ISSN 2007-9737

Comparison of Approaches for Querying Formal Ontologies via Natural Language

Anicet Lepetit-Ondo[1,*], Laurence Capus[1], Mamadou Bousso[2]

[1] Laval University, Department of Computer and Software Engineering, Canada

[2] Thies University, Department of Computer Science, Senegal

anicet-lepetit.ondo1@ulaval.ca, laurence.capus@ift.ulaval.ca, mbousso@univ-thies.sn

Abstract. The Semantic Web relies on the use of ontologies to ensure data sharing, reuse, and interoperability, thereby representing knowledge comprehensible to computers.However, querying ontologies, often performed using the SPARQL language, becomes a challenge, especially for non-expert users. Barriers include linguistic challenges due to syntactic complexity, the need to understand the underlying ontology structure, potential errors in query formulation, and difficulty expressing complex queries. To make knowledge access more user-friendly, this article explores ontology querying in natural language. We propose a reflection aimed at guiding future domain designers in the interrogation of ontologies in natural language, orienting them in their choice of approach according to the applications they will develop. The study relies on the application of Natural Language Processing (NLP) techniques, integrating tools such as Owlready2, RDFLIB, Protégé2000, and the Python programming language to achieve its goals. Three distinct approaches were evaluated for this purpose. The first approach, scenario-based, was tested on two separate ontologies: one related to university concepts and the other to estate settlement. This approach demonstrates remarkable adaptability across various ontologies. However, its effectiveness is closely linked to the types of scenarios and the specific jargon of the evaluated domain. The other two approaches, one based on SPARQL query patterns and the other on decision tree structure, were evaluated on a specific ontology, estate settlement. They show robust performance in terms of result accuracy. Nevertheless, their effectiveness depends on model training for named entity detection, node list management, and enrichment of SPARQL query patterns, operating exclusively within this particular ontology.

Keywords. Ontology, SPARQL language, natural language processing, query patterns, decision trees.

1 Introduction

Formal ontologies are used to represent the knowledge of a domain in a manner understandable to machines [1]. In this context, SPARQL stands out as the preferred language for querying these knowledge representation models [5]. However, due to its syntactic complexity, it can prove challenging for individuals inexperienced in the field to explore all the potential responses that the ontology could offer.

Therefore, the imperative to exploit these data sources in natural language becomes a major challenge to overcome various obstacles, such as linguistic barriers related to the syntactic complexity of the language, the need to understand the underlying structure of the ontology, potential errors in query formulation, and the difficulty of expressing complex queries.

In the scope of this study, we examine three flexible approaches for querying ontological sources to address this significant challenge. In the following sections of this article, we will cover the following aspects.

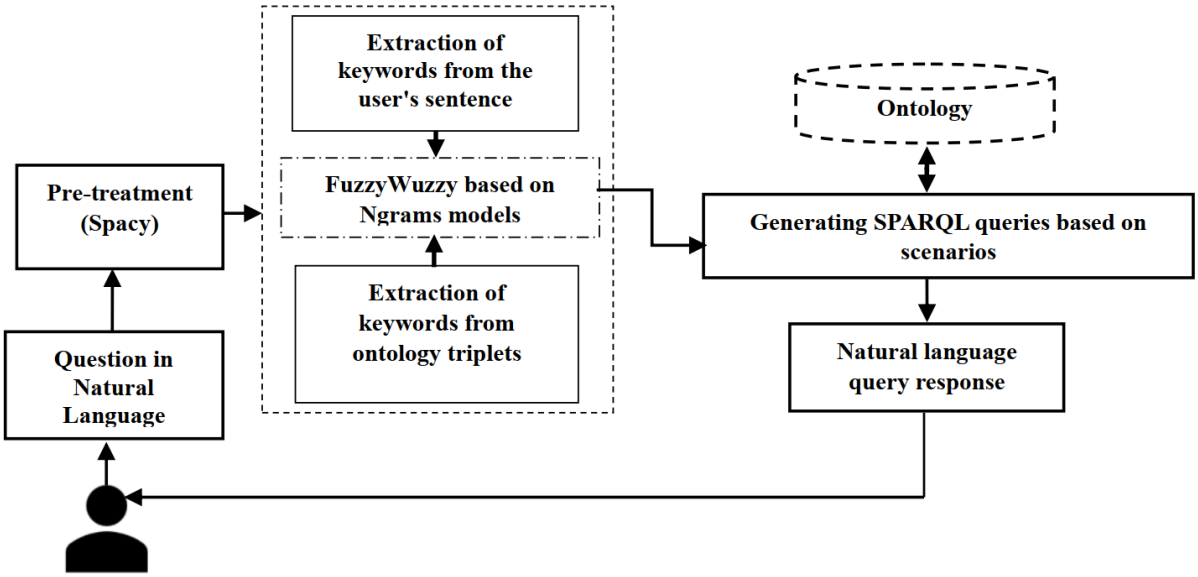


Fig. 1. Scenario-based approach

First, we will review prior work in this field to contextualize our contribution. Subsequently, we will describe our research methodology in detail, including the tools and techniques employed. We will then present our results and analyses. Finally, we will conclude with discussion and perspectives.

