Université de Montréal

Intégration de VerbNet dans un réalisateur profond

par

Daniel Galarreta-Piquette

Département de traduction et linguistique
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en linguistique

22 mars 2018

SOMMAIRE

Ce mémoire explore les patrons de régime en anglais provenant de la ressource lexicographique *VerbNet* et leur implémentation dans un système de génération automatique de texte.

SUMMARY

English summary and keywords...

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET DES ABRÉVIATIONS

GAT	Génération automatique de texte
GP	Patron de régime, de l'anglais *Government Pattern*
DPOS	Partie du discours profonde, de l'anglais *Deep Part of Speech*
TST	Théorie Sens-Texte
VN	*VerbNet*

DÉDICACES

Vos dédicaces.

REMERCIEMENTS

Remerciements...

INTRODUCTION

(FL) [1] Parler de la génération de texte automatique, de *VerbNet*, de la problématique (pas de consensus quant à l'architecture de la classe verbale en traitement automatique des langues (TAL)) de VerbNet, des dictionnaires. Pourquoi les verbes sont si importants ? (Kipper,2005, dissertation) Puisque les verbes sont porteurs du sens principal de la phrase, il faudrait donc créer une ressource qui puisse rendre compte du sens des verbes si on souhaite une bon fonctionnement des applications NLP.

1. utilise toujours un format particulier pour des commentaires, parce que c'est très facile de pas voir un petit commentaire dans un mémoire, et ça a l'air fou si tu te ramasses avec ça dans ton texte.

Chapitre 1

GÉNÉRATION AUTOMATIQUE DE TEXTE

La génération automatique de texte découle de la branche qu'est le TAL. Reiter et Dale (Reiter et Dale, 2000) définissent cette branche comme étant un domaine à la croisée des chemins entre l'intelligence artificielle et la linguistique computationnelle. L'objectif est de développer des sytèmes pouvant produire du texte compréhensible en langue naturelle à partir de données non-linguistiques. Bien que l'objectif est commun à tous ces systèmes, les moyens pour s'y rendre sont de divers ordres. Entre autre car il existe des inputs de diverses natures : données, texte et images(Thomason et al., 2014). Ensuite car il y existe diverses approches de réalisation de texte : templates, règles, stochastiques (Gatt et Krahmer, 2018).

Toutefois, avant d'entrer dans les détails de la génération automatique de texte (GAT), il serait intéressant de mentionner l'origine de ces systèmes. À la base, ils ont été conçus pour, entre autre, générer des rapports automatiquement afin de faciliter le travail des humains. Effectivement, il est très utile pour certains de pouvoir lire un résumé de données numériques analysées par un système informatique. Tel que JSreal le mentionne, avec la GAT on peut présenter un résumé d'information provenant d'input numérique qui serait incompréhensible, mais utile pour un humain qui n'est pas un expert, et donc incapable de déchiffrer ces données. En plus d'être capable de résumé ces informations, le système informatique a l'avantage de fournir ces rapports sans se fatiguer à faire une tâche extrêmement monotone, qui pourrait être très coûteuse en termes de temps et de ressources pour des être humains. D'ailleurs, pour qu'ils soient utiles, les textes générés automatiquement n'ont pas besoin d'être lus par une quantité phénomènale de gens. Dès qu'ils remplissent leur fonction, en étant utile à ne serait-ce que quelques personnes, leur raison d'être sera justifiée. Dans cette même optique, il s'est développé des systèmes pouvant générer du texte en fonction de l'utilisateur. Ainsi, on pourrait générer un rapport X en fonction du professionnel, du technicien ou du client (Mahammood Reiter,2011) qui lira ce texte. Cette utilisation de la GAT s'applique à de nombreux domaines dont les textes journalistiques. Les articles décrivant les matchs sportifs

qui ne bénéficient pas de couverture médiatique (Van der Lee, 2017) sont des applications concrètes et utile du développement en GAT. De même que des rapports générés automatiquement sur la qualité de l'air en fonction de l'utilisateur (Wanner, Bohnet, Bouayad-Agha, Lareau et Nicklass, 2010). Ainsi, via le développement de ces systèmes, l'effacité des ordinateurs et de l'accessibilité aux données grandissante, on est capable de générer un article de journal automatiquement suivant les minutes où un évènement se produit(Oremus,2014). On parle là de robo-journalisme à son plein essor.

De nos jours la GAT a grandement changé depuis que Dale et Reiter ont publié leur livre. Bien que ce soit le livre le plus complet entourant la NLG, le domaine a changé avec l'émergence de text-to-text generation et de vision-to-text generation(Hendricks et al., 2016a). qui reposent plus sur des méthodes statistiques que les modèles traditionnels de data-to-text qui étaient rule-based ou template based (Gatt et Krahmer, 2018). D'ailleurs, les frontières entre les diverses approches s'affaissent aussi avec le temps et nous sommes témoins d'une apparition constante de systèmes hybrides. Par exemple, des systèmes à base de patrons utilisant des règles de grammaire, des systèmes à base de règles utilisant des méthodes statistiques pour combler certaines tâches (Gatt et Krahmer, 2018). C'est donc un domaine très vaste qui, à ce jour, n'est pas uniforme. Ce qui est directement lié à une grande variété en NLG : divers types d'input (type de données), divers objectifs (tâches), diverses approches. Il est aussi important de préciser que la GAT a aussi une valeur linguistique théorique. Il existe des linguistes qui testent leur théories via le développement de générateur automatique de texte. C'est effectivement une bonne manière de vérifier si un modèle théorique fonctionne(Danlos, 1983).

Dans le cadre de ce mémoire, notre projet est un réalisateur. La réalisation linguistique étant une étape du processus entourant la GAT, nous ne travaillons que sur celle-ci. D'ailleurs notre réalisateur est créé avec des perspectives linguistiques. Ce qui exclut les systèmes à base de patrons et les systèmes statistiques, car ceux-ci ne nécessitent pas d'analyse linguistique pour le bon fonctionnement de la réalisation. Notre système se base sur des règles grammaticales. Cependant, avant de décrire notre projet, nous allons jeter les bases de la GAT en décrivant le pipeline classique et en exposant quelques réalisateurs linguistiques.

1.1. Pipeline classique GAT

À la base, le pipeline classique observé par Dale et Reiter est un processus séquentiel séparé en diverses sections (Reiter et Dale, 2000). Traditionnellement, les six étapes suivantes sont les plus utilisées selon Dale et Reiter, comme illustré à la figure 1.1.

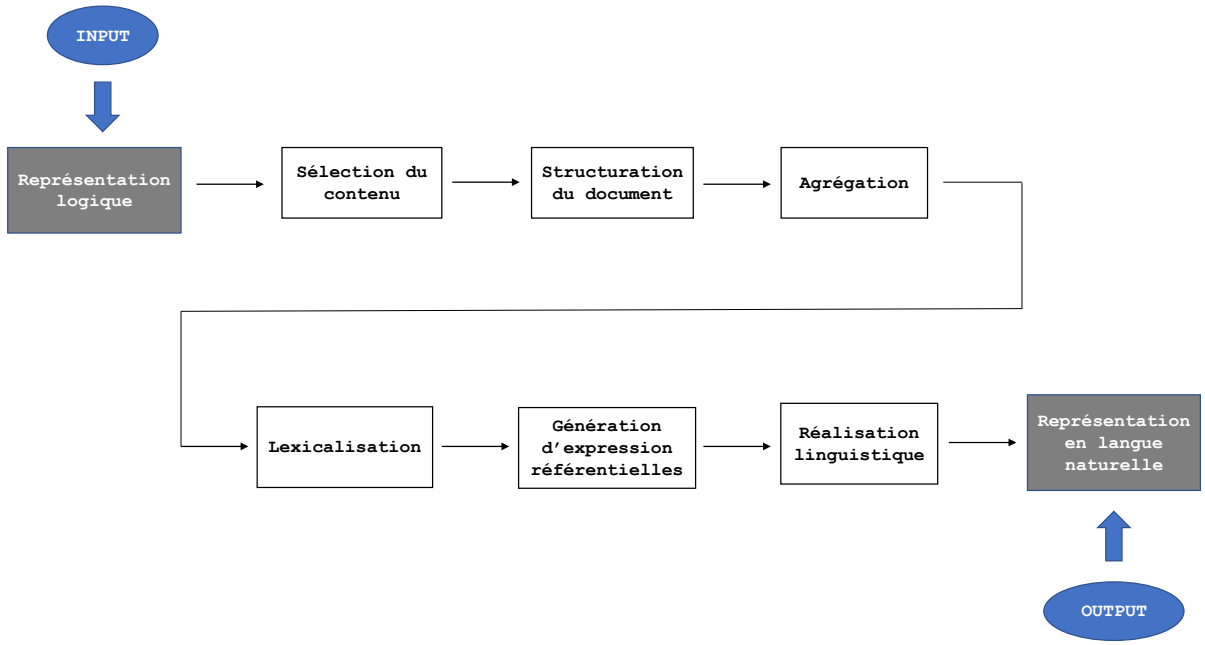

FIGURE 1.1. Pipeline classique

Beaucoup s'entendent pour dire qu'il y a deux parties. Le *quoi-dire* et le *comment-le-dire* selon (Danlos, 1983), puis le *early process* et le *late process* selon (Gatt et Krahmer, 2018). Le quoi-dire (*early process*) fait référence à blablabla et le comment-le-dire (*late process*) fait référence à blabla.

1.1.1. Sélection du contenu

Un système de GAT doit sélectionner quelles informations seront inclues dans le texte en construction et quelles informations n'y seront pas. La sélection du texte dépend entre autre de l'objectif de la tâche (par exemple si un texte est adressé à un débutant ou un expert), ainsi typiquement , il y a plus d'information ou de détails dans les données que d'information que nous voulons transmettre en texte. Ainsi, la sélection de contenu implique des choix. Par exemple, s'il s'agit de données pour un match de soccer, on ne voudrait probablement pas réaliser toutes les passes et les fautes commises durant le match, bien que ces informations figurent dans les données en input. Il faut donc les filtrer et créer des représentations sémantiques de l'information qui sont souvent exprimé en représentation formelle du langage comme des représentations logiques, des base de données, des matrices ou des graphes.

1.1.2. Structuration du document

Après avoir sélectionné le contenu, un système NLG doit décider l'ordre dans lequel les informations seront présentées. Par exemple, si on utilise encore l'exemple du soccer, on commencerait par les informations générales liées au match (où et quand le match a eu lieu, entre quelles équipes, qui était blessé cette journée, etc), puis la description des buts comptés en ordre chronologique. Ce qui résulte de cette étape du processus est le plan d'un texte : une représentation ordonnée et structurée de messages à transmettre. Durant les dernières années, il y a eu des tentatives d'implémenter des méthodes d'apprentissage machine pour que la structuration de document se fasse automatiquement (Dimitromanolaki et Androutsopoulos,2003 ; Althaus et al., 2004)

1.1.3. Agrégation

Ce ne sont pas tous les messages sélectionnés dans le plan qui doivent être exprimés dans des phrases individuelles. En combinant des phrases individuelles en une seule et même phrase, on génère un texte beaucoup plus fluide et agréable à lire (Cheng et Mellish, 2000). Ainsi, la majeure partie du temps, elle sert à enlever la redondance dans le texte. Encore une fois, des chercheurs tentent d'automatiser cette étape en implémentant des méthodes d'apprentissage machine pour que le système NLG apprenne les règles d'agrégation et les applique lorsqu'il se doit (Barzilay et Lapata, 2006).

1.1.4. Lexicalisation

À cette étape, nous avons des données sélectionnées, puis structurées et que les futures phrases ont été combinées, on peut commencer à traduire les données en langue naturelle. Cette partie est très importante car c'est là qu'on choisi les mots qui seront utilisés pour transmettre le message. Toutefois, cette section se complique parfois car il existe naturellement plusieurs manières différentes de dire la même chose. La complexité de ce processus de lexicalisation dépend des mécanismes intégrés au système pour gérer cela. Certains traitent de lexicalisation en surface, d'autres la traite profondément. Toutefois, ceux qui traitent la lexicalisation en surface sont beaucoup plus restreints dans leur choix et sont très rigides. Tandis qu'un modèle profond permet de mieux tenir compte de la richesse lexicale d'une langue, mais il faudra une approche qui unit les représentations conceptuelles avec des règles de grammaire qui encoderont les choix lexicaux et syntaxiques. D'ailleurs, (Elhadad et al. 1997) avait grandement contribué pour cela.

Convertir un graphe sémantique d'entités et de relations, en un graphe syntaxique de mots et de relations syntaxiques.

1.1.5. Génération d'expressions réferentielles

Souvent référée comme étape discriminatoire dont le but est de déterminer quelles informations doivent être générées pour qu'on puisse distinguer toutes les entités en jeu. Le but est de s'assurer que le lecteur peut idientifier chaque entité dans le texte. La meilleure manière de référer à une entité donnée. Il s'agit de la tâche pour sélectionner des mots ou des phrases qui identifieront des entités. C'est une étape du processus qui a reçu beaucoup d'attention, car il n'y a toujours pas de consensus quant à la manière de faire et c'est extrêmement difficle. (Compléter avec les autres articles)

1.1.6. Réalisation linguistique

La dernière étape de ce pipeline NLG classique est la réalisation linguistique. Lorsque tous les mots, puis successivement les phrases ont été choisies, il faut les combiner en des phrases grammaticales. Cette tâche implique couramment l'application de traits morpho-syntaxiques et la linéarisation. De même qu'insérer les mots fonctionnels (auxiliaires, etc.) et la ponctuation. (Lambrey,2016) Objectif : appliquer règles et procédés d'ordre grammaticaux aux représentations abstraites pour que l'output satisfasse les contraintes syntaxiques et morphologiques de la langue en question.

1.1.6.1. *À base de patrons*

Est utilisé lorsque le domaine est petit et que les variations sont minimes. (McRoy et al., 2003). Montrer un exemple avec le soccer. (p.19). L'avantage d'utiliser les templates est qu'on peut très bien prévoir ce qui sera généré en output, donc les risques d'erreurs grammaticales sont extrêmement faibles, du au contrôle qu'on a. Ils peuvent aussi être complémenter par des modules de règles pour pallier à certains problèmes, ce qui les rend très performant. Toutefois, l'inconvénient de tels systèmes est qu'ils sont très coûteux en termes de temps, puisque tout doit être fait à la main. Bien qu'avec la mode de l'apprentissage machine, certains systèmes apprennent à écrire des patrons (Angeli et al., 2012). Finalement, ils ne fonctionnent pas bien avec des tâches qui requièrent beaucoup de variation linguistique.

