Développement d'un module de génération de paraphrases pour la data augmentation

Sonia Ratsiandavana

Développement d'un module de génération de paraphrases pour la data augmentation

RATSIANDAVANA
Sonia

Sous la direction de Zied SELLAMI et Olivier KRAIF

UFR LLASIC
Département Sciences du Langage et Français Langue Etrangère (FLE)

Mémoire de master 2 professionnel - 20 crédits

Mention : Sciences du Langage - Spécialité : Industries De la Langue

Année universitaire 2018-2019

Développement d'un module de génération de paraphrases pour la data augmentation

RATSIANDAVANA
Sonia

Sous la direction de Zied SELLAMI et Olivier KRAIF

UFR LLASIC
Département Sciences du Langage et Français Langue Etrangère (FLE)

Mémoire de master 2 professionnel - 20 crédits
Mention : Sciences du Langage - Spécialité : Industries De la Langue

Année universitaire 2018-2019

Remerciements

Je tiens à remercier l'ensemble du corps enseignant du master IdL pour nous avoir transmis leur savoir. Je remercie particulièrement Olivier Kraif pour m'avoir encadrée durant ce stage, Claude Ponton et Georges Antoniadis pour avoir accepté d'être membres de mon jury.

Un grand merci à Zied Sellami pour son aide, sa disponibilité et son accompagnement durant ces 6 mois de projet. Merci à toute l'équipe de Linagora pour leur accueil et leur bonne humeur, en particulier Julie Hunter pour nos discussions toujours enrichissantes sur la linguistique.

Je souhaiterais aussi remercier mes camarades de promotion pour ces deux années riches en émotions.

Merci à mes parents et à ma sœur sans qui rien de tout cela n'aurait été possible.

UNIVERSITÉ Grenoble Alpes

DÉCLARATION

1. Ce travail est le fruit d'un travail personnel et constitue un document original.
2. Je sais que prétendre être l'auteur d'un travail écrit par une autre personne est une pratique sévèrement sanctionnée par la loi.
3. Personne d'autre que moi n'a le droit de faire valoir ce travail, en totalité ou en partie, comme le sien.
4. Les propos repris mot à mot à d'autres auteurs figurent entre guillemets (citations).
5. Les écrits sur lesquels je m'appuie dans ce mémoire sont systématiquement référencés selon un système de renvoi bibliographique clair et précis.

NOM : ..... RATSIANDAVANA .....

PRENOM : ..... Sonia .....

DATE : ..... 28/08/2019 .....

Sommaire

Introduction

Ce mémoire présente les résultats de 6 mois de travail au sein de l'entreprise LINAGORA. Durant ce stage, j'ai pu intégrer l'équipe Recherche et Développement de la branche toulousaine.

1 - L'entreprise

LINAGORA est une entreprise française spécialisée dans l'édition de logiciels libres. Elle est présente en France, au Vietnam, au Québec et en Tunisie.

Ses principaux axes de recherche sont la reconnaissance et la compréhension de la parole, le text mining, les architectures middlware distribuées, le Cloud Computing, l'ingénierie collaborative, la sécurité et les architectures Big-Data. Elle propose également des services de support pour les logiciels open-source.

Un de ses projets en cours est le développement d'un assistant vocal professionnel : LinTO. C'est dans le cadre de ce projet que j'ai pu travailler sur des problématiques liées au Traitement Automatique des Langues.

Figure 1: Logo de LinTO

2 - Le projet LinTO

LinTO est un assistant vocal open-source, capable de répondre aux commandes énoncées par les utilisateurs. Il est adapté au milieu professionnel et permet d'assister les employés lors des réunions en entreprise. Il possède des fonctionnalités permettant de piloter la salle de réunion, d'interroger un système d'informations, d'aider à animer la réunion, et propose également la rédaction de compte-rendu.

3 - Le stage

LinTO possède un moteur de reconnaissance des commandes : mon stage avait pour objectif le développement d'un module permettant d'étendre son corpus d'apprentissage de manière automatique. Ce corpus est composé de commandes transcrites (exemple : « quelle est la météo à Toulouse »). Nous le présenterons dans la première partie de ce mémoire.

Le module que nous allons développer permettra de prendre en entrée une commande textuelle et de produire en sortie des variantes de cette commande. Ces variantes ne seront pas des paraphrases strictes mais de nouvelles commandes pouvant être des équivalents fonctionnels. Elles pourront servir dans un contexte similaire. Dans l'exemple suivant l'objectif de l'utilisateur ne change pas, il souhaite connaître la météo :

« quelle est la météo à Toulouse » → « c'est quoi la température à Marseille »

Ce module s'appuiera sur une approche symbolique : définir une grammaire permettant de construire des phrases grammaticalement et sémantiquement correctes.

Dans une première partie, nous présenterons l'étape d'analyse de notre problématique : l'étude des travaux existants dans le domaine et l'étude du corpus de commandes mis à notre disposition. La deuxième partie sera consacrée à la description des différentes étapes de développement de notre module.

Partie 1

-

Construction du cahier des charges

1 - Approches existantes

1.1 - Data augmentation et paraphrase

Avant toute chose, il convient de définir les notions que nous allons aborder dans ce mémoire.

La « data augmentation » est une méthode permettant d'enrichir un corpus de données d'entraînement en créant de nouvelles données à partir des données déjà existantes.

Pour notre projet, il s'agit de produire de nouvelles commandes à partir d'un ensemble de commandes déjà existantes. Le procédé linguistique qui se rapproche le plus de ce que nous souhaitons faire est la paraphrase. Dans notre cas, nous pourrions plutôt parler de « quasi-paraphrase », puisque comme il a été dit précédemment, nous devons générer des équivalents fonctionnels : lors du processus, certains mots-clés sont susceptibles d'aller au-delà de la synonymie (exemple du cas où « Toulouse » est remplacée par « Marseille », « jour » par « mois ») et d'englober des catégories plus larges. Nous expliquerons la raison de ce phénomène dans la sous-partie 2.2 concernant l'annotation des entités dans l'outil Tock.

En Traitement Automatique des Langues, on définit la paraphrase comme étant « un *moyen alternatif* exprimant, dans une même langue, le *même contenu sémantique*, la *même information* ou la *même idée* que la forme originale » (Barzilay et McKeown, 2001 ; Fujita, 2005 ; Callison-Burch, 2007 ; Bhagat, 2009 ; Zhao et coll., 2009 ; Madnani, 2010, cités par Houda Bouanor, 2012).

Pour traiter cette problématique, il existe plusieurs approches : les approches par apprentissage et les approches symboliques.

1.2 - Approches par apprentissage

Les approches par apprentissages existantes utilisent les réseaux de neurones Seq2seq. Il s'agit d'un modèle qui prend une entrée d'une certaine longueur et produit une sortie pouvant être d'une longueur différente. On retrouve les modèles Seq2Seq dans la traduction automatique ou le résumé de texte par exemple.

Li et al. (2016) ont développé un générateur basé sur le modèle Seq2Seq permettant de produire la paraphrase d'une entrée donnée. Il y ont ajouté un évaluateur dont le rôle est

de vérifier si les deux phrases sont sémantiquement similaires. Le générateur produit la nouvelle phrase en prédisant chaque mot grâce à son contexte et au mot qui le précède. L'évaluateur vérifie sa sortie et le récompense pour améliorer l'apprentissage. Les mots utilisés dans la nouvelle phrase sont soit copiés dans la phrase d'entrée, soit générés à partir d'un lexique.

Jia et al. (2016) utilisent également les modèles Seq2Seq pour l'analyse sémantique, en y intégrant la *data recombination*. Elle consiste à traiter préalablement les phrases d'entrée avec une grammaire hors contexte afin de récupérer les instances d'entités qui s'y trouvent et de les réutiliser pour créer de nouvelles phrases. Cela permet d'amener de la variété aux données déjà existantes.

Pour notre projet, nous n'utiliserons pas ce type d'approches : ces méthodes nécessitent des volumes de données trop conséquentes. De plus, elles ne permettent pas de contrôler finement les données, ce qui rend le risque de sur-génération plus important.

1.3 - Approches symboliques

Parmi les approches symboliques qui existent dans le domaine, on retrouve les grammaires (Narayan et al., 2016). Des grammaires hors-contexte sont utilisées pour générer des variantes à partir d'une question donnée en entrée.

On crée une grammaire hors-contexte probabiliste pour déterminer la probabilité d'apparition des syntagmes d'une phrase. L'apprentissage se fait sur un corpus large de questions.

Un réseau sémantique est construit à partir des éléments de la phrase en entrée : il permet de restreindre les variantes de mots et phrases à utiliser dans les paraphrases générées. Sur la figure 2, nous avons un aperçu du processus suivi par la grammaire pour générer de nouvelles phrases.

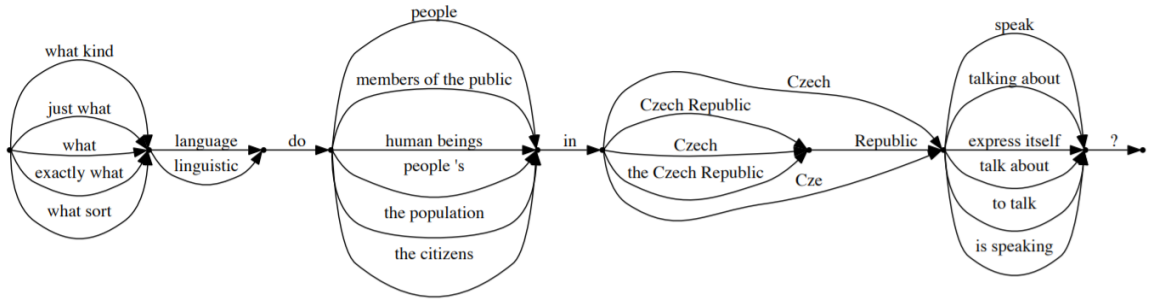

Figure 2: Exemple de grammaire de génération pour la question "What language do people in Czech Republic speak ?" (Narayan et al., 2016)

Pour le français, Kampeera (2013) propose une analyse linguistique pour le traitement automatique de la paraphrase. Il propose une formalisation basée sur les structures prédicat-argument : les prédicats sont les sens les plus importants dans les phrases source et dérivée, et les arguments, les autres unités de sens gravitant autour de ces prédicats. Cette formalisation principalement théorique est mise en pratique sur un corpus du domaine agroalimentaire.

2 - Les ressources existantes

2.1 - Présentation du corpus de commandes initial

Comme point de départ de ce travail, nous avions à notre disposition un fichier contenant plusieurs exemples de commandes destinées à l'apprentissage de notre assistant vocal. Ce corpus a été créé manuellement : il contient 676 commandes. Ces commandes sont unitaires, c'est-à-dire qu'elles ne demandent l'exécution que d'une action à la fois. Elles sont également classées par intentions[1], ce qui permet de formuler une même requête de plusieurs façons différentes. Au sein de notre corpus actuel, nous avons 22 intentions.

La figure 3 montre des exemples de commandes appartenant à l'intention *weather*. Chaque commande a été annotée pour dégager les entités[2] qui la compose. Ces phrases sont annotées au format markdown[3], ce qui permet d'identifier le mot entre crochets comme étant l'instance de l'entité entre parenthèses. Ces commandes ont été annotées manuellement et associées à leur intention grâce à l'outil TOCK que nous présenterons dans la partie 2.2 suivante.

[1] L'intention est l'objectif exprimé dans la phrase.
[2] Les entités sont des catégories regroupant les éléments importants d'un texte. Les éléments appartenant à ces catégories sont des instances. (Ex : l'entité « location » a pour instances « Toronto », « Paris » ect...
[3] Le markdown est un langage de balisage.

```
## intent:weather
- météo à [Toulouse](location)
- est-ce qu'il va neiger [demain](time)
- est-ce qu'il va pleuvoir [demain](time)
- quel temps fait-il à [Toronto](location)
- quel temps fait-il à [Hikone](location)
- quel temps fait-il à [Himeji](location)
- quel temps fait-il à [Totori](location)
- quel temps fait-il à [Hakone](location)
- quel temps fait-il à [Toronto](location) [demain](time)
- quel temps fait-il à [Hikone](location) [demain](time)
- quel temps fait-il à [Himeji](location) [demain](time)
- quel temps fait-il à [Totori](location) [demain](time)
- quel temps fait-il à [Hakone](location) [demain](time)
```

Figure 3: Exemple d'intention accompagnée de ses commandes

Notre objectif était de produire des variantes de chacune des commandes présentes dans le corpus.

2.2 - Présentation de Tock

Afin de construire le moteur de reconnaissances commandes de LinTO, nous nous appuyons sur Tock, une plateforme open-source permettant de construire des agents conversationnels. Tock permet de créer un modèle de reconnaissances des intentions. Il utilise les libraires pour le TAL Stanford CoreNLP et Apache OpenNLP.

Comme le montre la figure 4, il est possible d'annoter manuellement plusieurs mots ou expressions d'une phrase. Cela permet d'associer la présence de plusieurs entités à une intention spécifique.

Il faut d'abord définir l'annotation des entités manuellement sur plusieurs phrases, puis Tock l'effectue automatiquement sur les phrases suivantes.

Notre corpus est donc un échantillon de commandes annotées grâce à cet outil. Il est important de souligner que les instances des entités qui ont été identifiées dans ce corpus ne sont pas toujours des syonymes : il peut aussi s'agir de cohyponymes[4] (« Toulouse », « Paris »). En revanche, on ne retrouve pas d'antonymes[5] au sein des instances d'une même entité. Les antonymes se trouvent dans des entités différentes : nous avons par exemple l'entité « action_on » (lancer, démarrer, commencer) et l'entité

[4] Cohyponymes : mots ayant le même hyperonyme
[5] Antonymes : mots dont les sens s'opposent

« action_off » (arrêter, annuler, terminer) dont les instances sont antonymes. Nous verrons par la suite que pour générer d'autres variantes (en dehors des termes déjà existants dans notre corpus de référence), nous garderons aussi les mots voisins issus de plongements de mots, les hyponymes[6] ou hyperonymes[7]. Cela permettra de générer des commandes variées tout en restant dans l'intention générale de la phrase.

Figure 4: Exemple d'annotation des entités dans Tock

[6] Hyponyme : mot dont le sens est inclus dans celui d'un autre mot, plus général (*limousine* est l'hyponyme de *voiture*)
[7] Hyperonyme : mot dont le sens englobe celui d'autres mots (*voiture* est l'hyperonyme de *limousine*)

3 - Étude du sujet

3.1 - Étude du corpus de commandes initial

3.1.1 - Catégories principales : action / information

Avant le développement il était nécessaire d'étudier le corpus mis à notre disposition afin de déterminer le type de phrases que nous aurions en entrée de notre module, mais aussi le type de phrases que nous souhaitions générer en sortie.

Pour cela, nous avons étudié les phrases du corpus selon deux points de vue : syntaxique et sémantique.

Au niveau sémantique, outre les catégories liées aux intentions, on peut dégager deux types de commandes :

- Les commandes pour lesquelles l'utilisateur souhaite demander une information. On trouve les exemples suivants : « quelle est la météo ? », « quel est l'ordre du jour ? » ou simplement « actualités musicales ». Nous les avons appelées « commandes d'information ».

- Les commandes visant à effectuer une action, telles que « lance la visioconférence » ou « peux-tu démarrer le chronomètre ». Nous les avons appelées « commandes d'action ».

Selon Kampeera, on peut partir du postulat que deux paraphrases partagent des unités de sens similaires : l'une des phrases est la phrase source et l'autre est la phrase dérivée. Une unité de sens dans la phrase source est plus importante que les autres car c'est à partir de celle-ci que l'on peut démarrer la construction des paraphrases : c'est le « sens source ». Il appelle « sens dérivé » le sens issu de ce sens source et recréé dans la phrase dérivée. Ces deux sens sont appelés « prédicats ». Ici, le prédicat est pris au sens de « mot sémantiquement plein », il peut donc s'agir de n'importe quelle partie du discours (verbe, nom, adjectif ect…). Ces prédicats contrôlent d'autres unités de sens participant à la construction du sens de la phrase : les « arguments ».

Pour notre corpus, nous ne nous baserons pas sur les relations prédicat-arguments car la structure syntaxique de nos commandes n'est pas adaptée. En revanche, nous pouvons retenir le fait qu'un élément de la phrase a le rôle de sens principal.

Nous pouvons identifier les deux types de commandes (action et information) grâce à un élément essentiel qui se rapproche du prédicat que nous venons de décrire.

Lorsqu'une commande demande l'exécution d'une action, le verbe caractérisant l'action à effectuer est annoté avec l'entité « action », suivi d'une information sur l'action à effectuer (« start », « stop »...). C'est ce verbe annoté qui est notre élément essentiel. Il permet d'identifier une commande d'action quelque soit le type de phrase (impérative, interrogative…) et sa structure syntaxique. Dans ce type de commande, le verbe est accompagné d'un syntagme nominal qui peut être considéré comme argument. Sur la figure 5, nous avons par exemple « désactive le minuteur » : ici, le prédicat est « désactive » et son argument « minuteur ».

