Une autocomplétion générique de SPARQL dans un contexte multi-services

Karima Rafes
BorderCloud
Université Paris-Saclay
karima.rafes@bordercloud.com

Sarah Cohen-Boulakia
LRI, Université Paris-Sud, CNRS
Université Paris-Saclay
cohen@lri.fr

Serge Abiteboul
Inria Paris
Ecole Normale Supérieure
serge.abiteboul@inria.fr

ABSTRACT

SPARQL s'est imposé comme le langage de requêtes le plus utilisé pour accéder aux masses de données RDF disponibles sur le Web. Néanmoins, rédiger une requête en SPARQL peut se révéler fastidieux, y compris pour des utilisateurs expérimentés. Cela tient souvent d'une maîtrise imparfaite par l'utilisateur des ontologies impliquées pour décrire les connaissances. Pour pallier ce problème, un nombre croissant d'éditeurs de requêtes SPARQL proposent des fonctionnalités d'autocomplétion qui restent limitées car souvent associées à un unique champ et toujours associées à un service SPARQL fixé.

Dans cet article, nous démontrons, au travers d'une expérimentation, une approche permettant de proposer des complétions d'une requête en cours de rédaction en exploitant de nombreux types d'autocomplétion et ce dans un contexte multi-services. Cette expérimentation s'appuie sur un éditeur SPARQL auquel nous avons ajouté des mécanismes d'autocomplétion qui supportent une ontologie en perpétuelle évolution, ici avec la base de connaissances collaborative de Wikidata.

CCS CONCEPTS

• **Information systems → Service discovery and interfaces**; **Query suggestion**;

KEYWORDS

SPARQL, Autocompletion, Linked Data, Wikidata

1 INTRODUCTION

Le volume de données RDF disponibles sur le Web est en constante augmentation. L'exploitation de la richesse de ces données passe par leur interrogation. SPARQL s'est aujourd'hui imposé comme le langage de requêtes pour les données du Web. Cependant, rédiger une requête SPARQL peut se révéler fastidieux, y compris pour des utilisateurs expérimentés, et ce pour plusieurs raisons, incluant la maîtrise imparfaite par l'utilisateur des ontologies impliquées pour décrire les connaissances, la nécessité de suivre une syntaxe souvent très lourde, et la difficulté à réutiliser une requête écrite par un tiers.

La réponse à ces problèmes se concentre trop souvent sur l'implémentation d'interfaces qui masquent le langage d'interrogation, par exemple au travers de mots-clés, de questions en langage naturel, de pseudo-requêtes, de requêtes schématiques ou construites à partir d'exemples de résultats attendus. L'inconvénient de ces méthodes est double pour SPARQL. D'abord, l'utilisateur ne progresse pas dans l'apprentissage du langage et par conséquent, il n'est pas en mesure d'améliorer ses compétences au rythme des progrès des applications qui l'intéressent, ou de l'augmentation des données disponibles. Ensuite, l'utilisateur est limité dans sa recherche non seulement par l'ontologie supportée par l'interface mais aussi et surtout par les fonctions SPARQL supportées par l'interface.

La solution que nous proposons permet aux utilisateurs d'exploiter toute la puissance et l'expressivité de SPARQL tout en guidant l'utilisateur dans la rédaction de sa requête, en lui proposant des mécanismes d'*autocomplétion*. Bien que plusieurs éditeurs de requêtes SPARQL proposent depuis peu des fonctionnalités d'autocomplétion, chacune est limitée à quelques rares facettes spécifiques et elles ne portent que sur des requêtes s'adressant à un service SPARQL unique.

Le point de départ de cet article est une étude que nous avons menée auprès d'utilisateurs d'éditeurs SPARQL, lors de la rédaction de requêtes. Cette étude a été effectuée dans le cadre de l'utilisation de la plate-forme LinkedWiki [9] par différents types d'utilisateurs des plus novices aux plus experts dans leur utilisation de SPARQL, et issus de diverses disciplines (informatique, biologie, chimie, physique, science humaines et sociales...).

