

SORBONNE UNIVERSITÉ

RAPPORT

PLDAC 2019

---

# Interrogation SPARQL/RDF en Langage Naturel

---



*Auteurs :*

Dorian QUABOUL  
Wenshan WU

*Encadrants :*

Bernd AMANN  
Youry KHMELEVSKY



# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte . . . . .	3
1.2	Présentation des objectifs . . . . .	3
1.3	Présentation du plan . . . . .	4
<b>2</b>	<b>État de l’art</b>	<b>5</b>
<b>3</b>	<b>Un outil existant : NLQuery</b>	<b>7</b>
3.1	Présentation des outils du Web sémantique . . . . .	7
3.1.1	Ressource Description Framework . . . . .	7
3.1.2	WikiData . . . . .	8
3.2	Architecture logicielle . . . . .	9
3.3	Fonctionnement . . . . .	10
3.3.1	Démarrage de l’application . . . . .	10
3.3.2	Formulation des questions types . . . . .	12
3.3.3	Parser . . . . .	13
3.3.4	Matcher . . . . .	16
3.3.5	Requête SPARQL . . . . .	17
3.4	Limites . . . . .	18
<b>4</b>	<b>Notre approche : SPARQuery</b>	<b>19</b>
4.1	Motivations et objectifs . . . . .	19
4.2	Corrections et améliorations de NLQuery . . . . .	20
4.2.1	Ajouts de nouveaux types de questions . . . . .	20
4.2.2	Consistance des résultats . . . . .	26
4.3	Phase d’analyse . . . . .	28
4.3.1	Diagramme de cas d’utilisation . . . . .	28
4.3.2	Diagramme de package . . . . .	29
4.4	Architecture logicielle . . . . .	30
4.5	Phase de conception . . . . .	32
4.5.1	Diagramme de classes . . . . .	32
4.5.2	Diagramme de séquence . . . . .	34
4.6	Outils et langage utilisé . . . . .	36
4.7	Démonstration des fonctionnalités . . . . .	36

<b>5</b>	<b>Évaluation et comparaisons</b>	<b>49</b>
5.1	Benchmark sur les <i>wh-questions</i> . . . . .	49
5.2	Benchmark sur les <i>yes/no questions</i> . . . . .	51
5.3	Benchmark sur les <i>order questions</i> . . . . .	52
<b>6</b>	<b>Conclusion et perspectives</b>	<b>53</b>

# 1 Introduction

## 1.1 Contexte

Suite au développement des graphes RDF ainsi que de l'évolution des bases de connaissances, il est devenu difficile pour un utilisateur d'appréhender la complexité de ces nouveaux savoirs et donc de pouvoir formuler correctement une requête en langage SPARQL. L'utilisation de ce dernier exige non seulement la maîtrise de sa syntaxe et du modèle de données RDF, mais également la connaissance du vocabulaire exact des bases de connaissances et des URI utilisées pour désigner des entités. Tous ces aspects de données liées, de savoir sur les triplets RDF nous amènent à développer un système de traduction du langage naturel au langage SPARQL ainsi qu'un système d'interface permettant de résoudre le problème en donnant aux utilisateurs la possibilité d'exprimer leurs requêtes en langage naturel et en traduisant ces expressions en requêtes SPARQL.

## 1.2 Présentation des objectifs

L'objectif de ce projet est d'étudier les solutions et outils existants et de réaliser un prototype qui permet d'interroger différents endpoints SPARQL (Wikidata, Yago, DBPedia) par des expressions en langage naturel.

Voici quelques exemples d'expressions :

- Comment s'appelle la femme d'Emmanuel Macron ?
- Quelle est la population de la France en 2019 ?
- Quels pays ont plus de 30 millions d'habitants ?

Le travail à faire repose sur plusieurs grandes étapes, la première concerne une étude de l'état de l'art en nous basant sur des articles scientifiques ainsi que les outils existants, la suivante sur la spécification et réalisation de notre prototype et pour finir, réaliser une expérimentation en interrogeant une ou plusieurs bases de connaissances et être capable d'évaluer la performance de notre prototype.

### 1.3 Présentation du plan

Notre plan se scinde en plusieurs parties, dans un premier temps nous allons vous faire l'état de l'art des solutions existantes en présentant de façon non exhaustive plusieurs approches qui ont été mises en oeuvre dans l'élaboration d'un système de traduction. Après avoir listé ces différentes solutions, nous allons nous focaliser sur le projet NLQuery [14] en vous faisant une explication plus détaillée. Pour la suite, nous vous présenterons notre travail réalisé qui fut basé sur NLQuery [14] en nous appuyant sur notre compréhension du logiciel, son architecture, ses limites ainsi que sur les améliorations apportées. Et pour finir, une étude comparative sera effectuée sous la forme d'un benchmark ainsi que la conclusion apportée.

## 2 État de l'art

Compte tenu de la grande quantité d'information issue du web sémantique et de la nécessité de pouvoir y accéder facilement sans pour autant maîtriser le langage SPARQL, de nombreuses approches ont été proposées.

La plupart d'entre elles ont été décrites et évaluées sur des bases de données telles que (QALD [1], WebQuestions et SimpleQuestions [4]) dans l'étude [6]. Cette étude a permis d'identifier quatre grandes étapes communes à ces différents systèmes : (1) l'analyse de la question, (2) la mise en cohérence entre les données extraites de la question et celles de la base de connaissances (KB), (3) la désambiguïsation et enfin (4) la construction de la requête SPARQL.

Ces systèmes sont capables de trouver dans la base de connaissance (KB) l'information demandée par l'utilisateur en utilisant le langage naturel. RDF étant le format de stockage pour les bases de connaissance, la traduction du langage naturel vers le langage de requête SPARQL est primordiale.

Le bilan de cette étude montre qu'il est pratiquement impossible d'évaluer ces techniques individuellement car les systèmes évalués sont une combinaison de plusieurs composants. L'idéale serait de comparer des combinaisons de différentes techniques pour avoir une meilleure idée sur la performance d'un système.

Malgré ce bilan, des techniques se révèlent indispensables et doivent être intégrées dans notre prototype afin de le rendre plus robuste. Pour l'étape (1), les techniques les plus courantes sont celles de reconnaissance d'entité (*Named-Entity Recognition*), de segmentation utilisant le POS-tagging et d'identification de dépendances entre les mots. Dans l'étape (2), il y a des techniques qui consistent à chercher dans les labels (et leurs alias) de la base de connaissance et rechercher les correspondances. La technique la plus utilisée dans l'étape (3) est la *local disambiguation* qui vise à classer selon deux critères : la similarité des labels avec la phrase et la consistance avec le domaine (champ lexical). Pour finir, la construction de la requête se déduit grâce à la phase (1) ou grâce à des modèles (*templates*) de requêtes SPARQL.

Nous nous sommes intéressés à des outils répondants à la problématique posée. L'outil web [7] est multilingue et permet d'interconnecter différentes bases de connaissances. Pour la question de la désambiguïsation, il se base sur le *feedback* utilisateur. Il demande à l'utilisateur si la réponse est pertinente ou non.

Un autre outil [2] où la transformation du langage naturel à SPARQL est faite par des expressions régulières et arrive à donner un sens à la question avec des relations sémantiques.

Dans [11], une démonstration de l'outil *CANaLI* est faite. Il est très innovant car il est capable d'évaluer en temps réel ce que tape l'utilisateur à l'aide d'un système d'auto-complétion affichant des suggestions. Ainsi, cela assure que l'utilisateur ne tape que des questions sémantiquement correctes et que le système les interprète sans ambiguïté.

Le système LODQA [13] repose sur trois modules : le premier donne une représentation sous forme graphique des termes de la question, le second permet de trouver les URI des termes dans la base de connaissances et enfin le dernier module génère la requête SPARQL.

Certains outils ont préféré changer la forme de la question posée par l'utilisateur comme l'outil SQUALL [8] qui utilise une syntaxe qui s'apparente au langage naturel mais qui contient en réalité des variables SPARQL ou encore dans [10] qui se base sur des requêtes par "mots-clés" et une recherche approfondie sur les graphes RDF résultants.

Pour conclure, une étude [5] a été menée et propose un outil d'évaluation des systèmes QA (Question Answering). Il donne une évaluation plus fine et plus détaillée sur les performances d'un système.

## 3 Un outil existant : NLQuery

### 3.1 Présentation des outils du Web sémantique

#### 3.1.1 Ressource Description Framework

**Resource Description Framework (RDF)**[3] est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de telles descriptions.

Un document structuré en RDF est un ensemble de triplets, ce **triplet RDF** est une association (*sujet,prédicat,object*) :

- Le *sujet* représente la ressource à décrire
- Le *prédicat* représente un type de propriété applicable à cette ressource
- L'*objet* représente une donnée ou une autre ressource : c'est la valeur de la propriété

Le sujet, et l'objet dans le cas où c'est une ressource, peuvent être identifiés par un URI ou être des nœuds anonymes. Le prédicat est nécessairement identifié par un URI.

Afin de pouvoir manipuler ces données RDF, nous faisons appel à **SPARQL** qui est un langage de requête et un protocole qui permet de rechercher, d'ajouter, de modifier ou de supprimer des données RDF disponibles à travers Internet.



### 3.1.2 WikiData

**WikiData** est une base de données libre, collaborative, multilingue et secondaire collectant des données structurées.

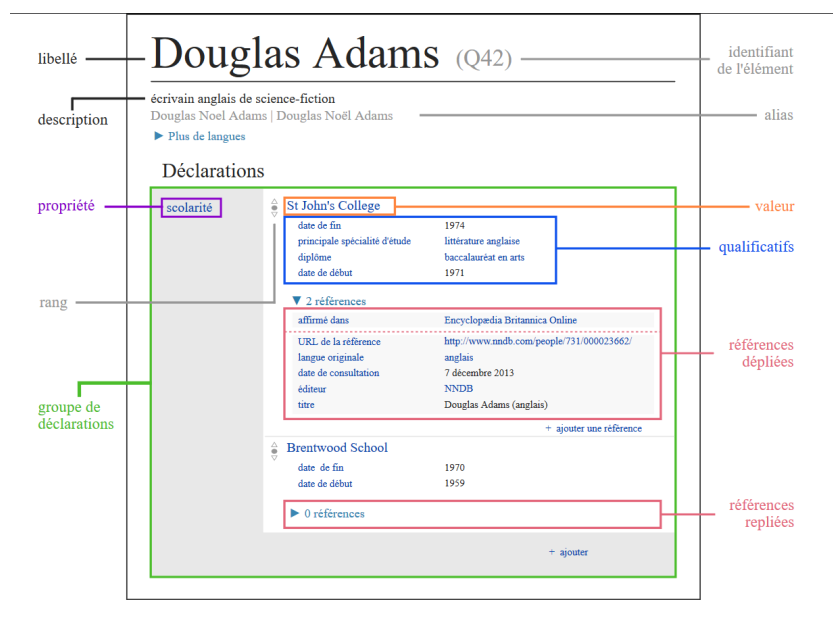


FIGURE 1 – Modèle de données dans WikiData

Voici une partie des informations que l'on peut obtenir via WikiData. Ici, le *sujet* est représenté par *Douglas Adams* ou bien par l'id *Q42*. La propriété affichée est *scolarité* et l'objet renvoyé est *St John's college*.

## 3.2 Architecture logicielle

L'architecture logicielle de NLQuery ainsi que son *workflow* sont présentés dans le schéma ci-dessous.