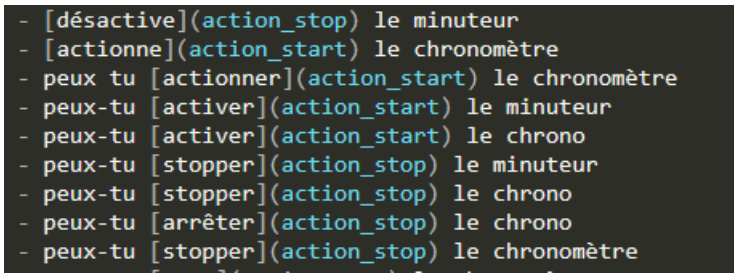

Figure 5: Exemples de commandes d'action dont les entités "action" sont annotées

Les commandes d'information n'ont pas de verbes « action » annotés et représentent le reste des commandes du corpus. Nous avons plusieurs manières de les repérer.

Dans l'exemple de la figure 6, la formulation de la phrase peut différer : on retrouve des interrogatives, des impératives et parfois des phrases nominales plus spécifiques au langage oral (« météo à Tunis »).

Dans le cas des commandes d'information, nous identifions le syntagme nominal principal comme l'élément essentiel de la phrase. Dans les exemples « quelles sont les prévisions météo à Rome demain » et « météo à Marseille » les éléments « prévision météo » et « météo » portent le sens principal de la phrase. Les autres éléments comme les syntagmes prépositionnels (« à Rome ») sont des compléments d'informations qu'il nous faudra aussi extraire.

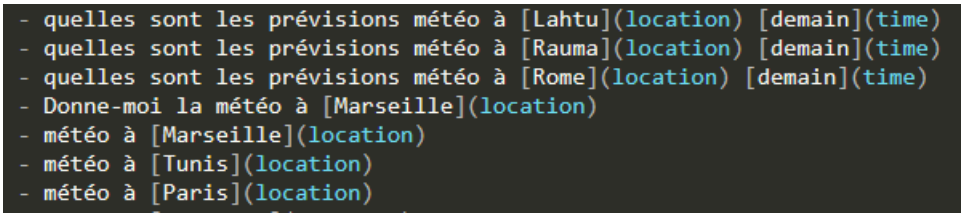

Figure 6: Exemples de commandes d'information

Il était important de distinguer les commandes d'action et d'information. Lors de la génération, nous ne pouvons pas les paraphraser en utilisant les mêmes combinaisons syntaxiques.

Ainsi, si nous prenons les deux commandes d'information suivantes :

- « Donne-moi la météo »

- « Quelle sont les actualités »

Nous pouvons récupérer les noms « météo » et « actualités », qui sont porteurs de l'information principale et leur appliquer les mêmes combinaisons syntaxiques :

- « Quelle est la météo », « je veux la météo », « peux-tu me donner la météo »

- « Quelles sont les actualités », « je veux les actualités », « peux-tu me donner les actualités »

En revanche, si l'on prend deux commandes d'action :

- « Lance l'enregistrement de la réunion »

- « Peux-tu arrêter le chronomètre »

Dans ce cas, les combinaisons syntaxiques vues précédemment ne sont pas applicables. Si on récupérait le syntagme nominal comme information principale, la phrase n'aurait plus le même sens : « Quelle est l'enregistrement de la réunion », « je veux le chronomètre ».

Nous remarquons que la phrase « donne-moi la météo » est considérée comme étant une commande d'information alors qu'elle comporte un verbe à l'impératif. Cela est dû au fait que sémantiquement, l'information qui doit être mise en avant est « météo ». De plus, l'annotateur ne l'a pas identifié comme instance d'une entité « action ».

3.1.2 - Sous-catégories de commandes information

Nous avons fait la distinction entre les commandes d'action et les commandes d'information, mais cela n'est pas suffisant pour savoir quelles combinaisons syntaxiques appliquer aux phrases.

En effet, dans le cas des commandes d'action, le verbe comme sens principal limite les combinaisons possibles. Nous n'avons identifié que deux possibilités :

- Forme impérative → une phrase débutant par le verbe conjugué à l'impératif : « allume la lumière »

- Forme interrogative → une phrase débutant par certains mots/expressions interrogatifs :

« peux-tu allumer la lumière », « est-ce que tu peux allumer la lumière »

Dans le cas des commandes d'information, le choix des combinaisons syntaxiques est plus délicat. Dans notre corpus, ce type de commande peut se trouver sous la forme de phrases interrogatives, impératives, ou de phrases nominales. Dans les deux derniers cas (impératives et phrases nominales), l'utilisateur souhaite connaître une information précise, et attend une réponse développée.

En revanche, dans le cas des phrases interrogatives, il existe deux situations. Coveney (2011) distingue les « interrogatives totales » et les « interrogatives partielles ». Nous verrons que les interrogatives partielles de ce corpus ont le même comportement que les impératives et les phrases nominales, et qu'elles pourront être traitées de la même façon.

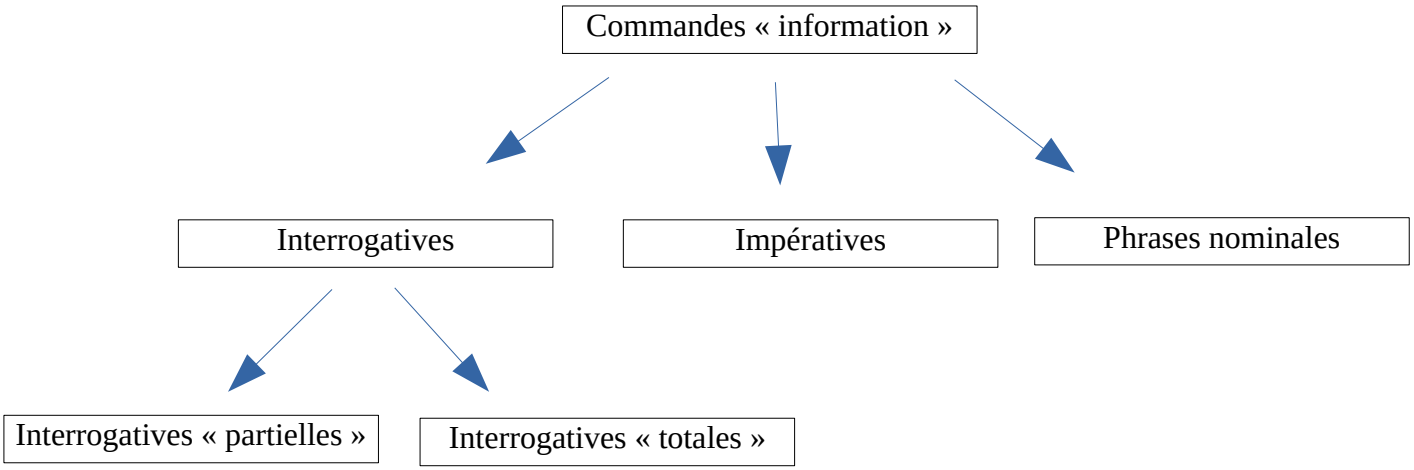

Figure 7: Types et sous-types de commandes d'information

Une interrogative est dite « totale » lorsque sa réponse ne peut être que « oui », « non » ou « si ». Elle porte sur l'action qui est exprimée par le verbe : « est-ce qu'il va pleuvoir ? ».

Dans le cas de l'interrogative « partielle », la question porte sur un élément particulier de la phrase.

Par exemple :

- « Qui intervient à la réunion ? » → le sujet

- « A quelle heure commence la réunion ? » → le complément circonstanciel

La figure 8 montre les différentes structures syntaxiques pour l'interrogation totale établies par Pohl (1965), Söll (1983) et Behnstedt (1973).

Pohl	**Söll**	**Behnstedt**
SV	SV	SV
ESV : 1. *est-ce que* 2. *n'est-ce pas que* 3. *hein que* 4. *-ti*	ESV	ESV
VS : 1. VScl (inversion simple) 2. SsnVScl (inversion complexe)	VS : 1. VScl 2. SsnVScl	VS : 1. VScl 2. SsnVScl
		SV *-ti*

S = sujet / Scl = sujet pronom clitique / Ssn = sujet syntagme nominal / V = verbe / E = *est-ce que* /

Figure 8: Structures syntaxiques de l'interrogative totale (Zumwald, Juin 2010)

Dans notre corpus, nous pouvons prendre l'exemple de la commande : « est-ce que la salle est libre maintenant ». Cette phrase correspond à la structure ESV, soit [est-ce que – sujet – verbe]. Ici le sujet est le syntagme nominal « la salle » suivi du verbe auxiliaire « est » (être). Cette phrase peut être déclinée avec les combinaisons syntaxiques présentes dans le tableau :

- SV [Sujet – Verbe] : « la salle est libre maintenant » (dans ce cas, l'interrogation est implicite, et sera marquée à l'oral par la prosodie)

- SsnVScl [Sujet syntagme nominal – Verbe – Sujet pronom clitique] : « la salle est-elle libre maintenant »

Pour les interrogatives partielles, les structures syntaxiques ne sont pas les mêmes, ce qui fait également différer les combinaisons appliquées lors de la construction des paraphrases. La figure 9 montre les différentes structures de l'interrogative partielle toujours identifiées par Pohl (1965), Söll (1983) et Behnstedt (1973).

Pohl	**Söll**	**Behnstedt**
KSV	KSV	KSV
KESV	KESV : 1. K *est-ce que* SV 2. K *c'est que* SV 3. K *que* SV	KESV : 1. K *est-ce que* SV 2. K *c'est que* SV 3. K *que c'est que* SV
SVK	SVK	SVK
K(S)VS 1. KVS 2. KSsnVScl	KVS[32]	K(S)VS 1. KVS 2. KSsnVScl
		K*que*SV

K = mot interrogatif / S = sujet / Scl = sujet pronom clitique / Ssn = sujet syntagme nominal / V = verbe / E = *est-ce que* /

Figure 9: Structures syntaxiques de l'interrogative partielle [Zumwald, Juin 2010]

Dans notre corpus, les interrogatives partielles sont les plus nombreuses. Parmi elles, la structure syntaxique la plus récurrente est la K(S)VS [mot interrogatif – Verbe – Sujet]. On retrouve en majorité l'adjectif interrogatif « quel.le.s » : « quels sont les gros titres », mais également la locution « c'est quoi » : « c'est quoi la météo à Bastille ». Le corpus comporte également des interrogatives partielles plus complexes à traiter : par exemple « combien j'ai de messages non lus » ou « comment on dit se tenir à carreaux en chinois », que nous ne traiterons pas dans ce mémoire.

Comme évoqué précédemment, outre la forme interrogative, les commandes d'information peuvent prendre la forme de phrases contenant un verbe à l'impératif ou de phrases nominales composées de simples syntagmes nominaux isolés. Les structures syntaxiques de ces deux catégories peuvent amener à la formation de paraphrases ayant les mêmes combinaisons syntaxiques que les interrogatives partielles. Cela est dû au fait que sémantiquement, l'intention de l'utilisateur est la même : il souhaite qu'on fournisse une réponse développée à sa demande.

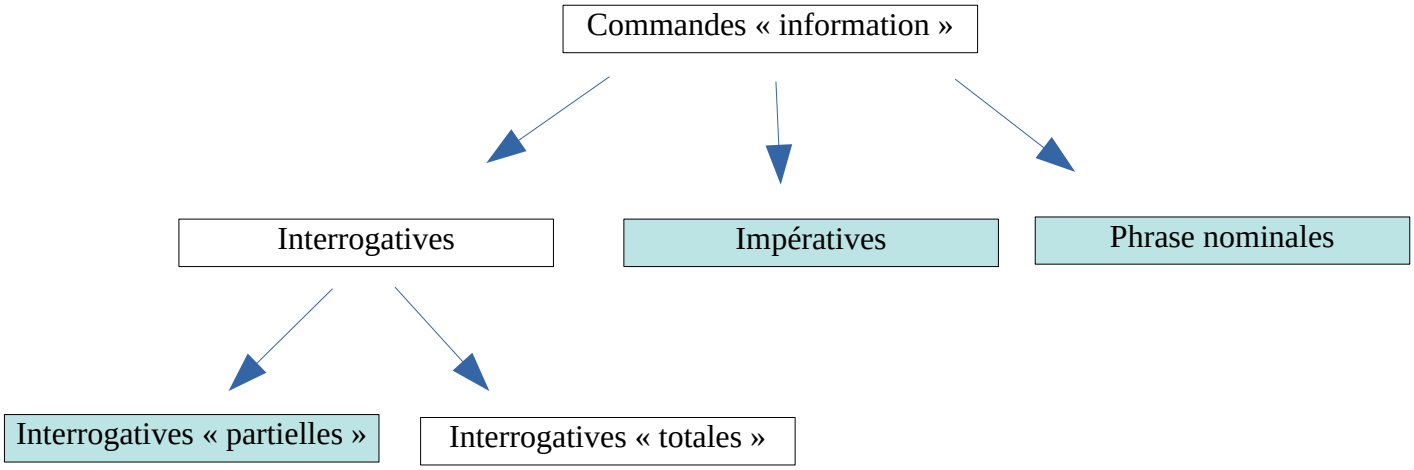

Figure 10: Types de commandes auxquels les mêmes combinaisons syntaxiques peuvent être appliquées lors de la génération des paraphrases

Plus concrètement, si l'on prend comme exemple :

- Un impératif : « donne-moi la température à Nantes »

- Une interrogative partielle : « quelle est la température à Nantes »

- Une phrase nominale : « température à Nantes »

On s'aperçoit que le sens de la phrase ne change pas. Ainsi pour les commandes d'information : les interrogatives partielles, les impératives et les phrases nominales ont un comportement identique et on peut leur appliquer les mêmes transformations.

3.2 - Définition des différentes étapes de développement

Dans cette sous-partie, nous décrivons les différentes étapes qu'il faudra mettre en œuvre pour le développement de notre module. La mise en pratique sera abordée plus loin dans la seconde partie du mémoire : *Réalisation du cahier des charges*.

3.2.1 - Récupération des parties du discours

Une fois le type de commande identifié, nous savons quels types de combinaisons syntaxiques nous pouvons leur appliquer pour la paraphrase. Mais avant la génération, nous devons récupérer les autres unités de sens essentielles de la phrase. C'est à partir de

ces éléments que l'on pourra générer des mots voisins (synonymie, hyperonymie...), et garder le sens initial de la commande.

Dans notre corpus, les phrases ne contiennent pas de signes de ponctuation puisqu'elles sont transcrites de l'oral. Les commandes sont unitaires, peu importe leur type : on ne trouve qu'un syntagme nominal par complément d'objet direct (ou un syntagme nominal seul). En revanche, ce COD peut comporter :

- un nom : « météo »

- un nom + adjectif : « évènements internationaux »

- un nom + une apposition du nom : « actualités science »

On peut également trouver un syntagme prépositionnel en fin de phrase. Celui-ci a un rôle de complément circonstanciel « météo à Toulouse », « quelles sont les actualités culturelles du mois » ou de complément du nom « enregistrement de la réunion ».

Notre objectif est de récupérer ces deux syntagmes (nominal et prépositionnel), remplacer les mots qui les composent par des mots voisins, puis selon le type de commande, les intégrer dans de nouvelles structures syntaxiques.

3.2.2 - Génération des phrases synonymes

Une fois les syntagmes récupérés, nous devons leur appliquer différentes combinaisons syntaxiques pour apporter de la variété au corpus, tout en gardant le sens général de la phrase.

Nous pouvons appliquer trois types de variantes :

→ les ajouts : ajout d'un terme ou d'une expression qui n'existait pas dans la commande initiale sans en modifier le sens

→ les changements : modification de la forme d'une partie du discours existante dans la commande initiale

→ les suppressions : suppression d'un mot existant dans la commande initiale

Les ajouts concernent essentiellement les mots et expressions de début de phrase (« est-ce que tu peux », « quel.le.s », « c'est quoi » …). Dans le cas des interrogatives

totales, l'ajout peut être une reprise pronominale. Par exemple : « la salle est libre » (interrogative marquée par la prosodie) → « la salle est-elle libre ».

Les changements sont de deux types :

Les modifications de nature morphosyntaxiques : dans notre cas, il s'agit du passage du verbe de l'infinitif (« peux-tu allumer la lumière ») vers l'impératif (« allume la lumière »), ou de l'impératif vers le subjonctif (« je veux que tu allumes la lumière »), et inversement.