1.1.6.2. *À base de règles grammaticales*

en parler brièvement

1.1.6.3. *Statistique*

en parler brièvement et nommer quelques systèmes mentionner que c'est à la mode

1.1.6.4. *Règles vs statistiques : avantages/inconvénients*

(Belz et Kow, 2009) et (Vicente et Al., 2015) Depuis le début de cette section, nous disons qu'il y existe les méthodes staitsitques, et bcp vantent leur mérite, mais certains se sont penchés pour voir jusqu'à quel point ils sont bons. Dans cette partie, parler du blog de E.Reiter et l'article de Belz (si le temps, l'article de Vicente)

Mentionner le blog de E.Reiter concernant les approches ML et la NLG (rule-based est meilleur, mais on pourrait se servir de ML, sans toutefois rely dessus à 100% comme certains chercheurs le font. source (`https://ehudreiter.com/2016/12/12/nlg-and-ml`)

1.2. Réalisation

Tel qu'explicité dans le tableau 1.1, la réalisation est la dernière étape dans le processus de GAT. Toutefois, pour beaucoup de chercheurs, elle ne représente pas toujours les tâches décrites précédemment. Il règne une ambiguité quant aux concepts qu'incarne la réalisation. Lambrey le mentionne (Lambrey, 2017) dans son mémoire, effectivement, *réalisation* peut faire référence aux engins qui font la tâche de réalisation linguistique telle que décrite par le pipeline classique, mais elle réfère aussi aux engins qui englobent l'ensemble des tâches du module de réalisation linguistique. Pour effectuer celles-ci, ces engins partent d'un niveau plus abstrait que la réalisation dans le pipeline classique. C'est pourquoi nous faisons une distinction entre réalisateur de surface et réalisateur profond.Comme le mentionne aussi Dale et Essers 1998 (Essers et Dale, 1998), puisque le pipeline est très long à construire en NLG, il n'est pas rare que des gens construisent leur système de NLG comme bon leur semble, mais qu'ils utilisent des réalisateurs déjà existant sur le marché. :The task of building a complete natural language generation nlg system is difficult and complex. Rather than start from scratch, it makes sense to make use of whatever reusable components are available, just as someone building a complete natural language analysis system would be likely to make use of an existing parser for syntactic analysis. In the context of nlg the state of the art is such that a number of reusable components are available for that subtask in the nlg process generally referred to as linguistic realisation.comment mettre une citation en LaTex : Comme Lavoie le mentionne (Lavoie et Rambow, 1997) : It is in the realizer that knowledge about the target language resides (syntax, morphology, idiosyncratic properties of lexical items). La nécessité de créer des réalisateurs provient du fait que Realization is fairly well understood

both from a linguistic and from a computational point of view, and therefore most projects that use text generation do not include the realizer in the scope of their research.Instead, such projects use an off-the-shelf realizer like Surge.

Polguère (Polguère) parle des différents modèles de GAT.

1.2.1. Réalisateurs de surface

Les réalisateurs de surface sont appelés ainsi car ils partent d'une représentation syntaxique et lexicalisée, donc peu abstraite en comparaison avec les réalisateurs profonds. Ils correspondent plus à la phase de réalisation linguistique classique mentionnée dans le pipeline classique de Dale et Reiter. Dans le sens où ils s'occupent de blablabla.

1.2.1.1. *SimpleNLG*

(Gatt et Reiter, 2009) Dans les années passés, un consensus s'est fait autour de la tâche de réalisation via RealPro, KPML, Surge. La réalisation implique deux tâches logiquement distinctes. La première est faire les bons choix linguisitques compte tenu de l'input sémantique donné. Puis une fois que c'est fait, construire la représentation syntaxique, faire la morphologie et linéariser les phrases sont des opérations assez mécaniques. Ainsi, Simple NLG est un engin de réalisation qui s'occupe des tâches plus mécaniques. Son rôle est de résumer un large volume de données numériques. Processus de réalisation : couvrir la syntaxe et la morphologie anglaise et la linéarisation. Simple NLG c'est une bibliothèque Java, qui fournit des interfaces offrant un contrôle direct sur le processus de réalisation (comment les phrases sont construites, fléchies, combinées et linéarisées).

Ce que SimpleNLG fait : défins un ensemble de types lexicaux et phrastiques correspondant aux catégories grammaticales majeures, il décrit aussi les manières de les combiner. SimpleNLG prend en entrée des constituants simples ou des morceaux de phrases préfabriquées.

Pour construire la structure syntaxique et la linéraiser, il faut faire les quatre étapes suivantes : Premièrement, initilaiser les constitutants de base avec leurs unités lexicales correspondantes, deuxièmement déterminer les trais lexicaux des unités, troisièmement combiner les constituents en de plus grandes structures syntaxiques et quatrièmement, passer les structures résultantes au linéarisateur qui s'occupe de la linérisation et de mettre les formes fléchies aux unités lexicales en fonction des règles morphologiques.

Ainsi, il y existe un module lexical et un module syntaxique. Le module lexical comprend : le dictionnaire comme tel qui contient les unités lexicale, leur appartenance à une catégorie

grammaticale, leurs propriétés syntaxiques et morphologiques. Le module syntaxique décrit les phénomènes morpho-syntaxiques et les règles qui permettent de créer des syntagmes jusqu'à créer une structure syntaxique complète d'un énoncé. Il se veut aussi un réaliseur facile d'utilisation, tel que son nom l'indique (Daoust et Lapalme, 2015).

Noter que SimpleNLG a été traduit dans plusieurs langues : espagnol, italien, et français, portugais (Mazzei et al., 2016, Ramos-Soto 2017, Vaudry ;Lapalme 2013 ; Oliveira et Sripada)

1.2.1.2. *JSreal*

(Daoust et Lapalme, 2015)

JSreal qui est en fait JavaScript Realiser. C'est un réaliseur de texte orienté pour les programmeurs web. C'est un réaliseur de texte en français qui génère des expressions et phrases bien formées qui elles peuvent être formattées en HTML pour être explosé dans un browser. Il peut aussi s'employer seul à des fins purement linguistiques ou être intégéré dans des générateurs de textes. Les Specs de JSreal sont similaires à ceux de SimpleNLG.

Pour générer du texte JSreal prend en input des arbres syntaxiques (écrits en java), qui seront ensuite parser en un arbre syntaxique, qui sera par la suite linéarisé. Pour générer du texte, JSreal possède les modules suivants : un lexicon, en ensemble de règles syntaxiques et un ensemble de règles morphologiques. Son dictionnaire défini la catégorie des mots qui le peuple, et les traits (genre, nombre, irrégularités, etc.). Les règles morphologiques : permet d'utiliser les bonnes formes fléchies. Les règles syntaxiques

Florie " : JSreal prend en entrée des spécifications d'arbres syntaxiques en constituants immédiats écrites en fonctions JavaScript (JS). Les mots, syntagmes, etc. sont les « unités » manipulées. Les unités de base sont les catégories grammaticales (nom, verbe, etc.). Chacune déclenche l'application d'une fonction prenant un lemme comme argument et retournant un objet JS avec les propriétés et méthodes correspondants. Les syntagmes sont des unités déclenchant des fonctions prenant d'autres unités comme arguments. Les lemmes et leurs caractéristiques (traits sont répertoriés dans un dictionnaire à part. L'application des fonctions permet de construire une structure de données sous forme d'arbre regroupant toutes les propriétés des lemmes. Cet arbre est ensuite parcouru et fournit la liste des tokens de la phrase finale."

Mentionner qu'il existe aussi une version bilingue de JSreal (Molins et Lapalme, 2015)

1.2.2. Réalisateurs profonds

Ce qui caractérise les réalisateurs profonds : nécessite un théorie sous-jacente pour traiter les phénomènes, traitent l'interface sémantique-syntaxe,

1.2.2.1. *RealPro*

(Lavoie et Rambow, 1997) RealPro est implémenté en C++, il peut donc se transposer à d'autres plateformes. Il prend en input des arbres de dépendances. Les connaissances syntaxiques et lexicales sont encodées dans un fichier ASCII, ce qui leur permet d'être mis-à-jour. Donc, il prend en input des strctures syntaxiques profondes, inspiré de la TST(melcuk). Les noeuds des arbres syntaxiques sont déjà lexicalisés, ce qui fait que ce système part d'une arbre syntaxique déjà lexicalisé et non d'un input sémantique. Les arcs qui lient les noeuds entre eux sont étiquettées par des relations syntaxiques. Ainsi, RealPro ne fait pas l'étape du choix lexical, donc la réalisation se fait à partir d'unités déjà lexicalisées. Ils ont préféré laisser la lexicalisation à un autre module. L'architecture de RealPro est basé sur la TST. Séquence de correspondance entre différents niveaux de représentations. Ainsi, pour passer de la structure profonde à la strucutre de surface, le logiciel vérifie dans son dictionnaire et ses règles de grammaires pour s'assurer que le passage à la seconde représentation syntaxique est correcte. Son linguistic Knowledge Base est réparti en divers modules dont le lexique, les règles de grammaires. Ceux-ci seront réquisitionnés par les diverses composantes qui composent le pipeline de ce système. Voici en ordre, les composantes les plus importantes : Deep-syntactic component, Surface syntactic component, Deep-morphological component.

Voici un graphique représentant le pipeline de ce système :

Le dictionnaire contient de l'information quant à la partie du discours et les irrégularités morphologiques pour contourner les règles de grammaires dont celle qui ajoute *-ed* à la fin des verbes au passé. Ainsi, cette information est encodée dans l'entrée de dictionnaire.

Listing 1.1. Entrée de dictionnaire

```
LEXEME: SEE
CATEGORY: verb
MORPHOLOGY:[([mood:past-part] seen [inv])
            ([tense:past]     saw  [inv])]
```

je pourrais aussi ajouter la structure de dépendance pour montrer l'input de : This boy sees Mary.

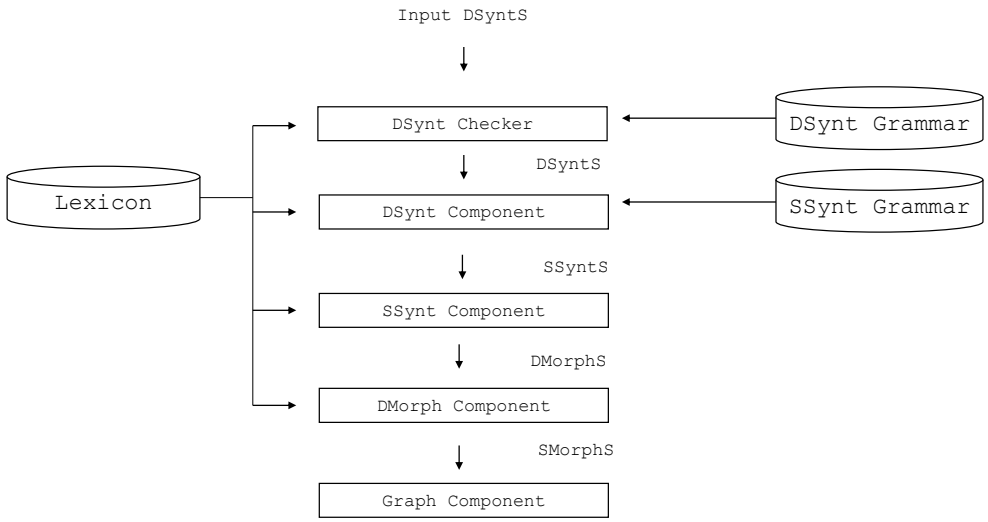

FIGURE 1.2. RealPro

1.2.2.2. *KPML*

KMPL(Bateman, 1997) est une extension multilingue du système de PENMAN (citation). C'est un système basé sur la grammaire fonctionnelle systémique (Halliday,1985). KPML contient trois éléments : un engin computationnel qui passe au travers de ressources grammaticales, une collection de ressources grammaticales, un environnement de développement pour écrire et débugger de nouvelles grammaires. Il s'agit d'un système puissant et complexe. Sa grammaire s'appelle NIGEL, il s'agit d'une grammaire de l'anglais. Toutefois, KPML se veut une ressource multilingue et elle couvre aussi des langues comme l'Allemand et le Néerlandais.

KPML prend des SPL en entrée. Il s'agit de *Sentence Planning Language*. Afin d'illustrer à quoi ressemble l'input, nous allons nous servir de Dale et Reiter qui montre une phrase exemple 'March had some rainy days'.

LISTING 1.2. SPL : input de KPML

```
(S1 \ generalized-possession
  :tense past
    :domain (N1 \ time-interval
                 :lex march
                              :determiner zero)
    :range (N2 \ time-interval
                :number plural
                           :lex day
```

```
                              :determiner some
                              :property ascription
                              (A1 \ quality :lex rainy)))
```

Compléter avec Florie

1.2.2.3. *MARQUIS*

(Wanner *et al.*, 2010)

1.2.2.4. *Forge : héritier de MARQUIS*

(Mille, Carlini, Burga et Wanner, 2017)

1.2.2.5. *Surge*

(Elhadad et Robin, 1998) Surge qui signifie *Systemic Unification Realisation Grammar of English*, c'est une grammaire à grande couverture. Elle est écrite en FUF *Functional Unification Formalism*. Il s'agit d'un langage de programmation créé pour construire des grammaires computationnelles, plus particulièrement pour les besoins de la réalisation linguistique dans un cadre de grammaire d'unification. Est basé sur FUG (Kay,1979) et ils se servent de graphes d'unification comme input à leur système. Cet input est sous forme de FD (Functional Description), il s'agit d'une collection de paires attributs-valeurs dont l'union fournit une spécification de l'énoncé à génerer. Chaque FD contient un trait CAT qui indique la catégorie syntaxique de la forme à produire. Les autres attributs dépendent de l'information requise par la grammaire pour générer une structure de cette catégorie

Par exemple, tel que trouvé dans Dale et Reiter, voici l'exemple d'une structure d'input pour 'March had some rainy days'.

Listing 1.3. FD : input de Surge

```
((cat clause)
 (proc ((type possessive)))
 (tense past)
 (partic ((possessor ((cat proper) head ((lex "March"))))
                     (possessed ((cat common) head ((lex day)))
                                                (describer ((lex rainy)))
                                                (selective yes) (number plural)))))
```

Compléter avec Florie

1.2.3. Différences entre réalisateur de surface et profond

Tel que Lambrey dans (Lambrey, 2017) l'a dit, les réalisateurs de surface sont plus facile à construire que les réalisateurs profonds. Entre autre parce qu'un RS est (à compléter). Tandis qu'un RP nécessite des linguistes pour développer l'application. Ces derniers sont plus complexes ce qui entraîne des coûts au niveau du temps et des ressources. Leur avantage réside dans le fait qu'ils sont dotés d'une profondeur linguistique leur permettant de faire l'analyse de phénomènes langagiers beaucoup plus complexes qu'un RS peut faire.

Les RP présentés dans cette section encodent sans exception leurs connaissances linguistiques dans des dictionnaires et des grammaires. On remarquera que leurs différences majeures sont directement liées aux théories linguistiques sous-jacentes qu'ils utilisent pour modéliser la langue. Ainsi des réalisateurs comme Surge ou G-TAG (que nous n'avons pas décrit ici), ne possèdent pas de dictionnaires car leur module de règles contient des arbres partiellement lexicalisés. Tandis que MARQUIS, Forge, RealPro et KPML possèdent respectivement un module de règles grammaticales et des dictionnaires. Ce qui rend ces systèmes plus enclin à traiter plusieurs langues en même temps, et ce avec facilité. Un exemple de système qui fonctionne ainsi est GenDR (Lareau, Lambrey, Dubinskaite, Galarreta-Piquette et Nejat, 2018), un réalisateur profond multilingue qui possède respectivement un dictionnaire séparé de sa grammaire. Ainsi, le système peut générer du texte à partir d'un même input sémantique dans plusieurs langues en même temps en faisant appel à des règles de grammaires partagées (certaines règles sont spécifiques à chaque langue) et des dictionnaires propres à chaque langue.

Dans le cadre de ce mémoire, nous allons nous baser sur GenDR, une extension de MARQUIS. GenDR est un projet en cours de développement dirigé par François Lareau à l'Observatoire de linguistique Sens-Texte. Nous entrerons dans les détails au chapitre suivant.

