'étape du prétraitement qui consiste à étiqueter un texte avec des étiquettes grammaticales pour permettre un traitement automatique par la suite. Dans GATE, le plugin Français contient une application pipeline qui comprend un TreeTagger, géré par le plugin Tagger Framework, qui assure l'étiquetage grammatical du Français. Le plugin Français représente en quelque sorte l'équivalent du système ANNIE présenté dans la partie précédente sauf qu'il a été adapté pour le traitement de la morphologie et de la syntaxe françaises. Le plugin Français comprend des modules tels que French Tokenizer, French Gazetteer, RegEx Sentence Splitter, TreeTagger …etc dont nous avons déjà expliqué le principe dans la partie 1.

- **Tagger Framework**

GATE a mis à la disposition des utilisateurs un cadre générique, le Tagger Framework, qui peut être configuré pour incorporer différents étiqueteurs externes de différentes langues dans GATE. La motivation pour ce plugin est de permettre aux utilisateurs d'avoir un support pour intégrer n'importe quel tagger externe. Le problème avec les taggers externes est la tokenisation : Généralement, le Tokenizer GATE utilisé dans un pipeline génère en sortie des annotations de type « Token », qui sont exploitées par le tagger de GATE. Cependant, la plupart des taggers externes utilisent leur propre tokenisation. Dans ce cas, **(1)** soit le tagger accepte un texte pré-tokenisé, **(2)** soit il utilise les tokens générés par le tagger externe et les importe sous le format de GATE (des annotations de type « Token »).

Dans cette application, nous avons utilisé le TreeTagger développé par Helmut Schmid **[26][27]** qui a été testé avec succès sur plusieurs langues (Allemand, Anglais, Français, Italien, Espagnol, Bulgare, Russe, Grecque, Portugais, Chinois, Latin ..etc). En ce qui concerne la tokenisation, le TreeTagger que nous avons utilisé accepte les textes pré-tokenisés. De ce fait, le module « Adapt Tokenizer to tagger »- voir la Figure 6.7- joue le rôle d'intermédiaire entre la tokenisation GATE et le TreeTagger afin d'adapter le texte tokenisé par GATE pour qu'il soit exploitable par le TreeTagger. Le schéma de la Figure 6.7 décrit le pipeline d'applications du plugin Français. On distingue plusieurs modules qui sont exécutés selon un ordre séquentiel, les sorties d'un module représentent les entrées du prochain.

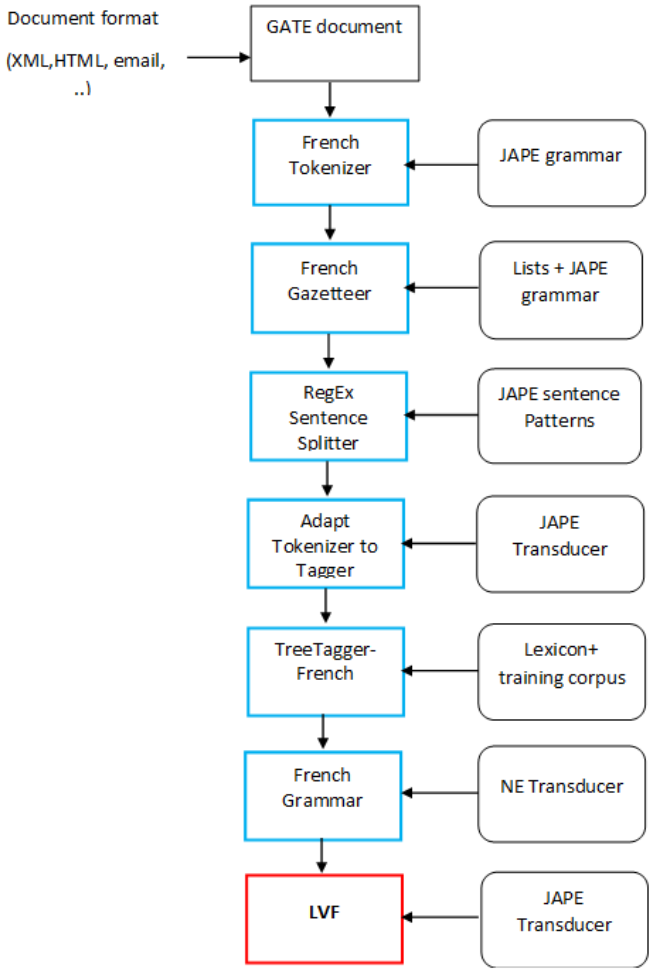


Figure 6.7 *Pipeline du plugin Français*

6.2.2.2 Annotation à partir de l'ontologie LVF

Nous avons déjà vu le processus de prétraitement en utilisant GATE. Dans ce qui suit, nous allons expliciter le processus d'annotation sémantique des verbes à partir de l'ontologie LVF.

Dans la ressource LVF, les verbes sont organisés selon leur schème syntaxique et sémantique par un ensemble d'entrées de verbes telles que « `abaisser01`, `abaisser02` **...** etc ». Chaque entrée est définie par un schème syntaxique et un opérateur sémantique correspondant aux différents cas d'utilisation du verbe dans la littérature. De ce fait, il serait intéressant d'annoter les verbes avec leurs entrées correspondantes dans l'ontologie LVF tout en sachant que chaque entrée est en relation avec les autres concepts de l'ontologie tels que le sens, le schème syntaxique, l'opérateur sémantique, le domaine, la conjugaison …etc. Par conséquent, si on veut annoter automatiquement un verbe avec son entrée correspondante, il faut parvenir à déterminer automatiquement le schème sémantique ou syntaxique du verbe en analysant la phrase dans laquelle le verbe apparait. Cependant, l'automatisation de ce processus au complet s'est avérée très difficile car elle exigerait l'intégration de plusieurs ressources comme plugins dans GATE, par exemple un parseur du français ou un analyseur morpho-syntaxique. Dans le cas de l'annotation automatique, on s'intéresse à déterminer automatiquement l'entrée correspondante en analysant les schèmes syntaxiques de chaque entrée (T1101, A30 …etc.). Pour ce faire, il va falloir analyser la construction grammaticale des phrases comme le sujet, le verbe, le complément et les prépositions à l'aide d'un parseur du français, qui consiste à définir la nature et la fonction grammaticale des constituants d'une phrase.

L'approche automatique n'a pas été appliquée dans le cadre de cette application d'annotation car l'intégration d'un parseur du français comme plugin dans GATE est un processus coûteux en termes de temps et de développement. De plus que l'analyse des schèmes syntaxiques ne s'est pas avérée très intéressante vu que deux entrées peuvent avoir deux schèmes syntaxiques similaires ce qui rend l'analyse syntaxique du texte pas très attrayante d'un point de vue d'une annotation déterministe et pertinente. Par conséquent, nous avons opté pour une approche semi-automatique pour la réalisation de cet annotateur. Elle consiste à annoter les verbes avec les différentes entrées possibles accompagnées par leurs sens et leurs schèmes syntaxiques, présentées sous forme d'une liste dans laquelle l'utilisateur pourra supprimer les entrées non pertinentes.

Nous avons développé dans le cadre de ce projet un module GATE de type JAPE transducer (Transducteur JAPE) nommé LVF et qui s'ajoute au pipeline précédent comme décrit dans la Figure 6.7. Ce module, écrit en langage JAPE, va être chargé de l'annotation sémantique des verbes à partir de l'ontologie LVF.

Afin d'annoter les verbes avec les entrées de l'ontologie LVF, nous avons suivi les étapes suivantes :

- Extraction automatique des verbes
- Lemmatisation
- Recherches des entrées dans LVF
- Création des annotations GATE

- Extraction automatique de verbe et lemmatisation

L'extraction automatique des verbes est l'une des premières étapes à considérer dans ce travail. L'extraction des verbes se fait très facilement grâce aux étiquettes grammaticales créées par le TreeTagger dans la phase du prétraitement, et qui sont sauvegardées dans la structure d'annotation de GATE. Nous avons utilisé le langage JAPE pour accéder à ces annotations en définissant les règles qui permettent de récupérer toutes les annotations/étiquettes qui commencent par « VER: ».

La figure 6.8 montre l'ajout des étiquettes grammaticales par le TreeTagger comme caractéristique (feature) nommée *category*.