Les changements impliquant les mots équivalents : à l'étape précédente, nous avons récupéré les syntagmes qui composent la phrase. Ces syntagmes sont composés de noms et d'adjectifs qui peuvent être remplacés par des équivalents porteurs du même sens. Dans notre corpus, nous avons l'exemple : « peux-tu stopper le chronomètre » → « peux-tu arrêter le minuteur ».

Les suppressions :

Dans le cas des commandes d'information par exemple, la suppression des mots/expressions interrogatifs dans la phrase interrogative permet de la transformer en phrase nominale (« quelle est la météo à Toulouse » → « météo à Toulouse »).

Partie 2

-

Réalisation du cahier des charges

1 - Pré-traitement d'une commande

1.1 - Différenciation des commandes d'action et d'information

La première tâche de notre module est d'identifier si la commande prise en entrée est une commande d'action ou d'information. Comme expliqué dans la première partie *Construction du cahier des charges*, cela permet de savoir quels types de paraphrases peuvent être appliqués sans perdre la sémantique initiale de la commande.

Pour les commandes d'action, l'identification se fait simplement. Nous parcourons la phrase à la recherche d'un mot annoté avec une entité débutant par « action ». Pour cela, nous utilisons une expression régulière permettant d'identifier les mots marqués par le format markdown (exemple : [ferme](action_off) le [volet](store) → « ferme » est le verbe d'action).

Pour les commandes d'information, nous ne pouvons pas nous appuyer sur l'annotation des entités. Pour les interrogatives partielles, l'identification se fait de plusieurs façons :

- des lexiques d'expressions-clés : ils contiennent des mots/expressions interrogatifs de début de phrase (« c'est quoi ») ou des expressions verbales (« donne-moi ») spécifiques aux commandes d'information (à différencier des verbes d'action déjà annotés des commandes d'action)

- des expressions régulières permettant de retrouver les différentes formes fléchies des adjectifs interrogatifs (« quel.le.s »)

- la présence d'un syntagme nominal en début de phrase (« météo à Toulouse » → forme inexistante pour les commandes d'action)

Le cas des lexiques d'expressions-clés est particulier. En parcourant notre corpus, nous avons pu établir des listes de mots et expressions présentes en début de phrase et caractéristiques des commandes d'information, plus spécifiquement, des interrogatives partielles.

Au départ, nous n'avons listé que les mots clés présents dans le corpus, mais il est très vite apparu que ces lexiques pouvaient servir aussi bien en reconnaissance qu'en génération.

Nous avons donc mêlé les expressions récupérées dans le corpus et d'autres pouvant servir à la génération des paraphrases. Ces nouveaux ajouts permettent de reconnaître des commandes générées grâce au module et de produire de nouvelles paraphrases. Exemple : « peux-tu », « merci de », « j'aimerais ».

Ce lexique a été divisé en plusieurs catégories, réparties dans des fichiers textes différents. Connaître les différences entre ces catégories n'est pas utile pour l'identification du type de commande, en revanche, cela l'est pour la génération des nouvelles phrases. Nous parlerons de leur rôle dans la partie consacrée à la grammaire générative.

Dans nos lexiques, nous avons omis certains mots interrogatifs apparaissant dans notre corpus et pouvant être reliés aux interrogatives partielles : « combien », « qui », « comment » … Cela est dû au fait que ces phrases appartiennent à des sous-catégories d'interrogatives partielles que nous n'avons pas encore pu identifier et traiter dans notre grammaire.

Pour les interrogatives totales, nous utilisons des patrons syntaxiques pour identifier les différentes variantes. Ces patrons sont appliqués sous la forme d'expressions régulières. Ils sont directement inspirés des constructions syntaxiques identifiées par Pohl (1965), Söll (1983) et Behnstedt (1973). Nous y ajoutons également les éventuels adjectifs et syntagmes prépositionnels, ayant un rôle d'adjectif, pouvant accompagner les noms.

```
# (est-ce que) la salle (de réunion) est libre
pattern1 = re.search('^DET NOUN( ADJ)?( ADP( DET)? NOUN( ADJ)?)* (AUX|VERB)(?! PRON)', gramCat)
# (est-ce que) il neige
pattern2 = re.search('^PRON (AUX|VERB)', gramCat)
# la salle (de réunion) est-elle libre
pattern3 = re.search('^(DET NOUN( ADJ)?|PROPN)( ADP( DET)? (NOUN|PROPN)( ADJ)?)* (AUX|VERB) PRON', gramCat)
# pleut-il
pattern4 = re.search('^(VERB|AUX) PRON', gramCat)
```

Figure 11: Patrons syntaxiques permettant d'identifier les interrogatives totales

Si un patron est identifié dans la commande, on l'identifie comme commande d'information – interrogatives totales, mais on précise également sa construction :

- SN_V : syntagme nominal + verbe

- PRON_V : pronom + verbe

- SN_V_PRON : syntagme nominal + verbe + pronom

- V_PRON : verbe + pronom

Lors de la génération de nouvelles phrases, cela permet de ne pas essayer de ré-appliquer la combinaison syntaxique d'origine à la phrase.

1.2 - Récupération des parties du discours

Lorsque le type de commande est identifié, nous avons besoin de récupérer les autres unités de sens essentielles, pouvant appartenir à différentes parties du discours.

Cette étape permet :

- de savoir où placer les mots dans les nouvelles commandes, selon leur catégorie grammaticale

- de trouver les termes qui leur sont équivalents

1.2.1 - Présentation de Standford CoreNLP et Spacy Python

Analyse des parties du discours et des dépendances syntaxiques

Pour l'analyse syntaxique, nous nous sommes principalement appuyés sur Standford CoreNLP. Il s'agit d'une boîte à outils développée contenant :

- Un étiqueteur syntaxique : un outil permettant d'identifier les différentes parties du discours d'une phrase ou d'un texte.

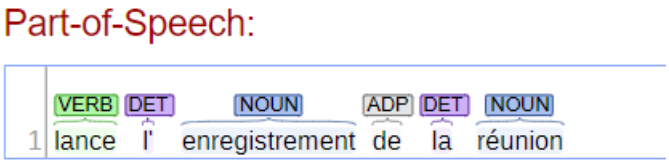

Figure 12: Exemple de sortie de l'étiqueteur syntaxique de CoreNLP

- Un analyseur de dépendances syntaxiques : un outil permettant d'étiqueter les relations syntaxiques entre les parties du discours.

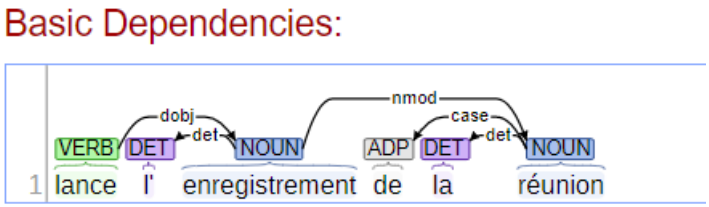

Figure 13: Exemple de sortie de l'analyseur de dépendances syntaxiques de CoreNLP

Il dispose également d'outils permettant l'analyse de sentiments ou la reconnaissance d'entités nommées. Nous n'utiliserons pas ces outils dans le cadre de notre projet.

CoreNLP a été développé en langage Java, mais notre module utilise le langage Python. Nous avons donc fait appel à un wrapper pour pouvoir l'utiliser.

Pour l'analyse des parties du discours et des dépendances syntaxiques, ce wrapper fournit une sortie sous la forme d'une liste de tuples :

- pour les parties du discours, il s'agit du mot suivi de sa catégorie grammaticale (voir figure 14)

```
[('lance', 'VERB'), ("l'", 'DET'), ('enregistrement', 'NOUN'), ('de', 'ADP'), ('la', 'DET'), ('réunion', 'NOUN')]
```

Figure 14: Sortie du wrapper CoreNLP pour les parties du discours

- pour l'analyse en dépendances, nous avons d'abord le nom de la relation, suivi de l'index (+1) de la tête et de l'index (+1) du dépendant dans la liste des parties du discours précédente (voir figure 15).

« La tête est l'élément qui impose ses propriétés syntaxiques aux autres éléments qui composent le constituant et qui sont ses dépendants. Les dépendants adoptent les propriétés de la tête. En français, par exemple, les déterminants, démonstratifs et adjectifs inclus dans un même constituant reflètent, en principe, des propriétés imposées par le nom avec lequel ils sont en relation. »

[Stefano Corno, 2016]

```
[('ROOT', 0, 1), ('det', 3, 2), ('dobj', 1, 3), ('case', 6, 4), ('det', 6, 5), ('nmod', 3, 6)]
```

Figure 15: Sortie du wrapper CoreNLP pour les dépendances syntaxiques

Dans ce projet, nous avons également utilisé Spacy. Il s'agit d'une libraire équivalente à CoreNLP mais pour le langage Python. Elle permet également l'identification des parties du discours et des dépendances syntaxiques dans les phrases.

Nous n'y avons fait appel que dans des cas spécifiques où CoreNLP ne parvenait pas à reconnaître un phénomène récurrent.

Par exemple, il ne parvenait pas à associer le nombre et le pourcentage dans les commandes telles que « baisse la lumière de 10 pourcents ».

Le parsing en constituants

Au début du projet, nous avons testé le parsing en constituants[8] de CoreNLP afin de récupérer les différents syntagmes présents dans la phrase en entrée. Les résultats n'ont cependant pas été concluants.

Nous avons donc créé notre propre méthode pour récupérer les mots et expressions essentiels dans la phrase d'entrée.

Cette méthode est spécifique à notre corpus : nos commandes se composent principalement d'un syntagme nominal unique suivi parfois d'un ou plusieurs syntagmes prépositionnels.

Récupération des éléments composant le syntagme nominal

Au début de notre projet, nous voulions nous baser sur la relation 'dobj' qui existe entre le verbe et son nom COD dans l'analyse en dépendances de CoreNLP.

Dans les faits, l'analyse en dépendances était trop incertaine pour récupérer le nom COD. De plus, certaines commandes n'ont pas de verbe. Pour cette tâche de récupération du syntagme nominal nous avons préféré utiliser l'étiquetage des catégories grammaticales qui donne des résultats plus constants.

[8] Il s'agit d'une analyse syntaxique qui permet d'identifier les différents syntagmes composants une phrase (syntagme verbale, nominal ...)

Nous avons récupéré l'ensemble des noms présents dans la phrase pour ensuite déterminer leur rôle. Dans notre corpus, le nom porteur du sens source est celui qui apparaît en premier.

Afin de vérifier que c'est bien le cas, nous identifions le rôle des autres noms de la liste :

- les noms appartenant à un syntagme prépositionnel : « baisse la luminosité de la salle ». Ici, notre liste contiendra « luminosité » et « salle ». Dans ce cas, nous utilisons l'analyse en dépendance pour identifier la relation 'case' qui relie une préposition et le nom qui l'accompagne. Dans notre exemple, « salle » est liée à la préposition « de » par la relation 'case'.

- les appositions du nom : nous pouvons les trouver dans une commande comme « joue le titre La Bohème ». Elles sont parfois identifiables grâce aux virgules présentes avant et après elle, mais dans notre cas les commandes sont transcrites de l'oral sans la ponctuation. Dans notre module, les noms qui ne sont ni le nom porteur du sens source ni les noms appartenant à un syntagme prépositionnel sont considérés comme des appositions. (Notons qu'une apposition du nom peut être un nom, un pronom, une proposition ou un infinitif. Dans notre corpus, il s'agit toujours d'un nom.)

Une fois que le nom principal de la phrase source est identifié, nous utilisons les relations de dépendance syntaxique pour récupérer son déterminant et l'éventuel adjectif qui l'accompagne :

Pour le déterminant nous utilisons la relation 'det' et pour l'adjectif la relation 'amod' :

dans la phrase « baisse la lumière bleue de la salle », nous récupérons les différents composants du syntagme nominal : « la », « lumière », « bleue ».

Le syntagme prépositionnel

Dans notre corpus, il a les fonctions :

- complément du nom : « quels sont les horaires de la salle »

- complément circonstanciel : « quelle est la météo à Toulouse »

CoreNLP ne différencie pas ces deux fonctions : elles partagent la relation ‘nmod’ avec le nom. Cependant, quelque soit leur fonction, les syntagmes prépositionnels se trouvent toujours en fin de phrase dans notre corpus.

Lors de l’étape de récupération des noms dans la phrase, nous avons identifié les noms ayant une relation ‘case’ avec une préposition (exemple : « de la salle » → la préposition « de » et le nom « salle » partagent la relation ‘case’). Dans le cas des compléments circonstanciels, cette relation peut être entre une préposition (notée ‘ADP’) et un nom propre (noté ‘PROPN’) comme présenté sur la figure 16 (« à Toulouse »).

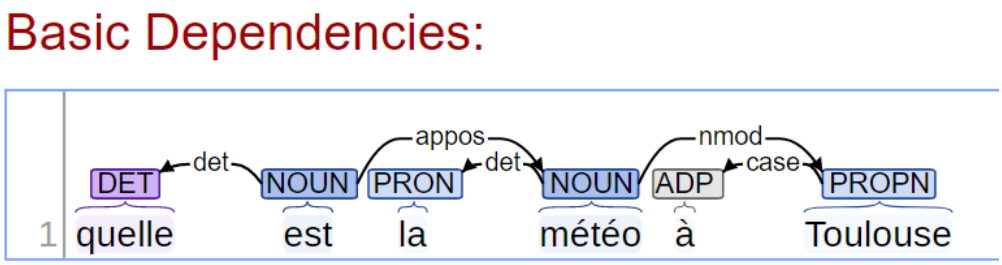

Figure 16: Exemple de relation 'case' entre une préposition et un nom propre

Une phrase peut être suivie de plusieurs syntagmes prépositionnels, nous les récupérons tous dans leur ordre d’apparition. Dans la phrase « quel est le niveau de pollution à Paris »), les expressions « de pollution » et « à Paris » sont les deux syntagmes prépositionnels. Le nom ou nom propre est récupéré avec son déterminant s’il existe (relation ‘det’) et sa préposition (relation ‘case’). Ils sont directement récupérés sous la forme de tuples.

Remarque :

Sur la figure 16 on peut voir que CoreNLP gère mal les phrases commençant par le déterminant interrogatif « quel ». Il identifie systématiquement le verbe « être » qui suit comme un nom, et parfois comme un déterminant (entraînant également des erreurs dans le reste de l’analyse). Ces erreurs sont récurrentes, nous les prenons donc en compte durant notre traitement.

Le verbe

La récupération du verbe est nécessaire pour les commandes d'action mais pas pour les commandes d'information.

Dans le cas des commandes d'action, nous l'identifions grâce à son annotation « action ». Nous n'utilisons pas l'analyse syntaxique de CoreNLP puisqu'il existe des commandes d'information contenant un verbe.

Comme nous l'avons évoqué précédemment, les verbes qui peuvent apparaître dans les commandes d'information ne sont pas essentiels à la sémantique de la phrase : il s'agit d'expressions de début de phrase comme « donne-moi » ou de l'auxiliaire être qui suit les déterminants interrogatifs « quel.le.s ». et qui peuvent être supprimés.

2 - Construction de la grammaire

2.1 - Construction des ressources lexicales

Dans la partie 1.1 concernant la reconnaissance des commandes d'action et d'information, nous avons évoqué les lexiques. Ces derniers permettent à la fois de reconnaître les commandes d'information et de les générer. Il en est de même pour les commandes d'action. Bien que nous ne nous servions pas de mots clés pour les reconnaître, nous avons également dressé des listes de mots et expressions permettant de créer de nouvelles phrases synonymes. Ces listes sont disponibles dans l'annexe 3.

Cette étape du développement est étroitement liée à l'étape de flexion des parties du discours qui suit. En effet, nous avons précisé plus haut que les lexiques étaient classés dans différentes catégories, séparées dans différents fichiers. Nous allons expliquer pourquoi.

Que ce soit pour les commandes d'action ou d'information, lorsque nous créons manuellement des paraphrases, nous remarquons que la partie qui subit les ajouts et suppressions est le début de la commande.

Ainsi, dans la commande d'information « météo à Toulouse », nous pouvons générer les phrases synonymes suivantes :

- « quelle est la météo à Toulouse »

- « c'est quoi la météo à Toulouse »

- « je veux la météo à Toulouse »

Et dans la commande d'action « arrête le minuteur » :

- « peux-tu arrêter le minuteur »

- « je veux que tu arrêtes le minuteur »

- « est-ce que tu peux arrêter le minuteur »

Nous ne parlons ici que du niveau syntaxique de la génération. Le remplacement des mots pleins par les termes qui leur sont équivalents est abordé plus loin.

Dans le cas des commandes d'information, toute la phrase est figée et seule l'expression indiquant une interrogation varie. Nous avons donc dressé une liste contenant

des expressions figées, qui peuvent être placées en début de phrase et précéder le syntagme nominal principal.