Chapitre 2

UN DICTIONNAIRE DE PATRONS DE RÉGIME : VERBNET

(ce début de chapitre sera probablement supprimé après la réorganisation du mémoire)
Dans ce chapitre, nous verrons l'apport que la ressource lexicale VerbNet peut offrir à des applications en traitement automatique du langage (TAL). Nous avons comme objectif d'extraire l'architecture de dictionnaire VerbNet pour l'implémenter dans un dictionnaire de type Théorie Sens-Texte (TST) qui servira à générer du texte. Cet objectif provient d'une problématique que nous avions rencontré auparavant. Nous voulions savoir comment organiser notre dictionnaire en ce qui concernait les verbes. Car ceux-ci sont si riches et complexes qu'il nous fallait trouver un moyen systématique d'encoder cette partie du discours. En ce qui concerne les dictionnaires, parmi la communauté TAL, il ne semble pas y avoir de consensus quant à la manière de procéder pour modéliser la classe verbale. La raison est simple, les verbes démontrent des comportements variables, très riches au niveau de l'éventail de patrons de régime possibles pour un même verbe, et assez complexes ce qui nécessite beaucoup plus d'attention que d'autres parties du discours comme les noms qui démontrent beaucoup moins de variétés d'usage quant au nombre de patrons de régime. Ce qui fait en sorte que comme tous les verbes sont des prédicats, et que les prédicats sont les noyaux des énoncés, il faut les traiter avec soin, faute de quoi leur application en NLP sera médiocre. Cette problématique nous a donc amené à constater que VerbNet s'était penché sur ce problème et nous voulions vérifier si les entrées de leur dictionnaire pouvaient s'appliquer en génération automatique de texte (GAT). De nos jours, avec les modèles stochastiques où il n'y a pas d'analyse linguistique, on ne construit plus de dictionnaires ou de règles grammaticales, mais on le laisse le soin au système d'apprendre les règles par lui-même et de développer le lexique par lui-même en en percevant les langues naturelles uniquement comme des suites de caractères. C'est une mode qui fonctionne présentement grâce à la quantité immense d'information en langue naturelle qui existe sur le web. En combinant ces nouveaux corpus incroyables avec

la puissance des ordinateurs et le développement en apprentissage machine, certains chercheurs font complètement fi de l'analyse linguistique dans leurs applications TAL et arrive à des résultats relativement bon (manque une citation ici). Toutefois, en ce qui concerne la GAT, le traitement de la langue se doit d'être impeccable (Lareau *et al.*, 2018). D'où la nécessité de développer des outils puissant et rigoureux. Toutefois, les systèmes de GAT à base de règles sont plus coûteux car il faudra développer les règles de grammaire et se doter d'un dictionnaire assez large pour couvrir une ou des langues. Cependant, tout en étant bien conscient du changement de cap dans le domaine vers le *machine learning*, nous pensons qu'il est encore primordial de développer de bons outils linguistiques computationnels à base de règles et de dictionnaires car c'est de cette manière qu'on pourra le mieux représenter les langues naturelles.

2.1. VerbNet

Ainsi, tel que mentionné précédemment, VerbNet a été créé dans un contexte où il y avait un réel besoin de réfléchir à la meilleure manière de procéder pour construire un dictionnaire qui saurait tenir compte de la richesse et la complexité que renferment les verbes (Kipper, Dang et Palmer, 2000). Les auteurs du projet trouvaient qu'il y avait un manque de lignes directrices sur l'organisation des verbes dans les dictionnaires destinés à des applications TAL. Malgré la quantité impressionnante de dictionnaires computationnels existant déjà, ils ont tout de même voulu créer un dictionnaire de verbe qui pourrait paliier à ce manque. (Faire un court retour sur les lacunes des autres dictionnaires)

2.1.1. Classes verbales de Levin

Le travail de Levin(Levin, 1993) a été de créer un dictionnaire où les verbes de la langue anglaise sont répartis dans une nombre fini de classes verbales. L'appartenance à une classe verbale est motivée par des comportements syntaxiques communs entre les verbes de cette classe. Son travail est le fruit d'observations sur les alternances de diathèses que démontrent les verbes. Levin remarquait que tout locuteur natif d'une langue est conscient des alternances de diathèses possibles d'un verbe, et ce sans avoir de connaissances linguistiques au préalable. Pour cette raison, elle a suivi son intuition et a tenté de délimiter tous les patrons de régime qu'un verbe possède, puis a évalué si d'autres verbes partageaient ces mêmes patrons. Lorsque c'était le cas, elle établissait une classe verbale qui allait regrouper les verbes se comportant de la même manière. Bien que son travail s'insère dans le cadre de la syntaxe, elle postulait que les verbes qui se comportent de la même manière syntaxiquement possèdent aussi des propriétés sémantiques communes. Ainsi, comme ces verbes partagent les mêmes

composantes sémantiques, il est normal que cela se reflète en surface via des comportements syntaxiques communs. Les composantes sémantiques sous-jacentes seraient à l'origine des comportements syntaxiques permis pour un verbe. Donc, des verbes qui partagent des comportements syntaxiques, partagent aussi des composantes sémantiques, mais ça ne veut pas dire que les verbes appartenant à la même phrase signifient la même chose. Cela veut dire qu'ils possèdent des caractérisitques sémantiques similaires. Levin met en garde que deux verbes synonymiques peuvent très bien appartenir à deux classes différentes tout comme deux verbes qui en apparence ne se ressemble pas du tout, peuvent très bien partager des composantes sémantiques similaires.

Un avantage de regrouper les verbes en classes verbales, à part la valeur théorique qui était de démontrer les propriétés sémantiques communes à ces verbes dégage aussi un côté pratique qui était de construire un dictionnaire où les entrées lexicales ne sont pas prises individuellement, mais regroupée pour faciliter l'ajout d'entrées lexicales. Lorsqu'on termine le traitement d'une entrée, on n'a pas besoin de décrire tous les patrons de régime associés, on n'a qu'à ajouter l'entrée dans la classe qui la représente. Les auteurs de VerbNet notent toutefois que le classement de certains verbes est un peu tiré par les cheveux, mais ils ont revisité le classement initial de Levin et y ont apporté quelques modifications(Schuler, 2005).

Voici un exemple tiré de la thèse de Schuler (Schuler, 2005) qui nous démontre l'idée derrière la construction du dictionnaire de Levin. On prend les verbes *break* et *cut*, puis on teste diverses configurations possibles pour confirmer s'ils appartiennent à la même classe ou bien à deux classes distinctes. On pourrait penser qu'ils appartiennent à la même classe compte tenu de leur signifié qui se ressemblent. Briser et découper partagent évidemment des composantes sémantiques car il y a le sens d'altérer quelque chose, mais le court exemple nous démontre qu'ils appartiendrait à deux classes distinctes.

(1) *Transitive construction*

a. John broke the window.
b. John cut the bread.

(2) *Middle construction*

a. Glass breaks easily.
b. This loaf cuts easily.

(3) *Intransitive construction*

a. The window broke.
b. *The bread cut.

(4) *Conative construction*

a. *John broke at the window.

b. John valiantly cut at the frozen loaf, but his knife was too dull to make a dent in it.

On voit d'abord que les constructions en (1) et en (2) sont possibles pour ces deux verbes. Toutefois, en (3) et en (4), on remarque qu'ils ne partagent pas ces cadres syntaxiques. *Break* prend seulement la construction intransitive et exclut la conative, tandis que *cut* prend la construction conative et exclut l'intransitive. Selon la logique de Levin, cela est due à des différences de composantes sémantiques. Le verbe *cut* décrit une série d'actions entreprises dans le but de séparer un objet en morceaux. Toutefois, il est possible de commencer à découper un objet sans que l'objet ne soit séparer. Dans ce scénario, on peut tout de même percevoir que l'objet a été découpé. En ce qui concerne *break*, le changement d'état, le fait d'être séparer en morceau est le coeur même de l'évènement. Si on n'arrive pas au résultat final, une tentavive de briser quelque chose ne peut être perçue.

Le projet de Levin a inspiré beaucoup de chercheurs, notamment l'équipe de VerbNet. C'est pourquoi ils ont repris une grande partie du travail de Levin. On nommera l'organisation hiérarchique de VerbNet en classe et en sous-classes et le regroupement des verbes en classes verbales. Toutefois, les auteurs de VerbNet ont retravaillé les entrées de Levin et y ont apporté des corrections et améliorations pour que le traitement des verbes soit meilleur(Kipper, Korhonen, Ryant et Palmer, 2006).

2.1.2. Composantes de VerbNet

Comme le système de Levin, VerbNet est aussi organisé en classes verbales. Chaque classe contient un ensemble de membres, une liste de rôles thématiques (accompagnés de restrictions sélectionnelles) utilisés pour décrire les arguments, puis un ensemble de cadres syntaxico-sémantiques possibles pour une classe. Chaque cadre est composé d'une brève description, suivi d'un exemple, puis d'une description syntaxique et des prédicats décrivant le cadre en question(Schuler, 2005).

2.1.2.1. *Classes verbales : organisation hiérarchique*

Les auteurs de VerbNet se sont fortement inspirés de Acquilex Lexical Knowledge Base (Copestake, 1992) qui avait organisé leur information lexicale en hiérarchie. Effectivement, les auteurs de VerbNet ont implémenté l'aspect hiérarchique en créant jusqu'à trois niveaux

de profondeur dans les classes verbales de Levin(Schuler, 2005). Ainsi, une sous-classe verbale hérite de tout le contenu lexical de la classe (ou de la sous-classe) qui la domine. Les sous-classes ont été créées pour spécifier qu'un sous-ensemble de verbes issus de la classe qui les domine démontrent des comportements différents du reste de la classe tout en étant des verbes qui partagent les restrictions de la classe dominée.(guidelines, (Schuler, 2005)). Les comportements différents comprennent : les constructions syntaxiques, les prédicats sémantiques et les restrictions sélectionnelles sur les rôles thématiques.

Prenons un exemple tiré de VerbNet pour en expliciter la hiérarchie.

- Spray-9.7
  - Spray-9.7-1
    - Spray-9.7-1-1
  - Spray-9.7-2

Spray-9.7 est le nom de la classe et celle qui dominera toutes les autres sous-classes. À l'intérieur de celle-ci, on spécifie tous les membres appartenant à cette classe, les rôles thématiques, les cadres syntaxiques et les prédicats sémantiques. Puis *Spray-9.7-1* est une sous-classe fille qui hérite de l'information de sa mère, mais précise d'autres informations. Comme un sous-ensemble de verbes propres à ces comportements différents. Puis *Spray-9.7-1-1* est une sous-classe d'une sous-classe et la hiérarchie continue. Elle héritera des traits de sa classe mère ainsi que de la classe qui domine sa classe mère. Finalement Spray-9.7-2 est la classe sœur de *Spray-9.7-1* donc, elle hérite aussi des traits de *Spray-9.7* mais ne partage pas les particularités de *Spray-9.7-1*

Tel que démontré dans l'exemple, les classes et sous-classes sont numérotées. D'abord, pour expliciter le lien hiérarchique qui transcende à l'intérieur d'une classe. Mais la numérotation est aussi directement héritée du système de Levin (Levin, 1993). De cette manière, les classes sont numérotées par des chiffres allant de 9-109 (guidelines). Le numéro associé à des classes sert à représenter le partage de caractéristiques sémantiques et syntaxiques entre les classes verbales. Par exemple, les classes signifiant 'mettre quelque chose' commenceront par le chiffre 9.

- put 9.1
- put spatial 9.2
- funnel 9.3
- put direction 9.4

- pour 9.5
- coil 9.6
- spray 9.7
- fill 9.8
- butter 9.9
- pocket 9.10

2.1.2.2. *Membres*

Ainsi, tel que mentionné précédemment, les entrées lexicales dans VerbNet sont des classes verbales. Contrairement à des dictionnaires où chaque entrée individuelle représente un verbe, ici on a une entrée lexicale qui représente une panoplie de verbes. Ce qui permet à VerbNet de couvrir très largement la langue anglaise dans un format adapté aux applications TAL. Ainsi, pour garnir leur section *Members*, qui regroupe les verbes appartenant à une classe verbale en question, les chercheurs ont puisé dans d'autres ressources lexicales dont la base de données LCS (Ayan et Dorr, 2002) pour enrichir leur lexique.

La figure suivante montre à quoi ressemble la section *Members* dans VerbNet. Chaque classe verbale représente un document XML qui contient toutes les sections sous formes de balises.

LISTING 2.1. les membres

```
<VNCLASS ID="give-13.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="vn_schema-3.xsd">
    <MEMBERS>
        <MEMBER name="deal"
                wn="deal%2:40:01 deal%2:40:02 deal%2:40:07 deal%2:40:06"
                grouping="deal.04"/>
        <MEMBER name="lend"
                wn="lend%2:40:00"
                grouping="lend.02"/>
        <MEMBER name="loan"
                wn="loan%2:40:00"
                grouping=""/>
        <MEMBER name="pass"
                wn="pass%2:40:00 pass%2:40:01 pass%2:40:13 pass%2:38:04"
                grouping="pass.04"/>
        <MEMBER name="peddle"
                wn="peddle%2:40:00"
                grouping="peddle.01"/>
        <MEMBER name="refund"
                wn="refund%2:40:00"
```

```
                grouping="refund.01"/>
        <MEMBER name="render"
                wn="render%2:40:02 render%2:40:01 render%2:40:00 render%2:40:03"
                grouping="render.02"/>
        <!--removed "trade" from class because doesn't take "to-PP"-->
        <!--removed "volunteer "from class because doesn't fit dative or-->
        <!--PP recipient PP frames-->
    </MEMBERS>
```

Ainsi, dans cet exemple, on voit que *deal, lend, loan, pass, peddle* et *refund* sont les membres de la classe *give-13.1*.

2.1.2.3. *Rôles thématiques*

VerbNet emploi 23 rôles thématiques pour identifier les arguments sélectionnés par les verbes. Ils ont opté pour cette approche d'identification car elle permet d'ajouter de l'information sémantique sur les participants. Contrairement à une approche où on énumère les arguments*Arg-1 Verbe Arg-2* comme dans PropBank (Palmer, Gildea et Kingsbury, 2005). À la base, les rôles thématiques ont été mis de l'avant par Fillmore (Fillmore, 1968) et Jackendoff (Jackendoff, 1972) pour identifier les arguments en leur assignant un rôle sémantique. Chaque argument se fait donné un rôle unique.

VerbNet critiquait les autres dictionnaires verbaux qui n'offraient pas de contenu sémantique. C'est pourquoi ils font la promotion de leur aspect sémantique via la section *Semantic frames* et les *Thematic Roles*(Schuler, 2005). Concrètement, parmi les 23 rôles thématiques choisis par VerbNet, certains proviennent de Fillmore et Jackendoff, d'autres s'en inspirent. C'est pourquoi ils se justifient en disant que le chiffre 23, et la nature des rôles est effectivement arbitraire, mais utile à la construction de leur dictionnaire. Voici la liste des rôles thématiques qu'ils ont choisi : *actor, agent, asset, attribute, beneficiary, cause, location, destination, source, experiencer, extent, goal, instrument, material, product, patient, predicate, recipient, stimulus, theme, time, topic*. Ces rôles ne sont pas spécifiques à des classes en particulier, les auteurs voulaient des rôles pouvant identifiés tous les arguments possibles dans leur corpus. Donc, des rôles assez génériques pouvant se prêter à divers cadres.