Plus précisément, nous présentons deux contributions. Nous proposons une première taxonomie des fonctionnalités d'autocomplétion attendues par nos utilisateurs (Section 2) et l'utilisons pour dessiner un état de l'art du domaine de l'autocomplétion SPARQL (Section 3). Nous présentons ensuite une expérimentation utilisant l'autocomplétion en nous appuyant sur un éditeur SPARQL pour démontrer que ces nouvelles fonctionnalités facilitent l'utilisation des ontologies en perpétuelles évolutions, et ce dans un contexte multi-services, ici avec la base de connaissances collaborative de Wikidata (Section 4).

2 BESOINS

Nous avons réalisé une étude des besoins en autocomplétion en collectant, durant un an, les souhaits d'utilisateurs aussi bien novices qu'experts. Cette expérimentation a été réalisée auprès de deux types d'utilisateurs : 103 étudiants issus de formations variées (en informatique, en administration des entreprises, du L3 au M2), dans le cadre de travaux pratiques de SPARQL et 60 professionnels au travers de formations internes et événements proposées au sein du *Center for Data Science* de l'Université Paris-Saclay.

Dans cette section, nous définissons d'abord les concepts essentiels relatifs à l'autocomplétion pour SPARQL. Nous présentons ensuite des besoins en terme de fonctionnalités d'autocomplétion attendues d'un éditeur SPARQL organisés suivant une taxonomie que nous proposons.

Dans l'ensemble des définitions suivantes, on se place dans le cadre de l'utilisation d'un éditeur SPARQL pour la rédaction d'une requête.

Un *service SPARQL* est le service délivré par une base de données RDF au travers du protocole SPARQL et qui va répondre à une requête décrite au sein d'un éditeur SPARQL. L'adresse Web ou URL d'un service SPARQL se nomme son *endpoint* SPARQL. Un service SPARQL a la possibilité de soumettre une sous-requête à un autre service SPARQL en spécifiant son endpoint. On parlera alors de requête fédérée et de contexte multi-services. Il est à noter que les éditeurs existants ne considèrent la connexion qu'à un service SPARQL à la fois. Un service SPARQL peut contenir plusieurs *graphes de données*. Chaque graphe possède un nom unique. Le langage SPARQL nomme au moyen d'*IRI* (*Internationalized Resource Identifier*) toutes ces adresses : *endpoint*, nom de graphe, identifiant d'une donnée dans ce graphe, adresse d'une page, etc. L'*autocomplétion* (dans notre contexte) est une fonctionnalité d'un éditeur de code SPARQL qui, sollicité par l'utilisateur, propose à ce dernier différentes façons de compléter la requête en cours de rédaction (cf. Figure 1).

Nous proposons maintenant une taxonomie des autocomplétions pour SPARQL fondée sur les retours de nos utilisateurs.

Autocomplétion syntaxique. Elle propose à l'utilisateur des mots clés ou des éléments de syntaxe du langage SPARQL. Elle peut se calculer sans difficulté à l'aide de la notation EBNF du langage SPARQL.

Néanmoins, en pratique, chaque service SPARQL ne supporte pas nécessairement l'ensemble de la syntaxe du langage. Par exemple, le service SPARQL de Wikidata ne supporte pas les requêtes contenant le mot-clé *GRAPH*. En conséquence, un éditeur, selon le service SPARQL auquel il accède, peut proposer des complétions déclenchant des erreurs parce qu'il génère des requêtes SPARQL syntaxiquement correctes mais qui utilisent des éléments de la syntaxe non supportés par le service.

Une autocompletion tenant compte de la syntaxe supportée par un service fait partie des fonctionnalités les plus attendues par les utilisateurs novices et experts. Ce besoin est encore plus fort dans le contexte de requêtes fédérées.

Autocomplétion des variables. Elle propose l'utilisation de variables qui apparaissent déjà dans la requête. Elle doit pouvoir être activée automatiquement par l'éditeur qui détecte que l'utilisateur commence à écrire une variable (commençant par *?*, notation EBNF 'VARNAME') et propose la liste des variables disponibles. Cette autocomplétion est attendue bien que certains développeurs préfèrent faire des copiés-collés.