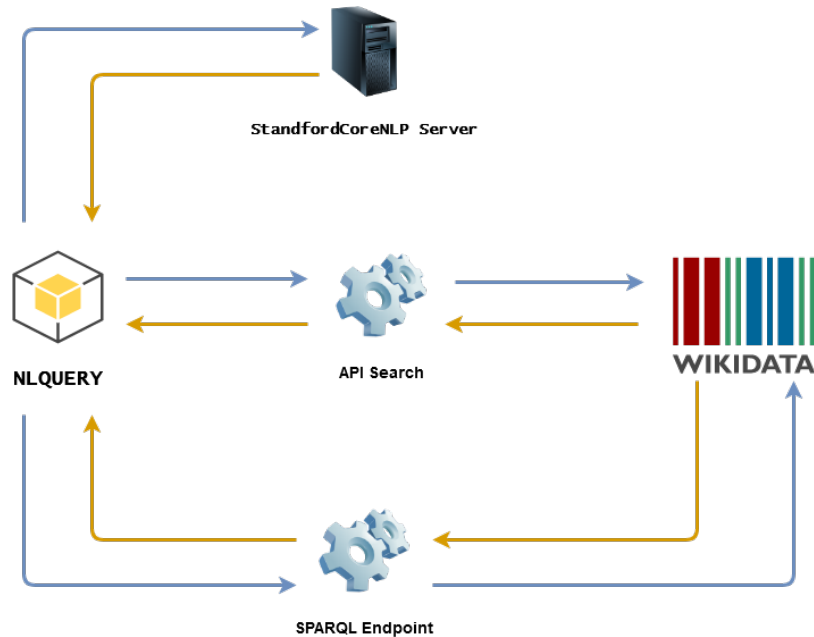


FIGURE 2 – Schéma représentant l'architecture logicielle de NLQuery

NLQuery fait appel à trois modules pour répondre à la question de l'utilisateur :

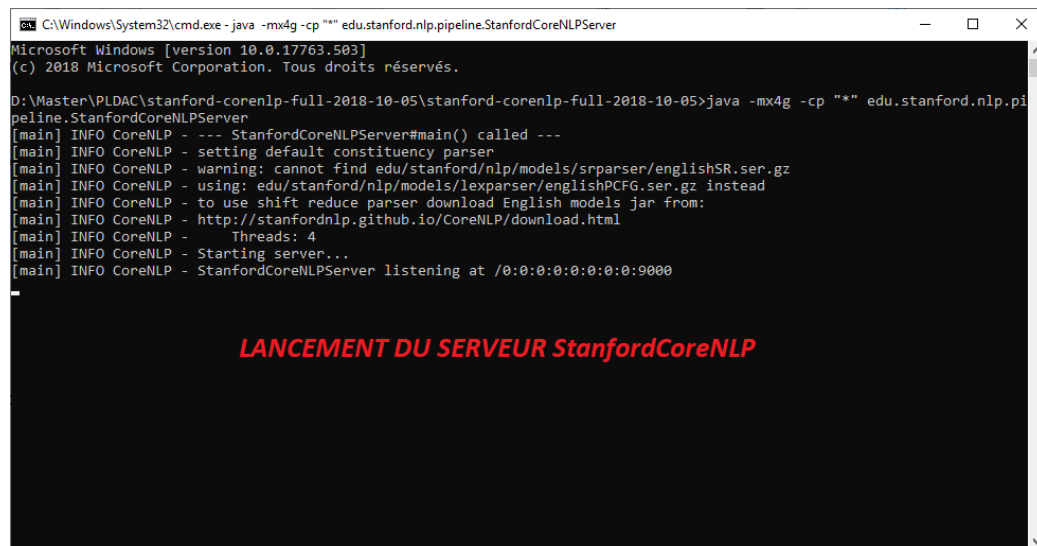
- Le serveur StanfordCoreNLP
- Un endpoint SPARQL
- Une api de recherche

La question de l'utilisateur passe par différents processus. Tout d'abord, elle est envoyée au serveur StanfordCoreNLP sous sa forme brute puis est retournée sous sa forme syntaxique (arbre POS tagging). Ensuite via une comparaison entre la forme syntaxique et des règles définies, NLQuery extrait les éléments importants de la question (par exemple un sujet et une propriété). NLQuery interroge une api qui va chercher dans Wikidata l'identifiant de chaque élément. NLQuery construit la requête SPARQL en utilisant les identifiants trouvés et envoie la requête à un endpoint SPARQL qui va se charger de l'exécuter. Finalement, le résultat est retourné à l'utilisateur.

## 3.3 Fonctionnement

### 3.3.1 Démarrage de l'application

Avant de lancer l'application NLQuery, il faut s'assurer que le serveur StanfordCoreNLP soit démarré. On le démarre en exécutant cette commande `java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer` dans le dossier du serveur.



```
C:\Windows\System32\cmd.exe - java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer
Microsoft Windows [version 10.0.17763.503]
(c) 2018 Microsoft Corporation. Tous droits réservés.

D:\Master\PLDAC\stanford-corenlp-full-2018-10-05\stanford-corenlp-full-2018-10-05>java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer
[main] INFO CoreNLP - --- StanfordCoreNLPServer#main() called ---
[main] INFO CoreNLP - setting default constituency parser
[main] INFO CoreNLP - warning: cannot find edu/stanford/nlp/models/srparser/englishSR.ser.gz
[main] INFO CoreNLP - using: edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz instead
[main] INFO CoreNLP - to use shift reduce parser download English models jar from:
[main] INFO CoreNLP - http://stanfordnlp.github.io/CoreNLP/download.html
[main] INFO CoreNLP - Threads: 4
[main] INFO CoreNLP - Starting server...
[main] INFO CoreNLP - StanfordCoreNLPServer listening at /0:0:0:0:0:0:0:0:9000

LANCEMENT DU SERVEUR StanfordCoreNLP
```

FIGURE 3 – Démarrage du serveur StanfordCoreNLP

```

C:\Windows\System32\cmd.exe - python27 main.py

D:\Master\PLDAC\nlquery_repository>python27 main.py Lancement de l'application (1)
Enter line: Who did Obama marry On pose la question (2)
[None, <nlquery.wikidata.WikiDataAnswer object at 0x0000000008D7E588>]
{'plain': u'Michelle Obama', 'params': {'prop': u'marry', 'qtype': u'who', 'subject': u'obama'}, 'sparql_query': u'\n
SELECT ?vallabel ?type\n
WHERE {\n
  \n
  wd:Q76 p:P26 ?prop . \n
  ?prop ps:P26 ?va
l .\n
  OPTIONAL {\n
    \n
    ?prop psv:P26 ?propVal .\n
    ?propVal rdf:type ?type .
  }\n
  }\n
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" } \n
}'
Michelle Obama La réponse à la question (4)
Enter line: Who was Malcolm Little known as (2)
[None, <nlquery.wikidata.WikiDataAnswer object at 0x00000000092D0160>]
{'plain': u'El-Hajj Malik El-Shabazz, Malcolm Little, Malcolm X', 'params': {'prop': u'known as', 'qtype': u'who', 'subject':
: u'malcolm little', 'sparql_query': u'\n
SELECT ?vallabel\n
WHERE {\n
  \n
  { wd:Q43303 skos:altLabel ?
val FILTER (LANG (?val) = "en") }\n
  UNION\n
  { wd:Q43303 rdfs:label ?val FILTER (LANG (?val) = "en") }\n
  }\n
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" } \n
}'
El-Hajj Malik El-Shabazz, Malcolm Little, Malcolm X (4)
Enter line: What is the hair color of Obama (2)
[None, <nlquery.wikidata.WikiDataAnswer object at 0x00000000092D0630>]
{'plain': u'None', 'params': {'prop': u'hair color', 'qtype': u'what', 'subject': u'obama'}, 'sparql_query': u'\n
SEL
ECT ?vallabel ?type\n
WHERE {\n
  \n
  wd:Q76 p:P1884 ?prop . \n
  ?prop ps:P1884 ?va
l .\n
  OPTIONAL {\n
    \n
    ?prop psv:P1884 ?propVal .\n
    ?propVal rdf:type ?type
  }\n
  }\n
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" } \n
}'
None Aucune réponse retournée (pas de justifications) (4)
Enter line: How tall is Yao Ming
[None, <nlquery.wikidata.WikiDataAnswer object at 0x0000000008D7E588>]
{'plain': u'229.0', 'params': {'prop': 'height', 'qtype': u'how', 'subject': u'yao ming'}, 'sparql_query': u'\n
SELEC
T ?vallabel ?type\n
WHERE {\n
  \n
  wd:Q58590 p:P2044 ?prop . \n
  ?prop ps:P2044 ?v
al .\n
  OPTIONAL {\n
    \n
    ?prop psv:P2044 ?propVal .\n
    ?propVal rdf:type ?typ
e .\n
  }\n
  }\n
  UNION {\n
    \n
    wd:Q58590 p:P2048 ?prop . \n
    ?prop ps:P2048 ?v
al .\n
  }\n
  OPTIONAL {\n
    \n
    ?prop psv:P2048 ?propVal .\n
    ?propVal rdf:type ?typ
e .\n
  }\n
  }\n
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" } \n
}
```

FIGURE 4 – Démarrage de NLQuery via la console

Le lancement de l'application se fait uniquement en ligne de commande en exécutant le script « main.py ». Une fois le script lancé, un champ de saisie apparaît demandant à l'utilisateur d'écrire une question. Pour obtenir une réponse, il faut qu'il appuie sur « Entrée » et le programme commence sa recherche. Notons qu'il faut disposer d'une connexion internet.

La réponse à la question s'affiche sous deux formes :

- Un dictionnaire contenant diverses informations comme le texte brut de la réponse, la requête SPARQL, les éléments extraits de la question (correspond au (3) sur la figure ci-dessus).
- En texte brut (correspond au (4) sur la figure ci-dessus).

### 3.3.2 Formulation des questions types

Il s'agit dans un premier temps de constituer une requête en langage naturel. Cette étape paraît évidente mais elle ne l'est pas pour autant. La requête se doit d'être formulée correctement, d'être sémantiquement et grammaticalement correcte et également prendre en compte l'ordre des mots. Dans notre cas, elle doit être formulée en anglais. Ces critères sont essentiels dans l'élaboration des règles car elles définissent le support du processus de traduction. Penchons-nous maintenant sur quelques exemples de requêtes en langage naturel :

Requête 1 :

— Is Donald Trump a better president than Emmanuel Macron ?

Cette requête est formulée correctement dans le sens où elle est grammaticalement correcte cependant elle n'a aucun sens sémantique. Il est très peu probable de retrouver dans une base de connaissances un triplet concernant une métrique d'évaluation de la réussite de chacune de ces personnes. Donc, cette requête a très peu de chance d'aboutir à un résultat.

Requêtes 2 :

— What is Joseph Meyer's job ?

— What is the job of Joseph Meyer ?

Ces deux dernières requêtes sont formulées différemment mais conservent le même sens, elles diffèrent sur le placement de certains mots et impactent donc la création des règles.

Requêtes 3 :

— Who is Barack Obam ?

— Who is Barak**c** Obama ?

Les fautes de frappe empêchent également d'arriver à un résultat correct car le "matching" ne sera pas complet, nous détaillerons cela dans les prochaines sous sections.

Requête 4 :

— Who is the doctoral advisor of Albert Einstein ?

Intéressons-nous de plus près à cette requête qui est syntaxiquement et sémantiquement correcte. Nous allons l'utiliser comme exemple afin d'illustrer le processus de traduction de NLQuery.

Maintenant que la requête structurée a été déterminée, il va falloir y extraire les mots-clés afin de pouvoir constituer le triplet (*Subject - Property - Object*) car comme on peut s'en douter plusieurs mots dans la requête n'apportent aucune information pertinente. Tels que *is*, *the* ou encore *of* qu'on qualifie de *stopwords* dans le domaine du traitement automatique des langues. Pour pouvoir identifier ces fameux mots-clés, NLQuery a recours au *part-of-speech tagging* qui consiste à associer aux mots d'un texte les informations grammaticales correspondantes comme la partie du discours, le genre, le nombre, etc. à l'aide d'un outil informatique présenté dans la prochaine sous section.

### 3.3.3 Parser

Comme évoqué précédemment, le logiciel fait appel à un outil informatique permettant d'appliquer le POS tagging. Pour cela, le choix de la librairie Stanford a été fait. C'est un excellent moyen de lire une phrase dans plusieurs langages et d'assigner à chaque mot de la phrase le type qui lui correspond (par exemple nom, pronom, verbe, déterminant, ...).

Voici l'aspect visuel de l'arbre obtenu généré par le serveur StanfordCoreNLP :

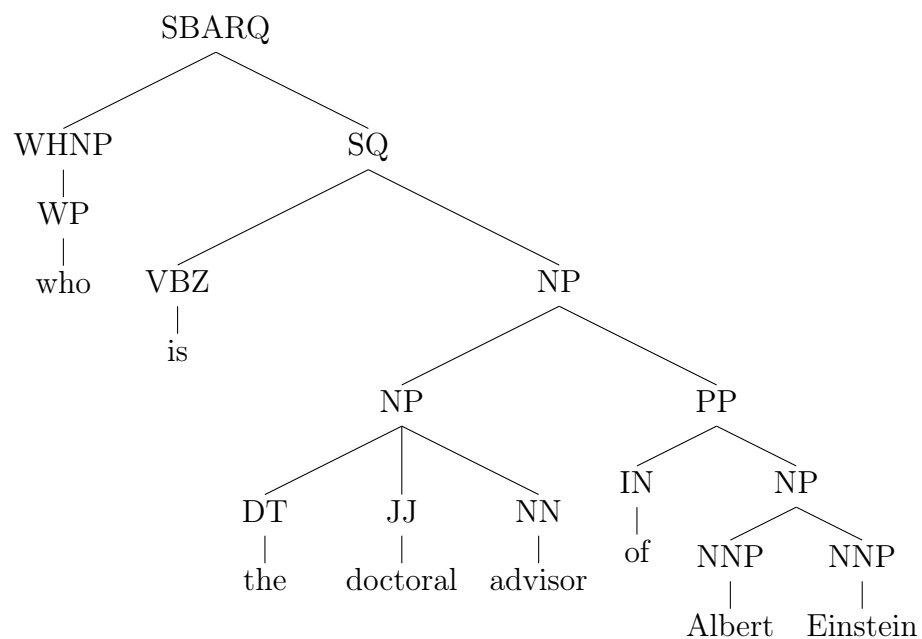


FIGURE 5 – Pos tagging tree  
Who is the doctoral advisor of Albert Einstein

Plusieurs labels ont été affectés aux mots de la requête. Chacun représente une signification particulière que nous allons étudier dans le tableau suivant :

Label	Signification
SBARQ	Introduit une question du type <i>wh</i>
WHNP	Signifie <i>Wh-Noun Phrase</i> indiquant une clause contenant un mot commençant par <i>wh</i>
WP	Signifie <i>Wh-Pronoun</i> qui contient le <i>wh</i> -mot en lui-même
SQ	Désigne une yes/no question ou bien une <i>wh</i> -question
VBZ	Verbe à la troisième personne du singulier au présent
NP	Signifie <i>Noun Phrase</i>
DT	Désigne un déterminant
JJ	Désigne un adjectif
NN	Désigne un nom singulier ou masculin
IN	Désigne une préposition ou conjonction de coordination
NNP	Désigne un nom propre singulier

TABLE 1 – Récapitulatif des labels utilisés

Comme nous pouvons l’observer, les mots principaux qui nous intéressent, résident dans les labels *NP* contenant *the doctoral advisor* et *Albert Einstein* tandis que le reste n’a pas grande importance. Il s’agit maintenant de pouvoir extraire ces mots afin de former la requête SPARQL.