À l'intérieur de chaque classe verbale (et sous-classe si c'est nécessaire), les rôles thématiques en jeu y sont listés dans la section <THEMROLES>. Ils sont ensuite mappés aux arguments sélectionnés dans les cadres syntaxiques et sémantiques (qu'on voit à la figure "cadres syntaxique").

Listing 2.2. les rôles thématiques

```
    <THEMROLES>
        <THEMROLE type="Agent">
```

```
            <SELRESTRS logic="or">
                <SELRESTR Value="+" type="animate"/>
                <SELRESTR Value="+" type="organization"/>
            </SELRESTRS>
        </THEMROLE>
        <THEMROLE type="Theme">
            <SELRESTRS/>
        </THEMROLE>
        <THEMROLE type="Recipient">
            <SELRESTRS logic="or">
                <SELRESTR Value="+" type="animate"/>
                <SELRESTR Value="+" type="organization"/>
            </SELRESTRS>
        </THEMROLE>
    </THEMROLES>
```

Pour les besoins de notre travail, nous n'utilisons pas les rôles thématiques dans notre travail, mais nous voulions souligner qu'ils étaient importants pour les créateurs de VerbNet. Voir les raisons de Melcuk p.230

2.1.2.4. *Restrictions sélectionnelles*

Les restriction sélectionnelles s'ajoutent aux rôles thématiques. Il s'agit de restrictions imposées aux rôles thématiques afin que certains types d'arguments soient sélectionnés. Ces traits fournissent encore plus d'informations sémantiques sur l'argument. Dans l'exemple fournit ici, on remarquera que l'Agent est de type animé ou une organisation.

LISTING 2.3. les restrictions sélectionnelles

```
    <THEMROLES>
        <THEMROLE type="Agent">
            <SELRESTRS logic="or">
                <SELRESTR Value="+" type="animate"/>
                <SELRESTR Value="+" type="organization"/>
            </SELRESTRS>
        </THEMROLE>
```

2.1.2.5. *Cadres syntaxiques*

Les cadres syntaxiques sont compris dans la section *FRAMES* de VerbNet. À l'intérieur de cette balise, on retrouve une autre balise, se nommant *FRAME*, qui contient les balise *SYNTAX* et *SEMANTICS*. Respectivement, la première décrit un comportement syntaxique régit par la classe verbale, tandis que la deuxième décrit les prédicats sémantiques impliqués pour un tel cadre. La balise *SYNTAX* nous donne une description d'une réalisation de surface d'une construction syntaxique.

Tel que mentionné plus tôt, comme le reste des informations mentionnées jusqu'à présent, les cadres syntaxiques sont partagés par l'ensemble d'une classe. Toutefois, lorsque certains cadres syntaxiques sont spécifiques à un sous-groupe, on crée une sous-classe qui aura successivement une balise *FRAMES* contenant les cadres syntaxiques propres à ce sous-groupe de verbes. Cette section nous donne de l'information de nature syntaxique. Elle explicite les liens qui unissent les rôles thématiques au verbe et l'ordre dans lequel ils peuvent apparaître en surface. Cette section est la section que nous voulions extraire à la base de notre travail. Nous voulions un dictionnaire qui énumérait exhaustivement tous les arguments sélectionnés par des verbes. Ainsi, cette section démontre explicitement comment le verbe se combine, avec quel type d'argument, quel genre de préposition il régie.

Dans la figure ci-bas, on voit la balise *SYNTAX* qui se trouve à l'intérieur de la balise *FRAME*. Puisque cette structure syntaxique est tirée de la classe *ID="give-13.1"*, une réalisation de surface possible pour ce cadre serait : *They lent a bicycle to me*. Dans ce contexte, Paul est l'agent, puis on a le verbe, suivi d'un thème, puis d'une préposition et finalement du récipiendaire.

Listing 2.4. cadres syntaxiques

```
<SYNTAX>
    <NP value="Agent">
        <SYNRESTRS/>
    </NP>
    <VERB/>
    <NP value="Theme">
        <SYNRESTRS/>
    </NP>
    <PREP value="to">
        <SELRESTRS/>
    </PREP>
    <NP value="Recipient">
        <SYNRESTRS/>
    </NP>
</SYNTAX>
```

2.1.2.6. *Prédicats sémantiques*

En lisant la revue de littérature de VerbNet, une caractéristique distincte sur laquelle les auteurs ont misé, est l'aspect sémantique du dictionnaire. Ils contestent le fait que beaucoup de dictionnaire faisait un traitement syntaxique superficiel et délaissait complètement la sémantique. C'est pourquoi, ils ont développé une section sémantique. Cette section est décrite par une suite de prédicats sémantiques. Chaque prédicat est décrit par une liste d'argument, qui sont à leur tour décrits par deux caractéristiques : *type* et *value*. Le cadre

sémantique ci-dessous complète le cadre syntaxique que nous venons d'exposer. Ainsi, il s'agit de la sémantique qu'on retrouverait si on analysait une phrase comme : *They lent a bicycle to me* ou *They lent me a bicycle*. Ça peut décrire ces deux situations puisqu'on fait abstraction de la syntaxe et on traite uniquement les prédicats en jeu.

LISTING 2.5. Les prédicats sémantiques

```
<SEMANTICS>
                <PRED value="has_possession">
                    <ARGS>
                        <ARG type="Event" value="start(E)"/>
                        <ARG type="ThemRole" value="Agent"/>
                        <ARG type="ThemRole" value="Theme"/>
                    </ARGS>
                </PRED>
                <PRED value="has_possession">
                    <ARGS>
                        <ARG type="Event" value="end(E)"/>
                        <ARG type="ThemRole" value="Recipient"/>
                        <ARG type="ThemRole" value="Theme"/>
                    </ARGS>
                </PRED>
                <PRED value="transfer">
                    <ARGS>
                        <ARG type="Event" value="during(E)"/>
                        <ARG type="ThemRole" value="Theme"/>
                    </ARGS>
                </PRED>
                <PRED value="cause">
                    <ARGS>
                        <ARG type="ThemRole" value="Agent"/>
                        <ARG type="Event" value="E"/>
                    </ARGS>
                </PRED>
            </SEMANTICS>
```

Pour mieux exposer leur sémantique, nous avons fait un graphique qui exemplifie la sémantique de ce cadre.

Est-ce que je l'explique en détail ? Car je n'utiliserai pas du tout leur approche sémantique.

2.1.3. Dictionnaires verbaux concurrents

Dans la section suivante, nous faisons une revue de littérature concernant les autres dictionnaires verbaux qui existent sur le marché. Focusent généralement sur les cadres de sous-catégorisation (valence, patron de régime, structure argumentale, etc.) . Devrais-je expliquer c'est quoi ici ?

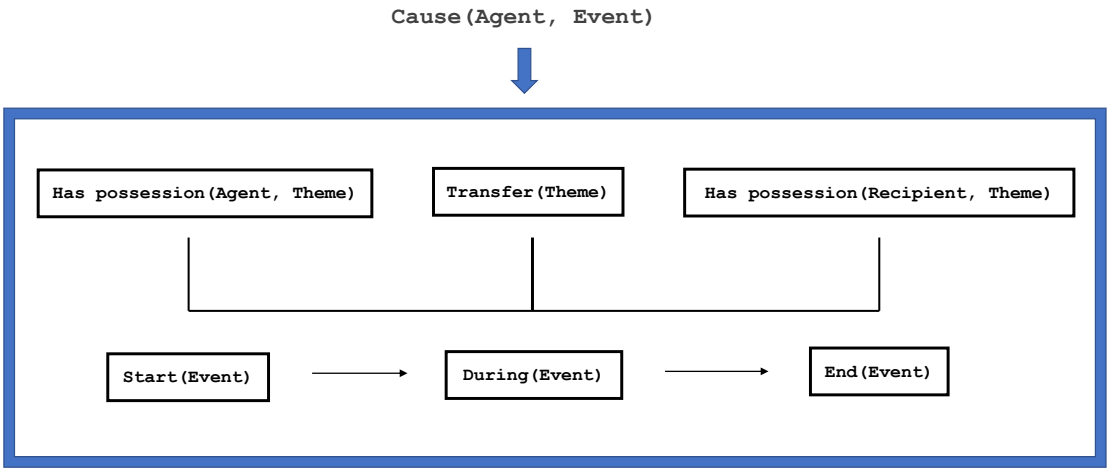

FIGURE 2.1. prédicats sémantiques

2.1.3.1. *WordNet*

Wordnet est une base de données lexicales traitant les verbes, noms, adjectifs et adverbes de la langue anglaise. Cette base de donnée s'organise en *synset*, des ensembles de synonymes. Il ne s'agit pas de synonymes exacts, loin de là, il s'agit plutôt d'un ensemble de mots unis par des traits conceptuo-sémantiques. Ces *synset* sont joints d'une définition et d'une phrase exemple. Comme VerbNet, WordNet est aussi une base de données hiérarchisée. Elle est construite via des liens d'hyperonymie et d'hyponomie entre les *synsets* ce qui nous permet de naviguer dans la base de données pour visualiser dans quel concept un synset donné est inclu. Ainsi, tel que mentionné, les entrées lexicales dans ce système sont des synset et ceux-ci appartiennent à l'une des classes suivantes. S'il s'agit de verbes dénotant des actions ou des évènements, ils seront classés parmi : *motion, perception, contact, communication, competition, change, cognition, consumption, creation, emotion, perception, possession, bodily care and functions, social behavior and interactions*. S'il s'agit de verbes dénotant des états, on le retrouvera parmi les classes de type : *resemble, belong, suffice,* ; ou des classes de type : *want, fail, prevent, succeed*, ou de types aspectuels comme *begin* (Fellbaum, 1998).À l'intérieur d'une entrée, on retrouve aussi des liens de divers ordres : synonymes, antonymes, troponymes, entailment et causation.(Schuler, 2005). Ce qui leur a permis de tisser une toile sémantique assez volumineuse.

À la base, WordNet a été conçu comme réseau sémantique, c'est pourquoi il contient explicitement aussi peu d'information syntaxique. La ressource fournit des définitions, des exemples et des *synsets*, mais ne nous donne pas d'information sur la structure sémantique

ou syntaxique des verbes. Elle est systématiquement implicite, contrairement à VerbNet qui l'explicite à l'aide de ses cadres syntaxiques. Il s'agit de la raison principale qui nous a poussé à ne pas utiliser cette ressource pour créer notre dictionnaire verbal. Nous voulions une base de données explicitant très clairement les différents actants régis par un verbe, ainsi que les prépositions sélectionnnées par celui-ci. Toutefois, notre dictionnaire a la possibilité d'être aussi mapper aux entrées de WordNet, car VerbNet a fait un mapping entre ses entrées lexicales et celles de WordNet. Chaque verbe dans VerbNet est mappé à un *synset* verbal de WordNet, si c'est possible et qu'il y existe un équivalent(Schuler, 2005).

manque les statistiques

2.1.3.2. *FrameNet*

Parallèlement au projet WordNet, s'est développé FrameNet. Le projet de Berkeley FrameNet est basé sur un corpus manuellement annoté, qui contient de l'information sur les noms, adjectifs et verbes de la langue anglaise. Dans FrameNet, les unités lexicales sont décrites en termes de *frame semantics*, qu'on traduirait par la sémantique des cadres.Les semantic frames sont définis comme des représentations schématiques de situations impliquant des participants , propositions et d'autres rôles conceptuels Le but de cette ressource est d'encoder la sémantique du lexique de l'anglais dans un modèle que peuvent lire les machines (Baker, Fillmore et Lowe, 1998). Ce projet couvre la sémantique des domaines suivants : santé, chance, perception, communication, transaction, temps, espace, corps , motion, étapes de la vie, contexte sociaux, émotion et cognition. Cette base de données lexicales est composée de trois modules. D'abord, un dictionanire dont les entrées sont les unités lexicales traitées. Suivi d'un dictionnaire de *frames* et complété par des exemples annotées manuellement correspondant aux *frames*. Ainsi, il faut passer par le dictionnaire d'entrées lexicales, pour ensuite identifier le cadre qui lui est associé dans le dictionnaire de cadres sémantiques. Ainsi, il faut d'abord chercher dans le dictionnaire d'entrées lexicales pour ensuite trouver le ou les cadres qui lui sont associés dans le dictionnaire de cadres sémantiques. D'ailleurs, les descriptions des frames sont encodés en structures conceptuelles. Et les phrases exemples manuellement annotées sont des preuves empiriques que les *frames* ont lieu d'être. Les frames décrivent la structure argumentale d'une unité lexicale. Ces arguments sont identifiés par des étiquettes similaires aux rôles thématiques. On les appelle des *frame elements* et ils sont extrêmement nombreux car ils sont spécifiques aux cadrex qu'ils décrivent.En frame semantics, un frame correspond à un scénario qui implique une intéraction et des participants (Shi et Mihalcea, 2005). À noter qu'il y existe aussi une organisation hiérarchique où on a des sous-frames qui héritent de traits des frames parents. Tout comme VerbNet l'a fait avec WordNet,

un mapping a été effectué entre les entrées de VerbNet et FrameNet. Cela s'est fait en deux étapes, ils ont mapper les classes de VN avec les frames de FN, puis les frames elements aux rôles thématiques (Shi et Mihalcea, 2005). Finalement, d'un point de vue pratique, FN est généralement utilisé comme *semantic parser*. Des chercheurs font des parse tree syntaxique mais qui tiennent compte des participants et de leur relation avec le verbe(Shi et Mihalcea, 2005).

manque les statistiques Expliquer pourquoi nous n'avons pas choisi FrameNet : n'explicite pas très clairement les patrons de régime. On les comprend via les phrases exemples et les frames elements, mais ce n'est pas suffisant, il y aurait beaucoup de travail à faire avant de l'implémenter dans MATE.

2.1.3.3. *XTAG*

Les chercheurs du projet XTAG ont construit une grammaire de la langue anglaise basé sur le formalisme de *Tree Adjoining Grammar* (TAG). Cette ressource offre des descriptions syntaxiques riches des verbes en anglais. Chaque unité lexicale se fait assigner un ensemble d'arbres-TAG décrivant ses comportements syntaxiques. Les arbres reflètent la structure argumentale de ces unités lexicales. Les arbres peuvent se construire via deux opérations : substitution et adjonction. En adjoignant de nouvelles branches ou en substituant des branches, la grammaire TAG permet de rendre compte des divers phénomènes linguistiques de la langue anglaise. XTAG inclut des descriptions syntaxiques pour 33 000 items lexicaux dont 9000 verbes (Research Group, 2001). XTAG organise son information syntaxique en créant des familles d'arbres. À l'intérieur de celles-ci, on distingue les arbres par des alternances syntaxiques. Ainsi, dans XTAG, Les classes verbales sont organisées de cette manière : chaque verbe dans le dictionnaire correspond à plusieurs familles d'arbres et chaque famille regorge d'arbres individuels issus de différentes transformation syntaxiques de surface pour une même structure argumentale canonique. Ainsi, on n'a pas à lister tous les arbres individuels possibles correspondant à un verbe, car celui-ci se fait assigner des familles d'arbres (Doran, Egedi, Hockey, Srinivas et Zaidel, 1994).