Dans le cas des commandes d'action, le fonctionnement de la phrase est à peu près identique, mais le verbe indiquant l'action à effectuer subit des changements morphologiques selon l'expression qui le précède. Il peut être laissé sous sa forme canonique ou fléchi. Dans notre cas, il ne peut y avoir que deux cas de figure : l'infinitif ou le subjonctif. La forme subjonctive apparaît après une expression de souhait : « je veux que », et la forme infinitive après des expressions telles que « merci de », « peux-tu ».

Nous avons donc séparé les mots et expressions de notre lexique selon la forme que doit prendre le verbe qui suit :

- *interro_infinitive :* les expressions suivies d'un verbe à l'infinitif

- *interro_subj :* les expressions suivies d'un verbe au subjonctif

- *interro_sn :* les expressions suivies d'un syntagme nominal

Il existe aussi d'autres catégories d'expression :

- *interro_adverb* : adverbes interrogatifs (« est-ce que ») de début de phrase qui doivent être complétés par une autre expression permettant d'introduire le verbe.

- *post_interro_ability :* expression introduisant le verbe et suit un adverbe interrogatif (exemple : « tu peux ») → « est-ce que tu peux ». Au début du projet, cette liste comportait des expressions telles que « ce serait possible de » ou « tu pourrais » qui illustrent la variété d'expressions possibles. Cependant, nous ne les utiliserons pas, car la formulation des commandes doit être plus directe.

2.2 - Formalisme des règles

Afin de formaliser les différentes combinaisons syntaxiques applicables à un type de commande, nous avons créé différentes règles.

Dans un premier temps, nous avons dressé la liste des éléments que nous pourrions trouver dans les paraphrases générées :

- Les lexiques de mots et expressions que nous avons dressés précédemment sont le premier élément qui apparaît en début de phrase.

- Pour les commandes d'action, le verbe peut être récupéré. Il subit également des changements morphologiques (infinitif, impératif, subjonctif) qu'il faut préciser dans la règle.

- Que ce soit pour les commandes d'action ou d'information, nous pouvons récupérer le nom principal. En revanche, ce nom peut ou ne pas être accompagné d'un adjectif. Sa présence n'est donc pas prévisible dans les règles. De même, nous ne pouvons pas prévoir les appositions du nom ni les noms et adjectifs issus de syntagmes prépositionnels.

Nous avons développé un système permettant de créer des règles correspondant à différentes combinaisons syntaxiques possibles pour la paraphrase. Ces règles sont spécifiques à chaque type de commande. La figure 17 montre un exemple de règles pour les commandes d'action, ainsi qu'un exemple de phrase pouvant être générée avec chaque règle. Un aperçu des règles pour les commandes d'informations spécifiques aux interrogatives partielles et aux interrogatives totales est disponible en annexe 1.

```
# tu pourrais allumer la lumière
'<interro_infinitive> <verb> <det> <noun>'
# est-ce que tu pourrais allumer la lumière
'<interro_adverb> <post_interro_ability> <verb> <det> <noun>'
# allume la lumière
'<verb_tense_impe> <det> <noun>'
# allumer la lumière
'<verb> <det> <noun>'
# j'aimerais que tu allumes la lumière
'<interro_subj> <verb_tense_subj> <det> <noun>'
```

Figure 17: Règles pour la génération de commandes d'action (aussi disponible en annexe 2)

Les règles mêlent les unités de sens récupérées dans la phrase d'entrée et des éléments issus du lexique (listés dans des fichiers séparés).

Une règle est composée de plusieurs balises faisant chacune référence à un élément qui composera la paraphrase finale.

Les balises que nous avons développées jusqu'à maintenant sont les suivantes :

→ les mots et expressions d'interrogation (leurs balises portent le même nom que les catégories vues précédemment dans la partie 2.1):

Éléments de règle pour les commandes d'action :

- <interro_infinitive>

  → expression suivie d'un verbe à l'infinitif

  → issue du lexique d'expressions pré-établi

  → exemple : « merci de »

- <interro_subj>

  → expression suivie d'un verbe au subjonctif

  → issue du lexique

  → exemple : « je veux que »

- <interro_adverb> :

  → adverbe interrogatif suivi d'une expression <post_interro_ability> (voir point suivant)

  → issue du lexique d'expressions pré-établi

  → exemple : « est-ce que »

- <post_interro_ability> :

  → expression de capacité précédée par un <interro_adverb> et suivie par un verbe à l'infinitif

  → issue du lexique d'expressions pré-établi

  → exemple : « tu peux »

Éléments de règle pour les commandes d'information :

- <interro_sn> :

  → expression suivie d'un syntagme nominal

  → issue du lexique d'expressions pré-établi

  → exemple : « c'est quoi »

- <interro_determinant> :

  → déterminant interrogatif suivi d'un <verb_auxiliary> (auxiliaire être)

  → issue du lexique d'expressions pré-établi

  → exemple : « quel.le.s »

→ les verbes :

Le verbe peut être le verbe de la phrase d'entrée ou un verbe dont le sens est voisin et issu d'une des méthodes que nous allons expliciter dans la partie dédiée à la *Génération de mots voisins*. Il en est de même pour les autres mots pleins comme les noms et adjectifs.

Éléments de règle pour les commandes d'action :

- <verb> : verbe sous sa forme infinitive
- <verb_tense_impe> : verbe à l'impératif
- <verb_tense_subj> : verbe au subjonctif

Éléments de règle pour les commandes d'information :

- <verb_auxiliary> : auxiliaire être
- <verb_demo> : verbe suivant une expression une expression <interro_infinitive>

→ nom :

<noun> : nom noyau du syntagme nominal

→ déterminant :

<det> : déterminant appartenant au syntagme nominal

Les éléments de règle pour les commandes d'information appartenant au sous-type des interrogatives totales sont spécifiques :

<interro_if> : expression issue du lexique d'expressions pré-établi contenant la conjonction « si » et précédant une <sentence> (exemple : « tu peux me dire si »)

<sentence> : phrase d'entrée que l'on débarrasse de l'expression interrogative qui la précède (exemple : « est-ce que la salle est libre » → « la salle est libre »)

<sentence_inverted_subj> : phrase sans expression interrogative à laquelle on applique la reprise pronominale (exemple : « la salle est libre » → « la salle est-elle libre »)

2.3 - Parseur de grammaire

2.3.1 - Grammaire des commandes d'action / information sous-type interrogatives partielles

Afin de lire les éléments de notre règle, nous avons conçu un parseur. Celui-ci permet de lire la règle et de remplacer chaque balise par le mot ou l'expression qui convient.

```
'<interro_adverb> <post_interro_ability> <verb> <det> <noun>'
```

Figure 18: Exemple de règle de génération pour les commandes d'action

Prenons l'exemple de la règle de la figure 18 : il faut commencer par générer le nom requis par la balise *noun.* C'est en fonction de ses traits morphologiques (ici le genre et le nombre) qu'il est ensuite possible de générer correctement ses dépendants (déterminants, adjectifs…). Si la commande prise en entrée est « ouvre les rideaux », nous pouvons récupérer le nom d'origine « rideaux » ou générer un terme équivalent. Pour un équivalent comme « stores », nous gardons en mémoire les informations masculin/pluriel.

Nous procédons ensuite à la lecture des autres éléments de règles : la balise *interro_adverb* correspond au nom d'une catégorie présente dans les lexiques vus précédemment. Le module choisira de manière aléatoire une des expressions de la liste (le seul adverbe interrogatif que nous traitons actuellement est « est-ce que »). Une action identique sera effectuée pour l'élément *post_interro_ability* (exemple : « tu peux »). L'élément suivant, *verb*, correspond à un verbe sous sa forme infinitive. Comme pour le nom, il s'agit de choisir de manière aléatoire un verbe équivalent au verbe d'entrée, ou de réutiliser ce dernier (« ouvrir »). Nous devons ensuite générer un *det*, un déterminant dont l'accord se fait en genre et en nombre avec le nom que nous avons choisi précédemment

« stores ». Grâce aux traits de genre et nombre récupérés précédemment, on peut générer « les ». La paraphrase finale sera « est-ce que tu peux ouvrir les stores ».

Cet exemple est simple, mais le cas de la flexion s'étend à d'autres éléments de la phrase. La figure 19 montre une règle dont le premier élément est un déterminant interrogatif (exemple : « quel.le.s »). Celui-ci doit être accordé en genre et en nombre avec le nom qui se trouve à la fin de la règle. Il en est de même pour l'auxiliaire et le déterminant qui suivent. Cette règle peut générer la commande « quelles sont les actualités ».

```
'<interro_determinant> <verb_auxiliary> <det> <noun>'
```

Figure 19: Exemple de règle dont la flexion de plusieurs éléments dépendent du nom

L'élément que nous devons traiter avant tout est donc le nom, puisqu'il est l'élément tête dont dépend tous les autres.

Développons le cas des parties du discours qui n'apparaissent pas dans nos règles :

Dans une phrase un nom peut être accompagné d'un adjectif. Il s'agit d'un élément dont la présence est variable et imprévisible. Nous ne pouvons donc pas avoir de balise pour l'adjectif.

Ainsi, nous supposons implicitement son traitement avec la balise du nom. Grâce à l'étape de récupération des parties du discours (voir sous-partie 1.2 du *Pré-traitement de la commande d'entrée*), nous savons si un adjectif est rattaché au nom et nous le récupérons. Au moment du traitement de la balise *noun*, nous prenons cet adjectif ou générons un terme équivalent que nous fléchissons selon le genre et le nombre du nom. Ainsi, la règle de la figure 18 peut également générer « quelles sont les actualités sportives ».

2.3.2 - Grammaire des commandes d'information, sous-type interrogatives totales

Au début de notre projet, nous avons commencé par identifier deux types de commandes : les commandes d'action et les commandes d'information de type interrogatives partielles.

Pour les commandes d'action, il n'y a pas de sous-types de phrase à reconnaître, les nouvelles commandes générées s'articulent essentiellement autour du verbe d'action.

Pour les commandes d'information, nous n'avions traité que les cas les plus récurrents :

- pour les interrogatives partielles : le schéma syntaxique [mot interrogatif – verbe – syntagme nominal], pour lequel des mots interrogatifs et expressions bien spécifiques sont reconnus (« c'est quoi », « quel.le.s », « donne-moi »...)

- les phrases nominales (« météo à Toulouse »)

Nous avons ensuite décidé d'ajouter la reconnaissance des interrogatives totales. Ces dernières apparaissent en minorité dans notre corpus, mais leurs structures sont très variables.

Afin de reconnaître leurs différentes structures syntaxiques dans la phrase d'entrée, nous nous sommes servis de patrons syntaxiques. Ces derniers nous fournissent les informations nécessaires à la construction des nouvelles phrases.

La formation des règles de notre grammaire reste la même, mais le parseur utilisé pour les interpréter est différent. Cela s'explique par la variabilité des parties du discours qui constituent les interrogatives totales prises en entrée.

Pour deux phrases données, grâce aux structures syntaxiques identifiées par Pohl (1965), Söll (1983) et Behnstedt (1973), nous obtenons les variantes suivantes :

[est-ce que – sujet – verbe]

→ « est-ce que la salle est libre ? »

→ « est-ce qu'il pleut ? »

[verbe – sujet]

→ inversion simple (verbe – sujet pronom clitique)

« pleut-il ? »

→ inversion complexe (syntagme nominal – verbe – sujet pronom clitique)

« la salle est-elle libre ? »

[sujet – verbe]

→ « la salle est libre ? »

→ « elle est libre ? »

Ces différentes structures sont utilisées pour la reconnaissance mais nous nous en servons également pour la génération.

Nous ne procédons pas à une récupération des parties du discours comme pour les interrogatives partielles ou les commandes d'action. Les interrogatives totales présentent dans notre corpus n'ont pas des structures aussi récurrentes que les deux autres types de commandes. Nous trouvons des phrases aussi diverses que « la pollution à Hakone est-elle élevée », « est-ce qu'il va pleuvoir demain » ou « Rennes est-elle polluée ». L'identification des parties du discours et la génération de leurs équivalents se fait lors du traitement de la balise <sentence> vu précédemment (voir partie 2.2 *Formalisme des règles*).

Une fois l'interrogative totale identifiée, nous la débarrassons du mot interrogatif qui la précède s'il existe : « est-ce que la salle est libre » → « la salle est libre »

Cette étape permet de normaliser les commandes d'entrée avant de les paraphraser. Le morceau de phrase restant est récupéré dans son ensemble et est identifié par la balise <sentence> dans les règles de génération. Dans le cas où la phrase ne contient pas de mot interrogatif mais est sujette à une reprise pronominale (exemple : « la salle est-elle libre ») ou qu'il s'agit d'une inversion sujet-verbe (exemple : « neige-t-il »), nous lui faisons reprendre la forme d'une phrase déclarative (« la salle est libre », « il neige »).

La règle de la figure 20 n'est composée que de deux éléments, mais la balise <sentence> évoquée précédemment joue le rôle des parties du discours dans la grammaire précédente.

```
'<interro_if> <sentence>'
```

Figure 20: Exemple de règle pour la génération d'interrogatives totales

Par exemple, pour la phrase « est-ce qu'il va pleuvoir », nous commençons par retirer l'expression interrogative que nous avons identifiée grâce à une expression régulière: « il va pleuvoir ». Nous pouvons ensuite commencer à lire notre règle. Le

premier élément (*interro_if*) est une expression interrogative contenant la conjonction « si » : nous choisissons une des expressions issue de notre liste (« dis moi si »). Le deuxième élément (*sentence*) est la phrase précédemment débarrassée de son adverbe interrogatif. La nouvelle phrase générée est donc « dis-moi si il va pleuvoir demain ».

La gestion des élisions[9] est effectuée à la fin de la génération grâce à des expressions régulières : « dis moi s'il va pleuvoir demain ».

Notons qu'ici, lors du traitement de la balise *sentence*, nous pouvons aussi remplacer les mots contenus dans la phrase normalisée par leurs équivalents.

La règle qui gère les inversions simple et complexe ne contient quant à elle qu'un seul élément : <sentence_inverted_subj>. Elle implique un changement au sein de la phrase normalisée. Ce changement dépend de sa structure. Il existe plusieurs possibilités :

→ la présence d'un syntagme nominal sujet : dans ce cas, nous procédons à une inversion complexe qui consiste à ajouter la reprise pronominale (exemple précédent : « la salle est libre » devient « la salle est-elle libre »)

→ la présence d'un pronom clitique sujet : on effectue une inversion simple (exemple : « il pleut » devient « pleut-il »)

Dans le cas de l'inversion simple, il peut y avoir deux possibilités en entrée :

« il pleut » vu précédemment, et « il va pleuvoir ». Dans la deuxième phrase, « va » est un semi-auxiliaire qui introduit le verbe « pleuvoir ». Ce phénomène implique que lors de la génération des termes équivalents, ce n'est pas le semi-auxiliaire qui nous intéressera mais plutôt le verbe qui le suit.

Cette exemple illustre la multiplicité des structures syntaxiques que nous devons gérer pour la génération des nouvelles phrases.

[9] L'élision est l'effacement de la voyelle qui finit un mot lorsque le mot suivant débute lui aussi par une voyelle

3 - Fléchir les parties du discours

3.1 - Présentation de GLAWI

GLAWI est un dictionnaire électronique open-source crée par Franck Sajous, Nabil Hathout et Basilio Calderone. Il s'agit de la version XML normalisée et structurée du Wiktionnaire français. Elle propose :

- des mots simples, mots composés et locutions
- des formes fléchies et leur lemme
- l'étymologie des mots
- leur prononciation
- des définitions (gloses et examples)
- des traductions
- des relations sémantiques
- des relations morphologiques (dérivatifs, antonymes, synonymes...)
- des variantes orthographiques

Pour notre usage, nous nous sommes intéressés aux lemmes et à leurs formes fléchies.

Avant la génération des nouvelles commandes, nous récupérons les parties du discours porteuses de sens (les mots dits « pleins »), notamment les verbes et adjectifs. Ces derniers apparaissent le plus souvent sous leur forme fléchie[10]. Nous devons retrouver leur forme lemmatisée[11] ou certaines de leurs formes fléchies nécessaires pour la génération des nouvelles phrases.

Contrairement aux dictionnaires classiques, les entrées de GLAWI ne sont pas que des lemmes. On peut également y trouver des formes fléchies.

Si nous prenons comme exemple la forme fléchie « culturelles » de l'adjectif « culturel » accordé au féminin pluriel, nous obtenons la figure 21.