2 Related Work

The early studies on SPARQL query generation primarily focus on manual query creation, particularly to assess the ability of ontology systems to cover and infer within a domain, as evidenced by the seminal works [3, 4].

With the aim of reducing the need for manual intervention, the researchers turned their attention to schema-based SPARQL query generation. These templates function as shells, i.e., standardized query forms with placeholders to be filled, as indicated by references [2, 11].

These methods aim to create an intermediate schematic representation of SPARQL queries. Following the schema-based query generation logic, other approaches have been developed to associate specific keywords with a domain to create a query model, as described by Zeng et al. [12]. New ideas for automatic query generation continue to emerge. For instance,Pradel et al. [8] describe a process for translating the user's query, expressed in natural language, into a SPARQL query suitable for querying a knowledge graph.

This method involves several steps: first, the identification of named entities carrying domain information in the user's sentence. Next, an analysis of syntactic dependencies is carried out, followed by the creation of a pivot query. This pivot query explicitly expresses the relationships inferred from the substrings of the user's query.

Since this pivot query is not directly understandable by the graph, the final step involves matching it with predefined query templates (which are well-structured queries for querying an ontology) to obtain a list of potential results for the user's query. An innovative approach was presented that integrates the linguistic aspects of a question with graphical data to optimize the generation of SPARQL queries [9].

SGPT[1] enhances the understanding of natural language questions by using a generative model and a stack of Transformer encoders, facilitating

[1]This abbreviation unveils its full significance when referenced to source [9]

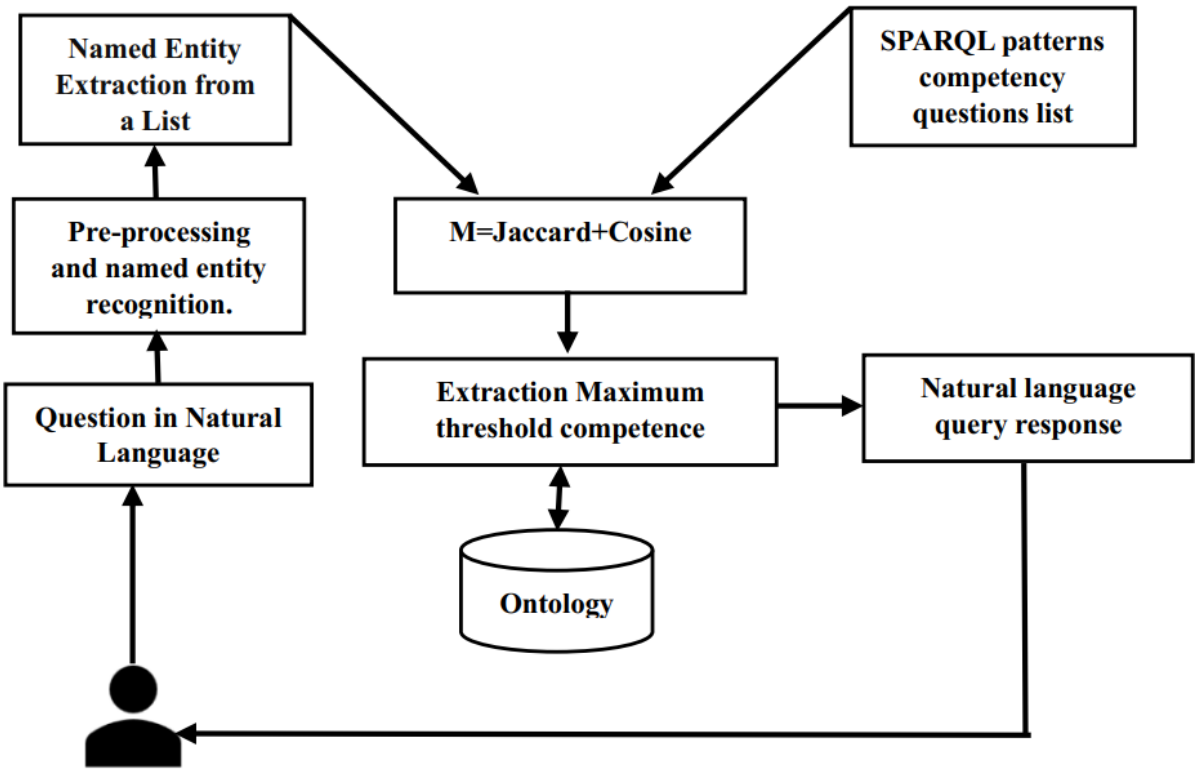


Fig. 2. General architecture of the one approach based on SPARQL query patterns

the injection of additional knowledge into the query generation process, such as entities. The goal is to ensure a more precise and thorough generation of SPARQL queries.

Another approach was proposed to generate queries following a specific methodology. Initially, it extracts named entities from the user's sentence, then, it constructs a dependency tree [13].

Query generation is based on the relationships present in the tree, associating each found entity or predicate with its syntactic equivalent in the ontology to produce a SPARQL query. Having reviewed articles in the field addressing similar approaches, within our framework, we develop three approaches with distinct goals:

Reducing manual interventions for consulting ontologies, enhancing schema-based systems using SPARQL query shell with innovative ideas, as illustrated by the approaches based on SPARQL query patterns and the decision tree structure outlined below, and optimizing more accurate SPARQL query generation to account for the various updates a domain may undergo, as demonstrated by the scenario-based approach.