### 3.3.4 Matcher

Précédemment, nous avons évoqué la notion de *règle* où il s'agit de définir certaines règles de grammaire. En voici un exemple concernant la requête "Who is the doctoral advisor of Albert Einstein".

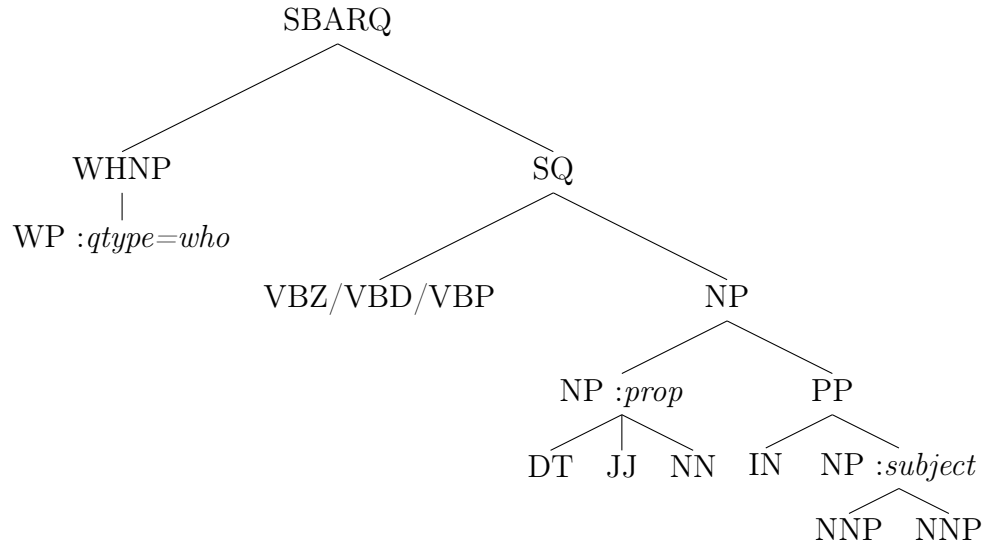


FIGURE 6 – Arbre de règles  
Who is the doctoral advisor of Albert Einstein

L'arbre a été construit en se basant sur celui généré par la librairie Stanford et les mots-clés suivants ont été insérés : *qtype*, *prop* et *subject*. Ces derniers feront office de "repères" pour parvenir à récupérer les mots pertinents.

Il s'agit maintenant de pouvoir récupérer le triplet suivant (*subject* - *prop* - *object*). La stratégie sera une évaluation séquentielle et récursive de l'arbre généré par la librairie Stanford dans un parcours en profondeur, pour cela il faut faire matcher tous les POS de la règle définie ci-dessus.

### 3.3.5 Requête SPARQL

Après avoir récupéré les éléments du triplet, il s'agit maintenant de constituer la requête SPARQL à l'aide de ces éléments. La base de connaissances utilisée est WikiData. Elle a la particularité d'utiliser des identifiants dans le cadre de l'interrogation à travers une requête. Ainsi, il est nécessaire de faire une pré-requête afin de pouvoir récupérer ces identifiants et donc s'en servir pour obtenir les résultats souhaités. Pour ce faire, NLQuery passe par une API de WikiData qui permet de récupérer les identifiants des **subjects** et **properties**. Ainsi pour le triplet suivant (*Albert Einstein* - *Doctoral Advisor* - object), on aura (*Q937* - *P184* - *unknown*). Enfin, il ne reste qu'à interroger WikiData en passant par un endpoint SPARQL. Voici la requête SPARQL obtenue ainsi que les réponses renvoyées par WikiData :

---

```
1 PREFIX wd: <http://www.wikidata.org/entity/>
2 PREFIX wdt: <http://www.wikidata.org/prop/direct/>
3
4 SELECT ?item ?itemLabel
5 WHERE
6 {
7     wd:Q937 wdt:P184 ?item.
8     SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
9 }
```

---

Listing 1 – Who is the doctoral advisor of Albert Einstein ?

item	itemLabel
wd :Q97154	Heinrich Burkhardt
wd :Q116635	Heinrich Friedrich Weber
wd :Q4175282	Alfred Kleiner

TABLE 2 – Résultats de la requête SPARQL

### 3.4 Limites

Après avoir analysé le logiciel NLQuery, nous avons pu faire une critique de cet outil pour mieux comprendre les inconvénients et les limites du système. En effet, cette solution n'est plus satisfaisante à plusieurs niveaux :

- Les règles sont stockées dans le code ce qui n'est pas pratique pour les manipuler.
- Le système couvre la majorité des « WH-questions » mais pas le reste comme les questions de type *yes/no* ou bien *order*, ce qui est contraignant pour l'utilisateur.
- Il n'y a pas d'interface graphique pour l'utilisateur, tout se passe en ligne de commande. Cela peut être difficile d'accès pour un utilisateur n'ayant pas de connaissance en programmation.
- La réponse n'est pas claire pour l'utilisateur et n'est pas bien organisée. Cela peut être difficile à comprendre pour un utilisateur n'ayant pas de connaissance en programmation.
- Lorsqu'il n'y a pas de réponse pour une question donnée, aucune justification n'est fournie. L'utilisateur ne sait pas pourquoi sa question n'a pas donné de réponse. L'erreur peut se trouver dans la formulation de la requête ou bien dans le contenu de la base de connaissances.
- L'utilisateur n'a aucun moyen de connaître quel type de question est accepté par le système, à moins d'accéder au code source.
- Le système ne fait pas de désambiguïsation, il retourne toujours le premier résultat trouvé sans chercher des liens entre les éléments de la question.

Bien que NLQuery ait des points négatifs, il sera exploité par notre prototype et nous verrons comment il va interagir avec.

## 4 Notre approche : SPARQuery

### 4.1 Motivations et objectifs

Notre prototype a pour principal objectif de corriger les faiblesses de NL-Query mais aussi d'apporter des fonctionnalités pertinentes et utiles. Voici à quoi il s'engage :

- **Décentraliser les règles** : Les règles seront écrites dans un fichier JSON. Il sera plus facile de les manipuler ou d'en ajouter à l'avenir.
- **Ajouter de nouvelles règles** : Les *yes/no* questions et *order* questions seront ajoutées. L'utilisateur pourra comparer deux entités et faire une vérification concernant une entité ou deux entités.
- **Ajouter de la consistance aux résultats** : Le système sera capable de trouver des liens entre les éléments de la question dans le but d'enlever toute ambiguïté. Les résultats seront plus précis et cohérents.
- **Ajouter une interface graphique** : L'utilisateur pourra facilement poser sa question et visualiser les diverses informations concernant la réponse en naviguant à travers des onglets. Cela sera plus compréhensible et plus intuitif pour l'utilisateur.
- **Ajouter des justifications concernant la réponse** : Des informations décrivant les éléments détectés par le système ainsi qu'une justification de la réponse seront disponibles. L'utilisateur pourra savoir pourquoi il n'a pas obtenu de réponse.
- **Ajouter des suggestions** : Des suggestions concernant les résultats de la détection des éléments de la question seront proposés. L'utilisateur pourra voir les éléments similaires à ceux de la question.
- **Ajouter un nouveau format de réponse** : Une image pourra être une réponse à une question posée par l'utilisateur. Au lieu d'avoir un lien dirigeant vers une image, il aura directement l'image intégrée dans l'interface graphique comme réponse.
- **Ajouter l'évaluation de la réponse** : L'utilisateur pourra évaluer la question pour dire s'il a obtenu une réponse. Nous pourrions visualiser les questions qui n'ont pas obtenu de réponse et réagir en conséquence en ajoutant ou corrigeant des règles.
- **Aider à reformuler une question** : Un système de reformulation permettra à l'utilisateur de reconstruire sa question. Cela sera utile s'il n'a pas eu de réponse car un des éléments n'a pas été reconnu par le système.

## 4.2 Corrections et améliorations de NLQuery

### 4.2.1 Ajouts de nouveaux types de questions

Après avoir testé et analysé le logiciel NLQuery [14], nous avons remarqué que la portée des questions types était particulièrement limitée à des questions de type *wh*. Afin de l’enrichir, nous avons pensé à rajouter d’autres types de questions pour rendre le programme plus flexible au moment d’une interrogation par l’utilisateur.

La première concerne l’ajout des **yes/no questions**.

En voici quelques exemples :

- Is Lake Baikal bigger than the Great Bear Lake ?
- Is Emmanuel Macron taller than Donald Trump ?
- Is Michael Jackson alive ?
- Is Fortnite developed by Epic Games ?
- Is Macron taller than Trump ?
- Is there a video game called Battle Chess ?
- Are tree frogs a type of amphibian ?
- Is Paris bigger than New York ?

Prenons l'exemple suivant :

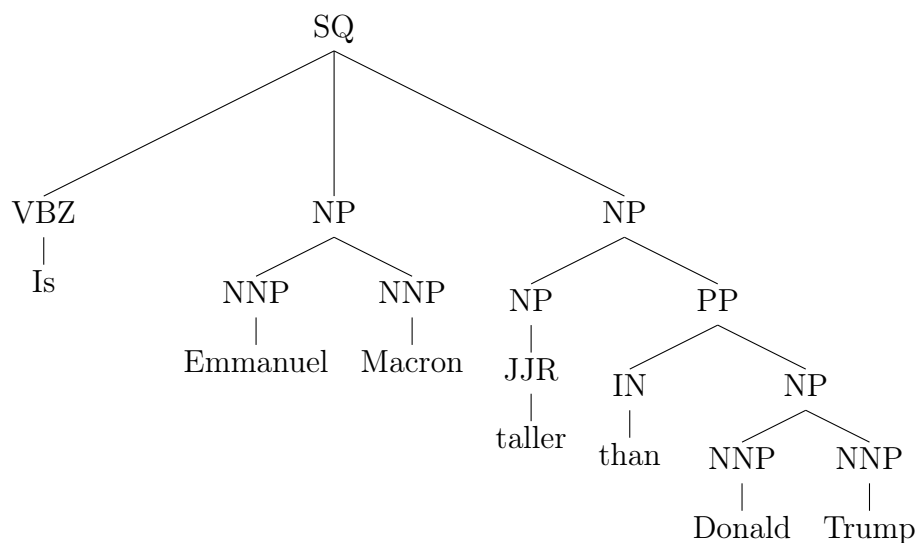


FIGURE 7 – Pos tagger tree  
Is Emmanuel Macron taller than Donald Trump

Nous remarquons cette fois que les termes pertinents reposent sur *Emmanuel Macron*, *taller* et *Donald Trump*. Commençons par définir une règle propre à ce genre de requête.

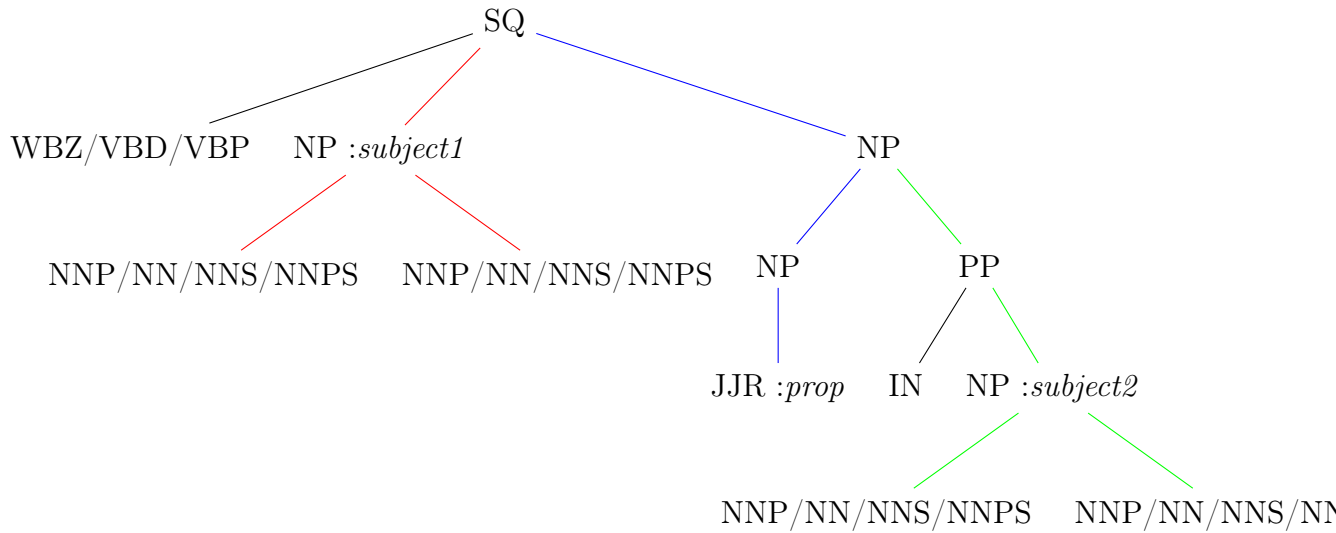


FIGURE 8 – Arbre de règle  
Is Emmanuel Macron taller than Donald Trump

L'évaluation reste la même que celle présentée dans la partie 3. Il est nécessaire de faire matcher tous les labels séquentiellement et de pouvoir récupérer les mots-clés définis (i.e *subject1*, *prop* et *subject2*). Il y a donc trois évaluations pertinentes avec celle en rouge, en bleu puis en vert afin d'obtenir le résultat suivant :