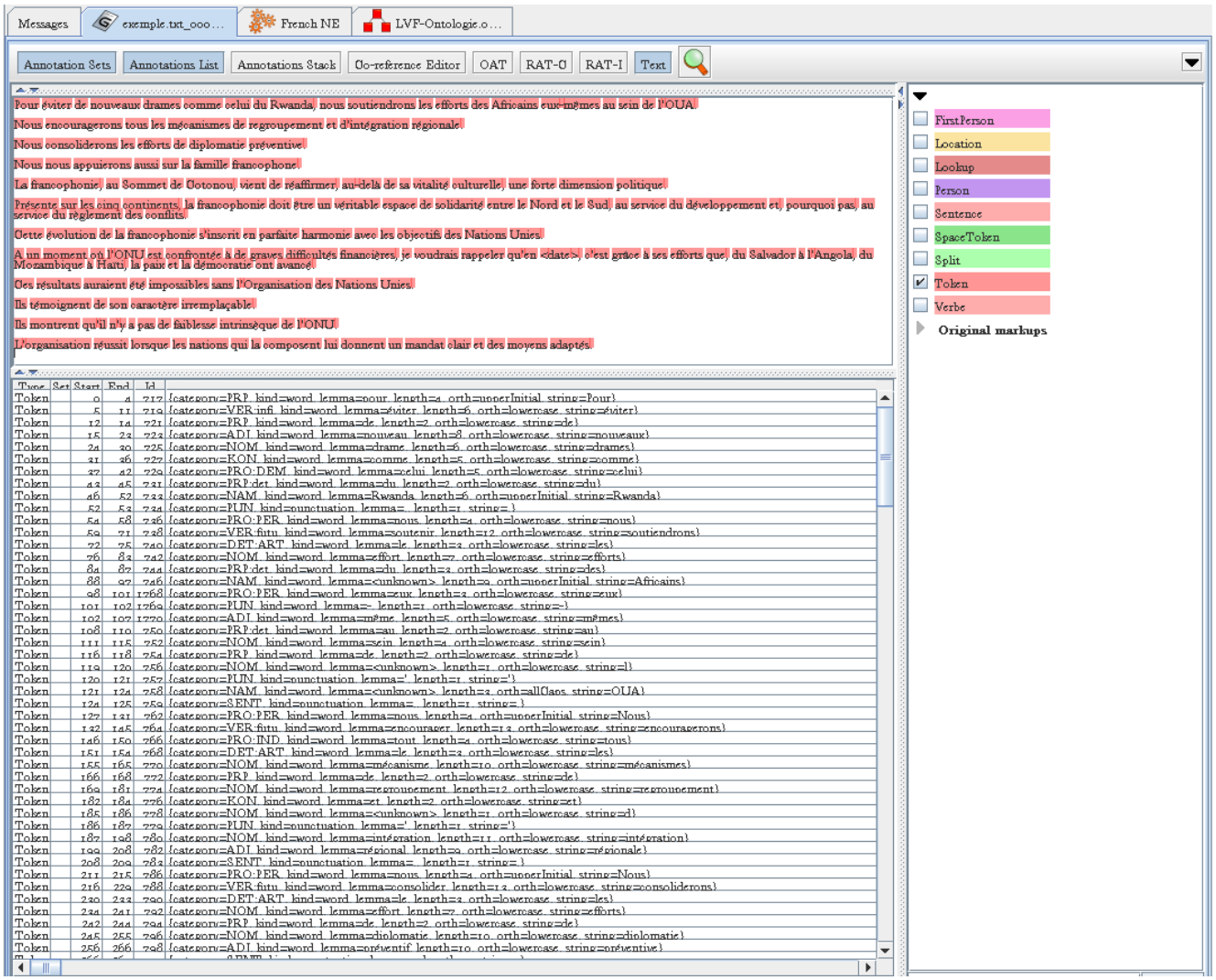


Figure 6.8 *Caractéristique des annotations « Token »*

La liste des étiquettes grammaticales des verbes fournies par le TreeTagger est la suivante :

VER:cond	Verbe au conditionnel
VER:futu	Verbe au futur
VER:impe	Verbe à impératif
VER:impf	Verbe à l'imparfait
VER:infi	Verbe infinitif
VER:pper	Verbe participe passé
VER:ppre	Verbe participe présent
VER:pres	Verbe au présent
VER:simp	Verbe au passé simple
VER:subi	Verbe imparfait subjonctif
VER:subp	Verbe présent subjonctif

Les verbes extraits sont sous plusieurs formes c'est pour cela qu'il faut les lemmatiser pour pouvoir procéder à la recherche de leurs entrées dans l'ontologie LVF. Les lemmes des verbes ont été récupérés directement à l'aide du TreeTagger, qui lui-même fournit en sortie les lemmes des mots traités en plus de leurs étiquettes grammaticales comme on peut le voir dans la figure 6.8, la caractéristique « lemma » est ajoutée après l'exécution du TreeTagger.

- Recherche des entrées correspondantes dans LVF

Comme nous l'avons déjà expliqué auparavant, chaque verbe devrait être annoté avec son entrée correspondante dans le LVF. La détermination automatique de l'entrée peut se faire grâce à l'analyse de son schème syntaxique ou sémantique. Le schème syntaxique d'une entrée est codé selon un modèle bien défini, tel que `T1318`, `P3008` ou `A16`, qui indique la nature du verbe (transitif direct ou indirect, intransitif ou pronominal), le type du sujet et de l'objet ainsi que la nature des compléments.

À cette étape, nous nous sommes basée sur une approche semi-automatique d'annotation des verbes à partir de l'ontologie LVF. Elle consiste à récupérer toutes les entrées de chaque verbe à partir de l'ontologie LVF, à les présenter à l'utilisateur sous forme d'une liste, ce dernier pourra par la suite supprimer les entrées qui ne correspondent pas au sens et à l'emploi du verbe dans la phrase pour ne garder que l'entrée appropriée. Cependant, une liste d'entrées possibles peut être peu significative pour un utilisateur. À cet effet, on a ajouté à chaque entrée d'autres informations à partir de l'ontologie LVF qui sont : le sens et le/les schèmes syntaxiques de chaque entrée. Pour pouvoir déterminer l'entrée correspondante, l'utilisateur pourra soit, apparier le schème syntaxique avec le verbe, soit, procéder à l'élimination des entrées selon le sens de chacune.

6.2.2.3 Description du module LVF

Nous détaillons maintenant certains aspects techniques de ce module. Le module LVF développé dans l'architecture GATE est un transducteur JAPE (Java Annotation Patterns Engine). Le langage JAPE, comme décrit dans le chapitre précédent, permet de décrire des expressions régulières sur des annotations ce qui nous permet de développer des applications d'extraction

d'informations et d'annotation variées. Le schéma de la figure 6.9 décrit le module LVF et ses composantes.

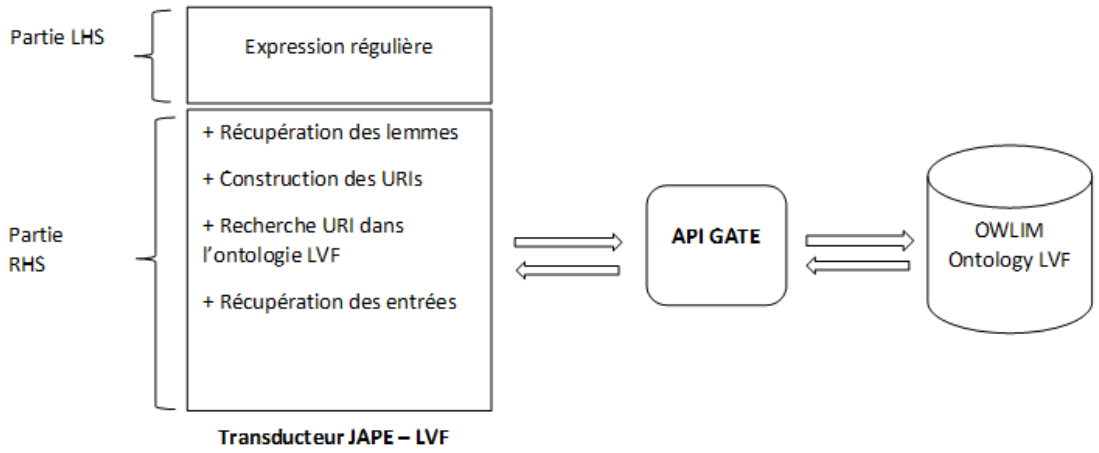


Figure 6.9 *Schéma descriptif du module LVF*

- Partie LHS

La partie LHS (Left-Hand-Side) d'un programme JAPE permet de décrire les expressions régulières pour matcher les annotations. Dans notre cas, la figure 6.10 présente l'expression régulière qui matche toutes les étiquettes grammaticales qui représentent des verbes, définies par le TreeTagger.