3 Three Approaches Tested

Querying an ontology in natural language is a complex task. This process requires the integration of various language processing techniques, as well as a language better adapted to ontologies. Querying involves reconciling a language that is close to human understanding with one that can be understood by an ontology. In this section, we describe three different approaches.

The first approach, based on user query scenarios, demonstrates how we can automatically construct a query from the triplets returned by the ontology, based on the similarity of tokens present in the user's question. The second explores SPARQL query models. This approach highlights

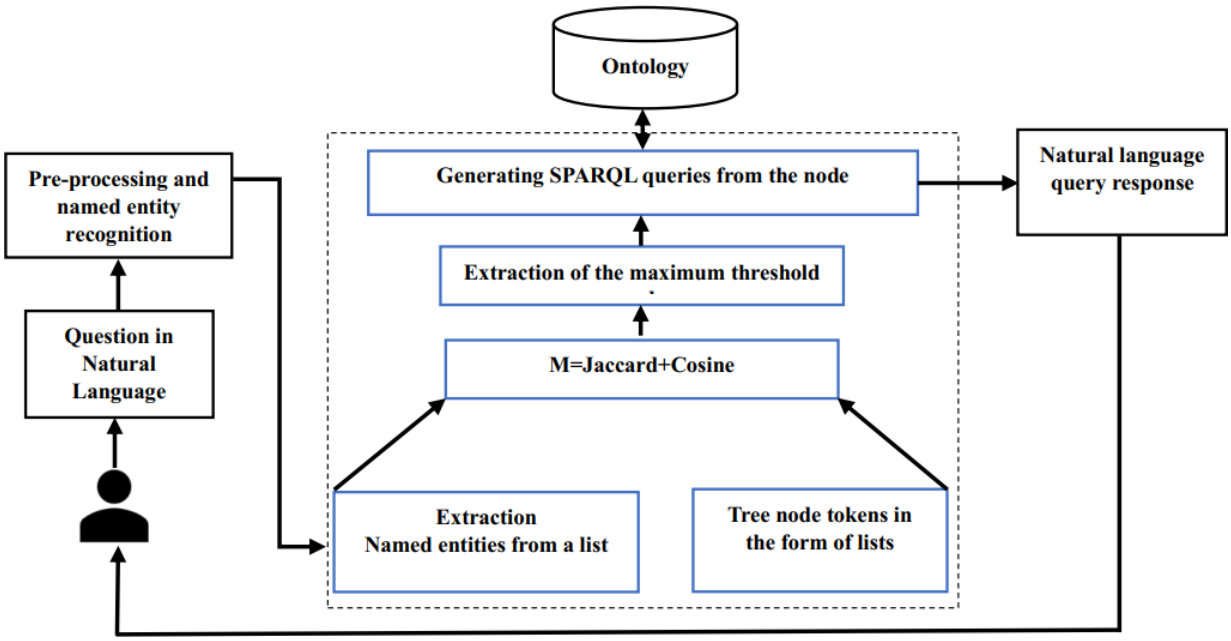


Fig. 3. General architecture of the approach based on a decision tree structure

a mechanism for extracting a query expressing the user's needs from predefined SPARQL query models. The third approach, based on the decision tree structure, describes a graph traversal to reach the potential node containing the information sought by the user.

3.1 Scenario-based Approach

The approach employed is based on extracting various triplet scenarios from an ontology to form a query capable of addressing user needs expressed in natural language. It aligns partially with the logic of the work conducted by Zlatareva and Amin [13]. However, we simplify it by excluding the identification of named entities and the dependency tree, because this entails the need to extensively train our model for each domain represented, since the effectiveness of this approach depends heavily on the ability of the training model to recognize more entities, creating a dependency on the model's performance on a specific dataset. This could also substantially extend the application time in various domains. Given that we have a list of sentence tokens, we instead explore a technique focused on identifying similar triplets in the ontology.

This approach aims to reduce the potential impact of a named entity recognition phase, as the extraction of different triplets already defines relationships between various entities. At this stage, we can automatically generate queries from triplets relevant to the sentence.

It is noteworthy that our approach also shares some similarities with the work of Rony et al. [9], which seeks to overcome challenges related to adding new knowledge to the ontology, avoiding continuous enrichment of predefined query models.

This overcomes constraints of predefined SPARQL query patterns approaches and those based on the decision tree structure. For a better understanding of the idea, let's examine specific scenarios for querying an ontology in Table 1.

Let: $\mathbf{Rg}$ = generated query. $\mathbf{To}$ = Similar triplet(s) extracted from the ontology:

$$\forall t \in T_{0,} = \{s, p, o\}\,, \tag{1}$$

$$\{s = \text{subject},\ p = \text{predicate},\ o = \text{object}\}. \tag{2}$$

Let Lo be a list of identified entities (class, data properties, object properties, and instances) in To, with:

Co = classes identified in To,
Pdo = data properties in To,
Poo = object properties in To,
Io = instances in To.

We obtained :

$$Rg = f(Lo). \tag{3}$$

The approach operates in several phases. Initially, in the first step, it extracts key tokens from the user-provided sentence and stores them in a list. Next, it proceeds to extract triplets from the ontology using the Owlready2 module developed by [6], storing them in a set of lists. Finally, it implements the similarity measurement using the 'fuzz.ratio()' function,elaborated by Kumari et al. [7], integrated into the library fuzzywuzzy[2].