Subject1 : *Emmanuel Macron*, Prop : *taller*, Subject2 : *Donald Trump*

Il s'agit à présent de formuler notre requête SPARQL en nous basant sur les éléments récupérés depuis le Pos tagger tree. Dans un premier temps, il va falloir interpréter la propriété *taller*. Cette dernière n'existe pas sous cette syntaxe sur WikiData. Il est donc nécessaire de la reformuler par *height* qui correspond à l'identifiant *P2048*

---

```

1    PREFIX wd: <http://www.wikidata.org/entity/>
2    PREFIX wdt: <http://www.wikidata.org/prop/direct/>
3
4    SELECT ?higher ?higherLabel
5    WHERE
6    {
7        wd:Q3052772 wdt:P2048 ?p1. # p1 représente la taille d'Emmanuel Macron
8        wd:Q22686 wdt:P2048 ?p2. # p2 représente la taille de Donald Trump
9        BIND (
10           IF(?p1>?p2, wd:Q3052772, wd:Q22686) AS ?higher
11        )
12        SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
13    }

```

---

## Listing 2 – Is Emmanuel Macron taller than Donald Trump?

Cette requête SPARQL renvoie la personne la plus grande entre Emmanuel Macron et Donald Trump. Cependant, elle ne répond pas encore à l'interrogation posée et c'est là qu'on remarque que la propriété utilisée a une grande importance :

---

```

1    # cas superlatif de supériorité
2    if (prop == "taller"): # ou autre propriété de même signification
3        if (higherLabel == subject2):
4            return False
5        else:
6            return True
7    # cas superlatif d'infériorité
8    elif (prop == "smaller"): # ou autre propriété de même signification
9        if (higherLabel == subject2):
10            return True
11        else:
12            return False

```

---

## Listing 3 – Réponse obtenue selon la propriété



La seconde règle ajoutée concerne les **order questions**.

En voici quelques exemples :

- Give me the capital of France
- Give me all the presidents
- Name a single-player video game
- Name someone buried in crown hill cemetery

Prenons l'exemple suivant :

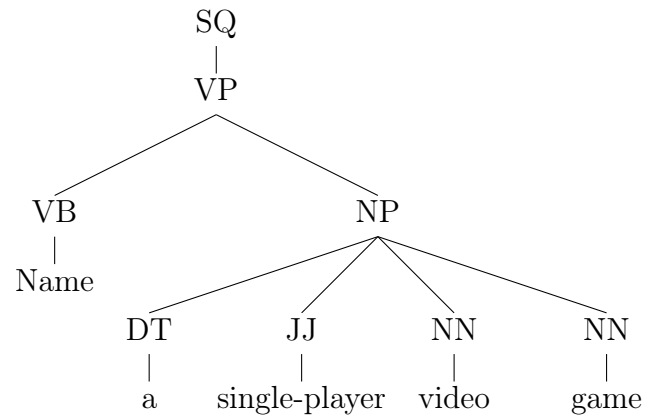


FIGURE 9 – Pos tagger tree  
Name a single-player video game

Le terme le plus important réside dans *single-player video game*. La particularité ici est que ce terme en question représente l'*object* dans le triplet et non plus le *subject* comme les exemples précédents.

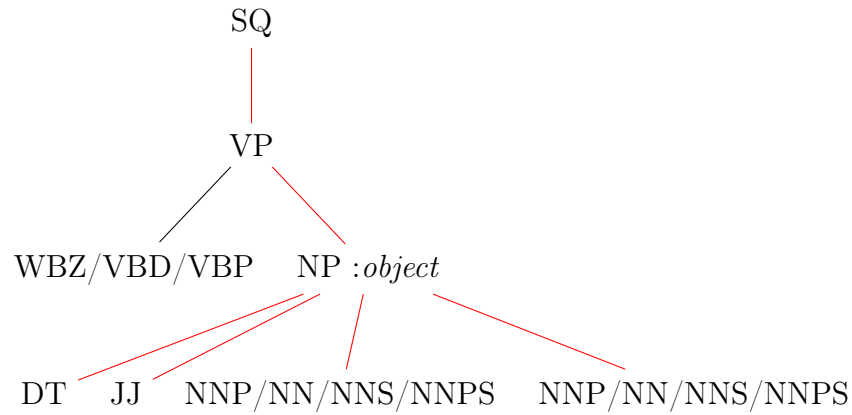


FIGURE 10 – Arbre de règles  
Name a single-player video game

Voici l'évaluation réalisée qui nous permet d'extraire l'information suivante :  
Object = *single-player video game*

---

```

1    PREFIX wd: <http://www.wikidata.org/entity/>
2    PREFIX wdt: <http://www.wikidata.org/prop/direct/>
3
4    SELECT distinct ?game ?gameLabel
5    WHERE
6    {
7        ?game ?prop wd:Q208850.
8        ?game (rdfs:label) ?gameLabel # récupère le résultat du label
9        SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
10   }
11   limit 1

```

---

Listing 4 – Name a single-player video game

game	gameLabel
wd :Q257469	Another World

TABLE 3 – Résultats de la requête SPARQL

#### 4.2.2 Consistance des résultats

Une des difficultés dans la récupération du résultat, c'est qu'il peut y en avoir plusieurs et probablement seulement un correspond aux attentes de l'utilisateur. Afin d'augmenter les chances d'obtenir un résultat "cohérent" avec la requête posée, nous avons décidé de retourner plusieurs résultats potentiels dans la base de connaissances WikiData.

Prenons l'exemple simple suivant : *Who is Obama ?*

Ici, on retrouve le sujet *Obama*. Lorsqu'on dit *Obama*, on a tendance à penser à l'ex-président des États-Unis mais on doit se poser la question "est-ce vraiment le résultat attendu ?". Pour palier à cette ambiguïté, on propose de récupérer un certain nombre d'identifiants faisant références à *Obama* et de suggérer plusieurs réponses.

Id Subject	Suggestion Subject	Description
Q76	Barack Obama	44th President of the United States
Q41773	Obama	city in Fukui prefecture, J
Q5280414	Obama	family name
Q18355807	Obama	genus of worms
Q26446735	Obama	Japanese family name (小浜)
Q39240943	Obama	Japanese family name (小汀)
Q33687029	C. Obama	None

FIGURE 11 – Suggestions - Who is Obama

Le fait d'essayer de faire matcher différents identifiants semble augmenter la performance de la requête, prenons les exemples suivants :

- Is Emmanuel Macron taller than Donald Trump ?
- Is Paris bigger than New-York ?

Dans l'exemple *Is Paris bigger than New-York*, on s'attend ici à comparer deux villes entre elles et déterminer laquelle est la plus grande. Cependant, sommes-nous certains que le premier résultat renvoyé par l'identifiant *Paris* soit vraiment *Paris* la capitale de France et non pas un club de foot ou encore l'université de Paris.

Ainsi, en faisant une recherche sur différents identifiants, il est possible d'obtenir des résultats cohérents comme par exemple :

---

```
1      subject1_ids # liste des ids retournée pour le sujet Paris
2      prop_ids # liste des ids retournée pour la propriété bigger
3      subject2_ids # liste des ids retournée pour le sujet New-York
4      answers = []
5
6      for i in range(subject1_ids):
7          for j in range(prop_ids):
8              for k in range(subject2_ids):
9                  res = getResultSparql(subject1_ids[i], prop_ids[j],
10                     subject2_ids[k])
11                  if len(res) > 0:
12                      answers.append(res)
13
14      return answers
```

---

#### Listing 5 – Code python avec multiple requêtes SPARQL

Par ailleurs, disposer de plusieurs identifiants permet de lever l'ambiguïté qui caractérise certaines entités (par exemple lors de la comparaison entre deux personnes ou entre deux villes). Ici, on ne compare pas la même granularité. Dans l'une on regarde la taille et dans l'autre la superficie. Pourtant on emploie plus ou moins le même terme lors de la requête. Ainsi s'affranchir d'un unique identifiant permet encore une fois d'augmenter l'efficacité des résultats.

## 4.3 Phase d'analyse

Une fois les améliorations de NLQuery terminées, nous pouvons commencer notre prototype. La phase d'analyse va nous permettre de préparer la conception et de spécifier les scénarios d'interaction avec le système et les utilisateurs.

### 4.3.1 Diagramme de cas d'utilisation

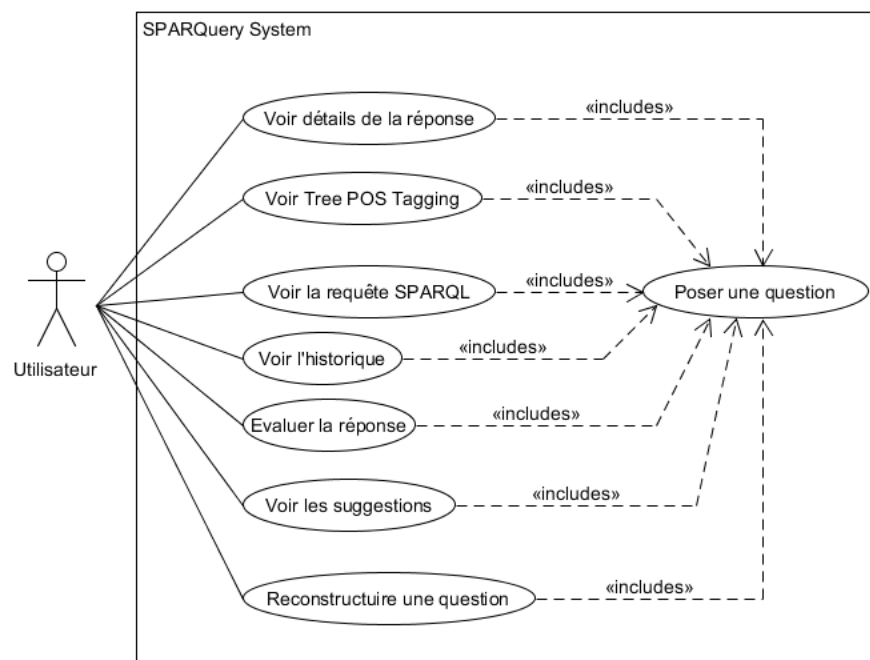


FIGURE 12 – Diagramme de cas d'utilisation de SPARQuery

Ce diagramme permet de visualiser le comportement du système en listant les grandes fonctionnalités. Il s'agit d'organiser et d'exprimer les besoins des utilisateurs. Le seul acteur qui puisse interagir avec le système est l'utilisateur. Il pourra donc agir de différentes façons : visualiser la réponse ainsi que des précisions apportées par le système, voir la transformation de la question ainsi que la règle qui a matché (toutes deux sous forme d'arbre POS tagging), voir la traduction de la question en requête SPARQL avec une légende associée, évaluer la réponse à la question en disant si elle a été pertinente

ou non, voir son historique de questions qu'il a évalué et enfin reconstruire une question à partir d'éléments présents dans celle-ci. Toutes ces actions nécessitent bien-sûr que l'utilisateur ait posé sa question (symbolisé dans le diagramme par les « includes »).

#### 4.3.2 Diagramme de package

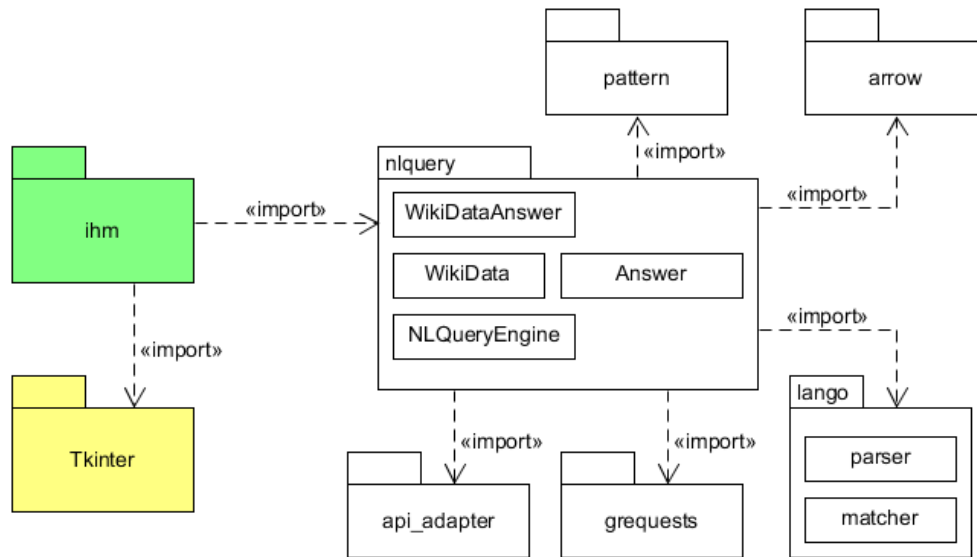


FIGURE 13 – Diagramme de package de SPARQuery

Ce diagramme permet de nous projeter par rapport à la décomposition du système en parties distinctes. En effet, nous allons utiliser l'outil NLQuery et nous aurons besoin d'une interface graphique pour interagir avec l'utilisateur. Cette interface graphique s'appuiera sur la librairie « Tkinter » spécialisée dans la conception GUI (Graphical User Interface). Pour répondre à la question posée par l'utilisateur, le package « ihm » aura besoin du package « nlquery » d'où la relation de dépendances.