[10] Les mots peuvent prendre plusieurs formes selon leur genre, nombre, temps, personne ect…
[11] Le lemme ou forme canonique correspond à la forme la plus simple d'un mot et est utilisé comme entrée des dictionnaires

```
<article>
  <title>culturelles</title>
  <pageId>332788</pageId>
  <meta/>
  <text>
    <pos type="adjectif" lemma="0" locution="0" gender="f" number="p">
      <pronunciations>
        <pron>kyl.ty.ʁɛl</pron>
      </pronunciations>
      <inflectionInfos>
        <inflectedForm gracePOS="Afpfp" lemma="culturel"/>
      </inflectionInfos>
      <definitions>
        <definition>
          <gloss>
            <xml><i>Féminin pluriel de</i><innerLink ref="culturel#fr">culturel</innerLink>.</xml>
            <txt>Féminin pluriel de culturel.</txt>
          </gloss>
        </definition>
      </definitions>
    </pos>
    <section type="homophones">
      <item>culturel</item>
      <item>culturelle</item>
      <item>culturels</item>
    </section>
  </text>
</article>
```

Figure 21: Article GLAWI correspondant à l'entrée "culturelles"

La section « inflectionInfos » nous fournit les différentes informations concernant la flexion du mot donné en entrée. Ici, l'attribut de l'élément « lemma » est « culturel ». On retrouve également les informations morphosyntaxiques du mot dans l'attribut « gracePOS ». Ces informations sont précisées au format GRACE (Rajman et al., 1997).

La figure 22 explique les codes attribués à chaque catégorie grammaticale, temps, modes et personnes.

Nouns	
Code	**Description**
Nc[mf][sp]	Common nouns + gender (m: masculine, f: feminine) + number (s: singular, p: plural)

Adjectives	
Code	**Description**
Afp[mf][sp]	Positive adjective + gender (m: masculine, f: feminine) + number (s: singular, p: plural)

All adjectives have been tagged as 'qualificative positive' (Afp).

Code	Description
Vmn----	Infinitive
Vmpp---	Present participle
Vm-ps-[sp][mf]	Past participle + number attribute [s/p] + gender attribute [m/f]
Vm[ism][pifs][123][sp]-	Inflected verb form + mood (i: indicative, s: subjunctive, m: imperative) + tense (p: present, i: imperfect, f: future, s: past) + person ([123]) + number (s: singular, p: plural)

Figure 22: Format GRACE explicité dans la documentation de GLAWI

Ainsi, pour l'exemple précédent, le code GRACE « Afpfp » correspond à :

- Afp : adjectif

- f : féminin

- p : pluriel

Pour notre projet, nous avions deux besoins :

- à partir de la forme fléchie d'un mot, trouver son lemme

- à partir du lemme d'un mot, trouver une forme fléchie précise : la figure 23 montre les différentes formes fléchies existantes pour le lemme « faire » dans GLAWI

```
<paradigm>
  <inflection form="firent" gracePOS="Vmis3p-" prons="fiʁ"/>
  <inflection form="fasse" gracePOS="Vmsp3s-" prons="fas"/>
  <inflection form="feraient" gracePOS="Vmcp3p-" prons="fə.ʁɛ"/>
  <inflection form="fais" gracePOS="Vmip1s-" prons="fɛ"/>
  <inflection form="fisse" gracePOS="Vmsi1s-" prons="fis"/>
  <inflection form="fis" gracePOS="Vmis2s-" prons="fi"/>
  <inflection form="fera" gracePOS="Vmif3s-" prons="fə.ʁa"/>
  <inflection form="fissiez" gracePOS="Vmsi2p-" prons="fi.sje"/>
  <inflection form="ferai" gracePOS="Vmif1s-" prons="fə.ʁe"/>
  <inflection form="fassions" gracePOS="Vmsp1p-" prons="fa.sjɔ̃"/>
  <inflection form="feront" gracePOS="Vmif3p-" prons="fə.ʁɔ̃"/>
  <inflection form="faisant" gracePOS="Vmpp---" prons="fə.zɑ̃"/>
  <inflection form="faisais" gracePOS="Vmii1s-" prons="fə.zɛ"/>
  <inflection form="fissions" gracePOS="Vmsi1p-" prons="fi.sjɔ̃"/>
  <inflection form="faite" gracePOS="Vmps-sf" prons=""/>
```

Figure 23: Exemples de formes fléchies du lemme "faire" dans GLAWI

3.2 - Méthode

Lors de nos premiers essais, nous nous sommes appuyés sur des scripts Perl proposés par les développeurs de GLAWI. Ceux-ci se présentent sous le nom *G-PeTo* (GLAWI Perl Tools). Il s'agit d'un ensemble de scripts permettant d'extraire des informations du dictionnaire de manière rapide.

Nous avons utilisé le script « extractArticles.pl », qui prend en entrée un mot et génère en sortie un fichier XML contenant l'article correspondant à ce mot. Cette méthode était coûteuse en terme de temps puisque le script était rappelé à chaque fois qu'une

information devait être cherchée. Nous avons donc créé un dictionnaire Python contenant les informations dont nous avions besoin à l'initialisation de notre programme.

La structure de ce dictionnaire est présentée sur la figure 24. Les entrées sont les mots (lemmes ou formes fléchies) et leur catégorie grammaticale. Si l'entrée est une forme fléchie les informations de genre et nombre ainsi que le lemme apparaissent. Pour les entrées lemmatisés, on peut trouver une liste des différentes formes fléchies et le code GRACE qui leur correspond.

```
# Dictionnaire Glawi

# faire_verbe : {    gender : "",
#                    number : "",
#                    lemma : "",
#                    inflection :
#                       {
#                           Vmis3p- : firent
#                           Vmsp3s- : fasse
#                           Vmcp3p- : feraient
#                       }
#               }
```

Figure 24: Structure du dictionnaire Python de GLAWI

Pour les commandes d'action, lorsque nous générons une nouvelle phrase, nous récupérons de manière aléatoire un verbe équivalent au verbe présent dans la commande d'entrée grâce à des outils que nous présenterons plus loin. Ce verbe est déjà sous sa forme lemmatisée. Puis, suivant la règle qui est choisie, il est gardé sous cette forme, ou fléchi selon le temps demandé. Pour l'instant, nos règles ne requièrent que l'usage de l'infinitif, du subjonctif et de l'impératif. La personne choisi est toujours la deuxième personne du singulier (« tu »). Les commandes de notre corpus ne contiennent pas d'exemple de commande utilisant le vouvoiement.

Lorsque le verbe d'action est déjà fléchi dans la phrase initial (« allume la lumière »), nous commençons par trouver son lemme dans le dictionnaire. A partir de ce lemme, nous cherchons la forme fléchie correspondant au temps requis (ex : le subjonctif) et à la deuxième personne du singulier.

Exemple :

verbe : « allume » → lemme trouvé dans GLAWI : « allumer »

temps demandé par la règle : subjonctif

personne : deuxième personne du singulier

informations converties au format GRACE :

Vm (verbe) + s (subjonctif) + p (présent) + 2 (personne) + s (singulier)

valeur GLAWI : « allumes »

Que ce soit pour les commandes d'action ou d'information, les noms et adjectifs sont aussi choisis parmi les termes équivalents, il faut donc effectuer le même travail pour respecter l'accord des adjectifs avec les noms qui les accompagnent.

4 - Génération des termes équivalents

4.1 - Récupération des équivalents à partir du fichier de commandes

Notre module peut prendre en entrée un fichier similaire à notre corpus actuel afin de générer des variantes pour chaque commande. Au début de notre projet, nous avons décidé de nous servir des parties du discours déjà présentes dans le fichier pour générer de nouvelles phrases. Cela permet d'avoir des termes équivalents déjà trouvés manuellement et d'être sûr de leur fiabilité.

4.1.1 - Méthode

Les équivalents que nous pouvons récupérer dans le fichier de commandes sont deux types :

- les mots qui sont des instances d'entités, annotés au format markdown

- les mots n'ayant pas été annotés

Équivalents issus d'instances de la même entité

Dans le cas des instances d'entités, nous ne pouvons pas simplement récupérer les instances présentes dans la commande d'entrée et les remplacer par d'autres présentes dans le fichier. Il faut pouvoir identifier à quelle partie du discours elles appartiennent afin de respecter la structure syntaxique des nouvelles phrases.

Nous avons commencé par créer un dictionnaire contenant toutes les instances présentes dans le fichier pour une entité donnée. (Nous partons du principe que toutes les instances d'une même entité appartiennent à la même catégorie grammaticale. Cela n'est pas toujours le cas dans notre corpus mais nous ne traitons pas cette possibilité.)

Par exemple, pour l'entité *action_on* nous récupérons les instances « lance », « démarre », « commence » ect…

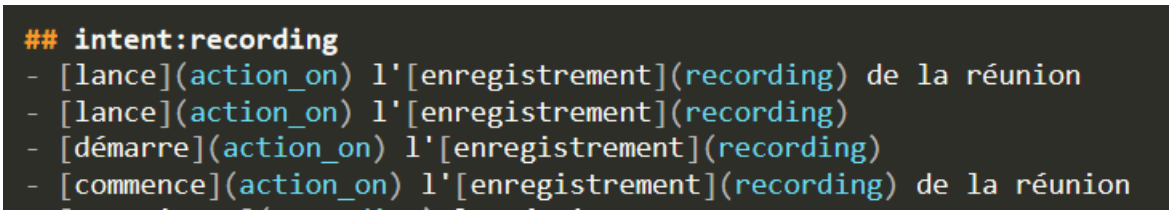

Figure 25: Instances de l'entité "action_on" dans les commandes appartenant à l'intention "recording"

Après quelques essais, nous nous sommes cependant rendus compte que pour une même entité, le sens des instances pouvait varier. Avec notre exemple précédent, nous pouvions également obtenir des verbes du type « ouvre » ou « allume ». Bien qu'appartenant à la même entité, ces instances n'étaient pas substituables entre elles.

Ces variations existaient entre des instances n'appartenant pas à la même intention dans le fichier. Nous avons donc limité la récupération aux instances présentes dans les phrases appartenant à la même intention.

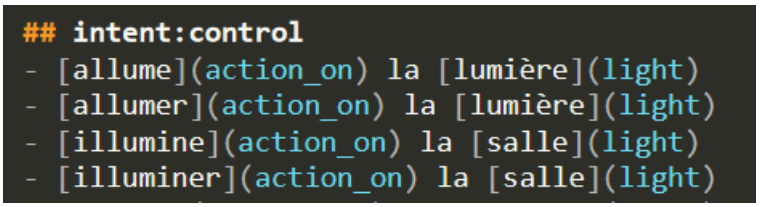

Figure 26: Instances de l'entité "action_on" dans les commandes appartenant à l'intention "control"

Une fois le dictionnaire créé, nous déterminons si certains mots de la phrase d'entrée sont identifiées comme des instances d'entité dans la phrase d'entrée. Dans le cas de la première phrase de la figure 26, c'est le cas de « allume » et « lumière ». Par exemple, dans le cas de « allume », qui appartient à l'entité *action_on*, on peut récupérer l'équivalent « illuminer ». Grâce à l'analyse préalable des parties du discours par CoreNLP, on sait également que « allumer » est le verbe et « lumière » le nom principal. Cela permet de générer des paraphrases syntaxiquement correctes.

Pour notre corpus actuelle, nous n'avons récupéré que les instances présentes dans l'intention à laquelle la phrase traitée appartient. Cependant, l'intérêt des instances d'entités est qu'elles doivent être substituables. Lorsque les annotations du corpus seront correctes nous pourrons récupérer toutes les instances présentes dans le fichier.

Les mots équivalents n'étant pas des instances d'entités

Dans le cas des mots n'ayant pas reçu d'annotation, nous ne pouvons nous appuyer que sur l'analyse des parties du discours fournie par CoreNLP. Sur la figure 27, les noms ne correspondent à aucune entité.

```
- donne moi les titres du [moment](time)
- donne moi les titres [sportifs](type_sport)
- quels sont les événements [musicaux](type_musical) du [jour](time)
- quels sont les événements [international](type_international) du [jour](time)
- quels sont les événements [politique](type_politique) du [jour](time)
- quels sont les événements [société](type_societe) de la [semaine](time)
- quels sont les événements [culturel](type_cultural) de la [semaine](time)
- quels sont les événements [planète](type_world) de la [semaine](time)
- quels sont les événements [science](type_science) du [mois](time)
- donne moi les événements [informatique](type_pixel) du [mois](time)
- donne moi les actualités [musicales](type_musical) du [mois](time)
```

Figure 27: Exemples de commandes en entrée ayant des noms non annotées

Ainsi, lorsque l'entrée de notre module est un fichier, nous commençons par effectuer la récupération des parties du discours sur l'ensemble des phrases (noms, adjectifs, appositions...). Avec l'exemple de la figure 27, on peut donc remplacer « titres » par « évènements » et « actualités ».

Le cas particulier des syntagmes prépositionnels

Les phrases en entrée peuvent contenir plusieurs noms, autres que le nom principal. Il peut s'agir d'appositions ou de noms appartenant à un syntagme prépositionnel. Dans la partie consacrée à la récupération des parties du discours, nous avions montré que les parties du discours appartenant au syntagme nominal principal étaient récupérés séparément (déterminant, nom, adjectif). Cela n'est pas le cas pour les parties du discours appartenant aux syntagmes prépositionnels, que nous récupérons comme un ensemble, sous la forme de tuples (préposition, déterminant, nom).

Si nous avons un fichier en entrée, nous récupérons les tuples contenant les syntagmes prépositionnels présents dans chaque phrase de l'intention. Nous vérifions quelles entités sont présentes dans le syntagme prépositionnel en entrée. Nous trions

ensuite ceux qui ont été récupérés précédemment dans l'intention en ne gardant que ceux qui ont les mêmes entités que l'original. Ils sont ensuite ajoutés aléatoirement aux nouvelles phrases :

Phrase originale → « quelle est la météo à [Toulouse](location) »

Syntagmes prépositionnels présents dans l'intention :

→ à [Cologne](location)

→ à [Hambourg](location) …

4.2 - Création de dictionnaires de termes équivalents à partir de ressources extérieures

Dans notre module, il est aussi possible de ne fournir qu'une phrase en entrée, et non un fichier. Dans ce cas, les équivalents doivent être récupérés dans des ressources extérieures. Ils peuvent également s'ajouter aux équivalents récupérés dans les intentions dans le cas où nous prenons un fichier en entrée du module.

4.2.1 - Présentation de ConceptNet

ConceptNet est un réseau sémantique libre d'accès. Il est issu du projet de crowdsourcing Open Mind Common Sense, lancé en 1999 par le MIT Media Lab. Depuis, il a été enrichi grâce d'autres projets de crowdsourcing, de ressources créées par des experts et de données issues de jeux sérieux.

Il prend un concept en entrée et propose des listes de mots liés à ce concept par différents types de relation. La figure 28 montre la structure du réseau sémantique. Nous n'allons pas la détailler, mais une définition de chaque relation est disponible dans l'annexe 4.

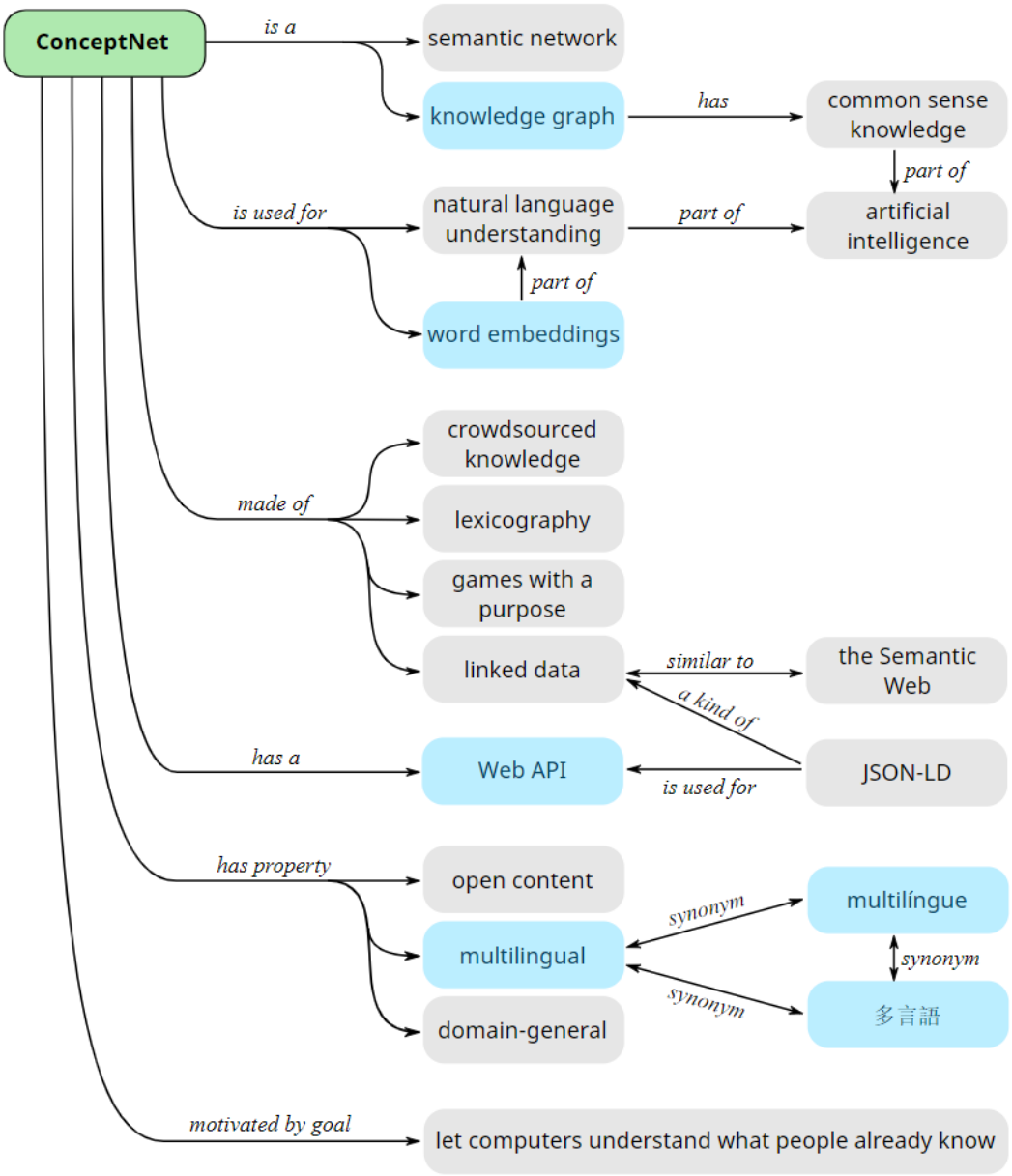

Figure 28: Structure du réseau sémantique ConceptNet

4.2.2 - Présentation de FastText

FastText est une libraire open-source crée par le laboratoire de recherche en intelligence artificielle de Facebook (FAIR). Elle permet d'entraîner des *words embeddings* (des « plongements de mots »). Il s'agit d'une méthode permettant de représenter les mots d'un corpus par des vecteurs. Cela permet d'évaluer la probabilité que des mots apparaissent dans un contexte similaire, et ainsi d'évaluer leur parenté sémantique.