This function is based on the Levenshtein algorithm (edit distance),it assesses the similarity between the character strings present in the ontology's triplet lists and the tokens extracted from the user's expressed need. To better understand the process of calculating this distance, let's consider the position indices i and j in the two strings between which we seek to apply Levenshtein's distance: Deducting the first two values by :

$$\mathbf{d}(0,0) = 0, \tag{4}$$

$$\mathbf{d}(0,\mathbf{i}) = \mathbf{d}(\mathbf{i},0) = \mathbf{i}. \tag{5}$$

And we have the recurrence:

$$\mathbf{d}(i,\ j) = \min \begin{cases} \mathbf{d}(i-1,j)+1, & \\ \mathbf{d}(i,j-1)+1 & \text{si} \quad c(i) \neq c(j), \\ \mathbf{d}(i-1,j-1) & \text{si} \quad c(i) = c(j). \end{cases} \tag{6}$$

Taking into consideration the nomenclature of entities in our ontology, which includes class names such as "area_of_expertise" or simply "area",we have implemented a method for extracting individual words or relevant adjacent words in the context of the ontology using unigram, bigram, and trigram groupings.

The goal of employing these n-gram models is to increase the chances of finding similar values in the dataset, even in the presence of slight variations in user input. This is primarily aimed at optimizing similarity search by taking into account partial matches or adjacent words.

Following the first phase, we obtain one or more lists of similar triplets extracted from the ontology. The second phase consists of generating a SPARQL query. The query generated will depend on the types of entities present in our list of similar triplets in our ontology. Figure 1 illustrates the scenario-based approach to user queries.

3.2 Approach based on SPARQL Query Patterns

The process begins with a named entity recognition phase, clarifying the user's research domain. Named entities play a crucial role, preventing misunderstandings and dispelling confusion between the user's query and the associated knowledge domain. We employed named entity recognition on a corpus of 300 to 500 questions, manually processed using the Doccano[3] software . The data was converted into JSON format, and a SpaCy [4] model was trained to recognize entities in user questions.

This approach is based on the principles outlined by Pradel et al. [8], but we simplify it by focusing exclusively on the named entities identified in the user's query. These entities contain crucial information regarding the user's intent. The main goal is to provide the user with a response based on their formulated query. To achieve this, the user's question undergoes a preprocessing pipeline to be associated with the corresponding SPARQL query in the predefined query model.

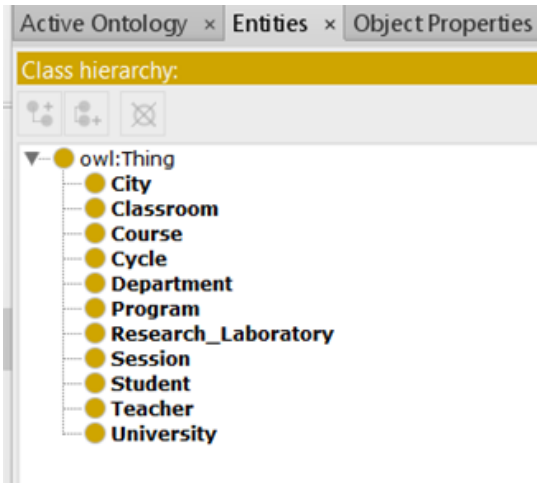
The question undergoes various preprocessing steps with SpaCy and NLTK[5], including normalization and named entity recognition. Then, the construction of extracted token lists will be performed after these treatments. We

[2] pypi.org/project/fuzzywuzzy/

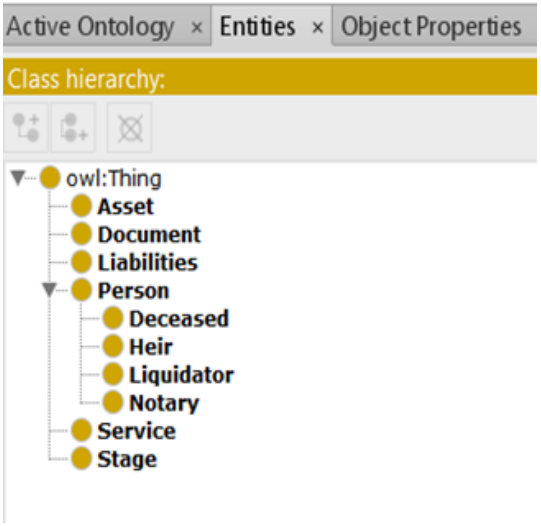
[3] https://doccano.github.io/doccano/

[4] realpython.com/natural-language-processing-spacy-python/

[5] realpython.com/nltk-nlp-python/



(a) University ontology



(b) Ontology of estate liquidation

Fig. 4. Ontology artefacts

limit ourselves to using named entities without considering their type. For example, for the named entity 'Notaire,' we take the identified value 'notaire' as the named entity, rather than its type, which could be 'personne'.

To make this intent understandable for our ontology, we identified a maximum set of questions that the ontology could answer. These questions, often referred to as domain competency questions [10], were then translated into SPARQL language and integrated into a Python file.