La règle « Verbe » est définie pour extraire tous les Token dont la caractéristique *category* est égale à une des étiquettes grammaticales des verbes.

```
Rule: Verbe
(
{Token.category== "VER :cond"} | {Token.category== "VER :futu"} | {Token.category== "VER :impe"} |
{Token.category== "VER :impf"}| {Token.category== "VER :infi"} | {Token.category== "VER :pres"} |
{Token.category== "VER :ppre"} | {Token.category== "VER :pper"} | {Token.category== "VER :simp"} |
{Token.category== "VER :subi"} | {Token.category== "VER :subp"}
): verb
```

Figure 6.10 *Partie LHS (Left-Hand-Side) du programme JAPE.*

- Partie RHS

La partie RHS, écrite en Java, représente la partie principale qui permet de définir le traitement à exécuter si la règle LHS est vérifiée. On va tout d'abord récupérer le lemme de chaque verbe pour pouvoir construire l'URI correspondant au verbe dans l'ontologie LVF. Ensuite, on va rechercher le verbe dans l'ontologie LVF qui est déjà chargée dans GATE à l'aide du plugin OWLIM Ontology. La communication avec l'ontologie se fait grâce à l'API GATE qui fournit un ensemble de fonctions prédéfinies pour la manipulation des ontologies.

Une fois le verbe trouvé, toutes ses entrées sont récupérées facilement car chaque verbe dans l'ontologie LVF est relié à ses entrées par une relation « Has_entree » de type ObjectProperty. Ensuite, nous récupérons le sens et le schème syntaxique de chaque entrée de la même manière car la classe « Entree » est reliée à la classe « Sens » par la relation « Has_sens » et à la classe schème par la relation « Has_scheme ».

Une nouvelle structure d'annotation appelée « Verbe » est créée dans laquelle les caractéristiques (features) représentent les instances de l'ontologie LVF, en l'occurrence les instances de la classe entrée, sens et schème.

La figure 6.11 montre le résultat de l'annotation après l'exécution du module LVF.

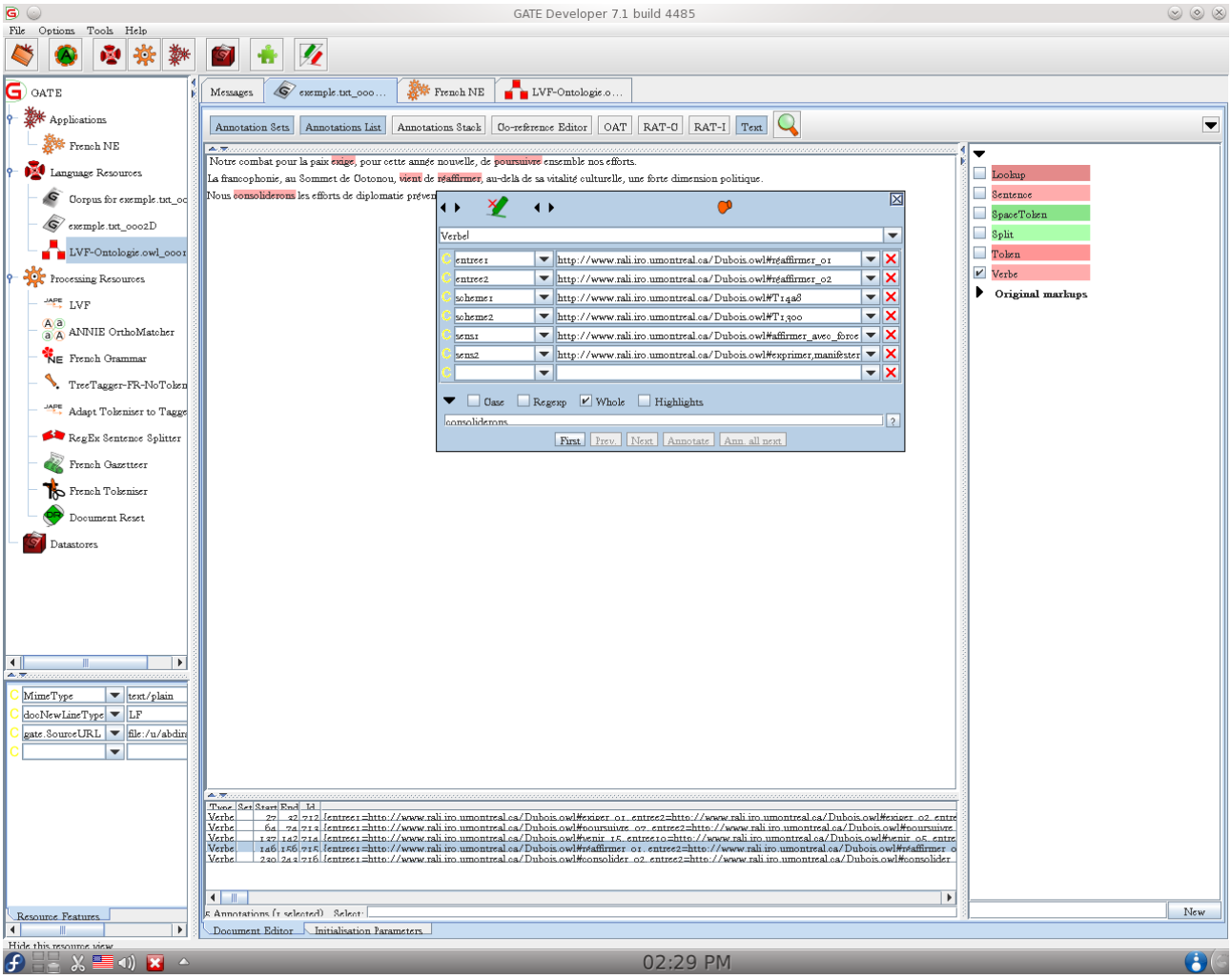


Figure 6.11 *Résultats de l'annotation des verbes*

Après l'exécution du pipeline French NE qui contient le module LVF, l'étiquette Verbe s'ajoute à la liste des annotations de GATE. La liste des entrées est affichée lorsque l'usager clique sur le verbe. Les entrées sont numérotées pour pouvoir retrouver leur sens et schème. Si une entrée possède deux schèmes, ces derniers sont numérotés aussi par exemple (*scheme3-0, scheme3-1*). Il est tout de même possible de supprimer toutes les entrées qui ne correspondent pas au contexte du verbe à partir de la liste des entrées et de sauvegarder par la suite le document avec les annotations pertinentes.

6.2.3 Conclusion

L'application GATE développée dans le cadre de ce travail s'appuie sur l'ontologie LVF pour annoter les verbes français d'une manière semi-automatique. La ressource LVF fournit une structure et une représentation syntaxique et sémantique des verbes très riche et complète ce qui nous a encouragée à exploiter ces avantages dans l'annotation sémantique des verbes. Chaque verbe correspond à une entrée syntactico-sémantique précise dans le LVF qui définit sa nature et son emploi dans la phrase. Cette application développée en langage JAPE permet d'extraire les verbes automatiquement et de proposer une liste d'annotations exhaustive de toutes leurs entrées possibles dans l'ontologie LVF pour l'utilisateur.

L'utilisation de la plateforme GATE s'est avérée très efficace et utile pour la réalisation de cette application. Rappelons que GATE représente une architecture et un Framework très riche en composants et plugins qui permettent de résoudre les problèmes de base les plus courants dans le traitement automatique de la langue notamment la tokenisation, l'étiquetage morpho-syntaxique, l'extraction d'information …etc. Un des avantages incontournables de GATE est la réutilisation des composants et des plugins CREOLE déjà intégrés ainsi que l'implémentation efficace et rapide de méthodes d'analyse et d'annotation de texte (Langage JAPE).

Chapitre 7 Conclusion et travaux futurs

Dans ce mémoire, nous avons tenté de mettre en valeur le dictionnaire LVF (Les Verbes Français) de Dubois et Dubois-Charlier en développant une version ontologique sous le format OWL et en l'exploitant dans des applications du traitement automatique de la langue et du web sémantique. Le LVF est caractérisé par une description sémantique et syntaxique des verbes et une classification selon une hiérarchie de classes syntactico-sémantiques.

Avec l'émergence du web sémantique et la diffusion rapide de ses technologies et standards dans la communauté du traitement automatique de la langue tel que XML, l'intérêt pour certains formalismes plus avancés comme RDF et OWL s'amorce à cause de certains besoins d'interopérabilité et de réutilisation. Notre ontologisation de la ressource LVF ajoute à cette ressource un nouveau mode d'accès sophistiqué et performant pour une meilleure exploitation par d'autres applications. Une ontologie LVF enrichit la ressource avec des capacités de raisonnement et d'inférence en permettant une utilisation éventuelle par les agents logiciels. Il est possible d'imaginer plusieurs cas d'utilisation de cette ontologie dans multiples domaines comme nous l'avons démontré par notre application d'annotation LVF développée dans la plateforme GATE.

Dans cette application, nous avons annoté les verbes français à partir des concepts et instances de l'ontologie LVF, plus précisément à partir des instances de la classe « Entree », « Sens » et « Schème » tout en sachant que l'entrée d'un verbe définit son schème syntaxique et sémantique et donc l'emploi du verbe. Le processus d'annotation des verbes est basé sur une approche semi-automatique qui propose une liste d'entrées possibles pour chaque verbe à l'aide de leurs sens et schème syntaxique correspondants.