Autocomplétion du service. C'est l'autocomplétion de l'*endpoint* du service qui résoudra une des sous-requêtes. Elle propose une liste de services SPARQL opérationnels et accessibles à l'utilisateur pour interroger plusieurs bases en une seule requête. Cette autocomplétion nécessite de disposer d'une base de connaissances des services SPARQL disponibles et accessibles par chaque utilisateur. Le service SPARQL qui calcule la requête fédérée doit également autoriser l'utilisateur à effectuer ce type de requête et les autres services doivent respecter le protocole SPARQL.

Il est important de noter que les implémentations du protocole SPARQL concernant les requêtes fédérées restent disparates entre les éditeurs entraînant des problèmes d'interopérabilité. Très peu de bases de données RDF permettent concrètement de rédiger des requêtes fédérées [10] et très peu d'utilisateurs accèdent donc à cette fonctionnalité pourtant fondamentale.

L'autocomplétion du service est donc indispensable pour permettre à l'utilisateur de découvrir les sources de données disponibles.

Autocomplétion du graphe. Cette autocomplétion permet de choisir un graphe (notation 'GraphRef' et 'NamedGraphClause') sur lequel portera la (sous-)requête. Il est aussi à noter que dans certaines implémentations, interroger le graphe par défaut du service revient à interroger tous les graphes du service simultanément alors que dans d'autres implémentations, interroger le graphe par défaut revient à n'interroger qu'un seul graphe. Préciser le nom du graphe permet donc de désambiguïser la requête et d'améliorer le temps de réponse de la requête. Cette fonctionnalité est attendue par les utilisateurs experts.

Autocomplétion des IRIs absolus. Elle permet de proposer une liste d'IRIs absolus (notation EBNF 'IRIREF') en s'appuyant sur l'IRI que l'utilisateur a commencé à écrire. Pour un utilisateur expert, l'utilisation d'un IRI absolu se fait par copié-collé. Pour un novice, écrire un IRI absolu est source d'erreurs. On notera que les utilisateurs utilisent surtout des IRIs relatifs, et le besoin de ce type de complétion est donc relativement faible.

Autocomplétion des IRIs relatifs via préfixe. Elle permet la complétion d'IRI relatifs (notation EBNF 'PrefixedName'). Dans la spécification d'une ontologie (décrite au moyen d'IRIs), un préfixe est toujours suggéré. L'utilisateur expert exploite souvent ce préfixe. L'autocomplétion des IRIs relatifs via préfixe propose alors une liste de préfixes à partir de quelques caractères puis, après sélection du préfixe, elle propose les IRIs associés à ce préfixe (types, propriétés...). Cette fonctionnalité est demandée en particulier par les utilisateurs experts.

Autocomplétion des IRIs relatifs via mots clés. Cette autocomplétion d'IRIs, à la différence des précédentes, ne présuppose aucune connaissance préalable du service SPARQL et des ontologies qu'il contient. L'utilisateur choisit des mots clés, dans la langue de son choix, pour obtenir une liste de suggestions d'IRIs relatifs. Il suffit alors d'en choisir une pour que l'outil puisse l'insérer dans la requête en cours avec la définition du préfixe. Ce type de fonctionnalité est particulièrement demandé par nos utilisateurs.

Autocomplétion de la déclaration des préfixes. C'est l'autocomplétion qui insère les déclarations des préfixes non explicitement spécifiés par l'utilisateur au sein d'une requête. Elle répond à un besoin récurrent rencontré par les utilisateurs qui réutilisent des exemples de requêtes disponibles sur le Web dans un nouveau contexte où les préfixes doivent être explicités. Cette fonctionnalité est très demandée par l'ensemble de nos utilisateurs.