At this stage, we applied a technique based on similarity aggregation to predict the most suitable query class that aligns with the user's formulated need. To achieve this, our approach relies on the similarity between two token lists: one containing named entities considered as keywords representing the user's intent, and the other based on keywords, particularly the variable nomenclature in SPARQL queries.

We aggregated the Jaccard similarity[6] metric with the Cosine similarity[7], assigning weights to the measures based on their relative importance and calculated a weighted average. This is done to capture a more accurate semantic similarity between our two lists. We followed this procedure:

– **Phase 1:** Let A and B be two sets formed from our lists. We obtain our Jaccard similarity using the following formula. Similarity of Jaccard:

$$\text{Jaccard} = \frac{A \cap B}{A \cup B}. \tag{7}$$

– **Phase 2:** Let u and v be two vector representations formed from our lists. We obtain our Cosine similarity using the following formula: The cosine similarity between two vectors $\mathbf{u}$ and $\mathbf{v}$ in the FastText embeddings space is given by cosine similarity:

$$(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}, \tag{8}$$

where $\cdot$ represents the dot product between the vectors, and $\|\mathbf{v}\|$ represents the Euclidean norm of vector $\mathbf{v}$.

– **Phase 3:** In the end, we obtain our weighted average:

$$\text{Final Similarity} = w_1 \cdot \text{Jaccard Sim} + w_2 \cdot \text{Cosine Sim} \tag{9}$$

With w_1 and w_2 being the associated weights, such that $w_1 + w_2 = 1$. After applying this average, we extract the SPARQL query with a high similarity value. This query is then executed with our ontology, and the corresponding response is sent back to the user. Figure 2 illustrates our approach based on SPARQL query patterns.

[6] www.learndatasci.com/glossary/jaccard-similarity/
[7] www.learndatasci.com/glossary/cosine-similarity/

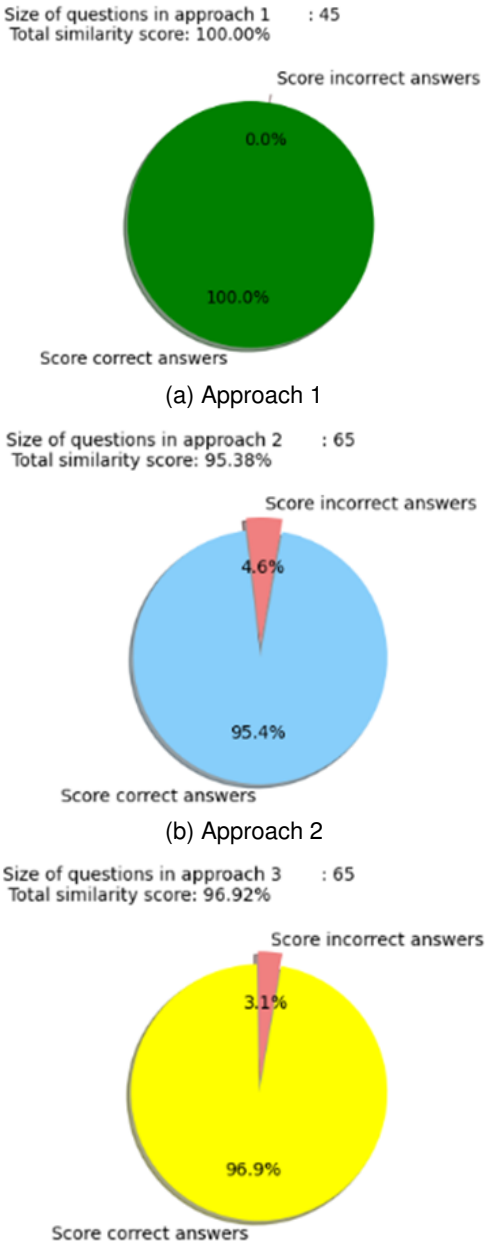
Table 1. Some ontology query scenarios

Type of Question	Possible scenarios (SC)	SPARQL Queries Generated
Unary questions: Questions involving a single constraint or parameter. They generally ask for the list of instances of a class.	Sc1: Instance values of classes, we have: $\mathbf{R_G} = \mathbf{f(Co)}$	`SELECT ?Var WHERE{?Var rdf:type table:Co.}`
Binary questions: Questions involving two constraints or parameters. They can include queries with specific property values or relations between classes.	Sc2: List of instances of a class with their respective values in a specific property. We have: $\mathbf{R_g} = \mathbf{f(Co,\ Pdo)}$	`SELECT ?Var1 ?Var2 WHERE {?Var1 rdf:type table:Co. ?Var1 table:Pdo1 ?Var2.}`
	Sc3: An specific instance with a particular property value. This corresponds to: $\mathbf{R_g} = \mathbf{f(Co,\ Pdo,\ Io)}$	`SELECT ?Var1 ?Var2 WHERE {?Var1 rdf:type table:Co.?Var1 table:Pdo1 ?Var2. FILTER (?Var1 = table:Io)}`
	Sc4: The instances of a class with the specific relationship they have with the instances of another class. We have: $\mathbf{R_g} = \mathbf{f(Co1,\ Co2,\ Poo)}$	`request Sc2(substituting Pdo by Poo) + UNION {{? Var2 rdf:type table:Co2. ?var2 table:Poo ? Var2.`
	Sc5: The specific instance of a class with the specific relationship it has with the instances of another class. It gives us: $\mathbf{Rg} = \mathbf{f(Co1,\ Co2,\ Poo,\ Io)}$	`SELECT ? var1? var2 WHERE {{{{Var1 rdf:type table:Co1. ? Var1 table:Poo ?Var1. FILTER (?var1 = table:Io).} }} UNION {{ ? Var2 rdf:type table: Co2.?var2 table:Poo ?Var2. FILTER (?var2 = table:Io).} }}`
Ternary Questions: Questions involving three constraints or parameters. They can include queries with two instance-specific property values.	Sc6: Obtaining the instances of a class with two specific property values. This corresponds to: $\mathbf{Rg} = \mathbf{f(Co,\ Pdo1,\ Pdo2)}$	`SELECT ?Var1 ?Var2 ?Var3 WHERE { ?Var1 rdf:type table:Co. ?Var1 table:Pdo1 ?Var2. ?Var1 table:Pdo2 ?Var3.}`
	Sc7: Obtaining the two specific property values of a particular instance of a class. This corresponds to: $\mathbf{Rg} = \mathbf{f(Co,\ Pdo1,\ Pdo2,\ Io)}$	`request Sc6 FILTER (?Var1 = table:Io)`