On peut choisir le corpus d'entraînement des *words embeddings* afin de répondre à un besoin spécifique. Les résultats que notre modèle donnera sont influencés par les sujets abordés dans ce corpus. Pour notre projet, le corpus utilisé est Wikipédia.

4.2.3 - Méthode

Pour commencer, nous avons récupéré les mots similaires fournis par chacun des deux outils pour un mot donné.

ConcepNet propose une sortie où les mots sont triés selon leur relation avec le mot en entrée. La figure 29 montre les résultats pour le mot « démarrer ». Ces résultats proviennent de sa version web (http:/conceptnet.io/). ConceptNet est multilingue, mais pour note projet, nous n'avons gardons que les termes français de sa version téléchargeable.

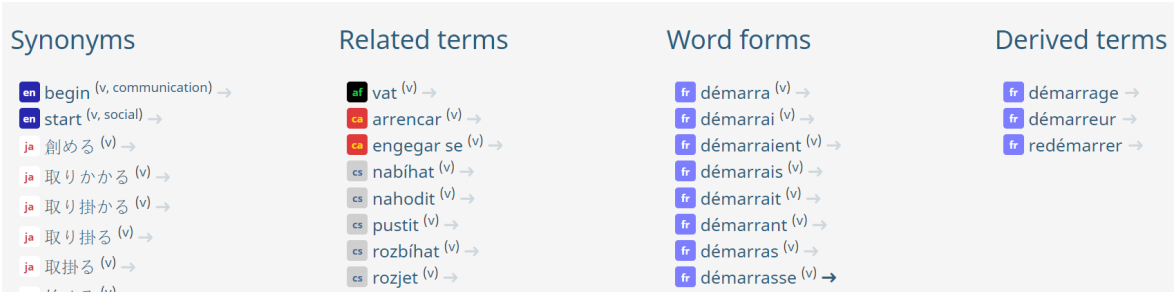

Figure 29: Exemple de sortie de ConceptNet triée par types de relation, pour le mot «démarrer»

Dans le cas de FastText, nous avons une sortie sous forme de listes de tuples (voir figure 31). Les paires sont composées du mot et de son score de similarité avec le mot en entrée.

Pour commencer, nous avons épuré la sortie de FastText en retirant les antonymes du mot d'entrée. Nous avons utilisé ConceptNet qui propose une catégorie dédiée (voir figure 30).

Figure 30: Antonymes du verbe "commencer" dans ConceptNet

Pour retirer les antonymes de la sortie de FastText, nous cherchons le mot source dans ConceptNet, récupérons sa catégorie « antonymes » et comparons la liste récupérée avec la sortie de FastText. Nous pouvons ainsi retirer « finir » et « terminer » qui étaient présents dans la sortie du verbe « commencer » (figure 31).

```
>>> model.most_similar([u'commencer'])
[('débuter', 0.8204286098480225), ('finir', 0.7753344774246216), ('terminer', 0.7655597925186157), (
'démarrer', 0.7269518375396729), ('entamer', 0.7256745100021362), ('Commencer', 0.7105789184570312),
 ('continuer', 0.7029359340667725), ('commençons', 0.7011582851409912), ('commencez', 0.695823848247
5281), ('faire', 0.6952711939811707)]
```

Figure 31: Sortie du verbe "commencer" dans FastText sans les antonymes trouvés dans la sortie de ConceptNet

Dans la sortie de ConceptNet, nous ne récupérons pas toutes les catégories dont nous disposons. Nous avons choisi les relations : *synonym*, *relatedTo*, *isA*, *similarTo* et *mannerOf*, que nous rassemblons dans une même liste.

La deuxième étape a pour but de fusionner les sorties de FastText et de ConceptNet. Au départ, nous comparons les deux sorties afin de ne garder en tête de liste que les mots communs aux deux sorties. Cependant, cette méthode ne permet pas de garder des mots qui pourraient aussi être intéressants bien que n'apparaissant que dans l'une des deux listes.

Afin de garder ces mots également, nous ajoutons une nouvelle étape d'analyse. Nous nous servons de ConceptNet pour opérer une vérification des mots non retenus.

Plus concrètement, dans le cas du verbe « stopper », nous obtenons les résultats présentés sur la figure 33. La première liste correspond à la sortie de FastText à laquelle on a retiré les antonymes de ConceptNet. La deuxième liste correspond à la sortie de ConceptNet, qui est la fusion des catégories *synonym*, *relatedTo*, *isA*, *similarTo* et *mannerOf*.

La troisième liste est issue de la fusion des deux listes précédentes. On y trouve :

- Les mots communs aux deux sorties : « cesser » et « empêcher » par exemple.

- Des mots présents dans FastText mais pas directement dans la sortie de ConceptNet : « interrompre ». Ici, le mot en entrée est « arrêter », et la sortie de ConceptNet ne comporte pas le mot « interrompre » pourtant présent dans FastText et qui serait intéressant pour la liste finale. Pour chaque mot de FastText n'apparaissant pas dans ConceptNet, nous le récupérons comme mot d'entrée et allons de nouveau chercher sa sortie dans ConceptNet. Pour notre exemple, nous cherchons « interrompre » dans ConceptNet, puis récupérons la liste des mots de chaque catégorie (*synonym*, *relatedTo*, *isA*, *similarTo* et *mannerOf*). Dans cette nouvelle liste, nous vérifions si « interrompre » est lié au mot source de départ « arrêter » (voir figure 32).

Figure 32: Relation entre le mot source "interrompre" et le mot cible "arrêter" dans ConceptNet

Si c'est le cas, le mot est gardé et intégré à la liste finale.

```
FastText similar words without ConceptNet antonyms :
[('stopper', 0.7697960138320923), ('arréter', 0.769567608833313), ('arreter', 0.7473807334899902), (
'cesser', 0.7436692118644714), ('continuer', 0.7298405170440674), ('Arrêter', 0.7277204990386963), (
'arrête', 0.7257634401321411), ('empêcher', 0.7183923721313477), ('interrompre', 0.7083262205123901)
, ('arrêter', 0.6919307112693787)]
ConceptNet similar words :
['alpaguer', 'appréhender', 'attraper', 'avoir', 'capturer', 'chauffer', 'choper', 'coffrer', 'coinc
er', 'coxer', 'interpeller', 'pincer', 'prendre', 'pécho', 'serrer', 'épingler', 'activité', 'aller'
, 'amuser', 'arrestation', 'arrêt', 'arrêtable', 'arrêtage', 'arrêtiste', 'arrêtoir', 'assurer', 'at
tention', 'avancer', 'bouger', 'cesser', 'compte', 'concert', 'continuer', 'demi temps', 'débusquer'
, 'empêcher', 'en avant', 'entreprise', 'faire', 'fil', 'fixer', 'fond', 'gibier', 'idée', 'individu
ellement', 'lieu', 'marche', 'nœud', 'prendre', 'prisonnier', 'quelque chose', 'quelqu' un', 'rester
', 'retenir', 'réarrêter', 'régler', 'résoudre', 'saisir arrêter', 'sans', 'service', 'sortir', 'sto
pper', 'sur', 'tarder', 'temps', 'usage', 'échapper']
final similar words list :
[('stopper', 0.7697960138320923), ('cesser', 0.7436692118644714), ('continuer', 0.7298405170440674),
 ('empêcher', 0.7183923721313477), ('interrompre', 0.7083262205123901)]
```

Figure 33: Résultat de la génération d'équivalents pour le verbe « stopper »

Ces mots identifiés comme mots similaires gardent le score de similarité qui leur a été attribuée dans FastText et apparaissent en tête de liste. Les autres ont un score de similarité par défaut de 0.0 et apparaissent en fin de liste.

Les listes de équivalents générées seront soumises à l'évaluation manuelle d'un utilisateur. Cela permettra de ne garder que les mots réellement adaptés au contexte de la phrase.

5 - Récapitulatif

Notre module de data augmentation repose sur une API Rest qui peut servir d'interface auprès de l'utilisateur et permet à notre système de proposer ses services à d'autres systèmes. Afin de l'interroger, nous passons par des requêtes Curl. Comme le montre la figure 34, il existe deux possibilités : générer des commandes alternatives depuis un fichier au format markdown ou depuis une commande seule.

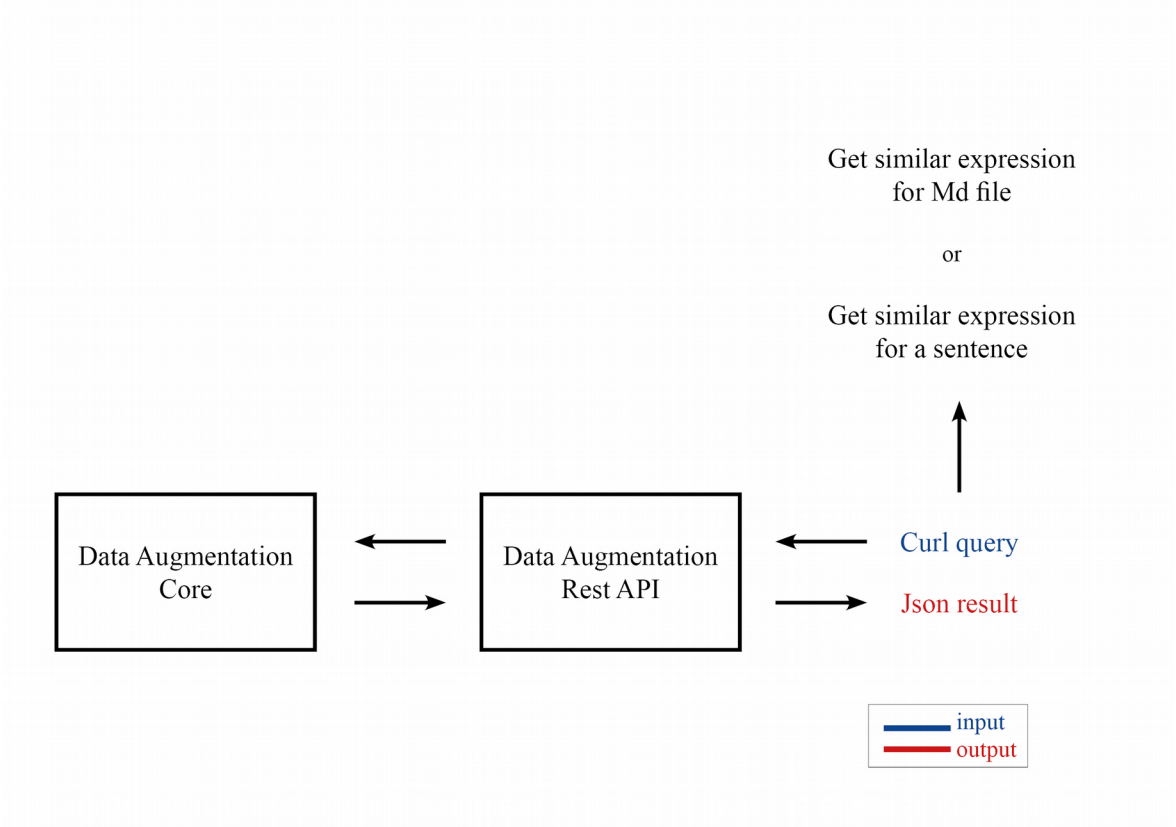

Figure 34: Système de requête via l'API Rest

La figure 35 nous présente un récapitulatif de la chaîne de traitement de notre module.

Le système prend en entrée une commande contenant des entités annotées au format markdown. Il commence par analyser ce texte d'entrée en s'appuyant sur les entités annotées qui s'y trouvent, et à l'analyse syntaxique fournie par CoreNLP et Spacy (*Text Analyzer*). Cette analyse permet de sortir le type de la commande ainsi que les parties du discours essentielles qui s'y trouvent. Ces parties du discours sont prises en entrée par notre générateur de mots équivalents (*Similar Words Generator*) afin de générer des listes de nouveaux mots adaptés au contexte de la phrase.

A partir du type de commande identifié, le module choisit ensuite aléatoirement une règle représentant une structure syntaxique possible pour ce type (*set of rules)*. La grammaire est lue par le parseur de grammaire (*Grammar Parser*). Lorsque c'est nécessaire, le parseur remplace les balises qui composent la règle par les expressions de début de phrase tirées des lexiques que nous avons manuellement établies (*lexicons)*. Il reprend également les parties du discours de la commande d'entrée en les lemmatisant grâce à Glawi. Soit il réutilise ces mots, soit les nouveaux mots générés grâce à ConceptNet et FastText, soit les mots tirés des autres commandes présentes dans le fichier (*Similar Words Generator*).

Il fléchit les parties du discours (*Text Generator*) en respectant la conjugaison des verbes exigée par la règle et la flexion des dépendants par rapport à leur tête. Il gère également les élisions (« de le », « de les » ect.).

La nouvelle commande est finalement générée.

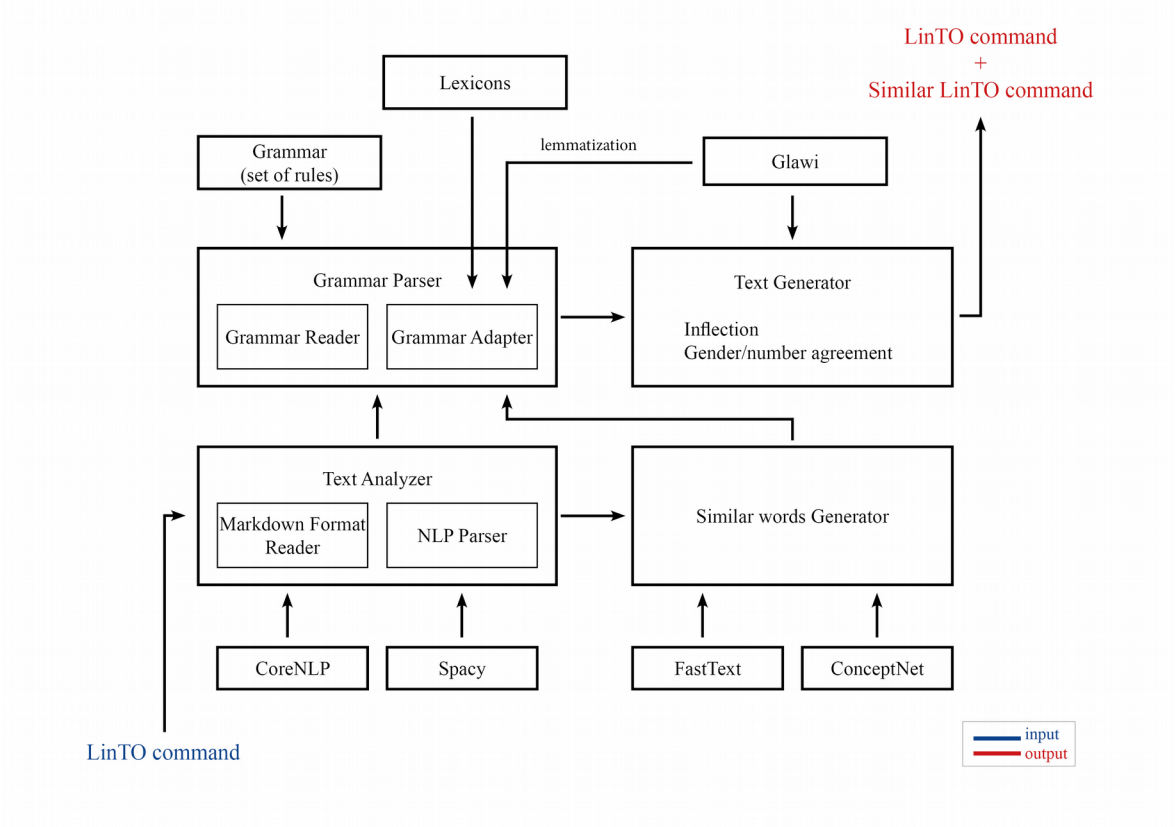

Figure 35: Vue d'ensemble de la chaîne de traitement

Partie 3

-

Résultats obtenus

Dans cette troisième partie, nous allons présenter des exemples de paraphrases générées par notre module : nous testons tour à tour les commandes d'action, les commandes d'informations et leurs sous-catégories, interrogatives partielles et totales.