Autocomplétion par suggestion de snippets. Elle propose à l'utilisateur un *snippet*, c'est-à-dire un morceau de code réutilisable, ici un morceau de code SPARQL. Cette fonctionnalité est très attendue par nos utilisateurs expérimentés et permettrait de guider les novices lors de leurs premières requêtes.

3 ETAT DE L'ART

Dans [11], dix-huit éditeurs textuels pour SPARQL ont été recensés parmi lesquels sept intègrent des fonctionnalités d'autocomplétion. Ils ont servi de base à notre étude. Nous présentons ici

un rapide état de l'art des fonctionnalités d'autocomplétion qui existent au sein de ces éditeurs.

A notre connaissance, aucun éditeur ne propose de solution au problème de l'autocomplétion *syntaxique* capable prendre en compte les différences entre les services SPARQL qui ne supportent pas toute la syntaxe du langage. On peut faire le même constat sur l'autocomplétion *du graphe*, nous n'avons pas observé d'éditeurs en mesure d'optimiser le temps de réponse d'une requête en limitant sa portée à un ou plusieurs graphes nommés au sein du service SPARQL. A contrario, l'autocomplétion *des variables* est une fonctionnalité que l'on retrouve dans plusieurs éditeurs.

L'autocomplétion *du service* n'est pas implémentée par les éditeurs qui ne considèrent aujourd'hui qu'un service à la fois. La prise en compte de cette fonctionnalité dans un contexte multi-services nécessiterait la constitution d'une plateforme collaborative pour référencer les services SPARQL. LinkedWiki [9] ou Datahub.io [4] permettraient d'alimenter ce type d'autocomplétion.

Les outils d'autocomplétion des *IRIs relatifs et absolus* utilisent plusieurs types d'approches.

Certains éditeurs exploitent les bases de données référençant toutes les ontologies du Web constituées par des approches comme [1] ou LOV [12]. Les IRIs peuvent aussi être extraits du service SPARQL à interroger, comme le font Flint [8] ou YASQE [11]. D'autres systèmes comme SPACE [6] observent les développeurs d'un service SPARQL (via les logs du service généralement) et le contenu du service pour constituer un système de recommandations d'IRIs. Certaines approches exploitent l'emplacement de l'IRI dans la requête. Le calcul peut alors se faire en temps réel pour chaque service SPARQL [2] ou en constituant à nouveau une unique base de données RDF avec toutes les données que peut utiliser le développeur [3].

La recherche des IRIs par *mots-clés* reste peu mise en œuvre et il n'existe à notre connaissance aucune recherche par mots-clés en langue naturelle.

Enfin, la *suggestion de snippets* a été introduite dans le cadre de langages de requêtes, par exemple pour SQL avec le système *SnipSuggest* [5]. Nous ne connaissons pas de tels outils pour SPARQL. Une approche sur la base du recensement des requêtes SQL précédentes des développeurs est esquissée dans [5].

Il apparaît de cet état de l'art qu'il n'existe pas d'éditeur proposant un panel complet de fonctionnalités d'autocomplétion et capable de traiter des requêtes fédérées (contexte multi-services). Si on peut s'inspirer des approches suivies par certains de ces éditeurs, les limitations des fonctionnalités actuelles sont nombreuses à devoir être surmontées pour permettre leur utilisation en environnement de production. Face aux masses de données RDF disponibles et à leur caractère fortement distribué, l'autocomplétion doit, en particulier, fonctionner dans un cadre multi-services, en isolation (pour protéger les données), et offrir de l'autocomplétion de snippets, comme le proposent typiquement les éditeurs de code de type professionnel (ne gérant pas du code SPARQL).

Avant d'implémenter l'infrastructure générique et multi-services nécessaire pour supporter toutes ces autocomplétions pour SPARQL, nous avons vérifié l'intérêt de cette approche au travers de l'expérimentation décrite dans la section suivante.

4 DÉMONSTRATEUR

Dans cette section, nous présentons des expérimentations avec l'autocomplétion pour SPARQL. Les fonctionnalités décrites ici sont disponibles dans les services **io.datascience-paris-saclay.fr** et **LinkedWiki.com**.