3.3 The Approach based on a Decision Tree Structure

As with the pattern-based approach, we began with a named entity recognition phase, thereby clarifying the user's research domain. This approach provides a progressive method to present the problem to the user, moving from generality to specificity. It allows starting from the user's expressed intent in their query to present a central point of the problem, from which multiple branches emerge, representing specific ideas that can be deduced.

Although this approach is inspired by different



(a) Approach 1

(b) Approach 2

(c) Approach 3

Fig. 5. Evaluation of approaches

methodologies of [2, 11], the schema-based one, it nonetheless exhibits a very distinctive operation. Its principle is as follows: we construct a tree composed of different nodes in the form of token lists, each representing a subgraph of the represented domain.

The decisions to be made depend on pre-defined conditions or criteria based on the maximum similarity values obtained between specific entities in the query and the nodes of the tree. Each branch connecting the nodes to the maximum similarity indicates the path of decisions to present to the user.

Just like in the pattern-based SPARQL query approach, Jaccard and cosine metrics were used for this purpose, where the output represents a weighted sum of the obtained similarity scores. Regarding query generation, we establish a predefined SPARQL query model representing the general structure of a formal SPARQL query.

Then, we assign the necessary information to each node of the graph to complete this structure so that it can query our ontology. Consequently, the finalized query output will depend on the selected node branch. Figure 3 illustrates the approach based on a decision tree structure.

4 Experiments

To test our approaches and assess their effectiveness in a real-time user-friendly environment, we practically implemented them. Specifically, we utilized a conversational bot based on Django [8] technology with Python.

Thus, experiments were conducted on two distinct datasets: one focusing on an ontology related to the liquidation of an estate, and the other on a university ontology. This section introduces the ontology artifacts used to test our approaches and outlines the various conducted tests.

4.1 Ontology Artefacts

In order to carry out our experiments, it was essential to test our approaches on a dataset adhering to the design standards of a formal ontology describing a specific domain.

To demonstrate the generality of our first approach, we chose to implement two distinct ontologies: one focusing on university concepts and the other on estate liquidation.

[8] dev.to/documatic/build-a-chatbot-using-python-django-46hb

The creation of these two artifacts was done using the Protégé2000[9] tool from Stanford University for graphical ontology creation, as well as Lamy's Owlready2 library [6], enabling the editing of our ontologies using the Python language. To verify the coherence of our ontologies, we used the Hemit reasoner integrated into the Protégé 2000 tool. Our two test ontologies are illustrated in Figures 4- (a)(b).

4.2 Test and Results

We evaluated our three approaches on two distinct ontologies using domain-specific competency questions extracted from forums discussing the two separate subjects. The results presented in this section were obtained using the estate settlement ontology, while the university ontology was used solely to assess the versatility of our first approach. The process involved:

For our first approach, we employed a total of 45 questions. Some were reformulated in two or three different ways, while others were unique. The test initially involved creating a dataset containing 45 new question reformulations, each associated with a desired SPARQL query, manually generated.

Then, using these questions, we let the model automatically generate its own SPARQL queries, based on its understanding, into a separate JSON file. Finally, we evaluated both the manually defined desired queries and those automatically generated by our approach.

For our second approach, we used a total of 65 questions. Some of these questions were reformulated in two or three different ways, while others were unique.

The test involved first manually creating a test file containing these 65 questions, each accompanied by its desired SPARQL query. Then, we let the model generate its own SPARQL queries into a separate JSON file for different needs. Finally, we evaluated the model's ability to correctly map the right queries across all 65 questions based on the previously established desired SPARQL queries.

For our third approach, we worked with a total of 65 questions. Some of these questions were reformulated in two or three different ways, while others were unique. The goal of the test was to evaluate whether each question posed to the chatbot could be correctly associated with the right node in our decision tree.

Given that our objective was to assess the model's ability to generate an appropriate SPARQL query or select the correct decision tree node for the entire set of questions, we utilized recall as the primary evaluation metric. We also highlight that the approach employed in designing your ontology involved defining competency questions to which our ontology should respond.