Dans le futur, nous envisageons d'intégrer un module qui automatiserait la sélection de l'entrée correspondante à l'emploi du verbe parmi l'ensemble des entrées possibles. Pour cela, il va falloir implanter un processus d'analyse automatique du schème syntaxique de chaque verbe car si on arrive à déterminer le schème syntaxique d'un verbe, donc on arrive à déduire automatiquement son entrée correspondante et donc sa nature sémantique et syntaxique. Pour l'analyse du schème syntaxique, il va falloir créer et intégrer un plugin d'analyseur syntaxique du français dans GATE qui sert à analyser la nature et les propriétés des constituants syntaxiques

d'une phrase afin de pouvoir déterminer automatiquement le schème syntaxique correspondant, car les schèmes syntaxiques sont présentés par des codes (T1308, P30g0…etc.) ce qui facilite leur analyse. L'opérateur sémantique du LVF codifie la syntaxe et la sémantique d'une entrée verbale ce qui serait une meilleure alternative pour la détermination automatique de l'entrée d'un verbe mais les opérateurs sémantiques présentés dans LVF ne fournissent aucun formalisme pour une utilisation par un programme informatique.

Produire des corpus annotés correctement avec des annotations encodées d'une manière formelle à l'aide du RDF/OWL permet de résoudre plusieurs difficultés d'interopérabilité mais aussi d'aider certaines applications TAL qui reposent sur des méthodes d'apprentissage ou classification automatique afin d'améliorer leurs résultats.

À terme l'ontologie LVF pourrait donc être exploitée par des méthodes de désambiguïsation de sens en traduction automatique ou en recherche d'informations.

Annexe I Feuille de style XSLT (XML -> OWL)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Ontology [
<!ENTITY xs "http://www.w3.org/2001/XMLSchema#">
<!ENTITY xml "http://www.w3.org/XML/1998/namespace">
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
]>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

    xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:dub="http://www.rali.iro.umontreal.ca/Dubois.owl"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    exclude-result-prefixes="xd xs"
    version="2.0">
    <xsl:strip-space elements="*"/>
    <xsl:output indent="yes"/>

    <xsl:variable name="node"/>

    <xsl:variable name="classe" select="'classe'"/>
    <xsl:variable name="domaine" select="'domaine'"/>
    <xsl:variable name="lexique" select="'lexique'"/>
    <xsl:variable name="ph" select="'phrase'"/>
    <xsl:variable name="domain"
select="document('FichierXMLDubois/domaines.xml',/)"/>
    <xsl:variable name="lexiqu"
select="document('FichierXMLDubois/lexiques.xml',/)"/>
    <xsl:variable name="class"
select="document('FichierXMLDubois/classes.xml',/)"/>
    <xsl:variable
name="operateu"select="document('FichierXMLDubois/operateurs.xml',/)"/>
    <xsl:variable
name="conjugaiso"select="document('FichierXMLDubois/codesConjugaison.xml',/)"
/>
    <xsl:variable name="schem"
select="document('FichierXMLDubois/schemes.xml',/)"/>
    <xsl:variable name="derivatio"
select="document('FichierXMLDubois/derives.xml',/)"/>
    <xsl:variable name="entree" select="'entree'"/>
    <xsl:function name="dub:encode-iri">
        <xsl:param name="in"/>
        <xsl:value-of select="translate($in,'&quot;&gt;&lt;\^%{}| ','--------
-_')"/>
    </xsl:function>

    <!-- Template pour le peuplement des classes Domaine, lexique,
conjugaison, scheme et derivation  -->

    <xsl:template name="temp">
```

```
        <xsl:param name="file"/>
        <xsl:variable name="lignesXML" select="$file//*[@*]"/>
        <xsl:for-each select="$lignesXML">
            <!--  <xsl:variable name="value" select="."/> -->
            <owl:Declaration>
                <owl:NamedIndividual IRI="#{dub:encode-iri(.)}"/>
            </owl:Declaration>
            <owl:ClassAssertion>
                <owl:Class IRI="#{local-name(..)}"/>
                <owl:NamedIndividual IRI="#{dub:encode-iri(.)}"/>
            </owl:ClassAssertion>
        </xsl:for-each>
<!-- DataPropertyAssertion pour Domaine, lexique, conjugaison, scheme et
derivation-->

<xsl:variable name="att" select="$file//*[@*]/@*"/>
      <xsl:for-each select="$att">
            <owl:DataPropertyAssertion>
                <owl:DataProperty IRI="#{local-name()}"/>
                <owl:NamedIndividual IRI="#{dub:encode-iri(../text())}"/>

                <owl:Literal datatypeIRI="xs:string"><xsl:value-of
select="."/></owl:Literal>
                </owl:DataPropertyAssertion>
            </xsl:for-each>
    </xsl:template>

    <xsl:template name="classe">
        <xsl:param name="classee"/>
        <xsl:variable name="cla" select="$classee//*[@*]"/>
        <xsl:for-each select="$cla">
            <xsl:choose>
                <xsl:when test="local-name()!='sem'">
                    <owl:Declaration>
                        <owl:NamedIndividual IRI="#{@code}"/>
                    </owl:Declaration>

                    <owl:ClassAssertion>
                        <owl:Class IRI="#{local-name()}"/>
                        <owl:NamedIndividual IRI="#{@code}"/>
                    </owl:ClassAssertion>

                    <owl:DataPropertyAssertion>
                        <owl:DataProperty IRI="#{@desc/local-name()}"/>
                        <owl:NamedIndividual IRI="#{@code}"/>
                        <owl:Literal datatypeIRI="xs:string"><xsl:value-of
select="@desc"/></owl:Literal>
                    </owl:DataPropertyAssertion>

                </xsl:when>
                <xsl:otherwise>
                    <owl:Declaration>
                        <owl:NamedIndividual IRI="#{dub:encode-iri(@desc)}"/>

                    </owl:Declaration>

                    <owl:ClassAssertion>
```

```
                        <owl:Class IRI="#{local-name(..)}"/>
                        <owl:NamedIndividual IRI="#{dub:encode-iri(@desc)}"/>
                    </owl:ClassAssertion>

                    <owl:DataPropertyAssertion>
                        <owl:DataProperty IRI="#{@code/local-name()}"/>
                        <owl:NamedIndividual IRI="#{dub:encode-iri(@desc)}"/>
                        <owl:Literal datatypeIRI="xs:string"><xsl:value-of
select="@code"/></owl:Literal>
                    </owl:DataPropertyAssertion>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:for-each>
    </xsl:template>

    <xsl:template name="ClassInstance">
        <xsl:param name="cls"/>
        <xsl:variable name="cl" select="$cls//*[@*]/*"/>
        <xsl:for-each select="$cl">
            <owl:ObjectPropertyAssertion>
                <owl:ObjectProperty IRI="#Has_{local-name()}"/>
                <owl:NamedIndividual IRI="#{../@code}"/>
                <owl:NamedIndividual IRI="#{@code}"/>
            </owl:ObjectPropertyAssertion>
        </xsl:for-each>
    </xsl:template>

    <xsl:template name="temp-operateur">
        <xsl:param name="operateur"/>
        <xsl:variable name="op" select="$operateur//*[@*]/@*"/>
        <xsl:for-each select="$op">

             <owl:Declaration>
                 <owl:NamedIndividual IRI="#{dub:encode-iri(.)}"/>
             </owl:Declaration>

             <owl:ClassAssertion>
                 <owl:Class IRI="#{local-name(../..)}"/>
                 <owl:NamedIndividual IRI="#{dub:encode-iri(.)}"/>
             </owl:ClassAssertion>

             <owl:DataPropertyAssertion>
                 <owl:DataProperty IRI="#desc"/>
                 <owl:NamedIndividual IRI="#{dub:encode-iri(.)}"/>
                 <owl:Literal datatypeIRI="xs:string"><xsl:value-of
select="../."/></owl:Literal>
               </owl:DataPropertyAssertion>
        </xsl:for-each>
    </xsl:template>

     <xsl:template match="verbe">
         <owl:Declaration>
             <owl:NamedIndividual IRI="#{dub:encode-iri(@mot)}"/>
           </owl:Declaration>
           <owl:ClassAssertion>
               <owl:Class IRI="#{local-name()}"/>
               <owl:NamedIndividual IRI="#{dub:encode-iri(@mot)}"/>
```

```
            </owl:ClassAssertion>

         <!-- DataPropertyAssertion pour Verbe-->

         <xsl:for-each select="@*">
             <xsl:variable name="attr" select="local-name()"/>
             <xsl:if test="$attr!='mot'">
                <owl:DataPropertyAssertion>
                   <owl:DataProperty IRI="#{local-name()}"/>
                   <owl:NamedIndividual IRI="#{dub:encode-iri(.)}"/>