Finalement, comme avec WordNet et FrameNet, Les auteurs de VerbNet ont aussi mappé leurs cadres syntaxiques aux arbres de XTAG (Ryant et Kipper, 2004). D'ailleurs, cela leur a permis de couvrir des descriptions syntaxiques qu'ils n'avaient pas répertoriés. VerbNet couvre surtout la voix déclarative, tandis que XTAG explore toutes les transformations possibles de voix. Cependant, XTAG ne fait pas de distinctions pour les différents sens des verbes et il s'agit là d'une composante cruciale à la construction d'un bon dictionnaire en GAT. C'est pourquoi nous n'avons pas opté pour ce dictionnaire. De plus, le formalisme

dans lequel les arbres sont encodés ne s'exporte pas facilement dans un format réutilisable en TST.

2.1.3.4. *Lexical conceptual structures-LCS*

La base de données LCS de Dorr s'est construite à partir des théories de sémantique lexicale de Jackendoff. Celui-ci argumente en faveur d'une approche de décomposition sémantique des verbes. Ceux-ci sont décrits en termes de leur structure conceptuelle lexicale(Dorr, 1992). Une structure LCS est un graphe, il s'agit d'une représentation sémantique du lexique. Dans ce système, la structure syntaxique découle des primitifs sémantiques. Un graphe LCS est une représentation sémantique où il y a des noeuds dont une racine. Chaque noeud a des spécifications avec des types d'information comme : type, primitif et champ. Type ; event, state, path, manner,etc. Puis après avoir spécifier le type, on spécifie le primitif sémantique du verbe (être, aller, rester,etc.) et les champs sont des traits qui agissent comme des restrictions sur les noeuds.Ces structures sont des représentations hiérarchiques non-linéaire composées d'une tête logique (la racine du graphe), d'un sujet logique (un seul) , d'arguments logiques et de modificateurs logiques. En ce qui concerne le traitement des verbes, la racine du graphe sera un verbe et les sujets/arguments logiques seront les participants sélectionnés par le verbe. Concrètement, ce qui décrit la sémantique des graphes est une combinaison de constituents et primitifs conceptuels et de champs sémantiques. D'abord, les constituents conceptuels appartiennent à un ensemble de catégories : chose, évènement, état, lieu, chemin, propriété, but, manière, montant, temps. Ensuite, les champs sémantiques sont des traits qui agissent comme des restrictions sélectionnelles(ex : +temp, +loc, +poss). Finalement, les primitifs conceptuels : ÊTRE, ALLER, RESTER , CAUSER, INCHOATIF, EXTENSION. Une décomposition sémantique des verbes en termes de structures lexicales conceptuelles explique leur propriété syntaxiques. Tel que Levin l'avait perçu, les propriétés sémantiques des verbes influenceront leur comportements syntaxiques. À l'intérieur de ce cadre théorique, on pense que les verbes avec des LCS similaires partagent aussi des comportement syntaxiques comme des alternances de diathèses. Ils utilisent aussi des rôles thématiques pour montrer la structure argumentale. La base de données de Dorr prend aussi la hiérarchie et se base sur les classes de Levin pour structurer son information. Dans cette base de données, les verbes sont aussi rassemblés en classes verbales. Ce qui unit les membres à une classe verbale est le partage d'une structure LCS commune. Ainsi, tous les membres d'une classe partage la même structure sémantique, mais le contenu sémantique selon chaque verbe pour satisfaire les contraintes lexicales de ceux-ci (Traum et Habash, 2000).

Nous n'avons pas pris cette ressource car nos représentations sémantiques opèrent déjà la même fonction que ces graphes LCS. De plus, le caractère syntaxique de ce système ne fournit pas du tout ce que nous cherchions. Nous voulons un dictionnaire qui énumère les différents patrons de régime possibles pour un verbe donné. Toutefois, ce système a été utilisé de la même manière que nous générons du texte automatiquement en différentes langues. Cette base de données lexicales a été utilisée pour faire de la traduction automatique (Dorr, 1992). Cela a été fait dans le cadre du projet UNITRAN qui traite :l'espagnol, l'anglais et l'allemand. À l'aide de représentation basées sur la LCS , ils pouvaient générer des traductions équivalentes entre les langues à partir d'une même représentation. Par la suite, les dictionnaires se chargent des spécificités de chaque langues. Notre système GenDR fonctionne aussi de cette manière.

manque les statistiques

2.1.3.5. *Comlex*

Comlex est une base de données lexicales développée pour l'anglais à NYU. C'est une ressource syntaxique riche, mais dont il faut débourser pour s'en servir. Les auteurs de ce système voulait créer un dictionnaire syntaxique sur les verbes de l'anglais à des fins computationelles (Grishman, Macleod et Meyers, 1994). Ils ont opté pour un système qui se voulait le plus neutre du point de vue de la théorie afin qu'il puisse être utilisé dans divers cadres de recherche. Ce dictionnaire ne traite pas uniquement que les verbes, mais c'est la partie qui nous intéresse. En ce qui concerne ceux-ci, le système décrit pour chaque verbe les compléments possibles qu'il pourrait sélectionner ainsi que les spécificités propres à certaines constructions (choix d'une préposition,etc.) Les entrées lexicales ont été manuellement ajoutées car ils ne pensaient pas que des méthodes automatiques pouvait bien rendre compte des verbes moins fréquents, et ils voulaient que leurs entrées sois dépourvues d'erreurs. Contient des descriptions syntaxiques pour 6000 verbes.

Nous n'avons pas pris ce système car, d'abord il faut payer la license, puis nous avions lu l'évaluation que VerbNet avait menée et il en ressortait que Comlex ne distingue pas les différents sens des verbes. Ce qui est problématique si on veut générer la phrase la plus correcte possible.

2.1.3.6. *A large SCF lexicon for NLP apps : Valex*

Valex est un projet de Korhonen, il s'agit d'un dictionnaire de cadre de sous-catégorisation (SCF) de l'anglais (Korhonen, Krymolowski et Briscoe, 2006). Elle a bâti son dictionnaire via des méthodes d'acquition automatiques. L'auteure vante les mérites d'une acquisition

automatique et se défend en stipulant que les dictionnaires bâtis manuellement comportent naturellement plus d'erreurs. Elle pense aussi qu'ils sont plus coûteux en termes de temps et de ressource, car il faut les entretenir et les enrichir. Finalement, elle ajoute que les dictionnaires manuellement acquis comportent une faille cruciale, il leur manque de l'information statistique. Par exemple, quel cadre de sous-catégorisation est le plus utilisé pour un verbe donné et les SCF les moins fréquents. Puisque de nombreuses applications TAL fonctionnent avec des méthodes probabilistes, la présence d'information statistique est cruciale à leur bon fonctionnement.

Dans son article, elle explique qu'elle a utilisé le système d'acquisition de Briscoe et Caroll (Briscoe, Carroll et Watson, 2006) qui se base sur la méthode RASP. À partir de textes non-annotés, les SCF sont extraits grâce au système RASP. Ainsi, les données brutes provenant des corpus sont d'abord tokénisées, étiquettées, lématisées puis parsées utilisant RASP. Puis les SCF sont extraits des phrases parsées. Ainsi, chaque entrée lexicale est une combinaison d'un verbe et d'un SCF ce qui résulte en un dictionnaire de base. Finalement, il est filtré car, puisque c'est une méthode automatique, le système déduit des SCF qui n'en sont pas. Il faut donc les retirer du dictionnaire. D'ailleurs, ces systèmes se retrouvent avec des problèmes de rappel. Certains SCF ne seront pas extraits puisque le système ne les reconnaîtra pas comme des SCF. Ils utilisent des dictionnaires construits manuellement pour trouver ces SCF manquants.

Dans Valex, une entrée lexicale comprend entre autre : la combinaison d'un verbe et d'un SCF, la syntaxe des arguments, la fréquence d'utilisation du SCF. Bien que ce système aurait été potentiellement bon, nous avons préféré nous tourner vers VerbNet. D'abord, ce dictionnaire ne différencie pas les sens des verbes, de plus, l'architecture du logicielle ne nous permet pas de tirer parti du principe d'héritage des traits. Contrairement à VerbNet qui le permet, d'autant plus que ça permet de réduire la quantité d'information dans notre dictionnaire. Finalement, comme notre système fonctionne avec des règles de grammaire et que ce n'est pas un générateur de texte basé sur des méthodes stochastiques, l'apport d'informations statistiques que Valex offre ne nous était pas utile pour l'instant.

statistiques : coverage

2.1.3.7. *LexSchem*

LexSchem est un dictionnaire de verbe pour le français créé par Messiant. Il jusitifiait la valeur de son projet en disant que que l'information la plus utile qu'un dictionnaire peut offrir sont les cadres de sous-catégorisation des verbes (Cédric Messiant et Korhonen, 2008). Ces *subcategorization frame* (SCF) capturent, au niveau syntaxique, les différentes combinaisons

d'arguments qu'un prédicat lie. Messiant ajoute que comme les verbes sont au centre des énoncés, un dictionnaire qui se concentre sur les cadres de sous-catégorisation peut être très bénéfiques à des applications TAL. Nous avons vu jusqu'à maintenant qu'ils peuvent être utilisés pour le parsing et la traduction automatique par exemple. Toutefois, suivant les pas de Korhonen (Korhonen *et al.*, 2006), Messiant a bâti un dictionnaire de SCF pour le français via une acquisition automatique. Il se justifie en disant que cette technique a déjà fait ses preuves dans des applications réelles malgré le fait qu'elle n'est pas aussi précise et détaillée qu'une approche manuelle. Mais elle est beaucoup moins coûteuse en termes de temps et de ressource. De plus, une approche automatisée permet d'extraire de l'information qui pourrait s'avérer très utile pour des applications TAL. Notamment, les statistiques et la fréquence d'utilisation d'un SCF. Son dictionnaire a été acquis à partir de corpus non-annoté. Par la suite,les SCF acquis automatiquement sont incorporés dans lexSchem. Voici la démarche qu'il utilisa, d'abord ils prennent des données brutes, puis il étiquette et lemmatise les mots pour ensuite parser le tout. Après il ne reste qu'à en extraire les SCF. Dans LexSchem, Les entrées lexicales sont composées essentiellement de : L'unité lexicale, ses cadres de sous-catégorisation et des phrases exemples tirées de corpus ainsi que la fréquence d'utilisation du SCF.

Ce que nous retenons de ce système, c'est qu'il pourrait être utile dans un avenir où nous voulions extraire VerbeNet qui est une version francophone de VerbNet. Nous pourrions ainsi complémenter la ressource francophone par une autre ressource franchophone. De plus, tel que VerbNet l'a fait, LexSchem construit ses entrées lexicales en misant sur les cadres de sous-catégorisation. Nous pensons aussi qu'un dictionnaire verbal en TAL devrait surtout incorporer ces données, ce qui nous intéresse sont les cadres de sous-catégorisation, car ceux-ci sont la partie la plus dure à traiter en TAL.

statistiques :

2.1.3.8. *VDE-Valency dictionary of English*

Le VDE est un dictionnaire de valence tout comme les dictionnaires précédents qui liste la manière dont un verbe peut se combiner avec ses arguments (Herbst, Heath, Roe et Götz, 2004). Le VDE contient les valences de 511 verbes (il traite aussi les noms et les adjectifs). Dans ce dictionnaire, chaque entrée décrit une valence possible pour un verbe accompagné d'un exemple provenant de la *Bank of English*. Lors de sa création, le VDE n'était pas destiné à des applications TAL, mais les auteurs se sont rapidement rendus compte que ça pourrait intéresser des linguistes computationnels. Ainsi est né le *Erlangen Valency Pattern Bank* (Herbst et Uhrig, 2009), un outil de TAL qui liste les patrons de valence identifié

par le VDE. Dans le VDE, les 511 verbes qui y figurent ont été choisis sur la base qu'ils sont fréquents dans la langue anglaise, qu'ils démontrent des propriétés complexes et qu'ils sont utiles pour des apprenants de l'anglais. Les patrons de valence qu'on retrouve dans le VDE proviennent d'une étude de corpus fait sur le COBUILD. Les patrons y sont décrits en termes de syntaxe de surface. Leur dictionnaire est réparti en deux où d'un côté on a la liste des 511 verbes et les patrons de valence leur étant associés, puis dans un autre dictionnaire les patrons de valence de la langue anglaise. Il s'agit aussi d'un système qui distingue les différents sens que peuvent prendre les verbes.

Bref, il s'agit d'un dictionnaire qui couvre de verbes, mais les plus fréquents. Toutefois, il s'agit d'un travail manuel, donc on s'attend à ce qu'il ne comporte pas beaucoup d'erreurs, et on pourrait ainsi en extraire une partie pour complémenter le dictionnaire de VerbNet si tel est le besoin.

2.1.3.9. *Dicovalence*

Le Dicovalence est un dictionnaire de valence pour la langue française. Une entrée lexicale dans ce dictionnaire correspond à la combinaison d'un verbe, d'un cadre valenciel et d'un exemple. Comme il y a 3700 verbes traités et que la plupart des verbes comptent plus d'un cadre valenciel, il y a plus de 8000 entrées lexicales dans ce dictionnaire. Les informations à l'intérieur des cadres valenciels sont décrites par l'approche pronominale en syntaxe (Département de linguistique, 2017). Dans leur système, ce que plusieurs appellent des participants, des actants ou des arguments, sont appelés des paradigmes. Le Dicovalence a aussi été créé dans une optique de TAL et d'enseignement de la langue. Contrairement à des systèmes comme FrameNet, ou VerbNet, ils identifient leurs arguments en les numérotant. Le paradigme p0 étant souvent le sujet, le p1 étant l'objet direct, le p2 étant l'objet indirect et tous les autres étant des obliques. Ce dictionnaire contient beaucoup d'information utile autre que les cadres valenciels. Il y a : des phrases exemples, des traductions du verbe en anglais et en allemand, des restrictions sélectionnelles sur les paradigmes, le type d'auxiliaire (avoir/être) et les constructions passives.

Nous n'utiliserons pas ce système puisque nous traitons l'anglais, mais le survol de ce système nous confirme que VerbNet était un bon choix, car les entrées importantes sont souvent les mêmes. Les cadres syntaxiques, les phrases exemples. Toutefois, nous pourrions nous en servir pour compléter l'implémentation de VerbeNet français dans notre système pour une génération multilingue.

2.1.4. Utilisation de VerbNet dans des applications TAL

VerbNet a été utilisé dans un nombre impressionnant de travaux de recherche. Ce qui en témoigne de son efficacité et de sa réputation. Voici notamment quelques projets de recherche où les chercheurs se sont servis de cette base de données lexicales pour effectuer leurs travaux.

2.1.4.1. *Construction de graphes conceptuels*

(Hensman et Dunnion, 2004) Les graphes conceptuels servent à représenter le sens d'un énoncé. Dans une recherche, ils se sont servis des informations lexicales de VerbNet pour créer des graphes automatiquement. Ainsi, ils prennent des documents provenant de corpus, ils en extraient les phrases, ils les parsent syntaxiquement, puis les matche à un patron correspondant dans VerbNet. Après avoir un patron correspondant, la phrase est maintenant dotée de toutes les informations lexicales contenues dans les entrées de VerbNet et la création du graphe conceptuelle se fait automatiquement. Ces graphes sont ensuite ajoutés dans une base de données et ils facilient l'information retrieval.