Ces expérimentations ont été réalisées dans le contexte de l'interrogation de la base de connaissances Wikidata qui propose un service SPARQL capable d'accéder aux données de tous les projets de la fondation Wikimédia, incluant notamment Wikipédia [7]. Ce service permet d'accéder à des données très volumineuses et supporte chaque jour de manière fiable une charge d'environ 3 millions de requêtes SPARQL [13].

Pour cette expérimentation, nous avons développé trois fonctionnalités spécifiques d'autocomplétion, motivées par des difficultés rencontrées par les utilisateurs des services.

La première fonctionnalité d'autocomplétion que nous avons implémentée est relative à la *déclaration des préfixes*. La Figure 1 présente un exemple issu de la documentation de Wikidata dans laquelle certains préfixes sont implicites. Ainsi, la fonctionnalité d'autocomplétion détecte qu'un préfixe est manquant (a.1) et propose à l'utilisateur le préfixe *wdt*. Un clic de l'utilisateur sur cette proposition déclenche alors l'insertion automatique de sa déclaration dans le code SPARQL (a.2).

La deuxième fonctionnalité d'autocomplétion que nous avons implémentée est une *autocomplétion des IRIs via mots-clés*. Pour utiliser cette fonctionnalité, l'utilisateur commence par spécifier la langue dans laquelle il exprimera ses mots-clés (b.1). L'utilisateur écrit ses mots-clés (ici "maladie infectieuse") dans l'éditeur SPARQL. En réponse, une liste de suggestions d'entités de Wikidata lui est proposée (b.2). La sélection d'une entité déclenche l'insertion d'un IRI relatif dans le code SPARQL (b.3). Pour faciliter la lecture de ces IRIs, le système sait les traduire automatiquement dans la langue sélectionnée en (b.1). De plus, grâce à cette fonctionnalité, l'utilisateur peut vérifier si l'IRI utilisé existe ou non et s'il correspond à la référence souhaitée à l'aide de son label.

La troisième fonctionnalité est une première implémentation d'*autocomplétion par suggestion de snippets* (Figure 2). Lorsque l'utilisateur appuie sur les touches Ctrl+Espace, quatre alternatives de complétion lui sont proposées dont deux (correspondants aux Cas 3 et 4) sont relatives à la proposition d'insertion de snippet. Le Cas 3 propose l'insertion d'un snippet correspondant à l'utilisation du service "label" qui permet à l'utilisateur d'accéder aux entités Wikidata renvoyées en réponse à sa requête non pas sous la forme d'IDs numériques mais exprimées dans la langue spécifiée (ici en anglais, *en* ou si le label de cet ID n'existe pas en anglais, en français, *fr*). Dans le Cas 4, le système propose l'insertion d'un snippet correspondant au filtre SPARQL langMatches qui permet de sélectionner la langue dans laquelle les entités Wikidata seront exprimées (ici en français, *fr*).