Voici quelques détails concernant la découpe en package de NLQuery :

- le package « nlquery » qui contient les outils nécessaires pour interroger Wikidata et stocker la réponse mais qui dépend fortement des packages ci-dessous pour fonctionner correctement.

- le package « grequests » qui permet d’envoyer des requêtes HTTP et de récupérer la réponse.
- le package « pattern » qui joue un rôle dans le traitement du langage naturel ainsi que dans l’analyse de graphes.
- le package « arrow » qui permet de manipuler ou convertir des dates.
- le package « api\_adapter » qui permet d’échanger avec des api’s grâce à un adaptateur REST (Representational State Transfer).
- le package « lango » qui est là pour obtenir la structure syntaxique d’une phrase (en arbre POS tagging) et en extraire des informations importantes.

## 4.4 Architecture logicielle

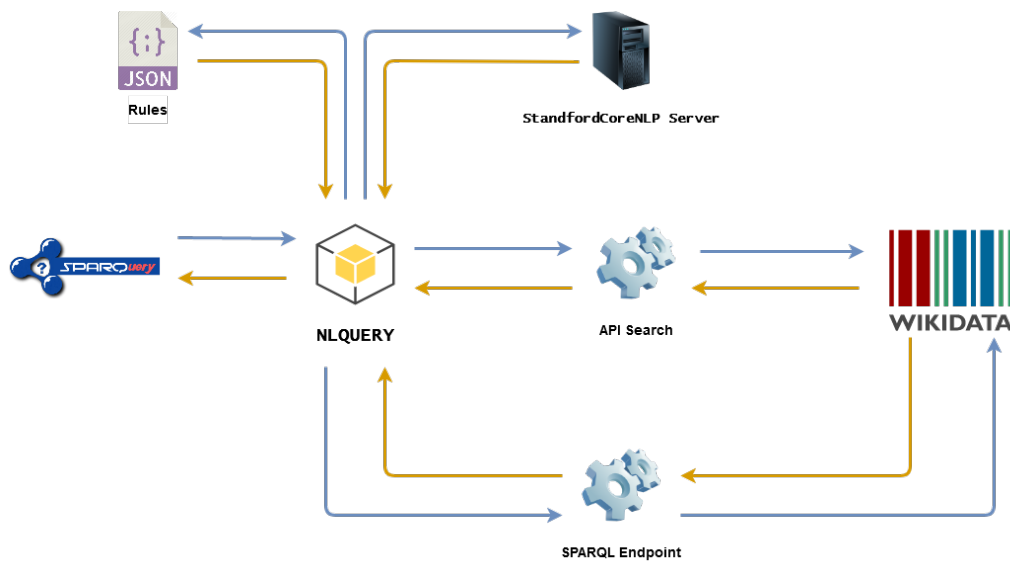


FIGURE 14 – Interactions entre SPARQuery et NLQuery (émission : en bleu / réception : en jaune)

Avant de se lancer dans la description de la structure interne du système, il faut qu’on s’attarde sur les échanges qui se déroulent au cours d’un scénario d’exécution. Ce diagramme vise à répondre à trois problématiques :

- Qui gère les interactions avec l’extérieur, en particulier l’utilisateur (saisie et affichage) ?
- Qui effectue des opérations sur les données ?

— Qui manipule les données ?

Notre application SPARQuery offre une interface graphique à l'utilisateur pour qu'il puisse poser ses questions. Pour cela, elle fait appel au module indépendant NLQuery en lui soumettant la question. Il fournit une interface accessible pour communiquer. Pour former le résultat final, NLQuery fait tout d'abord des processus internes que nous avons décrit auparavant. Nous avons décidé de décentraliser les règles dans un fichier JSON pour nous permettre d'en ajouter plus facilement. Finalement, il regroupe ces données et retourne le résultat final à notre application qui se charge de le délivrer à l'utilisateur.



## 4.5 Phase de conception

### 4.5.1 Diagramme de classes

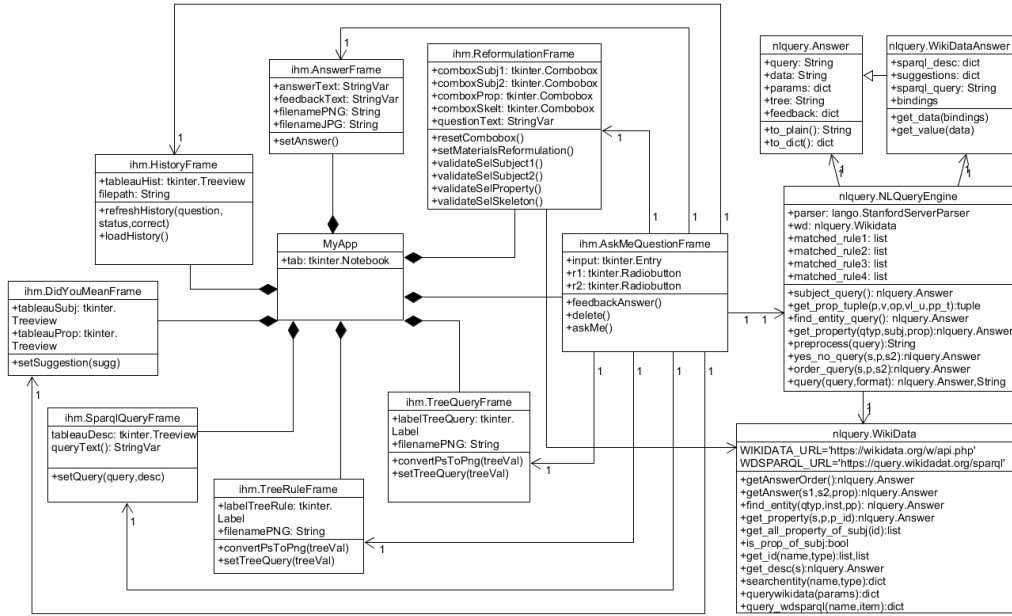


FIGURE 15 – Diagramme de classes de SPARQuery

Une fois avoir présenté une vue globale des interactions de l'architecture logicielle, nous allons décrire la structure interne du système. Ce diagramme permet d'avoir une vue précise des objets et nous aide ainsi pour l'implémentation de l'application.

L'application est représentée par la classe « MyApp » qui est composée de plusieurs « Frame » correspondant aux fenêtres de l'application et un « notebook » pour naviguer à travers les fenêtres.

On voit précisément où se situe l'interaction avec l'outil NLQuery. Elle se déroule dans la classe « AskMeQuestion ». Elle contient un champ de saisie et fait appel à la classe « nlquery.NLQueryEngine » pour récupérer l'ensemble des informations de la réponse. Ensuite, elle les distribue dans les diverses fenêtres de l'application. Chacune de ces fenêtres appliquera une opération pour se mettre à jour.

La réponse est décrite selon deux classes suivantes : la classe mère « Answer »

et la classe fille « `WikiDataAnswer` » pour les données spécifiques à Wikidata (requête, identifiants, suggestions).

On remarque que la classe « `ReformulationFrame` » doit utiliser la classe « `nlquery.Wikidata` » pour obtenir en temps réel les propriétés associées à l'identifiant d'une entité. Nous verrons plus en détail cette procédure grâce à un diagramme de séquence. Elle contient quatre listes déroulantes et un label textuel.

La fenêtre « `AnswerFrame` » contient des labels textuels pour la réponse et les commentaires du système. Elle contient aussi des noms de fichiers image car la réponse peut être sous forme d'image. Ils pourront être définis grâce à la méthode `setAnswer`.

La fenêtre « `HistoryFrame` » contient un tableau avec la liste des questions évaluées par l'utilisateur et un nom de fichier texte où est stocké l'historique.

La fenêtre « `SparqlQueryFrame` » contient un label textuel qui est la requête SPARQL et un tableau avec la description de la requête (identifiants + labels). Ils pourront être modifiés grâce à la méthode `setQuery`.

La fenêtre « `DidYouMeanFrame` » contient deux tableaux qui vont lister les suggestions et pourront être remplis grâce à la méthode `setSuggestions`.

Les fenêtres « `TreeRuleFrame` » et « `TreeQueryFrame` » contiennent toutes deux une image qui est une visualisation de l'arbre syntaxique de la question et de la règle qui a matché. Cette image pourra être modifiée avec la méthode `setTreeQuery`.

Grâce aux diagrammes de séquence, nous allons voir comment interagissent ces objets.

## 4.5.2 Diagramme de séquence

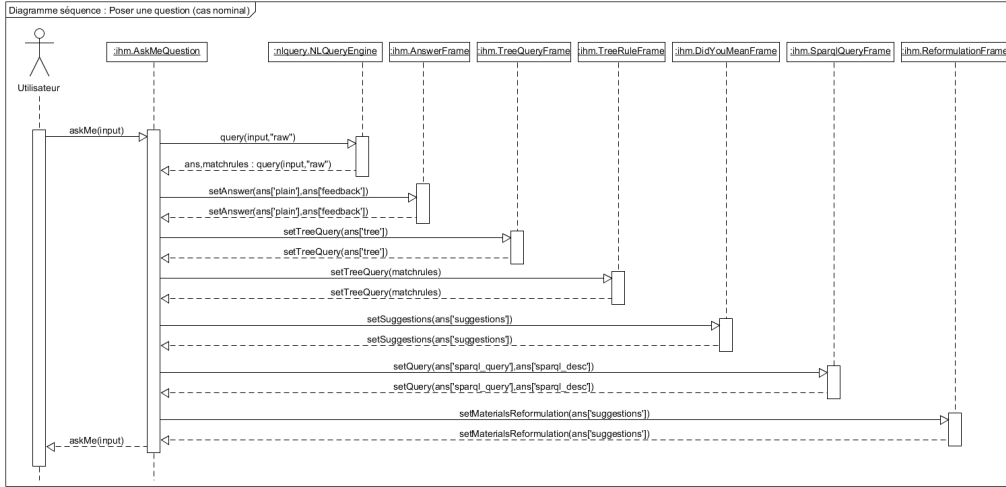


FIGURE 16 – Diagramme de séquence du cas d'utilisation "Poser une question"

Intéressons-nous au cas d'utilisation « Poser une question » et regardons les interactions entre les composants dans ce diagramme de séquence. L'utilisateur se trouve dans la fenêtre « AskMeQuestionFrame » et doit suivre la procédure suivante :

- L'utilisateur saisit sa question et lance l'opération.
- Cela fait appel à « NLQueryEngine » qui va se charger de récupérer l'ensemble des informations de la réponse. La réponse (sous forme de dictionnaire) et la règle qui a matché (matchrules) sont retournés.
- Ensuite, il s'agit de placer chaque élément de la réponse dans sa partie respective. La fenêtre « AnswerFrame » reçoit le texte brut de la réponse ainsi que des commentaires du système, les fenêtres « TreeRuleFrame » et « TreeQueryFrame » reçoivent les structures syntaxiques de la question et de la règle qui a matché, la fenêtre « DidYouMeanFrame » reçoit les suggestions concernant les éléments détectés dans la question, la fenêtre « SparqlQueryFrame » reçoit les informations de la requête SPARQL et enfin la fenêtre « ReformulationFrame » les mêmes suggestions que « DidYouMeanFrame ».
- Finalement, l'utilisateur voit s'afficher les éléments de la réponse.

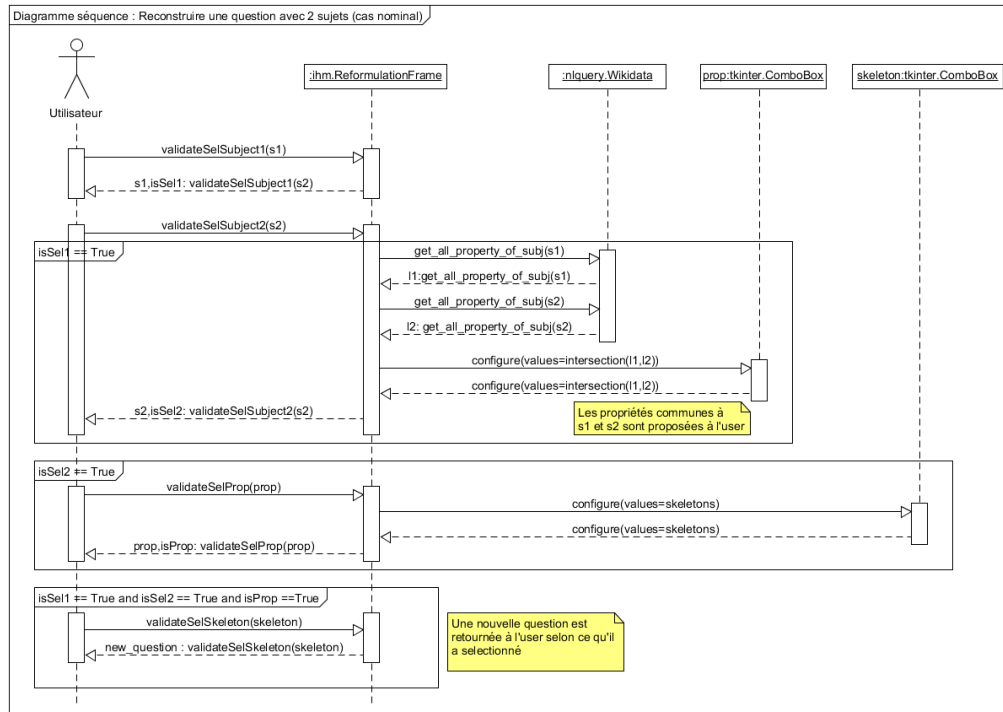


FIGURE 17 – Diagramme de séquence du cas d'utilisation "Reconstruire une question"

Nous allons décrire le cas d'utilisation « Reconstruire une question » à travers le diagramme de séquence ci-dessus. On considère que l'utilisateur a déjà posé sa question et qu'il s'agit d'une question composée de deux sujets : S1 et S2. L'utilisateur se trouve dans la fenêtre « ReformulationFrame » et doit suivre la procédure suivante :

- Il commence par choisir parmi des suggestions concernant les sujets S1 et S2. Une suggestion pour S1 (resp. S2) peut correspondre à un homonyme ou un terme contenant S1 (resp. S2).
- Une fois choisis, il peut débloquent le champ correspondant à la propriété et en choisir une. Cela fait appel à « Wikidata » pour récupérer la liste des propriétés (l1) du sujet S1 et la liste des propriétés (l2) du sujet S2. Ensuite, les propriétés en commun sont ajoutés dans la liste.
- Une fois les sujets et la propriété sélectionnés, l'utilisateur peut choisir parmi une liste de question-type (ex : «Is there a (S1) (P) (S2) ? ») et la nouvelle question est retournée à l'utilisateur en remplaçant les

champs correspondant aux sujets et à la propriété.