2.1.4.2. *Parsing sémantique*

(Shi et Mihalcea, 2005) Dans ce projet, VerbNet est combiné à FrameNet et WordNet pour faire du semantic parsing. La force de VerbNet dans ce projet est son exhaustivité des différents patrons de régime de l'anglais, et les rôles sémantiques qui sont liés aux verbes. Ils suggèrent que l'utilité de VerbNet provient aussi de sa couverture de l'anglais et des patrons de régime possible est extrêment riche et donc cruciale pour ce genre d'opération. Le but d'un semantic parser est d'analyser le sens de la structure d'une phrase. Donc, via les ressources dont VerbNet, lorsqu'il parse un texte non annoté, le parseur cherche des patrons similaires à ceux se trouvant dans les bases de données lexicales pour ensuite étiquetté les phrases et en faire sortir la structure syntaxique et les caractérisitques sémantiques via les rôles thématiques.

2.1.4.3. *Un système de question-réponse*

(Wen, Jiang et He, 2008) Pour un bon système de question-réponse, la précision de la réponse est cruciale. De parvenir à un haut niveau de précision pour une réponse, ils proposent un système de question-réponse reposant sur VerbNet. Leur système extait les informations syntaxiques et sémantiques des questions puis à partir du web formule des réponses candidates en fonction des informations extraites de la questions , puis VerbNet est appliqué dans leur système pour détecter les cadres syntaxiques des verbes dans les questions

et dans les réponses candidates afin d'obtenir les informations syntaxiques et sémantiques. Puis, leur système choisi la phrase candidate répondant le mieux à la question et tenant compte de la structure sémantique et syntaxique de la question.

2.1.4.4. *A supervised algorithm for verb desambiguasation into VerbNet classes*

(Abend, Reichart et Rappoport, 2008) Ils ont développé un modèle d'apprentissage supervisé pour mapper des lexèmes à des classes de VN. Comme la polysémie est un enjeu majeur en TAL, leur travail consistait à analyser un texte et lorsqu'il trouve un verbe, il doit l'associer à la bonne classe, en fonction des informations lexicales qu'on retrouve dans la classe. Développer ce genre d'outil en TAL est très important car, les machines ne voient les verbes que comme des chaînes de caractères dépourvues de sens. Le contexte d'utilisation d'un verbe nous permet de savoir à quelle classe VN il est présentement utilisé (comme bcp de verbes s'emploient avec des classes différentes selon le contexte). EN moyenne un verbe appartient à 2.5 classes différentes.

2.1.4.5. *VerbNet class assignment as a wsd task*

(Brown, Dligach et Palmer, 2011) Des représentations verbales riches sont cruciales en NLP pour un parsing sémantique profond. Utiliser un lexicon déjà annoté à plusieurs niveaux peut être très utile pour ce genre de tâches. par exemeple la désambiguisation en serait très profitable. Leur classifieur sémantique s'apparente beaucoup à celui de Abbend (Abend *et al.*, 2008).

2.1.5. Pourquoi avoir choisi VerbNet ?

Nous avons choisi VerbNet car cette une ressource qui s'est distinguée de ses concurrentes. D'abord, par son imposante couverture de la langue anglaise (statistiques). Ensuite, par son architecture interne héritée de Levin et améliorée. Le regroupement en classes verbales facilite énormément le travail, et le découpage hiérarchique suivi des mécanismes d'héritage en font un outil très performant et précis, sans être encombrant et facilement repérable. Les verbes sont aussi classés dans plusieurs classes verbales, donc ils sont partiellement désambigguisés, ce qui est extrêmement utile, car on veut être le plus précis possible en NLG. De plus, les descriptions syntaxiques encodées en XML sont facilement exportable et maléable dans un format qui nous convenait, par le fait même, le traitement en Python devenait très accessible puisque NLTK avait déjà fait un pré-traitement de VerbNet, ainsi il existait des modules dont nous pouvions nous inspirer pour extraire l'information dans les balises de VerbNet. Un autre point important est qu'il existe beaucoup de ressources

linguistiques à la VerbNet dans d'autres langues que l'anglais, dont le (mettre les citations de tous les verbnet étrangers) français, le portugais, l'italien, l'estonien, l'espagnol, le catalan, le tchèque. Finalement, puisque notre système GenDR se veut un générateur multilingue dans sa première version, c'est un premier pas pour facilement implémenter d'autres langues à notre système puisque nous avons déjà des scripts une architecture prête pour acceuillir des ressources lexicales similaires.

De plus, dans la section précédente, on montre qu'il est utilisé pour des applications de TAL diverses, mais on a aussi trouvé des systèmes récents qui utilisaient VerbNet pour faire de la NLG. Des chercheurs ont extrait le mapping qui avait été fait entre XTAG-VerbNet afin de donner une couverture imposante de l'anglais pour l'interface sémantique-syntaxe de son générateur. Ils se sont surtout servis de VerbNet pour créer leur grammaire. Leur projet s'appelle S-STRUCT (Pfeil, 2016), un générateur de texte automatique dont on a ajouté un module d'apprentissage machine pour la partie *discourse planning*. Ainsi, en améliorant la partie discourse planning avec du machine learning, le système apprend à générer les phrases dans un ordre statistiquement plus logique.

D'un autre côté, Wanner et Mille ont publié un court article expliquant qu'ils prévoyaient utiliser VerbNet comme base de données lexicales car ils avaient besoin d'une ressource lexicale riche dans le cadre de génération automatique de texte (Mille et Wanner, 2015). De telles ressources sont importantes pour la couverture et qualité des en GAT. Ils mentionnent que des réaliseurs classiques comme KPML, surge, et REALpro nécessitaient un enrichissement lexical pour être encore plus performant. Car, pour arriver à une structure syntaxique de surface, où tous les unités lexicales sont réalisées, il faut qu'un système de GAT possède des ressources lexicales capables de générer tous les actants d'un verbe en fonction des restrictions possibles et des bonnes prépositions, il faut que la génération soit impeccable aussi. Ces informations se retrouvent dans des dictionnaires de cadres de sous-catégorisation tel que ceux mentionnés plus haut. En anglais, VerbNet rempli très bien les caractéristiques recherchées pour des dictionnaires lexicaux car c'est un lexicon à grande surface qui couvre une grande partie de la grammaire anglaise et détaille avec précisions les patrons de régime possibles, ainsi que beaucoup d'autres informations. Toutefois, ils précisent quelque chose qui est très vrai, VerbNet est principalement utilisé pour des tâches comme semantic role labelling, information retrieval. En résumé, les ressources lexicales éxistantes sur le marché sont incomplètes et difficile à implémenter pour la NLG. Ce qui a découlé de ce travail est Forge (Mille *et al.*, 2017), un générateur de texte automatique. Toutefois, ils n'ont pas utilisé VerbNet dans leur système, tel que nous l'avons vu dans la section précédente. Ce qui démontre l'importance de ce travail car nous avons décidé d'utiliser VerbNet en tant que

ressource lexicale pour faire la tâche qu'ils avaient entamés, mais sur laquelle ils n'ont pas publié de résultats.

2.2. Python

À l'aide du module *xml.etree.cElementTree* nous avons pu faire des opérations sur l'ensemble des données de VerbNet qui sont encodées dans des fichiers XML. Le module Etree nous permet de naviguer dans les fichiers XML de VerbNet puis de manipuler et d'extraire les données qui nous intéresse. Après avoir manipulé et extrait les données intéressantes, nous les avons compilées dans des dictionnaires. Ceux-ci seront utilisés par notre système MATE pour générer du texte automatiquement. Par la suite, nous avons aussi utilisé Python et le module Etree pour extraire les phrases exemples qui accompagent chaque patron de régime couvert par la ressource lexicale. Dans le but de créer des structures sémantiques qui nous serviront d'input pour générer en syntaxe de surface la phrase exemple et ainsi vérifier si MATE est capable de générer les phrases exemples.

Nous voulions donc créer des documents DICT qui seraient l'information lexicale de notre système de GAT. Il est d'abord important de dire dans les grandes lignes comment notre système MATE fonctionne. Ainsi, notre système prend en input une représentation sémantique d'un énoncé, puis fait appel à des dictionnaires et des règles de grammaire pour générer en output une représentation syntaxique de surface de l'énoncé. Les unités lexicales s'encodent d'une certaine manière dans MATE et nous voulions créer nos dictionnaires à l'image des contraintes données par le système MATE. Nous avons donc codé la manipualtion et l'extraction des données en fonction de la manière dont le lexique est encodé dans MATE.

2.2.1. Création du dictionnaire de verbes : lexicon.dict

La première étape de notre projet consistait à créer le dictionnaire verbal. Bien que le dictionnaire final comporte 6393 entrées lexicales verbales, cette partie de code nous permettait de créer la base de notre système. Il serait important de décrire dans les grandes lignes à quoi ressemble le dictionnaire pour mieux expliquer pourquoi nous l'avons extrait de cette manière. Ainsi MATE possède la caractéristique d'héritage des traits. Donc, en ce qui concerne les verbes, nous avons architecturé la chose de cette manière. Il existe une classe des verbes qui ont les traits suivants : une dpos=V et une spos=verb.

```
verb {
  dpos = V
  spos = verb
}
```

Ensuite, les 278 classes verbales de VerbNet pointent vers la classe verb, donc les classes héritent de ces deux traits. Ce qui fait en sorte qu'on n'a pas a répété dans chaque entrée de classe verbale les traits dpos et spos.

```
"absorb-39.8": verb {
  gp = { id=NP_V_NP                          dia=12 } // Cotton absorbs water.
  gp = { id=NP_V_NP_PP_from_source           dia=123 } // Cattle take in nutrients from their fe
}
```

Ensuite, chaque membre de chaque classe verbale pointe vers la classe ou la sous-classe qui le représente ce qui fait en sorte qu'il hérite de tous les traits de la classe(sous-classe) vers laquelle il pointe. Par exemple : absorb, ingest, take in vont tous trois hériter des traits de l'entrée "absorb-39.8" qui est la classe verbale. C'est de cette manière que notre dictionnaire passe de 278 entrées lexicales à 6393. Car, les verbes de la langue anglaise ont été classés dans des classes verbales et nous les intégront à notre dictionnaire en gardant la même architecture que VerbNet avait pensé (basé sur les travaux de Levin).

```
absorb: "absorb-39.8"
take_in: "absorb-39.8"
ingest_1: "absorb-39.8"
```

2.2.1.1. *Création du dictionnaire initial contenant les 278 classes verbales comme entrées lexicales*

LISTING 2.6. code pour lexicon.dict

```
# BLOC 1
def supers(t, i):
    ID = t.get('ID')
    sc = {ID:i}
    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    if len(subclasses) > 0:
        for sub in subclasses:
            sc = {**sc, **supers(sub, ID)}
    return sc

# BLOC 2
def treeframes(t):
    ID = t.get('ID')
    z = []
    for frame in t.findall('FRAMES/FRAME'):
        description = re.sub(r"\s*[\s\.\-\ +\\\/\(\)]\s*", '_',
        frame.find('DESCRIPTION').get('primary'))
        if description in exclude:
            continue
        description = re.sub('PP', 'PP_{}', description)
        preps = [p.get('value') or
                p.find('SELRESTRS/SELRESTR').get('type').upper()
                for p in frame.findall('SYNTAX/PREP')+frame.findall('SYNTAX/LEX')]
```

```
        preps = [sorted(p.split()) for p in preps]
        examples = [e.text for e in frame.findall('EXAMPLES/EXAMPLE')]
        if len(preps)==1:
            description = description.format('_'.join(preps[0]))
        elif len(preps)==2:
            description = description.format('_'.join(preps[0]),
                                             '_'.join(preps[1]))
        elif len(preps)==3:
            description = description.format('_'.join(preps[0]),
                                             '_'.join(preps[1]),
                                             '_'.join(preps[2]))
        z.append((description, examples))

    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    subframes = [treeframes(subclass) for subclass in subclasses]
    subframes = sum(subframes, []) # flatten list of lists
    return [(ID, z)] + subframes

# BLOC 3
with open('lexicon.dict','w') as f:
    f.write('lexicon {\n')
    for file in [f for f in os.listdir('verbnet') if f[-4:] == '.xml']:
        root = ET.parse('verbnet/'+file).getroot()
        d = dict(treeframes(root))
        sc = supers(root, 'verb')
        for c in d.keys():
            f.write('"'+c+'"')
            if sc[c] == 'verb':
                f.write(': ' +sc[c] + ' {')
            else:
                f.write(': ' +'"'+sc[c]+'"' + ' {')
            [f.write('\n  gp = { id=' + gp[0] + (max(len(gp[0]), 30)-len(gp[0]))
                     *' ' + ' dia=x } // ' + ' '.join(gp[1])) for gp in d[c]]
            f.write('\n}\n')
    f.write('\n}')
```

Ce script sert à :

On l'a découpé en trois blocs pour faciliter l'explication.

Expliquons d'abord le premier Bloc Expliquos ensuite le deuxième Bloc Expliquons finalement le troisième Bloc

Faire du plus général au plus précis, puis ensuite bloc par bloc -Décrire ce que je veux que la fonction fasse dans les grandes lignes -Expliquer un peu plus concrètement ce que je veux que ça fasse -Expliciter mon code encore plus précisément -Mettre en commentaire la nature des variables

(Peut-être rajouter les numéros de lignes, ce sera utile pour faire référence à des lignes de code) Ce premier script Python se découpe en trois blocs. Les deux premiers blocs sont des fonctions que nous définissons pour ensuite les utiliser dans le BLOC 3 qui nous permettra

d'écrire le dictionnaire initial. Ainsi, pour une meilleure compréhension du script complet, nous décrirons les blocs 1 et 2 pour expliquer ce que les fonctions font, ensuite nous décrirons le bloc 3 car l'usage des fonctions sera maintenant clair.

Commençons par la première fonction supers. La fonction prend 2 arguments : t et i. Si vous regardez le bloc 3, dans la section sc, on voit que supers prend les arguments (root, 'verb'). Ces deux arguments sont les suivants : root donne accès à l'arbre XML et 'verb' est fait pour que chaque classe verbale pointe vers la catégorie verbe. Pour en hériter les traits dpos=V et spos=verb. On va chercher le trait ID dans l'un des deux arguments (l'argument arbre donc montré dans listing le trait ID dans le XML). Ensuite on crée un dictionnaire s'appelant sc dont la clé est le ID et la valeur est i. Puis on instancie une variable subclasses qui contient les racines de chaque sous-classes dans les documents XML (montrer avec listings à quoi ça ressemble dans le document XML). Puis ensuite, pour chaque élément dans sous-classe (si élément il y a) on lui passe la fonction supers et on va chercher son identifiant car en lui repassant la fonction, on retourne a sub.get('ID') donc on va chercher le ID qui se retrouve dans la partie sous-classe de l'arbre de la classe verbale et on met à jour le dictionnaire sc en y ajoutant les identifiants des sous-classes (les clés) qui pointeront vers l'identifiant de la classe qui le gouverne, car le deuxième argument de supers est une chaîne de caractère, dans ce cas, la chaîne ID qu'on avait pu chercher plus tôt dans la fonction. Cela est fait dans le but de construire : sous-classe : classe (donc la sous-classe hérite des traits de sa classe) grâce au mécanisme d'héritage de MATE, et tel que VerbNet le mentionne, chaque sous-classe hérite des traits donnés par la classe qui la domine.