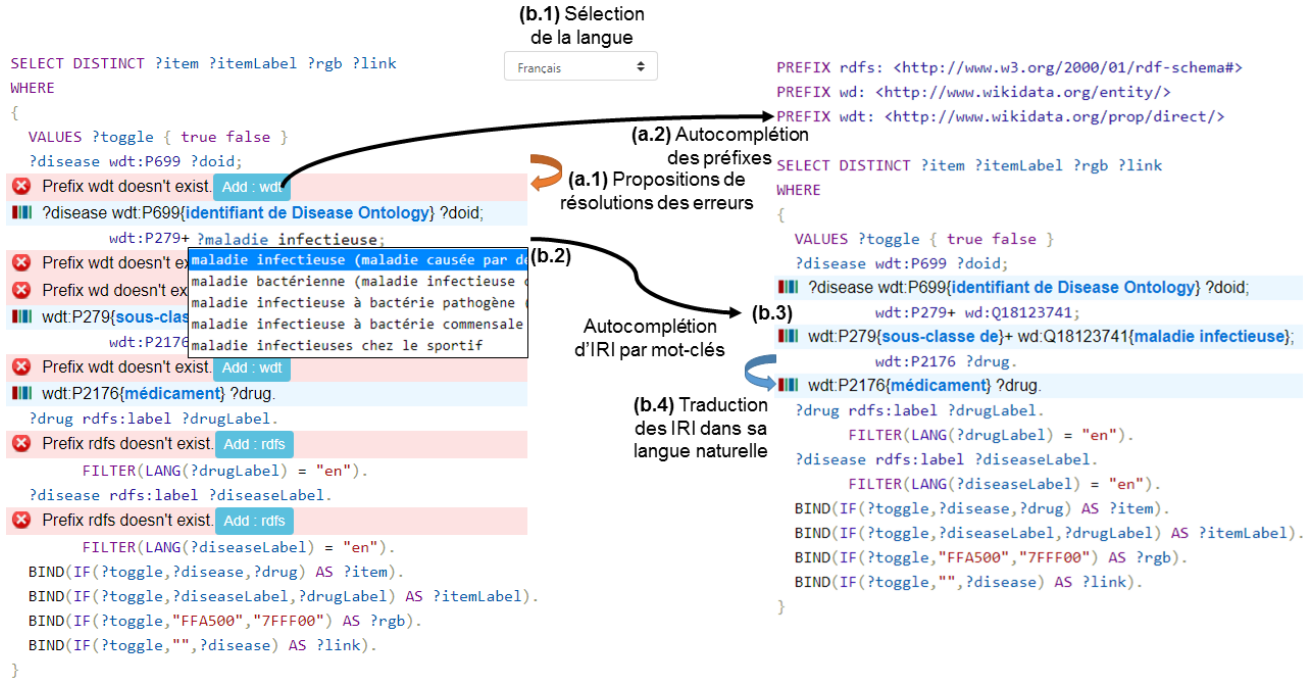

Figure 1: Démonstrateur de la plateforme LinkedWiki (http ://io.datascience-paris-saclay.fr) : (a) autocomplétion des préfixes inconnus dans une requête, (b) insertion de lignes explicitant la signification des IRIs dans en langue naturelle (c) autocomplétion par mots-clés dans la langue de l'utilisateur, ici pour Wikidata.