1 - Analyse qualitative des commandes générées

1.1 - Test des commandes d'action

Dans le fichier de la figure 36 nous plaçons un échantillon de l'intention *chrono* en essayant de varier la composition des phrases en entrée :

```
## intent:chrono
- peux-tu [stopper](action_stop) le chronomètre
- peux-tu [stopper](action_stop) le chrono
- peux-tu [démarrer](action_start) le chronomètre
- peux-tu [démarrer](action_start) le minuteur
- [débuter](action_start) le chronomètre
- [démarre](action_start) le minuteur pour [5](number) [minutes](time_unit)
- [débuter](action_start) le chronomètre pour [45](number) [minutes](time_unit)
- [commence](action_start) le chrono pour [1](number) [heure](time_unit)
- [désactive](action_stop) le chronomètre
- [arrête](action_stop) le minuteur
```

Figure 36: Échantillon de commandes "action" en entrée du module de data augmentation

Sur la figure 37 on peut observer les différentes paraphrases générées dans le cas où les termes équivalents sont récupérés dans le fichier lui-même. Nous voyons ici l'importance de ne pas mêler les instances de différentes entités entre elles : les instances des entités *action_on* et *action_stop* sont antonymes. Leurs équivalents ont été récupérés parmi les autres instances présentes dans l'intention *chrono*. Les termes équivalents qui ne sont pas des instances d'entités ont également été récupérés pour former les paraphrases.

```
## intent:chrono

<!-- Original sentence -->
- peux-tu [stopper](action_stop) le chronomètre
<!-- Alternative Sentences -->
- est-ce que tu peux [arrêter](action_stop) le chronomètre
- [stopper](action_stop) le chronomètre
- [désactiver](action_stop) le chronomètre
- [arrête](action_stop) le minuteur
- est-ce que tu peux [désactiver](action_stop) le minuteur
- [stoppe](action_stop) le chrono
- je veux que tu [stoppes](action_stop) le chronomètre
- [arrête](action_stop) le chronomètre
- est-ce que tu peux [arrêter](action_stop) le minuteur
- est-ce que tu peux [désactiver](action_stop) le chronomètre

<!-- Original sentence -->
- peux-tu [démarrer](action_start) le minuteur
<!-- Alternative Sentences -->
- tu vas [démarrer](action_start) le chronomètre
- je veux que tu [démarres](action_start) le chronomètre
- [démarre](action_start) le chrono
- tu vas [démarrer](action_start) le minuteur
- je veux que tu [débutes](action_start) le minuteur
- est-ce que tu peux [démarrer](action_start) le chronomètre
```

Figure 37: Exemples de paraphrases générées pour les commandes "action" à partir d'équivalents du fichier d'entrée

Sur l'exemple de la figure 38 on voit que les syntagmes prépositionnels ne sont ajoutés aux paraphrases que si la phrase originale en comporte.

```
<!-- Original sentence -->
- [démarre](action_start) le minuteur pour [5](number) [minutes](time_unit)
<!-- Alternative Sentences -->
- tu peux [débuter](action_start) le chronomètre pour [5](number) [minutes](time_unit)
- est-ce que tu peux [commencer](action_start) le chronomètre pour [45](number) [minutes](time_unit)
- peux-tu [débuter](action_start) le chrono pour [1](number) [heure](time_unit)
- [commencer](action_start) le chronomètre pour [1](number) [heure](time_unit)
- [débute](action_start) le chronomètre pour [1](number) [heure](time_unit)
- je veux que tu [débutes](action_start) le minuteur pour [45](number) [minutes](time_unit)
- [commence](action_start) le chronomètre pour [5](number) [minutes](time_unit)
- [débuter](action_start) le chrono pour [45](number) [minutes](time_unit)
- je veux que tu [débutes](action_start) le chrono pour [1](number) [heure](time_unit)
- tu peux [débuter](action_start) le minuteur pour [1](number) [heure](time_unit)
```

Figure 38: Exemples de paraphrases de commandes "action" comportant un syntagme prépositionnel

Maintenant, nous ajoutons des adjectifs aux deux premières commandes :

- « peux-tu stopper le chronomètre bleu »

- « peux-tu stopper le grand chrono »

Sur la figure 39, on constate que l'adjectif est bien pris en compte mais pas sa place dans la phrase :« grand » devrait se trouver avant le nom.

Cela est dû au fait que l'adjectif est placé par défaut derrière le nom. Il faudrait donc garder en mémoire sa place dans la phrase principale et dans les autres phrases d'où sont récupérées les équivalents.

```
<!-- Original sentence -->
- peux-tu [stopper](action_stop) le chronomètre bleu
<!-- Alternative Sentences -->
- est-ce que tu peux [stopper](action_stop) le chronomètre grand
- [stopper](action_stop) le minuteur bleu
- tu vas [arrêter](action_stop) le minuteur bleu
- tu peux [désactiver](action_stop) le minuteur grand
- je veux que tu [stoppes](action_stop) le chronomètre grand
- [arrêter](action_stop) le minuteur grand
- [désactive](action_stop) le minuteur grand
- [stopper](action_stop) le chrono bleu
- tu peux [stopper](action_stop) le chronomètre grand
- je veux que tu [stoppes](action_stop) le chrono grand
```

Figure 39: Exemples de commandes "action" générées avec adjectif

Toujours avec l'échantillon de la figure 36 précédente, nous ajoutons les équivalents tirés de FastText et Concept. Sur la figure 42, nous obtenons de nouveaux équivalents comme « interrompre » et « chronométrage » qui s'adaptent correctement au contexte de la phrase. Cependant, nous avons aussi le verbe « continuer » qui ne correspond pas à l'arrêt d'une action et « cesser » qui ne peut être employé comme transitif que dans des cas particuliers. Ces équivalents seront retirés lors de la phase d'évaluation manuelle, mais il est toujours intéressant de se demander comment traiter ces problématiques de manière automatique. Certains dictionnaires peuvent par exemple indiquer les informations concernant la transitivité des verbes.

```
<!-- Original sentence -->
- peux-tu [arrêter](action_stop) le minuteur
<!-- Alternative Sentences -->
- est-ce que tu peux [continuer](action_stop) le chronomètre
- est-ce que tu peux [stopper](action_stop) le chronométrage
- est-ce que tu peux [continuer](action_stop) le minuteur
- [arrête](action_stop) le chronomètre
- [arrêter](action_stop) le chronomètre
- je veux que tu [interrompes](action_stop) le minuteur
- je veux que tu [stoppes](action_stop) le minuteur
- [continue](action_stop) le chronomètre
- [cesser](action_stop) le minuteur
- est-ce que tu peux [arrêter](action_stop) le chronomètre
```

Figure 40: Exemple de commandes d'action utilisant les équivalents de FastText et ConceptNet

1.2 - Test des commandes d'information – interrogatives partielles

Nous testons ensuite les interrogatives partielles sur un échantillon de l'intention *news* (voir figure 41). Les phrases choisies comportent des adjectifs (« culturelles ») et appositions du nom (« science »), ainsi que différents syntagmes prépositionnels.

```
## intent:news
- quels sont les évènements [musicaux](type_musical) du [moment](time)
- quels sont les articles du [moment](time)

## intent:pollution
- quel est le niveau de pollution à [Paris](location)
- pollution à [Marseille](location)
```

Figure 41: Échantillon de commandes "information" comportant des interrogatives partielles

Les résultats de la génération pour l'intention *news* est présenté sur la figure 42. On peut voir que pour la première commande, presque aucun équivalent issu de FastText et ConceptNet n'a été retenu à part « musique ». Notons que lors de la génération des équivalents, nous avions inclus une fonction permettant de ne garder que les mots

appartenant à la même catégorie grammaticale que le mot source. Cependant, pour ce test nous ne l'avons pas utilisée et nous remarquons que cela n'est pas gênant ici puisque « musique » joue le rôle d'apposition du nom.

Pour la seconde commande, on retrouve le mot « entrefilet » comme équivalent d' « article ». Il s'agit effectivement d'un mot désignant un petit article de journal, mais qui est moins courant dans le langage quotidien.

```
## intent:news

<!-- Original sentence -->
- quels sont les évènements [musicaux](type_musical) du [moment](time)
<!-- Alternative Sentences -->
- quels sont les évènements [musicaux](type_musical) de la [semaine](time)
- les évènements [musique](type_musical) du [moment](time)
- évènements [musicaux](type_musical) de la [semaine](time)
- les articles [musicaux](type_musical) du [moment](time)
- articles [musicaux](type_musical) du [moment](time)
- titres [musicaux](type_musical) du [moment](time)
- tu pourrais trouver les évènements [musicaux](type_musical) de la [semaine](time)

<!-- Original sentence -->
- quels sont les articles du [moment](time)
<!-- Alternative Sentences -->
- entrefilet de la [semaine](time)
- trouve les articles de la [semaine](time)
- merci de trouver les articles du [moment](time)
- articles du [moment](time)
- les évènements du [moment](time)
- quels sont les évènements du [moment](time)
- quels sont les articles de la [semaine](time)
- c'est quoi les évènements du [moment](time)
```

Figure 42: Exemples de paraphrases pour les commandes "information" de l'intention «news» - interrogatives partielles

Sur les résultats de l'intention *pollution* (voir figure 43), on peut observer une plus grande variété d'équivalents (« niveau de pollution » → « degré de pollution »). En revanche, on retrouve également des termes qui s'éloignent du sens du mot source (« contamination » → « pollution » ?). On observe aussi un phénomène récurrent lorsque les commandes d'une même intention ont une structure syntaxique un peu différente : « la pollution de pollution », « niveau à Marseille ». Lorsque le système récupère les noms présents dans les commandes de la même intention, il sépare les noms et les compléments du noms qui les accompagnent.

```
## intent:pollution

<!-- Original sentence -->
- quel est le niveau de pollution à [Paris](location)
<!-- Alternative Sentences -->
- quel est le plan de pollution à [Paris](location)
- degré de pollution à [Paris](location)
- tu peux chercher le niveau de pollution à [Paris](location)
- tu pourrais donner la pollution de pollution à [Paris](location)
- le niveau de pollution à [Paris](location)
- niveau de pollution à [Paris](location)

<!-- Original sentence -->
- pollution à [Marseille](location)
<!-- Alternative Sentences -->
- donne-moi la contamination à [Marseille](location)
- quel est le niveau à [Marseille](location)
- le niveau à [Marseille](location)
- contamination à [Marseille](location)
- le polluant à [Marseille](location)
- donne-moi la pollution à [Marseille](location)
- quelle est la pollution à [Marseille](location)
- trouve le niveau à [Marseille](location)
```

Figure 43: Exemples de commandes "informations" générées pour l'intention "pollution"

1.3 - Test des commandes d'information – interrogatives totales

Pour tester les interrogatives totales nous avons choisi l'échantillon présenté sur la figure 44. La gestion des équivalents y est plus délicate. Pour ces tests, nous avons donc gardé les mots de la phrase originale afin d'illustrer les changements syntaxiques qui s'opèrent pour ce type de commandes.

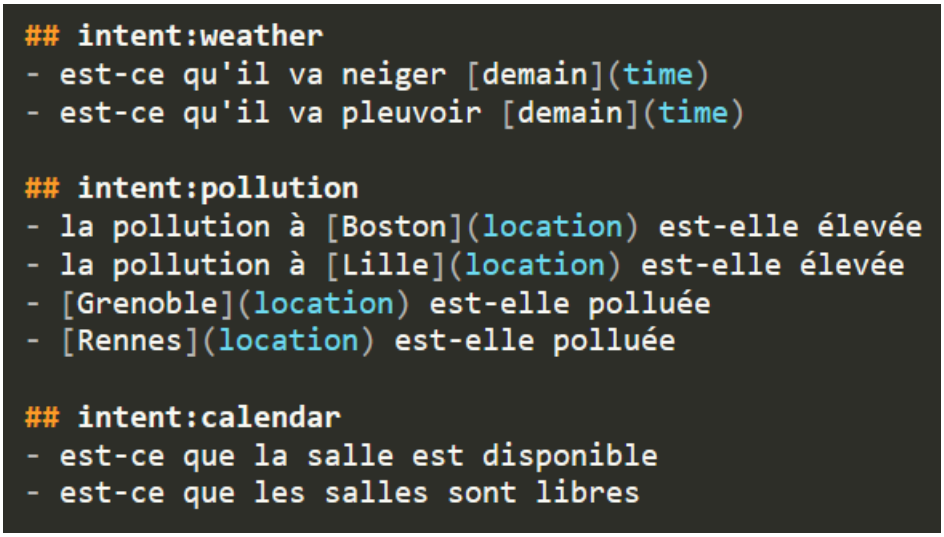

Figure 44: Échantillon de commandes "information" comportant des interrogatives totales

La figure 45 présente les résultats de quelques phrases : on remarque que lorsqu'il s'agit des interrogatives totales, notre module n'utilise pas le système de synonymie entre phrases d'une même intention. Les mots de la phrase originale sont réutilisées et seules les combinaisons syntaxiques de la commande changent. Les éléments qui composent les interrogatives partielles d'une même intention sont permutables, mais ils ne sont pas compatibles avec les éléments composant les interrogatives totales appartenant à la même intention.

Par exemple, dans l'intention *weather* de notre corpus original, nous pouvons trouver « c'est quoi la météo à Toulouse » (interrogative partielle) et « est-ce qu'il va neiger » (interrogative totale). La relation de synonymie entre les interrogatives partielles et totale est délicate.

Dans ce cas, nous pourrions penser à permuter les éléments des interrogatives partielles entre elles, et les éléments des interrogatives totales entre elles. Cependant, si les éléments des interrogatives partielles présents dans notre corpus sont, pour la grande majorité, substituables entre eux (au sein de la même intention), ceux des interrogatives totales le sont beaucoup moins. Par exemple, pour des interrogatives totales appartenant à l'intention *pollution*, nous pouvons trouver « est-ce que l'air est respirable » et « la pollution à Boston est-elle élevée ». Bien que ces deux phrases appartiennent à la même intention et soient toutes les deux des interrogatives totales, leurs éléments sont difficilement substituables.

Les équivalents qui seront utilisés pour générer les paraphrases des interrogatives totales viendront des ressources extérieures qui sont actuellement utilisés pour les commandes d'action et les interrogatives partielles.

```
## intent:weather
<!-- Original sentence -->
- est-ce qu'il va neiger [demain](time)
<!-- Alternative Sentences -->
- va-t-il neiger [demain](time)
- il va neiger [demain](time)
- tu peux me dire s'il va neiger [demain](time)
- je veux savoir s'il va neiger [demain](time)

## intent:pollution

<!-- Original sentence -->
- la pollution à [Lille](location) est-elle élevée
<!-- Alternative Sentences -->
- dis-moi si la pollution à [Lille](location) est élevée
- peux-tu me dire si la pollution à [Lille](location) est élevée
- est-ce que la pollution à [Lille](location) est élevée
- je veux savoir si la pollution à [Lille](location) est élevée
- la pollution à [Lille](location) est élevée

<!-- Original sentence -->
- [Grenoble](location) est-elle polluée
<!-- Alternative Sentences -->
- est-ce que [Grenoble](location) est pollué
- je veux savoir si [Grenoble](location) est pollué
- dis-moi si [Grenoble](location) est pollué
- tu peux me dire si [Grenoble](location) est pollué

## intent:calendar

<!-- Original sentence -->
- est-ce que la salle est disponible
<!-- Alternative Sentences -->
- la salle est disponible
- dis-moi si la salle est disponible
- la salle est-elle disponible
- peux-tu me dire si la salle est disponible
```

Figure 45: Exemples de paraphrases pour les commandes "information" - interrogatives totales

2 - Analyse quantitative des résultats

Afin d'évaluer quantitativement nos résultats, nous avons lancé notre module sur l'ensemble de notre corpus de référence puis nous avons sondé des échantillons représentatifs. La figure 46 présente le nombre de commandes valides sur le nombre de commandes sondées pour un échantillon donné.