Maintenant, passons à la seconde fonction treeframe. Cette fonction prend un argument. On va aller chercher ID. Puis on initialise une liste z. Pour tous les frames qu'on retrouve dans l'arbre en question, on va chercher la description primaire du frame. On va ensuite se servir des expressions régulières pour uniformiser les descriptions à notre bon vouloir. Parallèlement, nous avons aussi créé une liste s'appelant exclude qui contient toutes les descriptions que nous voulons exclure de notre système et si une description dans un frame se retrouve aussi dans exclude, alors on n'y touche pas et elle ne sera pas modifiée ni ajoutée à la liste description. Ensuite, on modifie encore le nom des descriptions car nous ferons une opération par la suite. Donc, on modifie toutes les descriptions qui contiennent PP pour qu'elles ressemblent à

2.2.1.2. *Extraction des 6393 membres des classes verbales pour enrichir le dictionnaire lexicon.dict*

LISTING 2.7. code pour ajouter des lexèmes à lexicon.dict

```
def treemember(t):
```

```
    ID = t.get('ID')
    members = [m.get('name') for m in t.findall('MEMBERS/MEMBER')]
    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    submembers = []
    if len(subclasses) > 0:
        for sub in subclasses:
            submembers = submembers + treemember(sub)
    return [(ID, members )] + submembers

files = [f for f in os.listdir('verbnet') if f[-4:] == '.xml']

members = dict(sum([treemember(ET.parse('verbnet/'+file).getroot()) for file in files], [])) #
values = sum(list(members.values()), []) # ici c'est uniquement les membres, sans infos sur le
dups = {m:[ID for ID in members.keys() if m in members[ID]] for m in values if values.count(m)
unique_member = {m:ID for ID in members.keys() for m in values if m in members[ID] and values.
lexemes = {d[0]+'_'+str(n+1):d[1][n] for d in dups.items() for n in range(len(d[1]))}

# Ici, je fusionne les dictionnaires ensemble
unified_dict = {**unique_member, **lexemes}

with open('members.dict','w') as f:
    f.write('members {\n')
    for key in sorted(unified_dict.keys()):
        f.write(key)
        f.write(': '),
        f.write('"'+str(unified_dict[key])+'"')
        f.write('\n')
    f.write('\n}\n')
```

2.2.2. Création du gpcon

LISTING 2.8. code pour gpcon.dict

```
def roman(n):
    return ['I', 'II', 'III', 'IV', 'V', 'VI'][n-1]

def gp(name, real_actant):
    s = name + ' {\n'
    i=0
    for actant in real_actant:
        i = i+1
        if type(actant) == list:
            for y in actant:
                s = s + "   " + roman(i) + "={" + y + "}\n"
        else:
            s = s + "   " + roman(i) + "={" + actant + "}\n"
    s = s + '}\n'
    return s


#SUBJECTIVE
subj = 'rel=subjective dpos=N'

#DIRECT OBJECTIVE
```

```
dir_N = 'rel=dir_objective dpos=N'
dir_V_ING = 'rel=dir_objective dpos=V finiteness=GER'
dir_V_INF = 'rel=dir_objective dpos=V finiteness=INF'

#INDIRECT OBJECTIVE
to_N = 'rel=indir_objective dpos=N prep=to'
indir_N = 'rel = indir_objective dpos = N'

#OBLIQUE
on_V = 'rel=oblique dpos=V prep=on'
to_obl_N = 'rel=oblique dpos=N prep=to'
for_obl_N = 'rel=oblique dpos=N prep=for'
as_N =  'rel=oblique dpos=N prep=as'
against_N = 'rel=oblique dpos=N prep=against'
at_N = 'rel=oblique dpos=N prep=at'
...

# LOC

locab = 'rel=oblique dpos=N prep=locab'
locad = 'rel=oblique dpos=N prep=locad'
locin = 'rel=oblique dpos=N prep=locin'

descriptions = {
'NP_agent_V': [subj],
'NP_agent_V_NP': [subj, dir_N],
'NP_asset_V_NP_PP_from_out_of': [subj, dir_N, [from_N, out_of_N]],
'NP_attribute_V': [subj],
'NP_attribute_V_NP_extent': [subj, dir_N],
'NP_attribute_V_PP_by_extent': [subj,by_N],
'NP_cause_V_NP': [subj, dir_N ],
'NP_instrument_V_NP': [subj, dir_N],
'NP_location_V': [subj],
'NP_location_V_NP_theme': [subj,dir_N],
'NP_location_V_PP_with_agent': [subj, with_N],
'NP_location_V_PP_with_theme': [subj, with_N],
'NP_material_V_NP': [subj,dir_N],
'NP_material_V_PP_into_product': [subj, into_N],
...

# CREATION DU GPCON
with open('gpcon.dict','w') as f:
    f.write('gpcon {\n')
    for d in descriptions.keys():
        f.write(gp(d, descriptions[d]))
    f.write('}')
```

Il serait important d'expliquer pourquoi nous n'avons pas repris les rôles sémantiques fournis par VerbNet, car la fonction gp et l'entièreté de ce script repose sur ces fondements de TST. Melcuk explique d'abord pourquoi nous étiquettons nos actants sémantiques de cette manière, puis pourquoi les rôles sémantiques n'ont pas leur place en sémantique.

Soit le mentionner ici, ou ailleurs, mais il a fallu faire un dictionnaire de patron de régime. D'abord, parce qu'on s'est rendu compte que du à toute l'information qu'on allait chercher et la différence dans le type d'information, on a jugé bon de créer un second dictionnaire qui ne contiendrait que les gps, autrement dit un gpcon. Celui l'information sur les patrons de régime (les actants syntaxiques). Il existe x nombre de gps répertoriés. Nous les avons trouvé en faisant un ensemble à partir de tous les descriptions que nous avons obtenus avec le script précédent. Une fois que nous avons l'ensemble des gps différents. Il nous fallait les créer, car tel que mentionné, nous ne pouvions pas extraire les gps de VerbNet dû à une différence trop grande (cadre théorique et application). Notre système de GAT fonctionne avec la théorie Sens-Texte et nous pensons que c'est la théorie qui s'y prête le plus pour faire ce type d'opérations et qui tient le mieux compte de la manière dont le langage fonctionne. Ainsi, nous avons créer le gpcon à partir de Python car un bon nombre d'opérations peuvent être automatisés (éviter les fautes, et c'est plus transparent). Pour la création du gpcon, notre dictionnaire en Python ressemblait à ça. Nos keys() étaient la description du gp et les valeurs étaient les actants syntaxiques impliqués dans ce gp (avec de l'information sur les actants syntaxiques nécessitant une préposition à réaliser). Selon l'ordre dans lequel figure nos objets dans la liste qui est ce qu'on retrouve dans values(), notre fonction va assigner le bon actant syntaxique(I, II, III, etc.) ainsi, cette partie est automatisée grâce à cette fonction. Après, pour l'objet "subj" on va lui assigner une string 'rel=subjective dpos=N' ce qui est encodé dans une autre cellule. Ainsi à chaque fois qu'un gp a un subj, on n'a pas à écrire ce que subj contient. Alors pour l'objet subj, on aura I et 'rel=subjective dpos=N'. C'est l'union de la fonction gp et de la fonction roman qui nous permettent d'assigner les bons actants syntaxiques aux objets dans la liste qui représente les values dans mon dictionnaire de gpcon.

2.2.3. Scripts pour faire les tests

2.2.3.1. *Extraction des exemples*

LISTING 2.9. code pour créer phrases.txt

```
def treeframes(t):
    z = []
    for frame in t.findall('FRAMES/FRAME'):
        description = re.sub(r"\s*[\s\.\-\ +\\\/\(\)]\s*", '_', frame.find('DESCRIPTION').get(
        if description in exclude:
            continue
        examples = [e.text for e in frame.findall('EXAMPLES/EXAMPLE')]
        z =  z + examples
    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    subframes = [treeframes(subclass) for subclass in subclasses]
    subframes = sum(subframes, []) # flatten list of lists
```

```
    return z + subframes

liste=[]
with open('phrases.txt','w') as f:
    for file in [f for f in os.listdir('verbnet') if f[-4:] == '.xml']:
        root = ET.parse('verbnet/'+file).getroot()
        d = (treeframes(root))
        final_liste = liste + d
        [f.write(x+'\n') for x in final_liste]
```

2.2.3.2. *Création des structures qui serviront de tests*

Dans la figure ci-bas, on explique comment on a créer les documents .str qui serviront d'input à notre système MATE qui prend ce genre de document en entrée.

LISTING 2.10. code pour créer des structures .str

```
phrases = open('phrases.txt','r')

with open('structures.str','w') as f:
    for(i,p) in enumerate(phrases):
        with open('s'+str(i)+'.str','w') as g:
            structure = 'structure Sem S'+str(i)+'{\n S {text="'+p.strip()+'"\n\n main-> \n }\
            f.write(structure)
            g.write(structure)
```

Chapitre 3

GENDR

Generic Deep Realizer (Lareau *et al.*, 2018) théorie sens-texte : trouver la meilleure citation pour ça

Le chapitre au complet sera dedié à GenDR, résumé de l'article de françois. Cette partie incluera tous les aspects théoriques que nous devons expliquer. (lexicalisation (collocations, FL), arborisation, patrons de régime,etc.) C'est l'ancienne version de GenDR, expliquée avec les règles et dictionnaires du tronc, et non de la branche. présenter les limites de ce système et pourquoi nous voulions aller chercher l'aide d'une ressource comme VerbNet pour pallier à ce problème.

basé sur la TST (dans une récente étude, elle est mentionnée comme étant importante dans le domaineVicentegeneracionlenguajenatural2015, fait ses preuves avec MARQUIS

les différentes approches, méthodes, la mode présente, nous allons dans une position plus traditionnelle pour les raisons X vue dans le premier chapitre, nous allons utiliser ça : GenDR qui fonctionne avec la plate-forme MATE.

Suivre l'article de françois pour détailler cette partie. Faire des graphiques en PPT pour exemplifier la chose.

GenDR c'est un réalisateur profond : faire un court retour sur ce que c'est.

parler de pourquoi ils utilisent la TST : tous les travaux répertoriés dans Zotero qui parle de TST, pour justifier l'emploi de MTT.

3.1. Architecture de MATE : le transducteur de graphe

3.2. Module interface sémantique-syntaxe dans GenDR

3.3. Les dictionnaires

3.4. Les règles de grammaire

3.4.1. domain-independant rules

3.4.2. domain dependent rule : per language

3.5. Les verbes

mentionner les dictionnaires qui disent que les verbes sont les plus importants Expliquer c'est quoi un patron de régime/cadre de sous-catégorisation, valence, etc. permettent de couvrir large patrons de régime des verbes y sont encodés problématique liée aux verbes

Chapitre 4

IMPLÉMENTATION ET ÉVALUATION DU SYSTÈME

4.1. Implémentation des patrons de régime

Où on les a implémenté dans GenDR Comment on a ajusté les règles à celles qu'on avait montré dans le chapitre GenDR

4.2. Adaptation des anciennes règles pour tenir compte des nouveaux dictionnaires

4.3. Mécanisme dsynt=created->constrained->OK pour générer un énoncé simple (sujet, verbe, objet)

4.3.1. Application de la règle root_standard

Cette règle crée un noeud qui sera la racine de l'arbre syntaxique. Ce noeud se fait imposer des contraintes. Notamment, on demande à ce que ce soit un lexème appartenant à la partie du discours : verbe et que sa finitude soit de type : fini. On impose à ce noeud le trait dsynt=constrained pour que ça s'harmonise avec le règle lex_standard, mais c'est à revoir.

4.3.2. application de la règle lex_standard pour lexicaliser le verbe principal

On assigne un lexème à un noeud créé en syntaxe profonde. Dans le contexte actuel, on l'utilise pour sélectionner l'unité lexicale qui matche les contraintes énoncées sur le noeud vide créé par la règle root_standard. Ce lexème provient du lexicon. Lorsque la règle s'applique et qu'elle consomme le noeud en y mettant la bonne lexicalisation (qui respecte les contraintes

sur le noeud), on ajoute un trait dsynt=OK pour signifier que le sémantème a été réalisé en syntaxe profonde et qu'on ne fasse plus d'opérations sur ce noeud.

4.3.3. Application de la règle actant_gp_selection

Cette règle s'applique lorsque nous avons un prédicat. On crée une variable[?GP] qui nous fournit un chemin vers l'information encodée sous l'attribut *gp* d'un lexème [?X]. Puis , on extirpe les traits *id* et *dia* pour chaque attribut *gp* de notre [?X]. Une fois qu'on a récupéré ces informations, on les appose au lexème en question car on se servira de ces informations pour l'application de règles subséquentes. Le trait *id* représente la description du patron de régime (chaque description se retrouve dans notre gpcon qui est un dictionnaire de *gp*) et le trait dia nous renseigne sur la diathèse de ce patron de régime, c'est-à-dire combien d'actants sont en jeu, et dans quel ordre sont-ils placés ? Il est essentiel qu'un lexème verbal aille chercher ces traits car il en a besoin pour appliquer les règles actancielles qui en découlent. Il faut que le système sache quel patron de régime utilisé pour un prédicat donné, et dans quel ordre les actants seront réalisés en syntaxe.

4.3.4. application des règles actancielles

Une fois qu'un gp est sélectionné, on appliquera la règle actancielle qui lui correspond. Nos règles actancielles ressemblent à : actant_gp_ijk. Les règles prennent en input les arcs sémantiques liant les actants à leur prédicat. Ces règles génèrent des noeuds vides auxquels on appose un trait dsynt=created pour signifier qu'on vient de créer des noeuds vides (on dit qu'ils sont vides car ils n'ont pas encore été consommés par une unité lexicale) en syntaxe profonde. On obtient ainsi des arcs syntaxiques au bout desquels se trouve un noeud vide. Ces règles se font imposer des conditions bien strictes. Il faut d'abord que le prédicat qui les gouverne soit lexicalisé. Ce qui se traduisait par l'ajout d'un trait dsynt=OK au lexème lexicalisé (avec la règle lex_tandard). La règle actant_gp_selection nous permettait de soutirer les traits *id* et *dia* et c'est ici que le trait dia entre en jeu. On s'en sert pour illustrer la diathèse du verbe. [à revoir]

4.3.5. application de la règle constraints_gp

Cette règle applique des contraintes à des noeuds nouvellement créés (autrement dit, des noeuds qui ont le trait dsynt=created). C'est à partir de cette règle que le mécanisme de *dsynt=created-> constrained->OK* prend vie. Il s'agit d'une étape intermédiaire entre la création du noeud avec les règles actancielles et la lexicalisation qui consommera le noeud en octroyant un lexème suivant un trait dsynt=OK (signifiant que la réalisation en syntaxe

profonde est terminée). Ainsi, la règle procède de cette manière. On prend un X (un prédicat) qui lie un Y (un actant) puis ce dernier se fait imposer des contraintes. On cherche à lui imposer ces contraintes car on souhaite qu'il respecte le patron de régime de X. Ainsi, s'il ne convient pas comme actant syntaxique, il ne pourra pas passer à la lexicalisation. Car on souhaite une correspondance entre les traits naturels contenus dans le dictionnaire pour le sémantème en question. Si ses traits ne convienne pas aux contraintes imposées au noeud, provenant du patron de régime de X, alors l'arbre sera incomplet. Lorsqu'on met les contraintes sur le noeud, on lui met aussi le trait dsynt=constrained pour montrer qu'il a été contraint et donc qu'il remplit les critères pour passer à la lexicalisation.

4.3.6. application des règles de lexicalisation

Il y a différentes règles de lexicalisation afin de donner un peu de latitude au système. Ainsi, si quelques informations sont manquantes dans le lexicon ou le semanticon, nous voulons que le système arrive quand même à effectuer une génération de texte. C'est pourquoi il existe la règle standard qui fonctionne lorsque nous avons accès à tous les éléments nécessaires. Les régles de types *guess* opèrent lorsqu'il nous manque certaines infos.