                   <owl:Literal datatypeIRI="xs:string"><xsl:value-of
select="."/></owl:Literal>
                 </owl:DataPropertyAssertion>
             </xsl:if>
         </xsl:for-each>

                <!-- ObjectPRopertyAssertion / DataPropertyAssertion -->

       <xsl:for-each select="entree">
            <owl:ObjectPropertyAssertion>
                <owl:ObjectProperty IRI="#Has_{local-name()}"/>
                <owl:NamedIndividual IRI="#{dub:encode-iri(../@mot)}"/>
                <owl:NamedIndividual IRI="#{dub:encode-iri(mot/text())}"/>

            </owl:ObjectPropertyAssertion>
            <xsl:for-each select="*">
                    <xsl:if test="local-name()!='mot'">
                       <xsl:choose>
                           <xsl:when test="local-name()='domaine'">
                               <owl:ObjectPropertyAssertion>
                               <owl:ObjectProperty IRI="#Has_{local-
name()}"/>
                               <owl:NamedIndividual IRI="#{dub:encode-
iri(preceding-sibling::mot/text())}"/>
                               <owl:NamedIndividual IRI="#{dub:encode-
iri(@nom)}"/>
                              </owl:ObjectPropertyAssertion>

                           </xsl:when>

                           <xsl:when test="local-name()='lexique'">
                               <owl:ObjectPropertyAssertion>
                                   <owl:ObjectProperty IRI="#Has_{local-
name()}"/>
                                   <owl:NamedIndividual IRI="#{dub:encode-
iri(preceding-sibling::mot/text())}"/>

                                   <owl:NamedIndividual IRI="#{dub:encode-
iri(@desc)}"/>

                               </owl:ObjectPropertyAssertion>
                           </xsl:when>
                           <xsl:when test="*">
                               <xsl:for-each select="*">
                                   <owl:ObjectPropertyAssertion>
```

```
                                        <owl:ObjectProperty IRI="#Has_{local-
name(..)}"/>
                                        <owl:NamedIndividual
IRI="#{dub:encode-iri(../preceding-sibling::mot/text())}"/>

                                        <owl:NamedIndividual
IRI="#{dub:encode-iri(.)}"/>

                                    </owl:ObjectPropertyAssertion>
                                </xsl:for-each>
                                     <!-- DataPropertyAssertion pour scheme  -
->
                                  <xsl:for-each select="*/@*">

                                        <owl:DataPropertyAssertion>
                                            <owl:DataProperty IRI="#{local-
name()}"/>
                                            <owl:NamedIndividual
IRI="#{dub:encode-iri(../.)}"/>

                                            <owl:Literal
datatypeIRI="xs:string"><xsl:value-of select="."/></owl:Literal>
                                        </owl:DataPropertyAssertion>
                                     </xsl:for-each>
                            </xsl:when>
                            <xsl:otherwise>
                                <owl:ObjectPropertyAssertion>
                                    <owl:ObjectProperty IRI="#Has_{local-
name()}"/>
                                    <owl:NamedIndividual IRI="#{dub:encode-
iri(preceding-sibling::mot/text())}"/>

                                    <owl:NamedIndividual IRI="#{dub:encode-
iri(.)}"/>

                                </owl:ObjectPropertyAssertion>
                                <xsl:if test="local-name()!='classe'">
                                    <!-- DataPropertyAssertion pour le reste
des classes sauf Classe, Domaine et lexique -->
                                    <xsl:for-each select="@*">
                                        <owl:DataPropertyAssertion>
                                            <owl:DataProperty IRI="#{local-
name()}"/>
                                            <owl:NamedIndividual
IRI="#{dub:encode-iri(../.)}"/>

                                            <owl:Literal
datatypeIRI="xs:string"><xsl:value-of select="."/></owl:Literal>
                                        </owl:DataPropertyAssertion>
                                    </xsl:for-each>
                                </xsl:if>
                            </xsl:otherwise>
                        </xsl:choose>
                    </xsl:if>
                    <xsl:if test="local-name()='mot'">
                         <xsl:for-each select="@*">
                            <owl:DataPropertyAssertion>
```

```
                                <owl:DataProperty IRI="#{local-name()}"/>
                                <owl:NamedIndividual IRI="#{dub:encode-
iri(../.)}"/>

                                <owl:Literal
datatypeIRI="xs:string"><xsl:value-of select="."/></owl:Literal>
                            </owl:DataPropertyAssertion>
                        </xsl:for-each>
                    </xsl:if>

                 <!-- ClassAssertion -->
                <xsl:if test="local-name()!=$classe">
                    <xsl:if test="local-name()!=$domaine">
                        <xsl:if test="local-name()!=$lexique">
                            <xsl:choose>
                                <xsl:when test="local-name()='phrases'">
                                    <xsl:for-each select="phrase">

                                        <owl:Declaration>
                                            <owl:NamedIndividual
IRI="#{dub:encode-iri(.)}"/>
                                        </owl:Declaration>

                                        <owl:ClassAssertion>
                                            <owl:Class IRI="#{local-
name()}"/>
                                            <owl:NamedIndividual
IRI="#{dub:encode-iri(.)}"/>
                                        </owl:ClassAssertion>
                                    </xsl:for-each>
                                </xsl:when>
                                <xsl:when test="local-name()='construction'">
                                    <xsl:for-each select="scheme">
                                        <owl:Declaration>
                                            <owl:NamedIndividual IRI="#{.}"/>
                                        </owl:Declaration>
                                        <owl:ClassAssertion>
                                            <owl:Class IRI="#{local-
name()}"/>
                                            <owl:NamedIndividual IRI="#{.}"/>
                                        </owl:ClassAssertion>
                                    </xsl:for-each>
                                </xsl:when>
                                <xsl:otherwise>
                                    <owl:Declaration>
                                        <owl:NamedIndividual
IRI="#{dub:encode-iri(.)}"/>
                                    </owl:Declaration>
                                    <xsl:variable name="local" select="local-
name()"/>
                                    <xsl:if test="$local='mot'">
                                        <owl:ClassAssertion>
                                            <owl:Class IRI="#{$entree}"/>
                                            <owl:NamedIndividual
IRI="#{dub:encode-iri(.)}"/>
                                        </owl:ClassAssertion>
                                 </xsl:if>
```

```
                                    <xsl:if test="$local!='mot'">
                                       <owl:ClassAssertion>
                                       <owl:Class IRI="#{local-name()}"/>
                                       <owl:NamedIndividual IRI="#{dub:encode-
iri(.)}"/>
                                           </owl:ClassAssertion>
                                       </xsl:if>
                                    </xsl:otherwise>
                                </xsl:choose>
                            </xsl:if>
                        </xsl:if>
                    </xsl:if>
                </xsl:for-each>
            </xsl:for-each>
        </xsl:template>

        <xsl:template match="/">
            <owl:Ontology
                xml:base="http://www.rali.iro.umontreal.ca/Dubois.owl"
                xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                xmlns:owl="http://www.w3.org/2002/07/owl#"
                xmlns:xs="http://www.w3.org/2001/XMLSchema#"
                xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                ontologyIRI="http://www.rali.iro.umontreal.ca/Dubois.owl">

                <xsl:call-template name="temp">
                    <xsl:with-param name="file" select="$domain"/>
                </xsl:call-template>

                <xsl:call-template name="temp">
                    <xsl:with-param name="file" select="$lexiqu"/>
                </xsl:call-template>

                <xsl:call-template name="temp">
                    <xsl:with-param name="file" select="$conjugaiso"/>
                </xsl:call-template>

                <xsl:call-template name="temp-operateur">
                    <xsl:with-param name="operateur" select="$operateu"/>
                </xsl:call-template>

                <xsl:call-template name="temp">
                    <xsl:with-param name="file" select="$schem"/>
                </xsl:call-template>

                <xsl:call-template name="temp">
                    <xsl:with-param name="file" select="$derivatio"/>
                </xsl:call-template>

                <xsl:call-template name="classe">
                    <xsl:with-param name="classee" select="$class"/>
                </xsl:call-template>

                <xsl:call-template name="ClassInstance">
                    <xsl:with-param name="cls" select="$class"/>
                </xsl:call-template>
```