## 4.6 Outils et langage utilisé

Pour développer l'application, nous avons utilisé le langage de programmation Python avec des bibliothèques pour la conversion d'URL en image (*urllib*), pour la manipulation d'image (*PIL*) et pour la création de l'interface graphique (*Tkinter*). Nous avons eu besoin de logiciels externes pour la conversion d'image au format ps en png : *MagickImage* et *GhostScript*.

## 4.7 Démonstration des fonctionnalités

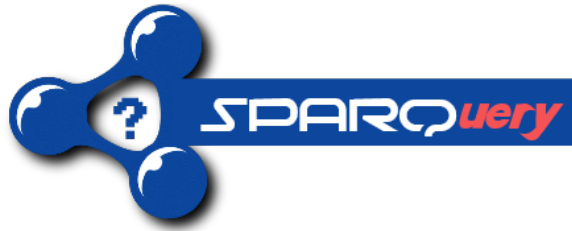


FIGURE 18 – Logo de SPARQuery

Nous avons créé un logo qui reflète assez bien l'esprit de notre application : design et innovant. Nous allons vous montrer comment s'organise l'application et comment l'utilisateur peut interagir avec.

Notre utilisateur a démarré SPARQuery avec son IDE préféré et a entré dans le champ de saisie la question suivante : "What sport practiced Michael McDowell ?".

L'application est organisée en deux blocs : un bloc fixe avec le champ de saisie accompagnée d'une barre de navigation et un autre bloc qui affiche la fenêtre active. Pour obtenir une réponse à sa question, il a simplement appuyé sur la touche "Entrée" de son clavier. Il a maintenant la possibilité de naviguer et de consulter les différents éléments de la réponse.

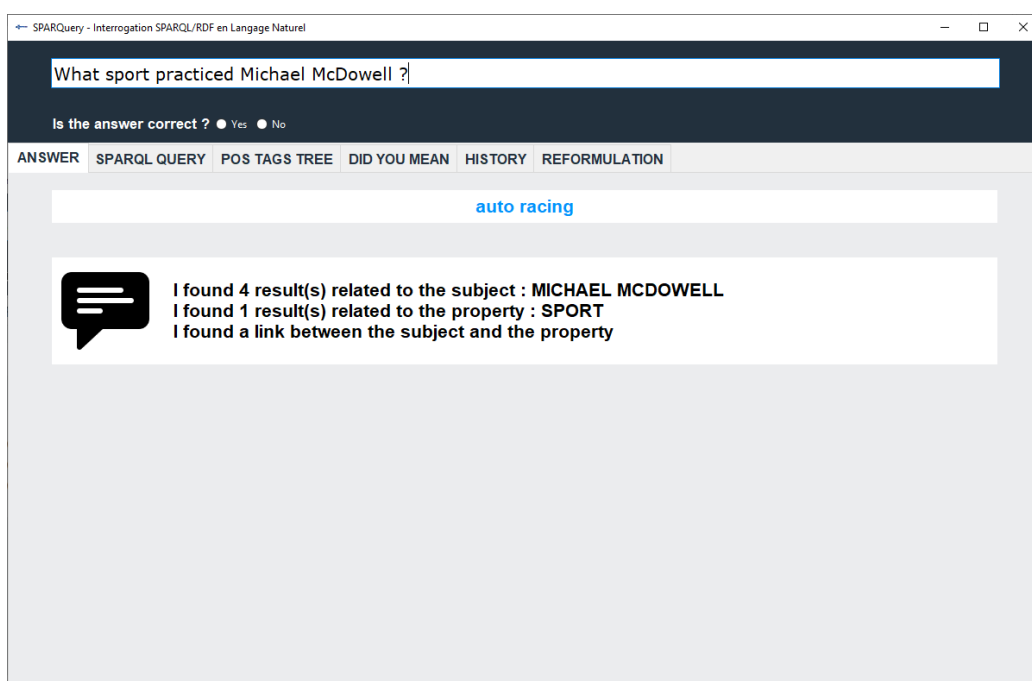


FIGURE 19 – Onglet "Answer"

Le premier onglet rencontré est l'onglet "Answer". Il contient le texte brut de la réponse (affiché en bleu) et en dessous un champ avec les commentaires du système. Les commentaires sont là pour justifier la réponse à l'utilisateur. On peut voir que la question contenait une propriété (*sport*) et un sujet (*Michael McDowell*). Cette extraction est rendue possible grâce aux "squelettes de règles" qui sont définis dans le système.

Notre utilisateur a décidé de passer à l'onglet suivant : celui de la traduction de la question en requête SPARQL. Cela peut avoir un côté ludique pour permettre à l'utilisateur d'appréhender le langage SPARQL. L'utilisateur peut

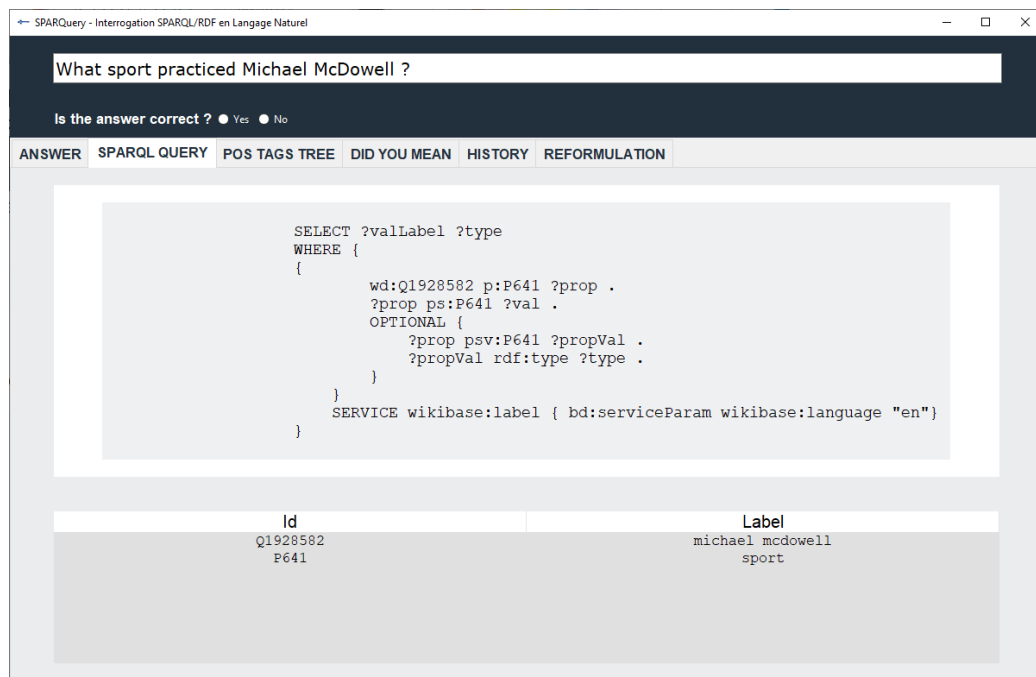


FIGURE 20 – Onglet "SPARQL QUERY"

connaître les identifiants des éléments extraits (ici sujet et propriété) et voir comment ils ont été placés dans la requête SPARQL.

Notre utilisateur continue de naviguer et atterrit dans l'onglet contenant les arbres syntaxiques de la question et de la règle qui a matché. Cette onglet nous est principalement destiné car il est utile pour créer de nouvelles règles à partir de l'arbre syntaxique de la question.

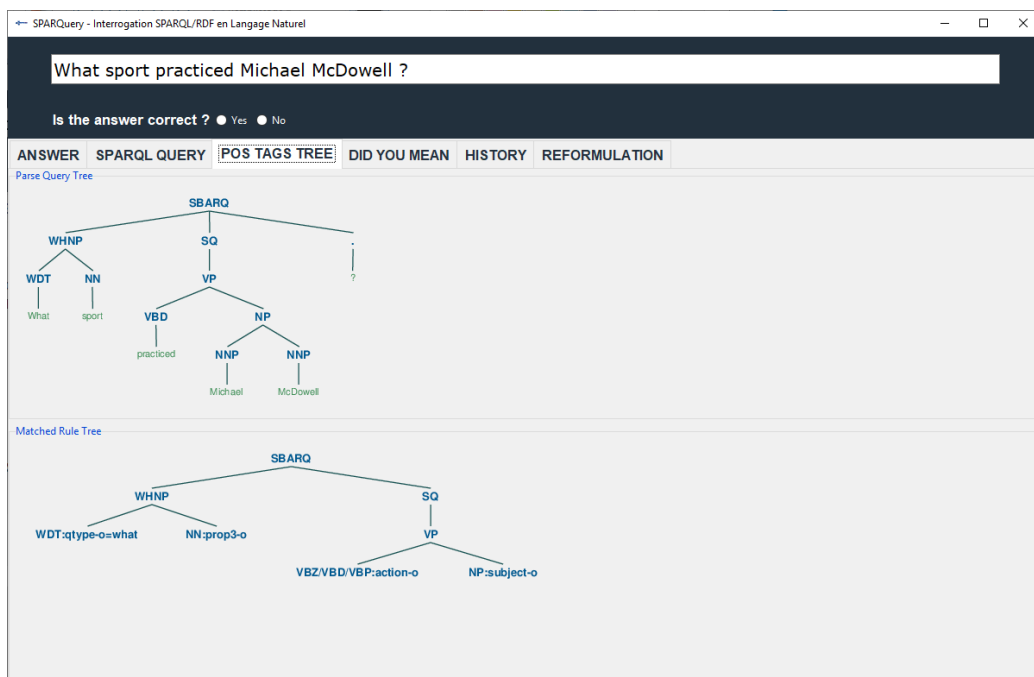


FIGURE 21 – Onglet "POS TAGS TREE"

Si jamais un des arbres n'est pas assez visible, il peut ajuster les panneaux pour une meilleure visibilité.

A l'avenir, un éditeur de règles pourrait être ajouté. L'utilisateur pourrait créer de nouvelles règles. Il suffirait qu'il assigne à chaque élément de la question, un attribut particulier dans l'arbre de règle (par exemple pour "sport" équivalent à "prop").



Dans l'onglet "Did You Mean", on propose à l'utilisateur des suggestions selon les éléments extraits de la question. Pour le sujet *Michael McDowell*, quatre entités sont possibles. Il s'agit d'un cas d'homonymie. On peut remarquer que le système a choisi le troisième individu, *le pilote*, car c'est le seul à avoir une propriété *sport*.

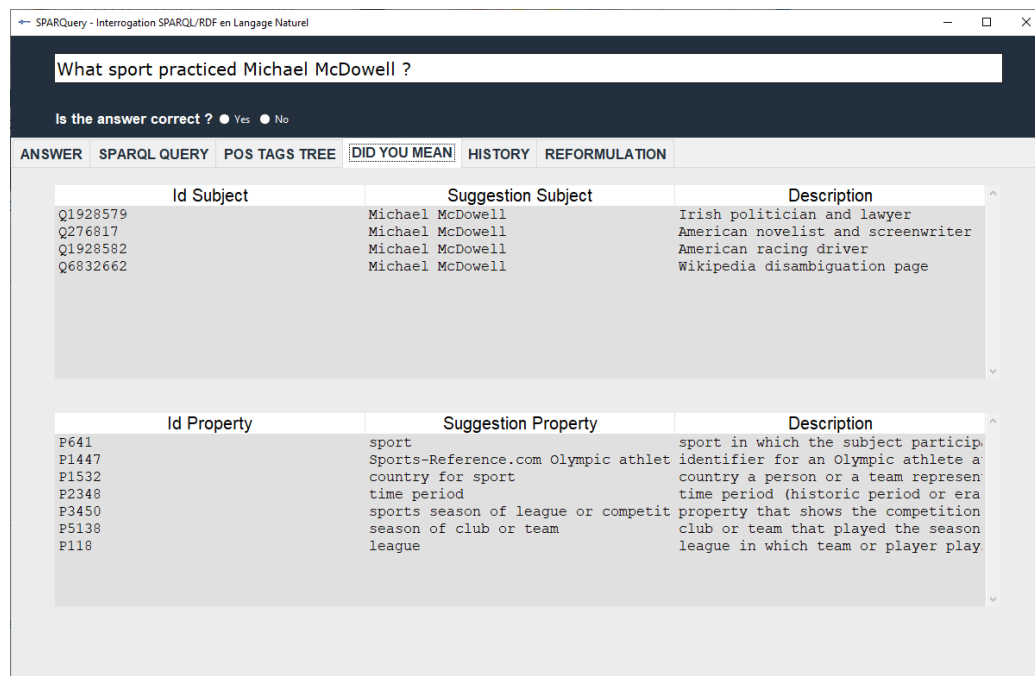


FIGURE 22 – Onglet "DID YOU MEAN"

La fenêtre active contient un tableau pour les sujets et un autre pour les propriétés. Chacun de ces tableaux est séparé en trois colonnes : une pour l'identifiant, la deuxième pour le label et la dernière pour la description.