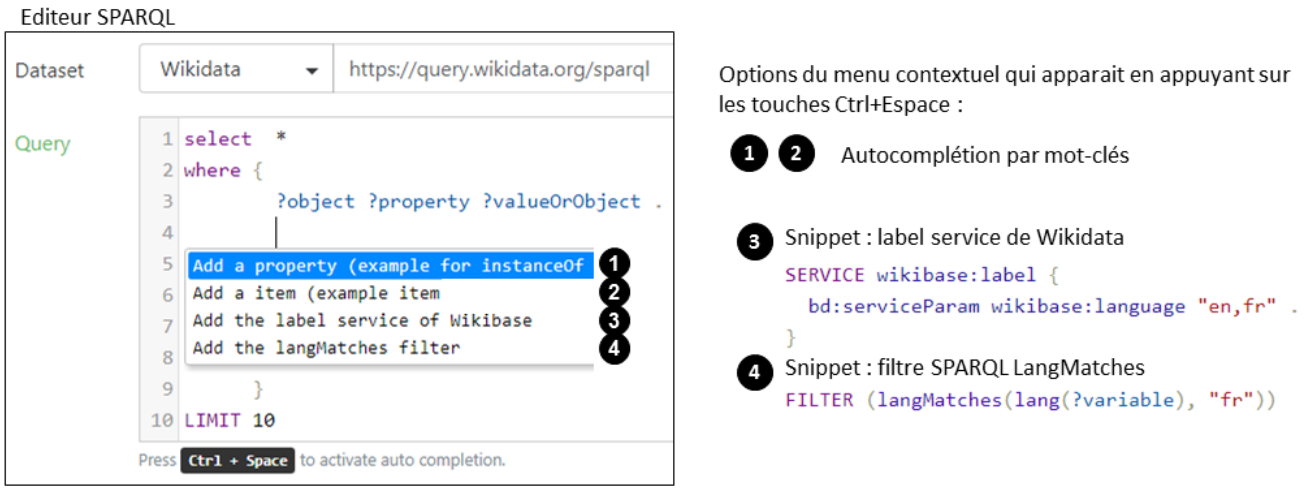

Figure 2: Editeur SPARQL de la plateforme LinkedWiki (http ://io.datascience-paris-saclay.fr) au moment où l'utilisateur choisit entre une autocomplétion par mots-clés pour rechercher (1) une propriété ou (2) un type ou une instance ou bien pour rechercher un snippet pour (3) appeler le service *label* de Wikidata ou pour (4) écrire le code SPARQL pour choisir un tag *lang*.

Le snippet du Cas 3 est pratique pour les utilisateurs de Wikidata mais il n'est supporté que par le service SPARQL de Wikidata (syntaxe non officielle de SPARQL 1.1). Dans un contexte multi-services, les utilisateurs ne peuvent utiliser que le Cas 4 pour afficher les labels de Wikidata au sein d'une requête fédérée. Ces deux premières classes d'autocomplétion par snippets permettent de simplifier l'écriture de requêtes. Leurs suggestions sont très appréciées des utilisateurs experts comme novices.

La même approche peut être utilisée pour proposer d'autres snippets. Cependant, rajouter au fil de l'eau des snippets va surcharger l'interface de l'utilisateur avec des snippets dont il pourrait ne jamais avoir besoin. Cela nous a conduit à réfléchir à une infrastructure générique pour supporter des fonctionnalités de suggestion de snippets personnalisés en fonction du service SPARQL, de l'utilisateur et de la requête en cours d'écriture, comme déjà considéré pour SQL par le système *SnipSuggest* [5].

```
PREFIX wikibase: <http://wikiba.se/ontology#>
PREFIX psv: <http://www.wikidata.org/prop/statement/value/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX schema: <http://schema.org/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>

select  ?lat ?long ?label ?description ?link
where {
  {
     ?link a wd:Q1622062 ;
 ?link a wd:Q1622062{college library} ;
             rdfs:label ?label ;
             geo:lat ?lat ;
             geo:long ?long .
    OPTIONAL{
      ?link schema:description ?description .
    }
  }UNION{
     SERVICE <https://query.wikidata.org/sparql> {
          {
            ?type wdt:P279 wd:Q7075 .
 ?type wdt:P279{subclass of} wd:Q7075{library} .
            ?object wdt:P31 ?type .
 ?object wdt:P31{instance of} ?type .
          }
          UNION
          {
             ?object wdt:P31  wd:Q7075 .
 ?object wdt:P31{instance of} wd:Q7075{library} .
          }
          ?object  rdfs:label ?label ;
                   p:P625 ?coord ;
 p:P625{coordinate location} ?coord ;
                   schema:description ?description ;
                   wdt:P856 ?link .
 wdt:P856{official website} ?link .
          ?coord   psv:P625 ?coordValue .
 ?coord psv:P625{coordinate location} ?coordValue .
          ?coordValue a wikibase:GlobecoordinateValue ;
                      wikibase:geoLatitude ?lat ;
                      wikibase:geoLongitude ?long .

          FILTER (langMatches( lang(?label), "en" ) )
    }
  }
}
```

Figure 3: L'éditeur SPARQL de la plateforme LinkedWiki supporte les requêtes fédérées entre Wikidata et la base de connaissances de l'Université Paris-Saclay.

L'exemple de requête en Figure 3 illustre une jointure entre Wikidata et une autre base de connaissances afin d'obtenir le résultat le plus complet possible. Le temps nécessaire à l'écriture de ce type de requête fédérée a été largement réduit grâce aux trois fonctionnalités d'autocomplétion disponibles au travers de notre éditeur multi-services.