```
            <xsl:apply-templates/>
        </owl:Ontology>
    </xsl:template>
</xsl:stylesheet>
```

Annexe II Feuille de style XSLT (XSD-> OWL)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Ontology [
    <!ENTITY xs "http://www.w3.org/2001/XMLSchema#">
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace">
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
]>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xml:base="http://www.rali.iro.umontreal.ca/Dubois.owl"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    exclude-result-prefixes="xd xs"
    version="2.0">
    <xsl:strip-space elements="*"/>
    <xsl:output indent="yes"/>

    <xsl:variable name="domain"
select="document('FichierXMLDubois/domaines.xsd',/)"/>
    <xsl:variable name="class"
select="document('FichierXMLDubois/classes.xsd',/)"/>
    <xsl:variable name="operateu"
select="document('FichierXMLDubois/operateurs.xsd',/)"/>
    <xsl:variable name="schem"
select="document('FichierXMLDubois/schemes.xsd',/)"/>
    <xsl:variable name="conjugaiso"
select="document('FichierXMLDubois/codesConjugaison.xsd',/)"/>
    <xsl:variable name="derivatio"
select="document('FichierXMLDubois/derives.xsd',/)"/>
    <xsl:variable name="lexiqu"
select="document('FichierXMLDubois/lexiques.xsd',/)"/>

    <xsl:variable name="var1" select="'domaine'"/>
    <xsl:variable name="var2" select="'classe'"/>
    <xsl:variable name="var3" select="'operateur'"/>
    <xsl:variable name="var4" select="'scheme'"/>
    <xsl:variable name="var5" select="'conjugaison'"/>
    <xsl:variable name="var6" select="'derivation'"/>
    <xsl:variable name="var7" select="'lexique'"/>
    <xsl:variable name="dom" select="'domaines'"/>
    <xsl:variable name="lex" select="'lexiques'"/>
    <xsl:variable name="def" select="'def'"/>
    <xsl:variable name="cl" select="'sous-classe-syntaxique'"/>
    <xsl:variable name="phrase" select="'phrase'"/>
    <xsl:variable name="entree" select="'entree'"/>
    <xsl:variable name="scheme" select="'scheme'"/>


    <xsl:key name="find"
```

```
match="xs:element/xs:complexType/xs:sequence/xs:element" use="@ref"/>

    <!-- template pour générer les classes de Domaine et Lexique -->
    <xsl:template name="Classe-externe1">
        <xsl:param name="classe"/>

        <xsl:for-each
select="$classe/xs:schema/xs:element[xs:complexType[not(@mixed)]]">
            <owl:Declaration>
                <owl:Class IRI="#{@name}"/>
            </owl:Declaration>
            <xsl:for-each
select="xs:complexType/xs:sequence/xs:element[not(@maxOccurs)]">
                <owl:SubClassOf>
                    <owl:Class IRI="#{@ref}"/>
                    <owl:Class IRI="#{../../../@name}"/>
                </owl:SubClassOf>
            </xsl:for-each>
        </xsl:for-each>
        <xsl:for-each
select="$classe/xs:schema/xs:element/xs:complexType/xs:attribute">
            <owl:Declaration>
                <owl:DataProperty IRI="#{@name}"/>
            </owl:Declaration>

            <owl:DataPropertyDomain>
                <owl:DataProperty IRI="#{@name}"/>
                <xsl:variable name="ref" select="../../@name"/>
                <owl:Class IRI="#{key('find', $ref)/../../../@name}"/>
            </owl:DataPropertyDomain>

            <xsl:if test="@type">
                <owl:DataPropertyRange>
                    <owl:DataProperty IRI="#{@name}"/>
                    <owl:Datatype IRI="#{@type}"/>
                </owl:DataPropertyRange>
            </xsl:if>
            <xsl:if test="not(@type)">
                <owl:DataPropertyRange>
                    <owl:DataProperty IRI="#{@name}"/>
                    <owl:Datatype IRI="#xs:string"/>
                </owl:DataPropertyRange>
            </xsl:if>
        </xsl:for-each>
    </xsl:template>

    <!-- template pour générer les classes de conjugaison, scheme et derivation
-->

    <xsl:template name="Classe-externe2">
        <xsl:param name="class"/>
        <xsl:param name="nam"/>

        <owl:Declaration>
            <owl:Class IRI="#{$nam}"/>
        </owl:Declaration>
```

```
        <xsl:for-each
select="$class/xs:schema/xs:element[xs:complexType[not(@mixed)]]">
            <owl:Declaration>
                <owl:Class IRI="#{@name}"/>
            </owl:Declaration>

            <xsl:for-each
select="xs:complexType/xs:sequence/xs:element[not(@maxOccurs)]">
                <owl:SubClassOf>
                    <owl:Class IRI="#{@ref}"/>
                    <owl:Class IRI="#{../../../@name}"/>
                </owl:SubClassOf>
            </xsl:for-each>
        </xsl:for-each>

        <xsl:for-each
select="$class/xs:schema/xs:element/xs:complexType/xs:attribute">
            <xsl:variable name="att" select="@name"/>
            <xsl:variable name="type" select="@type"/>
            <xsl:variable name="ref" select="../../@name"/>
            <xsl:for-each select="key('find', $ref)">
            <owl:Declaration>
                <owl:DataProperty IRI="#{$att}"/>
            </owl:Declaration>
            <owl:DataPropertyDomain>
                <owl:DataProperty IRI="#{$att}"/>
                <owl:Class IRI="#{../../../@name}"/>
            </owl:DataPropertyDomain>
            <xsl:if test="$type">
                <owl:DataPropertyRange>
                    <owl:DataProperty IRI="#{$att}"/>
                    <owl:Datatype IRI="#{$type}"/>
                </owl:DataPropertyRange>
            </xsl:if>
            <xsl:if test="not($type)">
                <owl:DataPropertyRange>
                    <owl:DataProperty IRI="#{$att}"/>
                    <owl:Datatype IRI="#xs:string"/>
                </owl:DataPropertyRange>
            </xsl:if>
            </xsl:for-each>
        </xsl:for-each>
    </xsl:template>

    <!-- template pour générer la classe operateur -->
    <xsl:template name="Classe-externe3">
        <xsl:param name="clas"/>
        <xsl:param name="nom"/>

        <owl:Declaration>
            <owl:Class IRI="#{$nom}"/>
        </owl:Declaration>
        <xsl:for-each
select="$clas/xs:schema/xs:element[xs:complexType[not(@mixed)]]">
            <owl:Declaration>
                <owl:Class IRI="#{@name}"/>
            </owl:Declaration>
```

```
            <xsl:for-each
select="xs:complexType/xs:sequence/xs:element[not(@maxOccurs)]">
                <owl:SubClassOf>
                    <owl:Class IRI="#{@ref}"/>
                    <owl:Class IRI="#{../../../@name}"/>
                </owl:SubClassOf>
            </xsl:for-each>
        </xsl:for-each>
        <xsl:for-each
select="$clas/xs:schema/xs:element/xs:complexType/xs:sequence/xs:element[@maxOc
curs]">
            <owl:Declaration>
                <owl:DataProperty IRI="#{$def}"/>
            </owl:Declaration>

            <owl:DataPropertyDomain>
                <owl:DataProperty IRI="#{$def}"/>
                <owl:Class IRI="#{../../../@name}"/>
            </owl:DataPropertyDomain>

            <owl:DataPropertyRange>
                <owl:DataProperty IRI="#{$def}"/>
                <owl:Datatype IRI="xs:string"/>
            </owl:DataPropertyRange>
        </xsl:for-each>
    </xsl:template>

    <xsl:template match="xs:element">
        <xsl:choose>
            <xsl:when test="@name and (child::xs:complexType) or (@name and
@type)">
                <xsl:choose>

                    <!-- Classe Domaine -->
                    <xsl:when test="@name=$var1">
                        <xsl:call-template name="Classe-externe1">
                            <xsl:with-param name="classe" select="$domain"/>
                        </xsl:call-template>
                    </xsl:when>

                    <!-- Classe Lexique -->
                    <xsl:when test="@name=$var7">
                        <xsl:call-template name="Classe-externe1">
                            <xsl:with-param name="classe" select="$lexiqu"/>
                        </xsl:call-template>
                    </xsl:when>

                    <!-- Classe Operateur  -->
                    <xsl:when test="@name=$var3">
                        <xsl:call-template name="Classe-externe3">
                            <xsl:with-param name="clas" select="$operateu"/>
                            <xsl:with-param name="nom" select="$var3"/>
                        </xsl:call-template>

                    </xsl:when>

                    <!-- Classe conjugaison -->
```

```
                    <xsl:when test="@name=$var5">
                        <xsl:call-template name="Classe-externe2">
                            <xsl:with-param name="class" select="$conjugaiso"/>
                            <xsl:with-param name="nam" select="$var5"/>
                        </xsl:call-template>
                    </xsl:when>

                    <!-- Classe scheme -->
                    <xsl:when test="@name=$var4">
                        <xsl:call-template name="Classe-externe2">
                            <xsl:with-param name="class" select="$schem"/>
                            <xsl:with-param name="nam" select="$var4"/>
                        </xsl:call-template>
                    </xsl:when>

                    <!-- Classe derivation -->
                    <xsl:when test="@name=$var6">
                        <xsl:call-template name="Classe-externe2">
                            <xsl:with-param name="class" select="$derivatio"/>
                            <xsl:with-param name="nam" select="$var6"/>
                        </xsl:call-template>
                    </xsl:when>

                    <!-- Classe Classes -->
                        <xsl:when test="@name=$var2">
                        <xsl:for-each select="$class/xs:schema/xs:element">
                            <xsl:if test="@name!='sem'">
                                <owl:Declaration>
                                    <owl:Class IRI="#{@name}"/>
                                </owl:Declaration>
                            </xsl:if>
                            <xsl:choose>
                                <xsl:when
test="xs:complexType[child::xs:attribute]/xs:sequence/xs:element">
                                    <xsl:for-each
select="xs:complexType/xs:sequence/xs:element">
                                        <owl:Declaration>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                        </owl:Declaration>
                                        <owl:ObjectPropertyRange>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                            <owl:Class IRI="#{@ref}"/>
                                        </owl:ObjectPropertyRange>
                                        <owl:ObjectPropertyDomain>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                            <owl:Class
IRI="#{../../../@name}"/>
                                        </owl:ObjectPropertyDomain>
                                        <owl:SubClassOf>
                                            <owl:Class IRI="#{@ref}"/>
                                            <owl:Class
IRI="#{../../../../child::node()[1]/@name}"/>
                                        </owl:SubClassOf>
                                    </xsl:for-each>
```

```
                                </xsl:when>
                                <xsl:when
test="xs:complexType[not(child::xs:attribute)]/xs:sequence/xs:element">
                                    <xsl:for-each
select="xs:complexType/xs:sequence/xs:element">
                                        <xsl:if test="@ref!='sem'">
                                            <owl:SubClassOf>
                                                <owl:Class IRI="#{@ref}"/>
                                                <owl:Class
IRI="#{../../../@name}"/>
                                            </owl:SubClassOf>
                                        </xsl:if>
                                    </xsl:for-each>
                                </xsl:when>
                                <xsl:when
test="xs:complexType[child::xs:attribute]">
                                    <xsl:variable name="att"
select="//xs:complexType/xs:attribute"/>
                                    <xsl:for-each select="$att">
                                      <xsl:if test="../../@name!='sem'">
                                          <xsl:if test="@name='desc'">
                                              <owl:Declaration>
                                                  <owl:DataProperty
IRI="#{@name}"/>
                                              </owl:Declaration>
                                              <owl:DataPropertyDomain>
                                                  <owl:DataProperty
IRI="#{@name}"/>
                                                  <owl:Class
IRI="#{../../@name}"/>
                                              </owl:DataPropertyDomain>
                                              <owl:DataPropertyRange>