L'utilisateur peut à tout moment évaluer la réponse. Il n'a qu'à cocher un des boutons radio situés en dessous du champ de saisie.

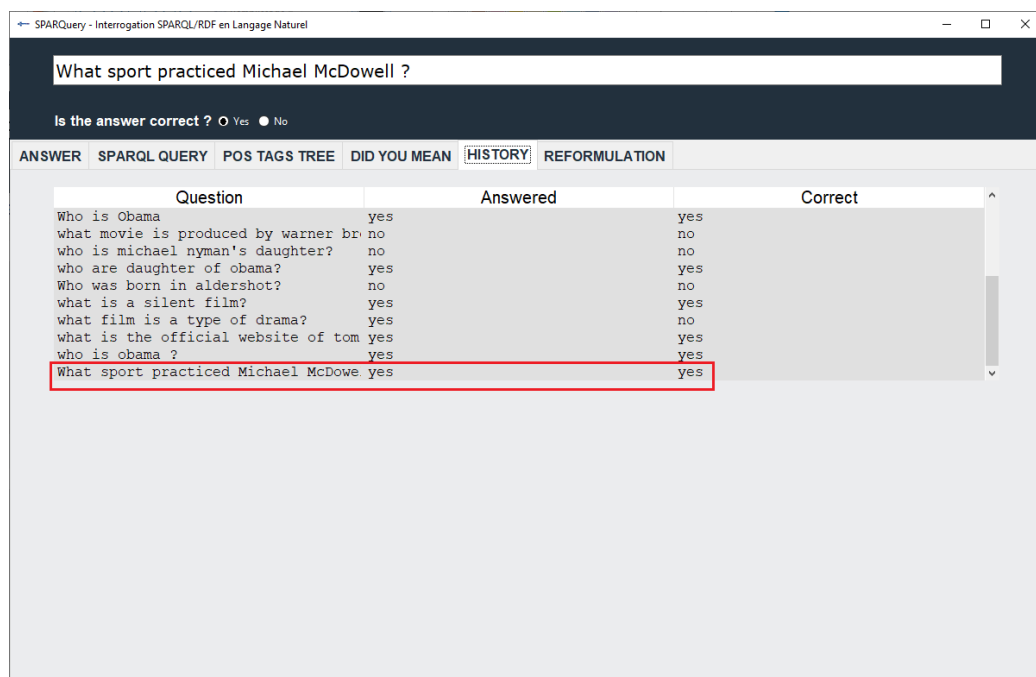


FIGURE 23 – Onglet "HISTORY"

A chaque évaluation, la réponse est enregistrée dans un fichier avec deux mentions : si la question a obtenu une réponse correcte et si la question a obtenu une réponse.

Cela nous assure un suivi et permet de corriger ou d'ajouter de nouvelles règles si besoin.

Il existe plusieurs raisons pour que l'utilisateur n'obtienne pas de réponse à sa question. Le premier cas, lorsque le type de la question n'est pas reconnu par le système.

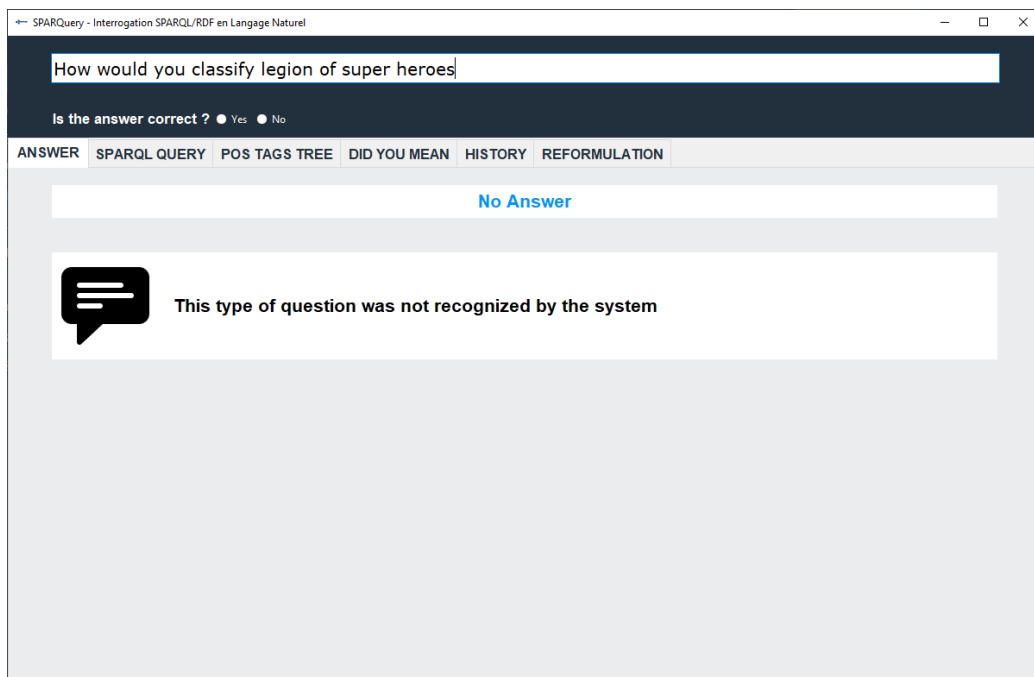


FIGURE 24 – Pas de réponse - Type de question non reconnu

Un message est affiché à l'utilisateur et il est donc invité à reposer une question compatible avec le système. De plus, l'utilisateur peut nous aider à améliorer le système en cochant la case "No" pour que nous puissions l'ajouter en tant que nouvelle règle compatible avec le système.

Le second cas, pour que l'utilisateur n'ait pas de réponse, peut avoir différentes raisons :

- un des éléments a mal été orthographié
- un des éléments n'existe pas dans WikiData
- pas de lien entre les éléments

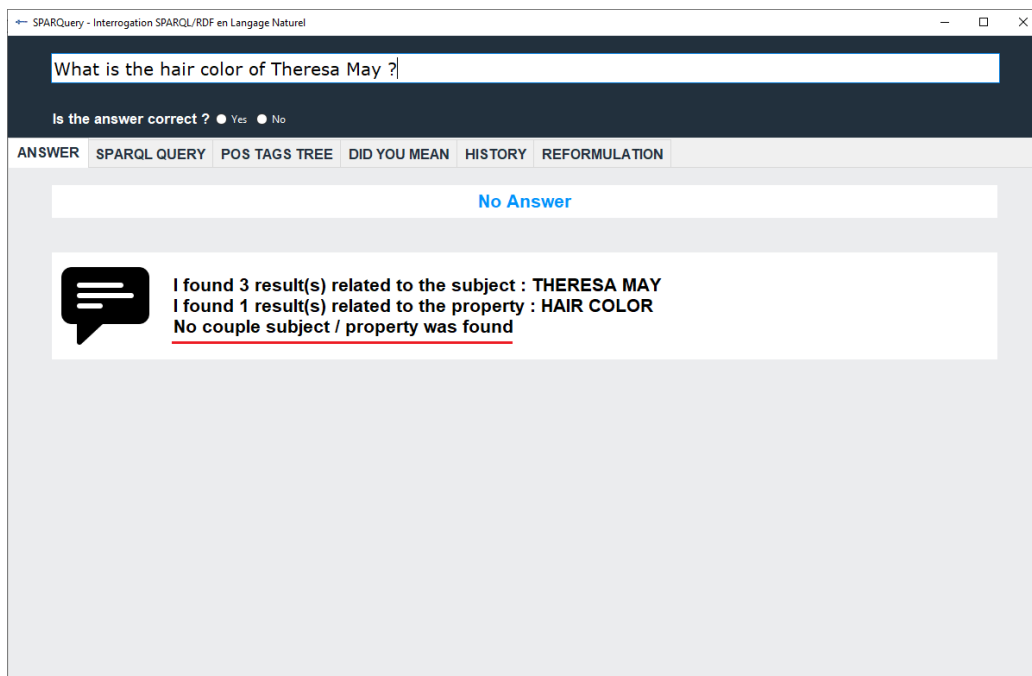


FIGURE 25 – Pas de réponse - pas de liens entre les éléments

Notre utilisateur voudrait connaître la couleur de cheveux de Theresa May mais n'obtient pas de réponse. Le système a reconnu la propriété *hair color* ainsi que le sujet *Theresa May* mais n'a pas trouvé de lien entre ces deux éléments.

C'est pour une de ces raisons que la fonctionnalité de "Reformulation de question" a été créée. L'utilisateur pourra connaître en temps réel les propriétés associées au sujet *Theresa May*.

Voyons comment l'utilisateur peut reformuler sa question. L'utilisateur doit se rendre dans le dernier onglet. La reformulation se fait étape par étape.

SPARQuery - Interrogation SPARQL/RDF en Langage Naturel

What is the hair color of Theresa May ?

Is the answer correct ? ☐ Yes ☐ No

ANSWER | SPARQL QUERY | POS TAGS TREE | DID YOU MEAN | HISTORY | **REFORMULATION**

**(1) Subject 1**

- Theresa May (Q264766)
- Theresa May (Q264766)
- Theresa May (Q30161835)
- Theresa May to become UK Prime Minister as opposition begins leadership election (Q25820611)
- Theresa May's Conservative Party wins UK election but loses majority, leaving Brexit plan in question (Q30237873)
- Theresa May's Brexit speech rules out single market membership (Q28378501)
- Theresa May calls for June general election (Q29566784)
- Theresa May (Q30684574)

**Subject 2**

**Property**

**Query type**

No question was built

FIGURE 26 – Onglet "REFORMULATION - étape 1"

La première étape concerne le choix du sujet. La question posée par l'utilisateur contenait seulement un sujet (*Theresa May*). Donc le premier champ est pré-rempli avec les suggestions du sujet (les homonymes). L'utilisateur a choisi le premier sujet qui correspond à la "première ministre".

Puis une fois un sujet sélectionné, la liste déroulante pour le champ "property" se remplit.

SPARQuery - Interrogation SPARQL/RDF en Langage Naturel

What is the hair color of Theresa May ?

Is the answer correct ? ☐ Yes ☐ No

ANSWER | SPARQL QUERY | POS TAGS TREE | DID YOU MEAN | HISTORY | REFORMULATION

(1) Subject 1: Theresa May (Q264766)

Subject 2:

(2) Property: image

Query type:

No question was built

FIGURE 27 – Onglet "REFORMULATION" - étape 2

En effet, il s'agit uniquement des propriétés relatives au sujet choisi. L'utilisateur a choisi la propriété *image*.

Il ne lui reste plus qu'à choisir un "squelette" de question compatible c'est-à-dire contenant une propriété et un sujet.

SPARQuery - Interrogation SPARQL/RDF en Langage Naturel

What is the hair color of Theresa May ?

Is the answer correct ? ☐ Yes ☐ No

ANSWER | SPARQL QUERY | POS TAGS TREE | DID YOU MEAN | HISTORY | **REFORMULATION**

(1) **Subject 1** Theresa May (Q264766)

**Subject 2**

(2) **Property** image

(3) **Query type**

No question was built

- who are the (P) of (S)?
- how many countries (P) (S)?
- which mountain is the (P) after the (S)?
- when was (S) (P)?
- where is (S) (P)?
- who is the (P) of (S) ?
- who (P) (S)?
- who was (S) (P)?
- what is the (P) of (S)?**
- what (P) is (S)?

FIGURE 28 – Onglet "REFORMULATION" - étape 3

Notre utilisateur a choisi le squelette "What is the (P) of (S) ?" ce qui donnera après remplacement "What is the image of Theresa May ?". Il faut rappeler que la liste des questions-types se base par rapport aux questions déjà posées dans l'application. En effet, à chaque question posée son squelette est enregistré dans un fichier. Par exemple, le squelette pour la question "What sport practiced Michael McDowell ?" serait "What (P) practiced (S) ?".

SPARQuery - Interrogation SPARQL/RDF en Langage Naturel

what is the image of Theresa May? (5) COLLER

Is the answer correct ? ● Yes ● No

ANSWER SPARQL QUERY POS TAGS TREE DID YOU MEAN HISTORY REFORMULATION

(1) Subject 1 Theresa May (Q264766)

Subject 2

(2) Property image

(3) Query type what is the (P) of (S)?

what is the image of Theresa May? (4) COPIER

FIGURE 29 – Onglet "REFORMULATION" - phase finale

Si l'utilisateur veut poser sa nouvelle question, il copie-colle dans le champ de saisie et il relance l'opération.