Commandes d'action	Total = **69**	Commandes d'information	Total = **70**
Impératives	49	**Interrogatives partielles**	39
Avec une structure simple (verbe + nom)	10 / 10	Avec une structure simple (adjectif interrogatif + auxiliaire + nom)	7/10
Contenant des expressions polylexicales (« prévisions météo »)	0 / 9	Contenant des expressions polylexicales	8/9
Contenant un syntagme prépositionnel (« pour 45 minutes »)	10 / 10	Contenant un syntagme prépositionnel	8/10
Contenant une subordonnée (« que j'ai un repas d'affaire vendredi midi »)	0 / 10	Sans entités annotées	4/10
Contenant un nom propre (« ajoute Michaël à la visioconférence »)	4 / 10		
		Interrogatives totales	11/11
Interrogatives	20/20	**Phrases nominales**	14/20
Commandes acceptables	**44 / 69**	Commandes acceptables	**52 / 70**

Figure 46: Tableau récapitulatif des résultats des échantillons de commandes sondés

En observant 139 phrases générées par notre module, nous obtenons 96 commandes valides, soit une précision de **69 %**. Une commande est considérée comme valide lorsque la syntaxe, la conjugaison et les accords en genre/nombre y sont respectés. Il faut

également qu'elle puisse assurer le rôle d'équivalent fonctionnel par rapport à la commande d'origine.

Pour conclure cette partie, nous pouvons dire que la gestion des termes équivalents entre phrases d'une même intention est assez bonne, mais elle peut aussi rencontrer des problèmes lorsque ces commandes ont des structures syntaxiques plus éloignées ou que les commandes contiennent des expressions polylexicales. L'utilisation des synonymes issus de FastText et ConceptNet donne également des résultats intéressants, mais la validation manuelle des listes est nécessaire pour gérer les catégories grammaticales, la transitivité des verbes et les mots peu courants.

Conclusion

3 - Bilan et perspectives

3.1 - Prise de recul

Pour ce projet, nous avons essayé d'identifier la structure syntaxique de notre commande d'entrée afin de lui appliquer un traitement adapté. Au sein de notre corpus, les commandes peuvent être divisées en deux grandes catégories selon le type de requête (l'exécution d'une action ou la demande d'information) et en sous catégories (interrogatives partielles, interrogatives totales ...). Les commandes partageant le même type de commande partagent plusieurs structures syntaxiques identiques dans lesquelles les parties du discours porteuses du sens principal de la phrase sont les mêmes (nom pour les commandes d'information et verbe pour les commandes d'action). Notre approche à base de règles nous permet de contrôler plus finement nos données, mais c'est aussi une méthode qui requiert un important travail d'analyse et de développement selon les sous-catégories de structures syntaxiques présentes dans le corpus.

Grâce aux travaux de Pohl (1965), Söll (1983) et Behnstedt (1973), nous avons pu identifier les structures principales des différents types de phrases interrogatives. Cependant, ces catégories ne présentent que des schémas syntaxiques de surface, et comme nous allons le voir dans les améliorations possibles, plusieurs phrases peuvent correspondre au même schéma, sans pour autant pouvoir être paraphrasées de la même manière.

3.2 - Améliorations possibles

Le traitement des paraphrases dépend surtout du corpus que l'on traite. Pour ce projet, nous avions un échantillon de 676 phrases dans lequel une structure syntaxique était dominante (expressions interrogatives – verbe – nom). Cependant, il faut aussi prendre en compte le traitement de structures syntaxiques identiques en entrée mais au comportement différent lors du paraphrasage. Par exemple « quelle est la météo » et « qui sont les intervenants à la réunion » suivent le même schéma syntaxique de surface, mais nous ne pouvons pas les paraphraser de la même manière.

Il faudrait également modifier la récupération des parties du discours afin de prendre en compte l'index des mots dans la phrase d'entrée. Cela pourrait être utile pour les syntagmes prépositionnels rattachés à un nom et qui ne se trouveraient pas en fin de phrase (exemple possible : « quel sont les films à l'affiche le mois prochain »). Pour l'instant, seul leur ordre d'apparition par rapport aux autres syntagmes prépositionnels est retenu, et non pas leur ordre d'apparition dans la phrase.

Nous sommes également confrontés à la problématique des expressions polylexicales : pour l'instant, les noms, adjectifs, appositions et compléments du nom sont récupérés séparément, et la génération de leurs équivalents se fait donc individuellement. Nous avons pu configurer les outils que nous utilisons actuellement (ConceptNet et FastText) pour générer des expressions contenant plusieurs termes. Cependant, il existe plusieurs types d'expressions polylexicales (locutions, collocations) et lors de la récupération des termes originaux dans la commande en entrée, notre module ne peut pas différencier les locutions ou collocations « bande originale » (qui se comportent comme un seul mot) des cooccurrences « sujet de la réunion » (dont les termes peuvent être pris indépendamment et remplacés par des équivalents : « thème de la réunion », « agenda de la réunion »). Plusieurs solutions pourraient être envisagées :

- Nous pourrions représenter ces termes comme un ensemble en les identifiant comme l'instance d'une même entité dans Tock.

- Nous pourrions envisager d'utiliser un dictionnaire de collocations open-source. Glawi propose déjà quelques locutions, mais sa couverture n'est pas suffisamment large.

Une autre amélioration possible concerne le corpus de référence : il faudrait préciser certaines entités afin de pouvoir récupérer leurs instances dans l'ensemble du fichier. Cela permettrait de les rendre substituables entre elles (exemple : *action_on* → « allumer », « ouvre » ne se réfèrent pas à la même action.).

Pour conclure ce mémoire, nous pouvons dire qu'il est important de garder en tête le lien de dépendance qui existe entre la structure syntaxique des phrases en entrée , et les combinaisons syntaxiques possibles en sortie du module. C'est cette étape qui est essentielle dans le mécanisme de paraphrasage.

4 - Bilan personnel

Ce stage a été une expérience très enrichissante, il m'a permis de gagner en autonomie et d'apprendre à gérer un projet depuis l'analyse du sujet jusqu'à son développement. La principale difficulté a été de proposer une solution à une problématique large dans un délai bien défini. Bien que de petite taille, notre corpus est représentatif des particularités et exceptions qui existent dans la langue française.

Il aussi été difficile de jongler entre linguistique et informatique, mais c'est ce qui nous amène aussi à mieux comprendre les mécanismes de la langue que l'on traite.

Bibliographie

Barzilay, R. et McKeown, K. R. (2001). Extracting Paraphrases from a Parallel Corpus. Dans *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics* (2001), Toulouse, France.

Behnstedt, P., (1973), Viens-tu? Est-ce que tu viens? Tu viens? : Formen und Strukturen des direkten Fragesatzes im Französischen, Tübingen : G. Narr

Callison-Burch, C. (2007). Paraphrasing and Translation. Thèse de doctorat, University of Édimbourg.

Fujita, A. (2005). Automatic Generation of Syntactically Well-formed and Semantically Appropriate Paraphrases. Thèse de doctorat, Nara Institute of Science and Technology, Japon

Kampeera Wannachai. (2013). Analyse linguistique et formalisation pour le traitement automatique de la paraphrase. Thèse de doctorat, Université de Franche-Comté.

Madnani, N. (2010). The Circle of Meaning : From Translation to Paraphrasing and Back. Thèse de doctorat, University of Maryland, College Park, États-Unis.

Pohl, J., (1965), « Observations sur les formes d'interrogation dans la langue parlée et dans la langue écrite non littéraire », *Linguistique et philologie romanes : Xe Congrès international de linguistique et philologie romane* (1962), Paris : Librairie Klincksieck, tome II, 4e partie

Robin Jia and Percy Liang. (2016). Data recombination for neural semantic parsing. Dans *Proceedings of ACL*.

Shashi Narayan, Siva Reddy, and Shay B Cohen. (2016). Paraphrase generation from latent-variable PCFGs for semantic parsing. arXiv:1601.06068 [cs.CL].

Söll, L., (1983), « L'interrogation directe dans un corpus de langage enfantin », *Etudes de grammaire française descriptive*, F.-J. Hausmann (éd.), Heidelberg : J. Groos, Studies in descriptive linguistics

Zhao, S., Lan, X., Liu, T. et Li, S. (2009). Application-driven Statistical Paraphrase Generation. Dans *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, Suntec, Singapour.

Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. (2017). Paraphrase generation with deep reinforcement learning. arXiv preprint arXiv:1711.00279 .

Zumwald G., (Juin 2010). Traiter la diversité des formes interrogatives en français : apports et limites de l'approche variationniste.

Sitographie

Corno Stefano, « Autour de la relation tête-dépendant dans les langues indo-européennes anciennes. Typologie et reconstruction », *Éducation et sociétés plurilingues* [En ligne], 40 | 2016, mis en ligne le 28 octobre 2016, consulté le 24 juillet 2019. URL : http://journals.openedition.org/esp/886 (consulté le 01/08/2019)

Coveney Aidan, « L'interrogation directe », Travaux de linguistique, 2011/2 (n°63), p. 112 – 145. DOI : 10.3917/tl.063.0112. URL : https://www.cairn.info/revue-travaux-de-linguistique-2011-2-page-112.htm (consulté le 17/06/2019)

Table des illustrations

Table des annexes

Annexe 1
Règles pour la génération de commandes d'information

Sur la capture d'écran ci-dessous, nous avons un aperçu des règles utilisées pour traiter les commandes d'information qui apparaissent sous la forme d'interrogatives partielles et d'interrogatives totales.

```
# Règles interrogatives partielles

# c'est quoi la température
'<interro_sn> <det> <noun>'
# quelle est la température
'<interro_determinant> <verb_auxiliary> <det> <noun>'
# pourrais-tu montrer la température
'<interro_infinitive> <verb_demo> <det> <noun>'
# température
'<noun>'
# la température
'<det> <noun>'

# Règles interrogatives totales

#est-ce que la salle est libre
'est-ce que <sentence>'
# dis-moi si la salle est libre
'<interro_if> <sentence>'
# la salle est libre
'<sentence>'
#la salle est-elle libre
'<sentence_inverted_subj>'
```

Annexe 2
Règles pour la génération de commandes d'action

```
# tu pourrais allumer la lumière
'<interro_infinitive> <verb> <det> <noun>'
# est-ce que tu pourrais allumer la lumière
'<interro_adverb> <post_interro_ability> <verb> <det> <noun>'
# allume la lumière
'<verb_tense_impe> <det> <noun>'
# allumer la lumière
'<verb> <det> <noun>'
# j'aimerais que tu allumes la lumière
'<interro_subj> <verb_tense_subj> <det> <noun>'
```

Annexe 3
Lexiques d'expressions de début de phrase

interro_adverb	est-ce que
interro_infinitive	peux-tu tu peux merci de tu vas
interro_subj	je veux que
post_interro_ability	tu peux
interro_sn	c'est quoi trouve cherche donne-moi j'ai besoin de je veux connaître je veux avoir je veux
verb_demo	fournir chercher trouver donner

Annexe 4
Liste des relations proposées par ConceptNet

Relation URI	Description	Examples
/r/RelatedTo	The most general relation. There is some positive relationship between A and B, but ConceptNet can't determine what that relationship is based on the data. This was called "ConceptuallyRelatedTo" in ConceptNet 2 through 4. Symmetric.	learn ↔ erudition
/r/FormOf	A is an inflected form of B; B is the root word of A.	slept → sleep
/r/IsA	A is a subtype or a specific instance of B; every A is a B. This can include specific instances; the distinction between subtypes and instances is often blurry in language. This is the *hyponym* relation in WordNet.	car → vehicle; Chicago → city
/r/PartOf	A is a part of B. This is the *part meronym* relation in WordNet.	gearshift → car
/r/HasA	B belongs to A, either as an inherent part or due to a social construct of possession. HasA is often the reverse of PartOf.	bird → wing; pen → ink
/r/UsedFor	A is used for B; the purpose of A is B.	bridge → cross water
/r/CapableOf	Something that A can typically do is B.	knife → cut
/r/AtLocation	A is a typical location for B, or A is the inherent location of B. Some instances of this would be considered meronyms in WordNet.	butter → refrigerator; Boston → Massachusetts
/r/Causes	A and B are events, and it is typical for A to cause B.	exercise → sweat
/r/HasSubevent	A and B are events, and B happens as a subevent of A.	eating → chewing
/r/HasFirstSubevent	A is an event that begins with subevent B.	sleep → close eyes
/r/HasLastSubevent	A is an event that concludes with subevent B.	cook → clean up kitchen
/r/HasPrerequisite	In order for A to happen, B needs to happen; B is a dependency of A.	dream → sleep
/r/HasProperty	A has B as a property; A can be described as B.	ice → cold

/r/MotivatedByGoal	Someone does A because they want result B; A is a step toward accomplishing the goal B.	compete → win
/r/ObstructedBy	A is a goal that can be prevented by B; B is an obstacle in the way of A.	sleep → noise
/r/Desires	A is a conscious entity that typically wants B. Many assertions of this type use the appropriate language's word for "person" as A.	person → love
/r/CreatedBy	B is a process or agent that creates A.	cake → bake
/r/Synonym	A and B have very similar meanings. They may be translations of each other in different languages. This is the *synonym* relation in WordNet as well. Symmetric.	sunlight ↔ sunshine
/r/Antonym	A and B are opposites in some relevant way, such as being opposite ends of a scale, or fundamentally similar things with a key difference between them. Counterintuitively, two concepts must be quite similar before people consider them antonyms. This is the *antonym* relation in WordNet as well. Symmetric.	black ↔ white; hot ↔ cold
/r/DistinctFrom	A and B are distinct member of a set; something that is A is not B. Symmetric.	red ↔ blue; August ↔ September
/r/DerivedFrom	A is a word or phrase that appears within B and contributes to B's meaning.	pocketbook → book
/r/SymbolOf	A symbolically represents B.	red → fervor
/r/DefinedAs	A and B overlap considerably in meaning, and B is a more explanatory version of A.	peace → absence of war
/r/MannerOf	A is a specific way to do B. Similar to "IsA", but for verbs.	auction → sale
/r/LocatedNear	A and B are typically found near each other. Symmetric.	chair ↔ table
/r/HasContext	A is a word used in the context of B, which could be a topic area, technical field, or regional dialect.	astern → ship; arvo → Australia
/r/SimilarTo	A is similar to B. Symmetric.	mixer ↔ food processor
/r/EtymologicallyRelatedTo	A and B have a common origin. Symmetric.	folkmusiikki ↔ folk music
/r/EtymologicallyDerivedFrom	A is derived from B.	dejta → date
/r/CausesDesire	A makes someone want B.	having no food → go to a store
/r/MadeOf	A is made of B.	bottle → plastic
/r/ReceivesAction	B can be done to A.	button → push
/r/ExternalURL	Instead of relating to ConceptNet nodes, this pseudo-relation points to a URL outside of ConceptNet, where further Linked Data about this term can be found. Similar to RDF's `seeAlso` relation.	knowledge → http://dbpedia.org/page/Knowledge

Table des matières

MOTS-CLÉS : data augmentation, paraphrase, traitement automatique de la langue, syntaxe, sémantique

RÉSUMÉ

Ce document résume 6 mois de travail au sein de l'équipe Recherche et Développement de Linagora Toulouse. Le but de ce projet était de développer un module de paraphrase permettant d'enrichir le corpus d'apprentissage de l'agent conversationnel LinTO. Nous avons commencé par analyser les différentes commandes présentes dans notre corpus initial : les structures syntaxiques récurrentes et les mécanismes de paraphrasage qu'on peut leur appliquer. A partir de ces observations, nous avons créé une grammaire à base de règles pour générer plusieurs paraphrases d'une commande en entrée.

KEYWORDS : data augmentation, paraphrase, natural langage processing, syntax, semantics

ABSTRACT

This paper summarizes the work completed during a six-month internship in the Research & Development team of Linagora. The aim of this project is to develop a paraphrasing tool able to expand the training datasets of the smart vocal assistant LinTO. We started by analyzing the existing commands in the original corpus : the recurrent syntactic structures and the paraphrasing mecanisms that can be applied to them. From these observations, we created a rule-based grammar to generate semantically and synctactically correct sentences.