                                                  <owl:DataProperty
IRI="#{@name}"/>
                                                  <owl:Datatype
IRI="xs:string"/>
                                              </owl:DataPropertyRange>
                                          </xsl:if>
                                      </xsl:if>
                                    </xsl:for-each>
                                </xsl:when>
                            </xsl:choose>
                        </xsl:for-each>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:if test="@name!='verbes'">
                          <xsl:if test="@name!='phrases'">
                              <xsl:if test="@name!='mot'">
                                  <xsl:if test="@name!='construction'">
                                      <owl:Declaration>
                                          <owl:Class IRI="#{@name}"/>
                                      </owl:Declaration>
                                  </xsl:if>
                              </xsl:if>
                          </xsl:if>
                      </xsl:if>
                    </xsl:otherwise>
```

```
                </xsl:choose>
                 <!--              Génération ObjectProperty / Range/ Domain à
partir de Dubois.xsd                 -->

            <xsl:choose>

                <xsl:when
test="xs:complexType[not(@mixed)]/xs:sequence/xs:element">
                    <xsl:for-each
select="xs:complexType[not(@mixed)]/xs:sequence/xs:element">
                        <xsl:choose>
                                    <xsl:when test="@ref=$var1">
                                        <owl:Declaration>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                        </owl:Declaration>
                                        <owl:ObjectPropertyRange>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                            <owl:Class IRI="#{$dom}"/>
                                        </owl:ObjectPropertyRange>
                                        <owl:ObjectPropertyDomain>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                            <owl:Class
IRI="#{../../../@name}"/>
                                        </owl:ObjectPropertyDomain>
                                    </xsl:when>
                                    <xsl:when test="@ref=$var7">
                                        <owl:Declaration>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                        </owl:Declaration>
                                        <owl:ObjectPropertyRange>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                            <owl:Class IRI="#{$lex}"/>
                                        </owl:ObjectPropertyRange>
                                        <owl:ObjectPropertyDomain>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                            <owl:Class
IRI="#{../../../@name}"/>
                                        </owl:ObjectPropertyDomain>
                                    </xsl:when>
                                    <xsl:when test="@ref=$var2">
                                        <owl:Declaration>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                        </owl:Declaration>
                                        <owl:ObjectPropertyRange>
                                            <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                            <owl:Class IRI="#{$cl}"/>
                                        </owl:ObjectPropertyRange>
                                        <owl:ObjectPropertyDomain>
                                            <owl:ObjectProperty
```

```
IRI="#Has_{@ref}"/>
                                        <owl:Class
IRI="#{../../../@name}"/>
                                    </owl:ObjectPropertyDomain>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:if test="@ref!='mot'">
                                        <xsl:if test="@ref!='verbe'">
                                            <xsl:if test="@ref!='phrase'">
                                                <xsl:if
test="@ref!='scheme'">
                                                    <owl:Declaration>
                                                        <owl:ObjectProperty
IRI="#Has_{@ref}"/>
                                                    </owl:Declaration>
                                                    <xsl:if
test="@ref='phrases'">

<owl:ObjectPropertyRange>

<owl:ObjectProperty IRI="#Has_{@ref}"/>
                                                            <owl:Class
IRI="#{$phrase}"/>

</owl:ObjectPropertyRange>

<owl:ObjectPropertyDomain>

<owl:ObjectProperty IRI="#Has_{@ref}"/>
                                                            <owl:Class
IRI="#{../../../@name}"/>

</owl:ObjectPropertyDomain>
                                                    </xsl:if>
                                                    <xsl:if
test="@ref='construction'">

<owl:ObjectPropertyRange>

<owl:ObjectProperty IRI="#Has_{@ref}"/>
                                                            <owl:Class
IRI="#{$scheme}"/>

</owl:ObjectPropertyRange>


<owl:ObjectPropertyDomain>

<owl:ObjectProperty IRI="#Has_{@ref}"/>
                                                            <owl:Class
IRI="#{../../../@name}"/>

</owl:ObjectPropertyDomain>
                                                    </xsl:if>
                                                    <xsl:if
test="@ref!='phrases'">
                                                        <xsl:if
```

```
test="@ref!='construction'">

<owl:ObjectPropertyRange>

<owl:ObjectProperty IRI="#Has_{@ref}"/>
                                                                        <owl:Class
IRI="#{@ref}"/>

</owl:ObjectPropertyRange>


<owl:ObjectPropertyDomain>

<owl:ObjectProperty IRI="#Has_{@ref}"/>
  <owl:Class IRI="#{../../../@name}"/>

</owl:ObjectPropertyDomain>
                                                                    </xsl:if>
                                                                </xsl:if>
                                                            </xsl:if>
                                                        </xsl:if>
                                                    </xsl:if>
                                                </xsl:if>
                                            </xsl:otherwise>
                                        </xsl:choose>
                                </xsl:for-each>
                            </xsl:when>
                        </xsl:choose>

                            <!--          DataProperty declaration / Range declaration
-->

                    <xsl:choose>
                        <xsl:when test="not(@type or @ref) and @name!='verbes' and
child::xs:complexType//xs:attribute">
                            <xsl:choose>
                            <xsl:when test="xs:complexType/xs:attribute">
                                <xsl:for-each select="xs:complexType/xs:attribute">
                                    <owl:Declaration>
                                        <owl:DataProperty IRI="#{@name}"/>
                                    </owl:Declaration>
                                    <owl:DataPropertyRange>
                                        <owl:DataProperty IRI="#{@name}"/>
                                        <xsl:if test="@type !='texte'">
                                            <owl:Datatype IRI="{@type}"/>
                                        </xsl:if>
                                        <xsl:if test="@type='texte'">
                                            <owl:Datatype IRI="xs:string"/>
                                        </xsl:if>
                                        <xsl:if test="xs:simpleType">
                                            <owl:Datatype
IRI="{xs:simpleType/xs:restriction/@base}"/>
                                        </xsl:if>
                                    </owl:DataPropertyRange>
                                    <owl:DataPropertyDomain>
                                        <owl:DataProperty IRI="#{@name}"/>
                                        <owl:Class IRI="#{../../@name}"/>
```

```
                        </owl:DataPropertyDomain>
                    </xsl:for-each>
                </xsl:when>
                <xsl:when
test="xs:complexType/xs:simpleContent/xs:restriction/xs:attribute">
                    <xsl:for-each
select="xs:complexType/xs:simpleContent/xs:restriction/xs:attribute">
                        <xsl:if test="../../../../@name!='domaine'">
                           <xsl:if test="../../../../@name!='classe'">
                              <xsl:if test="../../../../@name!='lexique'">
                                  <owl:Declaration>
                                      <owl:DataProperty IRI="#{@name}"/>
                                  </owl:Declaration>
                                  <owl:DataPropertyRange>
                                      <owl:DataProperty IRI="#{@name}"/>
                                      <xsl:if test="@type !='texte'">
                                          <owl:Datatype IRI="{@type}"/>
                                      </xsl:if>
                                      <xsl:if test="@type='texte'">
                                          <owl:Datatype IRI="xs:string"/>
                                      </xsl:if>
                                      <xsl:if test="xs:simpleType">
                                          <owl:Datatype
IRI="{xs:simpleType/xs:restriction/@base}"/>
                                      </xsl:if>
                                  </owl:DataPropertyRange>
                                  <owl:DataPropertyDomain>
                                      <owl:DataProperty IRI="#{@name}"/>
                                      <!--  -->
                                      <owl:Class IRI="#{../../../../@name}"/>
                                  </owl:DataPropertyDomain>
                               </xsl:if>
                           </xsl:if>
                        </xsl:if>
                    </xsl:for-each>
                </xsl:when>
                <xsl:when
test="xs:complexType/xs:simpleContent/xs:extension/xs:attribute">
                    <xsl:for-each
select="xs:complexType/xs:simpleContent/xs:extension/xs:attribute">
                        <xsl:if test="../../../../@name!='domaine'">
                            <xsl:if test="../../../../@name!='classe'">
                                <xsl:if test="../../../../@name!='lexique'">
                                <owl:Declaration>
                                    <owl:DataProperty IRI="#{@name}"/>
                                </owl:Declaration>
                                <owl:DataPropertyRange>
                                    <owl:DataProperty IRI="#{@name}"/>
                                    <xsl:if test="@type !='texte'">
                                        <owl:Datatype IRI="{@type}"/>
                                    </xsl:if>
                                    <xsl:if test="@type='texte'">
                                        <owl:Datatype IRI="xs:string"/>
                                    </xsl:if>
                                    <xsl:if test="xs:simpleType">
                                        <owl:Datatype
IRI="{xs:simpleType/xs:restriction/@base}"/>
```

```
                                    </xsl:if>
                                </owl:DataPropertyRange>
                                <owl:DataPropertyDomain>
                                    <owl:DataProperty IRI="#{@name}"/>
                                    <xsl:variable name="verif"
select="../../../../@name"/>
                                    <xsl:if test="$verif='mot'">
                                        <owl:Class IRI="#{$entree}"/>
                                    </xsl:if>
                                    <xsl:if test="$verif!='mot'">
                                        <owl:Class
IRI="#{../../../../@name}"/>
                                    </xsl:if>
                                </owl:DataPropertyDomain>
                            </xsl:if>
                        </xsl:if>
                    </xsl:if>
                </xsl:for-each>
            </xsl:when>
        </xsl:choose>
    </xsl:when>
</xsl:choose>
</xsl:when>
</xsl:choose>
</xsl:template>

<xsl:template match="/">
    <owl:Ontology
        xml:base="http://www.rali.iro.umontreal.ca/Dubois.owl"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:xs="http://www.w3.org/2001/XMLSchema#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        ontologyIRI="http://www.rali.iro.umontreal.ca/Dubois.owl">
        <xsl:apply-templates/>
    </owl:Ontology>
</xsl:template>
</xsl:stylesheet>
```