Finalement, grâce à la reformulation, l'utilisateur pourra obtenir une réponse.

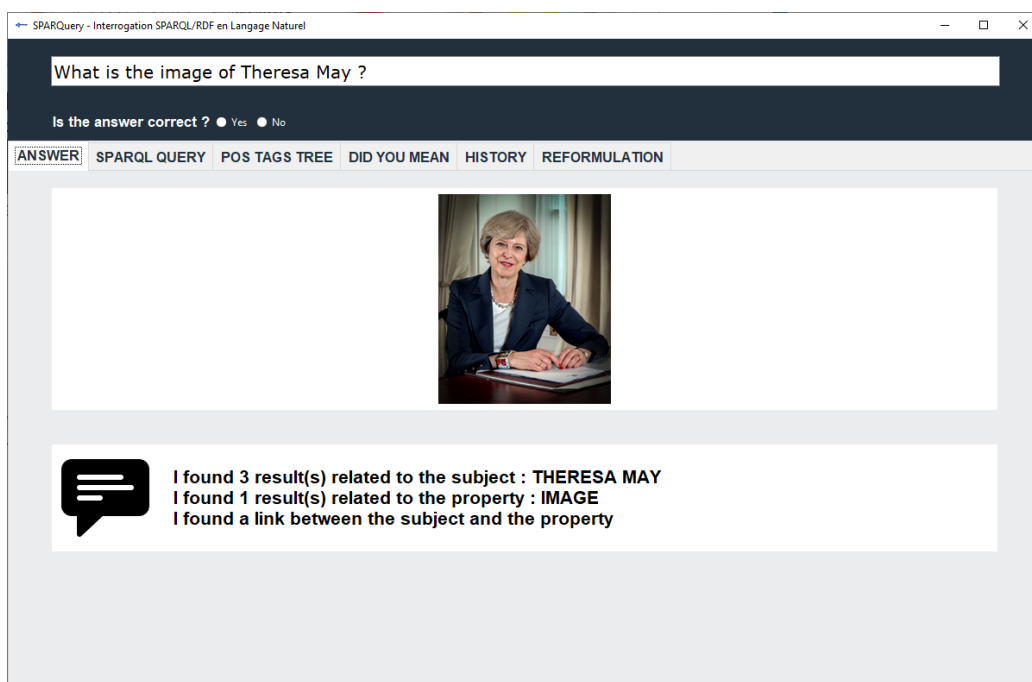


FIGURE 30 – Onglet "Answer" - réponse sous forme d'image

L'application peut aussi retourner la réponse sous forme d'image, comme le montre la capture d'écran ci-dessus. L'application accepte trois formats d'image : JPG, PNG et SVG. On notera ici qu'il s'agit de l'image *Theresa May* (première ministre). Lorsqu'il y a plusieurs sujets en concurrence, on prend toujours celui en tête de liste.

## 5 Évaluation et comparaisons

Notre évaluation sera sous la forme de plusieurs benchmarks mettant en jeu la précision des résultats entre NLQuery [14] et notre prototype.

### 5.1 Benchmark sur les *wh-questions*

	Réponses obtenues des différents systèmes concernant les <i>wh-questions</i>	
Requête NL	NLQuery	SPARQuery
Who is Obama ?	44th President of the United States of America	44th President of the United States of America
Who is the doctoral advisor of Albert Einstein ?	Alfred Kleiner Heinrich Burkhardt Heinrich Friedrich Weber	Alfred Kleiner Heinrich Burkhardt Heinrich Friedrich Weber
What artist performed Colors ?	Aucun résultat	Beck
What sport practiced Michael McDowell ?	Auto racing	Auto Racing
What religion is Donald Trump ?	Presbyterianism	Presbyterianism
In which films directed by Garry Marshall was Julia Roberts starring ?	Aucun résultat	Aucun résultat
Which mountain is the highest after the Annapurna ?	Aucun résultat	Aucun résultat

TABLE 4 – Benchmark *wh-questions*

Le fait qu'il n'y ait aucun résultat pour la requête *What artist performed Colors ?* provient de la récupération d'un seul résultat pour le terme *Colors*. Ici *Colors* se réfère à une musique et non à une couleur comme on pourrait y penser. Mais sur WikiData, en ne récupérant qu'un seul résultat on récupère le résultat avec le plus haut ranking donc *colors* la couleur.

Concernant les deux dernières requêtes n'aboutissent pas pour les raisons suivantes : il faudrait définir leur règles dans un premier temps, ensuite identifier les termes pertinents afin de former les requêtes SPARQL. La difficulté

repose sur le fait que cette requête demande plusieurs sous-requêtes SPARQL pour arriver au résultat final. Comme par exemple *In which films directed by Garry Marshall was Julia Roberts starring?* dont les termes pertinents sont *film*, *directe*, *Garry Marshall*, *Julia Roberts* et *starring*. Ces derniers sont assez nombreux, rendant la requête plus complexe.

## 5.2 Benchmark sur les *yes/no questions*

	Réponses obtenues des différents systèmes concernant les <i>yes/no questions</i>	
Requête NL	NLQuery	SPARQuery
Is Lake Baikal bigger than the Great Bear Lake ?	Aucun résultat	Yes
Is Emmanuel Macron taller than Donald Trump ?	Aucun résultat	No
Is Michael Jackson alive ?	Aucun résultat	No
Is Fortnite developed by Epic Games ?	Aucun résultat	Yes
Is there a video game called Battle Chess ?	Aucun résultat	Yes
Is Paris bigger than New York ?	Aucun résultat	No
Was the Cuban Missile Crisis earlier than the Bay of Pigs Invasion ?	Aucun résultat	Aucun résultat
It was in Munich this German city actor Erland Erlandsen died ?	Aucun résultat	Aucun résultat

TABLE 5 – Benchmark *yes/no questions*

La présence ici du manque de résultat sur le logiciel NLQuery [14] s'explique par le fait que les règles pour répondre à ce type de question n'ont pas été implémentées. De même pour SPARQuery, par exemple en prenant la dernière requête *It was in Munich this German city actor Erland Erlandsen died*, il s'agirait ici de constituer le triplet suivant : *Erland Erlandsen - died - Munich* et d'interroger WikiData et voir si on trouve ou non une relation entre Erland Erlandsen et Munich avec la propriété *died*.

### 5.3 Benchmark sur les *order questions*

	Réponses obtenues des différents systèmes concernant les <i>order questions</i>	
Requête NL	NLQuery	SPARQuery
Give me the capital of France	Aucun résultat	Paris
Give me all the cosmonauts	Aucun résultat	Lee Archambault George D. Zamka Ivan Bella David Saint-Jacques ...
Name a single-player video game	Aucun résultat	Civilization III
Name someone buried in crown hill cemetery	Aucun résultat	Benjamin Harrison
Name the place where John Mica was born	Aucun résultat	Aucun résultat
List all episodes of the first season of the HBO television series The Sopranos	Aucun résultat	Aucun résultat
Give me all cities in New Jersey with more than 100000 inhabitants	Aucun résultat	Aucun résultat

TABLE 6 – Benchmark *order questions*

Le manque de résultat réside encore une fois dans l'inexistence des règles qui n'ont pas été introduites, mais également dans la complexité des dernières requêtes nécessitant plusieurs requêtes SPARQL afin de pouvoir obtenir le résultat final.

## 6 Conclusion et perspectives

Si un utilisateur veut interroger une base de connaissances telle que Wikidata ou Dbpedia, il risque d'être confronté à des obstacles. Pour pouvoir exprimer une requête, il doit connaître le langage SPARQL ainsi que le « vocabulaire » associé à la base de connaissances.

Voilà pourquoi de nombreux travaux ont émergé pour essayer de répondre à cette problématique. Nous en avons fait l'étude pour essayer de dégager une méthodologie commune et finalement notre attention s'est dirigé vers un outil : NLQuery.

Nous l'avons analysé de fond en comble pour pouvoir comprendre au mieux son fonctionnement et l'exploiter par la suite. Il se base principalement sur des « squelettes de règles » afin de récupérer des mots-clés caractérisant la question posée et utilisés pour la requête SPARQL. Durant cette analyse, nous avons constaté des défauts comme le fait qu'il ne dispose pas d'interface graphique, qu'il soit limité au niveau du type de question et que parfois les résultats n'étaient pas concluants. C'est pour cela que nous avons développé notre application SPARQuery dans le but de corriger ces défauts et apporter un renouveau à l'outil NLQuery.

L'approche que nous vous avons présenté est novatrice sous différents aspects. Nous avons développé un outil polyvalent, informatif et ludique pour l'utilisateur. Nous avons porté un effort particulier sur l'analyse et la conception logicielle afin de répondre à la problématique de la manière la plus efficace. Nous avons pu améliorer NLQuery en ajoutant d'autres types de questions comme les « Yes/No » questions ainsi que les « order » questions. Aussi nous l'avons rendu plus robuste en enlevant l'ambiguïté quand il le fallait. Nous avons mis au point un système de reconstruction de question à partir des questions déjà répondues par le système.

Ensuite, nous avons pu comparer l'ancienne version de NLQuery avec nos améliorations à l'aide d'un jeu de données. Bien que notre prototype n'en soit qu'à ses balbutiements, les premiers résultats sont très encourageants.

Revenons sur les limites de notre prototype afin de proposer de nouvelles perspectives pouvant l'enrichir et le faire évoluer.

Nous avons choisi comme langue principale l'anglais car c'est une langue universelle et grammaticalement plus simple comparée à d'autres langues. Mais il serait intéressant de le rendre multilingue. Nous nous sommes focalisés sur une seule base de connaissances Wikidata. Il faudrait l'étendre à de nouvelles bases de connaissances et pouvoir les interroger simultanément afin d'avoir

des réponses plus pertinentes. Notre solution est limitée aux requêtes simples mais pas composées. L'idéale serait de pouvoir répondre aux requêtes séparées par l'union, l'intersection ou l'agrégat. Le système pourrait être déployer sur internet pour laisser aux utilisateurs la possibilité d'évaluer les réponses à leurs questions. Cela permettrait d'avoir un niveau de confiance pour chaque réponse. Une autre perspective serait de lier le prototype à un système de reconnaissance vocale. A l'avenir, nous aimerions expérimenter des méthodes pour apprendre de nouvelles règles. Cela pourrait en créant un éditeur de règles permettant à l'utilisateur de préciser les éléments comme le type de la question, le sujet ou encore la propriété.

## Références

- [1] QALD : Question Answering over Linked Data, February 2019. original-date : 2015-03-12T11 :21 :39Z.
- [2] Quepy : A python framework to transform natural language questions to queries in a database query language, March 2019. original-date : 2012-12-03T15 :46 :14Z.
- [3] Resource Description Framework (RDF), April 2019. Page Version ID : 158503533.
- [4] SimpleQuestions dataset to Wikidata, March 2019. original-date : 2017-06-29T12 :58 :40Z.
- [5] Takuto Asakura, Jin-Dong Kim, Yasunori Yamamoto, Yuka Tateisi, and Toshihisa Takagi. A Quantitative Evaluation of Natural Language Question Interpretation for Question Answering Systems. *arXiv :1809.07485 /cs*, September 2018. arXiv : 1809.07485.
- [6] Dennis Diefenbach, Vanessa Lopez, Kamal Singh, and Pierre Maret. Core Techniques of Question Answering Systems over Knowledge Bases : a Survey. *Knowledge and Information Systems (KAIS)*, September 2017.
- [7] Dennis Diefenbach, Kamal Singh, and Pierre Maret. WDAqua-core1 : A Question Answering service for RDF Knowledge Bases. In *Companion of the The Web Conference 2018*, Lyon, France, April 2018. ACM Press.
- [8] Sébastien Ferré. SQUALL : The expressiveness of SPARQL 1.1 made available as a controlled natural language. *Data and Knowledge Engineering*, 94 :163 – 188, November 2014.
- [9] Lushan Han, Tim Finin, and Anupam Joshi. Schema-free Structured Querying of DBpedia Data. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM '12*, pages 2090–2093, New York, NY, USA, 2012. ACM. event-place : Maui, Hawaii, USA.
- [10] Shuo Han, Lei Zou, Jeffrey Xu Yu, and Dongyan Zhao. Keyword Search on RDF Graphs - A Query Graph Assembly Approach. April 2017.
- [11] Giuseppe M. Mazzeo and Carlo Zaniolo. Answering Controlled Natural Language Questions on RDF Knowledge Bases. In *EDBT*, 2016.
- [12] Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. Passage de la langue naturelle à une requête SPARQL dans le système SWIP.



In *IC - 24èmes Journées francophones d'Ingénierie des Connaissances*,  
Lille, France, July 2013.

- [13] S. Shaik, Prathyusha Kanakam, S. Mahaboob Hussain, and Dr U. Suryanarayana. Transforming Natural Language Query to SPARQL for Semantic Information Retrieval. 2016.
- [14] Michael Young. Natural Language Engine on WikiData. Contribute to ayounprogrammer/nlquery development by creating an account on GitHub, February 2019. original-date : 2016-08-17T09 :47 :40Z.
- [15] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over RDF : a graph data driven approach. In *SIGMOD Conference*, 2014.