With that in mind, we evaluate the returned responses across the entire set of responses deemed positive. The results obtained are presented in Figures 5 (a, b, and c). As an illustration of how our chatbot effectively responded to questions, we present in Figure 6(a and b) a few screenshots of the application.

5 Discussion and Perspectives

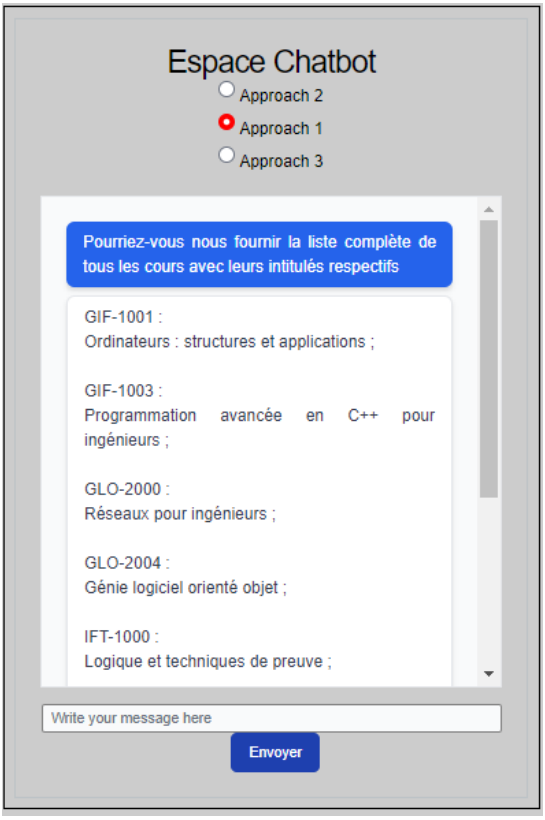
Our contribution revolves around several innovative aspects. Firstly, we have developed a straightforward method for matching SPARQL queries to predefined patterns based solely on named entities extracted from the sentence. Indeed, these entities already carry the essential information needed for query extraction, thus avoiding the complexity of a syntax tree.

Additionally, we propose a dynamic method for automatically generating SPARQL queries by extracting named entities from a sentence and utilizing the syntactic values of keywords returned by RDF triple lists from the ontology.

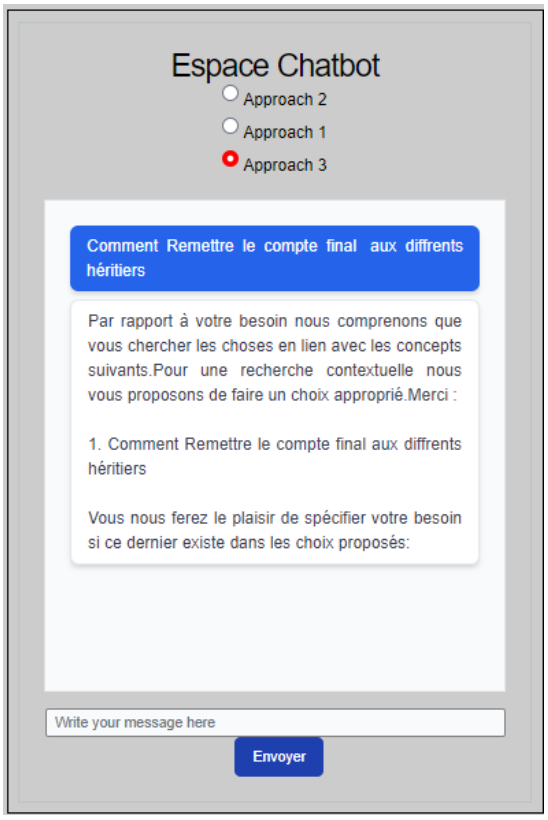
This approach allows for flexibility in application across different ontologies, as the formed queries are not predefined, thereby offering adaptability to various knowledge contexts. Lastly, we leverage an innovative approach that segments a knowledge domain into a decision tree format, where each node represents the needs of a subgraph of knowledge, thus providing intuitive interaction with knowledge bases.

The results obtained from the tested approaches are promising, demonstrating very high precision (Figure 5). The three tested

[9] protege.stanford.edu/



(a) Approach 1



(b) Approach 3

Fig. 6. Chatbot test

approaches each have their advantages and limitations. However, due to the frequent updates that a knowledge domain may undergo, the pattern-based and decision tree-based approaches would take longer to integrate new updates into the ontology.

Indeed, because of their less adaptable structures, each new piece of knowledge added to the ontology would require additional enrichment of query patterns for the pattern-based approach, as well as the addition of more nodes to address new requirements in the case of the decision tree-based approach.

It is important to highlight that pattern-based and decision tree-based approaches show a marked sensitivity to specific elements, such as model training for accurate named entity management, and their effectiveness in the context of a particular ontology.

The significant limitation associated with the continuous updates that a knowledge domain may undergo, observed in the two previous approaches, further underscores the relevance of the scenario-based approach in this context.

This approach has proven to be perfectly suited to address questions related to two distinct ontologies, aligning its responses with the logic of the scenarios used to validate our method. In contrast to the pattern-based SPARQL query approach and the decision tree-based approach, which focus on a specific ontology, the scenario-based approach excels across different ontologies.