Ces fonctionnalités, malgré leur simplicité (il s'agit d'une première expérimentation), ont rencontré un vif succès auprès de nos utilisateurs, en particulier les plus novices, leur facilitant notamment l'écriture de requêtes fédérées entre leurs bases de connaissances et le Web des données.

5 CONCLUSION

Dans cet article, nous avons dressé un premier panorama des besoins en autocomplétion de requêtes SPARQL. Nous avons introduit une taxonomie des fonctionnalités d'autocomplétion attendues de nombreux utilisateurs et développeurs du Web sémantique. Nous avons démontré l'intérêt d'une approche via autocomplétions au travers d'une expérimentation sur une ontologie en constante évolution, la base de connaissances collaborative de Wikidata. Notre expérimentation s'est appuyée sur un éditeur SPARQL au sein duquel nous avons développé plusieurs fonctionnalités d'autocomplétion.

Dans des travaux en cours, nous nous focalisons maintenant sur la conception et l'implémentation d'une infrastructure générique, capable de prendre en compte un grand ensemble de fonctionnalités d'autocomplétion, et ce dans un contexte multi-services, tout particulièrement attendues par les utilisateurs de la platforme LinkedWiki. Le défi est de pouvoir tirer le meilleur parti de toute la connaissance accumulée aux travers des requêtes déjà conçues par les utilisateurs du système.

REMERCIEMENTS

Ce travail est soutenu par le Center for Data Science (CDS), financé par l'IDEX Paris-Saclay, ANR-11-IDEX-0003-02.

RÉFÉRENCES

[1] Aleksandar Andreevski, Riste Stojanov, Milos Jovanovik, and Dimitar Trajanov. 2015. Semantic Web Integration with SPARQL Autocomplete. In *The 12th International Conference on Informatics and Information Technologies*. 1–4.
[2] Stéphane Campinas. 2014. Live SPARQL auto-completion. In *Proceedings of the 2014 International Conference on Posters & Demonstrations Track-Volume 1272*. CEUR-WS. org, 477–480.
[3] Stephane Campinas, Thomas E Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. 2012. Introducing RDF graph summary with application to assisted SPARQL formulation. In *Database and Expert Systems Applications (DEXA), 2012 23rd International Workshop on*. IEEE, 261–266.
[4] Open Knowledge Foundation. 2015. Datahub. (2015).
[5] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. 2010. SnipSuggest : Context-aware autocompletion for SQL. *Proceedings of the VLDB Endowment* 4, 1 (2010), 22–33.
[6] Kasjen Kramer, Renata Dividino, and Gerd Gröner. 2013. Space : Sparql index for efficient autocompletion. In *Proceedings of the 2013th International Conference on Posters & Demonstrations Track-Volume 1035*. CEUR-WS. org, 157–160.
[7] Daniel Mietchen, Gregor Hagedorn, Rafes Karima, Anonymous, and Many others. 2015. Enabling Open Science: Wikidata for Research. (2015). https://doi.org/10.5281/zenodo.13906
[8] TSO (The Stationery Office). 2011. Flint SPARQL editor released into semantic web community. (2011). http://www.tso.co.uk/news/2011/07/flint-sparql-editor-released-semantic-web-community
[9] Karima Rafes and Cécile Germain. 2015. A platform for scientific data sharing. In *BDA2015-Bases de Données Avancées*.
[10] Karima Rafes, Julien Nauroy, and Cécile Germain. 2015. Certifying the interoperability of RDF database systems. In *LDQ 2015-2nd Workshop on Linked Data Quality*. Springer.
[11] Laurens Rietveld and Rinke Hoekstra. 2017. The YASGUI family of SPARQL clients. *Semantic Web* 8, 3 (2017), 373–383.
[12] Pierre-Yves Vandenbussche, Ghislain A Atemezing, María Poveda-Villalón, and Bernard Vatant. 2017. Linked Open Vocabularies (LOV) : a gateway to reusable semantic vocabularies on the Web. *Semantic Web* 8, 3 (2017), 437–452.
[13] Wikimedia. 2017. Wikidata Query Service Metrics. (2017).