Bibliographie

[1] Van Assem, Mark, Aldo Gangemi, and Guus Schreiber. "Conversion of WordNet to a standard RDF/OWL representation." *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06), Genoa, Italy*. 2006.

[2] Niles, Ian, and A. Pease. *Mapping WordNet to the SUMO ontology*. Teknowledge Technical Report, 2003.

[3] Berners-Lee, Tim, James Hendler, and Ora Lassila. "The semantic web." *Scientific american* 284.5 (2001): 28-37.

[4] Masinter, Larry, Tim Berners-Lee, and Roy T. Fielding. "Uniform resource identifier (URI): Generic syntax." (2005).

[5] Lapalme, Guy. "XML: Looking at the Forest Instead of the Trees." *Disponible sur http://www. iro. umontreal. ca/~ lapalme/ForestInsteadOfTheTrees* (2007).

[6] McGuinness, Deborah L., and Frank Van Harmelen. "OWL web ontology language overview." *W3C recommendation* 10.2004-03 (2004): 10.

[7] Handschuh, Siegfried, and Steffen Staab, eds. *Annotation for the semantic web*. Vol. 96. IOS Press, 2003.

[8] Keita, Abdel Kader, Catherine Roussey, and Robert Laurini. "Un outil d'aide à la construction d'ontologies pré-consensuel les: le projet Towntology." *INFORSID*. 2006.

[10] Bourigault, Didier, et al. "Syntex, analyseur syntaxique de corpus." *Actes des 12èmes journées sur le Traitement Automatique des Langues Naturelles*. 2005.

[11] Bourigault, Didier. "Upery: un outil d'analyse distributionnelle étendue pour la construction d'ontologies à partir de corpus." *Actes de la 9ème conférence annuelle sur le Traitement Automatique des Langues (TALN 2002), Nancy*. 2002.

[12] Gómez-Pérez, Asunción, and David Manzano-Macho. "An overview of methods and tools for ontology learning from texts." *The knowledge engineering review*19.3 (2004): 187-212.

[13] Ide, Nancy, Alessandro Lenci, and Nicoletta Calzolari. "RDF instantiation of ISLE/MILE lexical entries." *Proceedings of the ACL 2003 workshop on Linguistic annotation: getting the model right-Volume 19*. Association for Computational Linguistics, 2003.

[14] Dini, Luca. "Nlp technologies and semantic web: Risks, opportunities and challenges." *Intelligenza artificiale* 1.1 (2004): 67-71.

[15] DENOUE, Laurent, and Laurence VIGNOLLET. "L'importance des annotations: application à la classification des documents du Web." *Document numérique*4.1-2 (2000): 37-57.

[16] Ma, Yue, Laurent Audibert, and Adeline Nazarenko. "Ontologies étendues pour l'annotation sémantique." *20es Journées Francophones d'Ingénierie des Connaissances* (2009).

[17] Amardeilh, Florence. *Web Sémantique et Informatique Linguistique: propositions méthodologiques et réalisation d'une plateforme logicielle*. Diss. Université de Nanterre-Paris X, 2007.

[18] Heflin, Jeff, and James Hendler. "Dynamic ontologies on the web." *AAAI/IAAI*. 2000.

[19] François, Jacques, Denis Le Pesant, and Danielle Leeman. "Présentation de la classification des Verbes Français de Jean Dubois et Françoise Dubois-Charlier." *Langue française* 1 (2007): 3-19.

[20] Hadouche, Fadila, and Guy Lapalme. "Une version électronique du LVF comparée avec d'autres ressources lexicales." *Langages* 3 (2010): 193-220.

[21] Noy, Natalya F., and Deborah L. McGuinness. "Développement d'une ontologie 101: Guide pour la création de votre première ontologie." *Université de Stanford, Stanford, Traduit de l'anglais par Anila Angjeli. ht tp://www. bnf. fr/pages/infopro/normes/pdf/no-DevOnto. pdf* (2000).

[22] Ferdinand, Matthias, Christian Zirpins, and David Trastour. "Lifting XML schema to OWL." *Web Engineering*. Springer Berlin Heidelberg, 2004. 354-358.

[23] H. Bohring, and S. Auer, "Mapping XML to OWL Ontologies", In Leipziger Informatik-Tage, vol. 72, 2005, pp. 147–156. Society, Washington, DC, USA, (2007).

[24] Sergej Melnik. Bridging the gap between XML and RDF. http://wwwdb.stanford.edu/ melnik/rdf/fusion.html, 1999.

[25] Corbin, Harold, and Aaron Temin. "TIPSTER lessons learned: the SE/CM perspective." *Proceedings TIPSTER Phase III* (1999).

[26] Schmid, Helmut. "Improvements in part-of-speech tagging with an application to German." *In Proceedings of the ACL SIGDAT-Workshop*. 1995.

[27] Schmid, Helmut. "Probabilistic part-of-speech tagging using decision trees."*Proceedings of international conference on new methods in language processing*. Vol. 12. 1994.