Despite the remarkable performance observed for this approach, its effectiveness is closely tied

to the types of scenarios and the specific jargon of the evaluated domains. Therefore, our goal is to further refine more complex scenarios to address more demanding questions, such as those that link one entity to several others in the ontology.

Our current studies aim to significantly enhance our approach based on predefined SPARQL query patterns to automatically generate queries after a learning phase on new knowledge.

To achieve this goal, we have initiated a series of tests integrating two Transformers models, T5 and GPT2. This will enable us to perform optimal classification and generation of the user's query formulated in SPARQL language, while minimizing errors resulting from linguistic variations present in user queries, thus reducing rigid dependency on predefined patterns.

6 Conclusion

In our article, the main objective was to test three approaches designed to facilitate querying ontologies in natural language. This initiative stems from the desire to simplify access for users seeking to explore ontologies, avoiding the complexity of SPARQL, commonly used to query these semantic databases. The implementation of these approaches required the application of natural language processing techniques.

The approaches we deployed demonstrated robust performance in terms of result accuracy. The scenario-based approach stands out for its ability to query any ontological source, but its effectiveness heavily depends on the types of scenarios and the nomenclature of entities present in the ontology.

On the other hand, the two latter approaches (pattern-based and decision tree-based) seamlessly map each question to its corresponding category in predefined SPARQL query patterns or the associated node in the tree, respectively.

Whether formulating the question with or without the use of domain-specific synonyms, they respectively detect the corresponding query in the pattern or identify the node where the user will find the answers to their needs with high precision.

However, their performances are influenced by factors such as model training for named entity detection, the list of nodes, and the enrichment of SPARQL query patterns. Additionally, these approaches operate within a specific ontology.

It's worth noting that our article provides insights to guide future designers in the field toward querying ontologies in natural language, helping them choose an approach based on the applications they will develop.

References

1. **Abderrahmane, K. (2017).** Développement d'un environnement pour l'alignement des ontologies: Une approche à base d'instances. PhD Thesis, Université Ahmed Ben Bella.

2. **Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G. H. L., Lemay, A., Advokaat, N. (2017).** Gmark: Schema-driven generation of graphs and queries. IEEE Transactions on Knowledge and Data Engineering, Vol. 29, No. 4, pp. 856–869. DOI: 10.1109/tkde.2016.2633993.

3. **Guo, Y., Pan, Z., Heflin, J. (2005).** LUBM: A benchmark for owl knowledge base systems. Journal of Web Semantics, Vol. 3, No. 2–3, pp. 158–182. DOI: 10.1016/j.websem.2005.06.005.

4. **Haase, P., Mathäß, T., Ziller, M. (2010).** An evaluation of approaches to federated query processing over linked data. Proceedings of the 6th International Conference on Semantic Systems, pp. 1–9. DOI: 10.1145/1839707.1839713.

5. **Lasolle, N. (2022).** Un système d'interrogation flexible pour le Web sémantique: Application au corpus de la correspondance d'Henri Poincaré. PhD Thesis, Université de Lorraine.

6. **Jean-Baptiste, L. (2021).** Ontologies with python: Programming owl 2.0 ontologies with python and owlready2. Apress. DOI: 10.1007/978-1-4842-6552-9

7. **Kumari, V., Godbole, P., Sharma, Y. (2023).** Automatic subjective answer evaluation. Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods, pp. 289–295. DOI: 10.5220/0011656000003411.
8. **Pradel, C., Haemmerlé, O., Hernandez, N. J. (2013).** Passage de la langue naturelle à une requête SPARQL dans le système SWIP. 24èmes Journées francophones d'Ingénierie des Connaissances.
9. **Rony, M. R. A. H., Kumar, U., Teucher, R., Kovriguina, L., Lehmann, J. (2022).** SGPT: A generative approach for sparql query generation from natural language questions. IEEE Access, Vol. 10, pp. 70712–70723. DOI: 10.1109/access.2022.3188714.
10. **Thiéblin, E., Haemmerlé, O., Trojahn, C. (2021).** Automatic evaluation of complex alignments: an instance-based approach. Semantic Web, Vol. 12, No. 5, pp. 767–787. DOI: 10.3233/sw-210437.
11. **Unger, C., Bühmann, L., Lehmann, J., Ngonga-Ngomo, A., Gerber, D., Cimiano, P. (2012).** Template-based question answering over RDF data. Proceedings of the 21st international conference on World Wide Web, pp. 639–648. DOI: 10.1145/2187836.2187923.
12. **Zenz, G., Zhou, X., Minack, E., Siberski, W., Nejdl, W. (2009).** From keywords to semantic queries—incremental query construction on the semantic web. Journal of Web Semantics, Vol. 7, No. 3, pp. 166–176. DOI: 10.1016/j.websem.2009.07.005.
13. **Zlatareva, N., Amin, D. (2021).** Natural language to SPARQL query builder for semantic web applications. Journal of Machine Intelligence and Data Science, Vol. 2, pp. 44–53. DOI: 10.11159/jmids.2021.006.

Article received on 14/04/2024; accepted on 29/06/2024.
**Corresponding author is Anicet Lepetit-Ondo.*