4.3.7. lex_standard

Nous avons vu cette règle un peu plus haut, car il fallait l'appliquer dès le début pour lexicaliser le noyau principal afin de chercher le bon lexème qui pouvait remplir cette fonction. Une fois que nous l'avons lexicalisé, nous avons pu aller chercher les informations sur la nature de son gp etc. Nous sommes maintenant rendus au moment où il existe des arcs syntaxiques au bout desquels nous avons des noeuds contraints par des traits comme : la dpos, la finitude, la définitude, etc. Nous allons donc lexicaliser ces noeuds afin de poursuivre la construction de l'arbre, du haut vers le bas. En gros, comment ça fonctionne. On cherche la lexicalisation d'un sémantème donné dans le semanticon, en allant chercher le trait *lex* du sémantème. Puis, on colle cette lexicalisation a un noeud déjà existant. Ce noeud existe déjà car il a soit été créé par root_standard ou par une des règles actancielles. On va octroyer au noeud 3 traits : un trait dlex, un trait dpos, et finalement un trait dsynt. Le trait dlex est la lexicalisation profonde, celle qu'on retrouve dans le lexicon, le trait dpos est la partie du discours profonde qui doit correspondre à la dpos demandée par constraints_gp. Et finalement, un trait dsynt=OK qui s'ajoute à la lexicalisation du sémantème pour signifier au système que le noeud a été consommé et qu'il a été réalisé en syntaxe profonde. Donc, qu'on ne fasse plus d'opération sur ce noeud. Finalement, on peut uniquement faire des opérations sur des noeuds qui ont let trait dsynt=constrained afin de lexicaliser seulement

les noeuds qui se sont fait attribués des contraintes. Ainsi, s'ils respectent les contraintes, ils pourront être lexicalisé, sinon, l'arbre sera incomplet et la génération échouera.

4.3.8. Avantages d'utiliser ces mécanismes

"Dsynt=OK" indique que le noeud a été consommé. Autrement dit, il existe maintenant une unité lexicale réalisé en syntaxe profonde, là où il y avait un noeud vide. "Dsynt=constrained" indique qu'un noeud vide s'est fait imposé des contraintes. Ces contraintes peuvent être de plusieurs ordres. Notamment, on impose une dpos, une finitude, un mood, etc. Il s'agit du passsage intermédiaire entre la création d'un noeud et sa lexicalisation. On veut s'assurer qu'il respecte certaines contraintes pour ne pas lexicaliser n'importe quoi lors de la génération de notre arbre syntaxique. Finalement, le trait dsynt=created

4.4. Méthodes d'évaluation en TAL

4.5. Analyse des données

4.6. Méthodes d'évaluation en NLG

notre système ne passera pas par la phase de morpho et linéarisation. Mais,avec la structure de surface, nous pouvons déjà savoir si l'arbre en output est grammaticalement correcte selon les arbres de dépendance. Les outils de linéarisation ne font que prendre les arbres et linéariser en fonction des dépendances.

voir : The first Surface Realisation Shared task

l'évaluation de notre système sera principalement humaine, mais notre manière de voir si c'était bon sera de comparer les structures de surface avec les phrases données en input provenant de VerbNet et voir si ça a de l'allure

évaluation manuelle : avec les jpg, la version textuelle des graphes, et les phrases exemples

notre évaluation concernera : les limites de GenDR et les limites de VerbNet, une critique théorique de ce qui font, en testant une application pratique de leur lexicon. Et fournir une ligne directrice aux futurs lexicons, ou bien aux genres de lexicon qu'il nous faut. Dans le fond, une critique de Levin aussi, qui ne traite pas des fonctions lexicales, car c'est une analyse syntaxique qui sous-tend des principes sémantiques. Et la partie sémantique de VerbNet est plus axé sur les rôles sémantique et sur les prédicats sémantiques, mais qui clash avec notre théorie qui rend mieux compte de l'interface sémantique/syntaxe.

dire que c'est normal que en NLG, les méthodes d'évaluation sont complexes. Surtout que notre truc est pas linéarisé, pourquoi on peut pas utiliser de F-mesure, etc.

dans le chapitre 5 de verbnet, ils font la propre évaluation de leur système. Dans notre mémoire, nous ferons aussi une évaluation de leur système, sur des bases théoriques et appliquées.

Bibliographie

Abend, O., R. Reichart et A. Rappoport. 2008, «A Supervised Algorithm for Verb Disambiguation into VerbNet Classes», dans *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, Association for Computational Linguistics, Stroudsburg, PA, USA, ISBN 978-1-905593-44-6, p. 9–16.

Ayan, N. F. et B. J. Dorr. 2002, «Generating A Parsing Lexicon from an LCS-Based Lexicon», dans *In : Proceedings of the LREC-2002 Workshop on Linguistic Knowledge Acquisition and Representation, Las Palmas, Canary Islands*, p. 4352.

Baker, C. F., C. J. Fillmore et J. B. Lowe. 1998, «The Berkeley FrameNet Project», dans *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1*, COLING '98, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 86–90, doi : 10.3115/980451.980860.

Bateman, J. A. 1997, «Enabling Technology for Multilingual Natural Language Generation : The KPML Development Environment», *Nat. Lang. Eng.*, vol. 3, n° 1, doi :10.1017/S1351324997001514, p. 15–55, ISSN 1351-3249.

Briscoe, T., J. Carroll et R. Watson. 2006, «The Second Release of the RASP System», dans *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, COLING-ACL '06, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 77–80, doi :10.3115/1225403.1225423.

Brown, S. W., D. Dligach et M. Palmer. 2011, «VerbNet Class Assignment As a WSD Task», dans *Proceedings of the Ninth International Conference on Computational Semantics*, IWCS '11, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 85–94.

Cédric Messiant, T. P. et A. Korhonen. 2008, «LexSchem : A Large Subcategorization Lexicon for French Verbs», dans *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, édité par Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Daniel Tapias, European Language Resources Association (ELRA), Marrakech, Morocco, ISBN 2-9517408-4-0. Http ://www.lrec-conf.org/proceedings/lrec2008/.

Copestake, A. 1992, «The ACQUILEX LKB : Representation issues in semi-automatic acquisition of large lexicons», dans *Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP-92*, p. 88–96.

Danlos, L. 1983, «Présentation d'un modèle de génération automatique», *Revue québécoise de linguistique*, vol. 13, n° 1, doi :10.7202/602510ar, p. 203–228, ISSN 0710-0167, 1705-4591.

Daoust, N. et G. Lapalme. 2015, «JSREAL : A Text Realizer for Web Programming», dans *Language Production, Cognition, and the Lexicon*, vol. 48, édité par N. Gala, R. Rapp et G. Bel-Enguix, Springer International Publishing, Cham, ISBN 978-3-319-08042-0 978-3-319-08043-7, p. 361–376, doi :10.1007/978-3-319-08043-7_21.

Département de linguistique. 2017, «Dicovalence», ORTOLANG (Open Resources and TOols for LANGuage) –www.ortolang.fr.

Doran, C., D. Egedi, B. A. Hockey, B. Srinivas et M. Zaidel. 1994, «XTAG System : A Wide Coverage Grammar for English», dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 2*, COLING '94, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 922–928, doi :10.3115/991250.991297.

Dorr, B. J. 1992, «The Use of Lexical Semantics in Interlingual Machine Translation», *Machine Translation*, vol. 7, n° 3, p. 135–193, ISSN 0922-6567.

Elhadad, M. et J. Robin. 1998, «SURGE : A Comprehensive Plug-in Syntactic Realization Component for Text Generation», cahier de recherche.

Essers, V. et R. Dale. 1998, «Choosing a Surface Realiser», dans *In Proceedings of PRICAI98*, Singapore.

Fellbaum, C., éd.. 1998, «WordNet : An Electronic Lexical Database», Language, Speech, and Communication, MIT Press, Cambridge, MA, ISBN 978-0-262-06197-1.

Fillmore, C. J. 1968, «The Case for Case», dans *Universals in Linguistic Theory*, édité par E. Bach et R. T. Harms, Holt, Rinehart and Winston, New York, p. 0–88.

Gatt, A. et E. Krahmer. 2018, «Survey of the State of the Art in Natural Language Generation : Core tasks, applications and evaluation», *Journal of Artificial Intelligence Research*, vol. 61, p. 65–170.

Gatt, A. et E. Reiter. 2009, «SimpleNLG : A Realisation Engine for Practical Applications», dans *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 90–93.

Grishman, R., C. Macleod et A. Meyers. 1994, «Comlex Syntax : Building a Computational Lexicon», dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, Association for Computational Linguistics, Kyoto, Japan, p. 268–272, doi :10.3115/991886.991931.

Hensman, S. et J. Dunnion. 2004, «Automatically Building Conceptual Graphs Using VerbNet and WordNet», dans *Proceedings of the 2004 International Symposium on Information and Communication Technologies*, ISICT '04, Trinity College Dublin, Las Vegas, Nevada, USA, ISBN 978-1-59593-170-2, p. 115–120.

Herbst, T., D. Heath, I. F. Roe et D. Götz. 2004, «A Valency Dictionary of English, A Corpus-Based Analysis of the Complementation Patterns of English Verbs, Nouns and Adjectives», De Gruyter Mouton, Berlin, Boston, ISBN 978-3-11-089258-1, doi :10.1515/9783110892581.

Herbst, T. et P. Uhrig. 2009, «Erlangen Valency Patternbank. A corpus-based research tool for work on valency and argument structure constructions.», .

Jackendoff, R. 1972, «Semantic Interpretation in Generative Grammar», Cambridge : Mass., Mit Press.

Kipper, K., H. T. Dang et M. Palmer. 2000, «Class-Based Construction of a Verb Lexicon», dans *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, AAAI Press, ISBN 978-0-262-51112-4, p. 691–696.

Kipper, K., A. Korhonen, N. Ryant et M. Palmer. 2006, «Extending VerbNet with Novel Verb Classes», dans *Proceedings of the Fifth International Conference on Language Resources and Evaluation – LREC'06 (http ://verbs.colorado.edu/ Mpalmer/Projects/Verbnet.Html)*.

Korhonen, A., Y. Krymolowski et T. Briscoe. 2006, «A large subcategorization lexicon for natural language processing applications», dans *In Proceedings of LREC*.

Lambrey, F. 2017, «Implémentation des collocations pour la réalisation de texte multilingue», thèse de doctorat, Université de Montréal.

Lareau, F., F. Lambrey, I. Dubinskaite, D. Galarreta-Piquette et M. Nejat. 2018, «GenDR : A Generic Deep Realizer with Complex Lexicalization», dans *Proceedings of 11th Edition of the Language Resources and Evaluation Conference (LREC)*, Miyazaki.

Lavoie, B. et O. Rambow. 1997, «A Fast and Portable Realizer for Text Generation Systems», dans *Proceedings of the Fifth Conference on Applied Natural Language Processing*, ANLC '97, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 265–268, doi :10.3115/974557.974596.

Levin, B. 1993, «English verb classes and alternations : A preliminary investigation», .

Mille, S., R. Carlini, A. Burga et L. Wanner. 2017, «FORGe at SemEval-2017 Task 9 : Deep sentence generation based on a sequence of graph transducers», dans *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017*, p. 920–923, doi :10.18653/v1/S17-2158.

Mille, S. et L. Wanner. 2015, «Towards Large Coverage Detailed Lexical Resources for Data-to-Text Generation.», dans *Proceedings of the First International Workshop on D2T Generation*, Edinburgh, Scotland.

Molins, P. et G. Lapalme. 2015, «JSrealB : A Bilingual Text Realizer for Web Programming», Association for Computational Linguistics, p. 109–111, doi :10.18653/v1/W15-4719.

Palmer, M., D. Gildea et P. Kingsbury. 2005, «The Proposition Bank : An Annotated Corpus of Semantic Roles», *Comput. Linguist.*, vol. 31, n° 1, doi :10.1162/0891201053630264, p. 71–106, ISSN 0891-2017.

Pfeil, J. W. 2016, «Algorithms and Resources for Scalable Natural Language Generation», thèse de doctorat, Case Western Reserve University.

Polguère, A. «Pour un modèle stratifié de la lexicalisation en génération de texte», *Traitement Automatique des Langues (TAL)*.

Reiter, E. et R. Dale. 2000, «Building Natural Language Generation Systems», Cambridge University Press, New York, NY, USA, ISBN 978-0-521-62036-9.

Research Group, T. X. 2001, «A Lexicalized Tree Adjoining Grammar for English», *IRCS Technical Reports Series*.

Ryant, N. et K. Kipper. 2004, «Assigning XTAG Trees to VerbNet», dans *Proceedings of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms*, Vancouver, Canada, p. 194–198.

Schuler, K. K. 2005, «Verbnet : A Broad-coverage, Comprehensive Verb Lexicon», PhD Thesis, University of Pennsylvania, Philadelphia, PA, USA.

Shi, L. et R. Mihalcea. 2005, «Putting Pieces Together : Combining FrameNet, VerbNet and WordNet for Robust Semantic Parsing», dans *Proceedings of the 6th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'05, Springer-Verlag, Mexico City, Mexico, ISBN 3-540-24523-5 978-3-540-24523-0, p. 100–111, doi :10.1007/978-3-540-30586-6_9.

Traum, D. et N. Habash. 2000, «Generation from Lexical Conceptual Structures», dans *Proceedings of the 2000 NAACL-ANLP Workshop on Applied Interlinguas : Practical Applications of Interlingual Approaches to NLP - Volume 2*, NAACL-ANLP-Interlinguas '00, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 52–59, doi :10.3115/1117554.1117561.

Wanner, L., B. Bohnet, N. Bouayad-Agha, F. Lareau et D. Nicklass. 2010, «MARQUIS : GENERATION OF USER-TAILORED MULTILINGUAL AIR QUALITY BULLETINS», *Appl. Artif. Intell.*, vol. 24, n° 10, doi :10.1080/08839514.2010.529258, p. 914–952, ISSN 0883-9514.

Wen, D., S. Jiang et Y. He. 2008, «A question answering system based on VerbNet frames», dans *Proceedings of the 4th International Conference on Natural Language Processing and Knowledge Engineering, NLPKE 2008, Beijing, China, October 19-22, 2008*, p. 1–8, doi :10.1109/NLPKE.2008.4906769.

Annexe A

LE TITRE

A.1. Section un de l'annexe A

La première annexe du document.

Pour plus de renseignements, consultez le site web de la FESP. Pour plus de renseignements, consultez le site web de la FESP.

Tableau A. I. Liste des parties

Les couvertures conformes	obligatoires
Les pages de garde	obligatoires
La page de titre	obligatoire
Le résumé en français et les mots clés français	obligatoires
Le résumé en anglais et les mots clés anglais	obligatoires
Le résumé de vulgarisation	facultatif
La table des matières, la liste des tableaux, la liste des figures	obligatoires
La liste des sigles, la liste des abréviations	obligatoires
La dédicace	facultative
Les remerciements	facultatifs
L'avant-propos	facultatif
Le corps de l'ouvrage	obligatoire
L'index analytique	facultatif
Les sources documentaires	obligatoires
Les appendices (annexes)	facultatifs
Le curriculum vitæ	facultatif
Les documents spéciaux	facultatifs

Annexe B

LE TITRE2

texte annexe B