Towards Transforming Natural Language Queries into SPARQL Queries

Majid Askar [0000-0001-5501-1645]

Assiut University, Egypt
majid.askar@aun.edu.eg

Abstract. Ontology Based Data Access (OBDA) improves knowledge sharing and reuse and grants a set of options to enhance the power of knowledge management. In OBDA systems, user queries can be expressed in SPARQL. This prevents ordinary users from formulating their own queries. For such lay users, it should be possible to express queries in their own languages and terms instead of being forced to learn SPARQL. To enable this, a transformation from a natural language to SPARQL is needed. To cope with such translation, several challenges, such as natural language processing, ontology handling, string matching, and SPARQL query building should be addressed. In this PhD study, we will use natural language processing techniques and investigate the role of user involvement in the query translation process. Also, string-matching algorithms will be used. We will start with the processing of the natural query. Then the mapping between the query entities and corresponding entities in the datasets. The user interaction validating and confirming this mapping should take place. Finally, building the SPARQL query. The proposed approach will be evaluated based on a bench mark by means of query efficiency and accuracy.

Keywords: Knowledge management, OBDA, Natural Language, Query translation.

1 Introduction

Ontology-Based Data Access (OBDA) is concerned with querying data sources in the presence of domain-specific knowledge provided by ontologies [1,2,3]. In OBDA, knowledge resources can be used as a mediator to facilitate the access to different and multiple data sources. In the OBDA paradigm, an ontology defines a high-level global schema of data sources and provides a vocabulary for user queries [2,3,4,5]. OBDA has three main components. First, the data layer contains data sets that the system provides access to, including structured, semi-structured, and unstructured data. Second, the conceptual layer provides the conceptual components, where the end user interacts with the system, including the pre-developed ontologies and query interface. Finally, the mapping layer maintains the links and mappings between concepts and entities from the conceptual layer and data set attributes.

OBDA scheme aims to answer user queries by allowing the access to data in the data layer. For the user to get an answer for her query, it goes through a query processing pipeline. This query processing pipeline is a continuous challenge in the context of OBDA-based systems, where enhancing such process has become a must.

A fundamental feature of knowledge management systems is to provide and allow a way to access data. This can be always achieved through a query-based method. In the context of OBDA systems, query processing is about performing a set of steps on the user query. This is executed against data sources and return results back to the user. Again, in the context of OBDA, the user query usually written in SPARQL as in MASTRO [6] and Ontop [7]. Also, VQS [8] can be used to edit the user query. However, there is some work on natural language to SPARQL translation [9-13], but the user interaction in the intermediate steps is not considered except in [12], and it is not integrated to an existing OBDA system. From the user point of view, using natural language in queries instead of SPARQL is much better. When using natural language any user can write the query, but with SPARQL only expert users can.

1.1 Motivation

In the oil and gas industry, about 30-70% of engineers' time is exhausted in looking for and in evaluating the quality of data [14]. In a practical way, almost all of this time will be saved if the engineers can fire their queries directly (i.e. without an external help or without learning new things). In other words expressing their queries in their own language (natural language) will be better. Another case in Siemens, it usually takes weeks to generate a new SQL query [15]. So, this about generating the query, which indeed costs time (about 2 weeks) and money (IT experts who works on that query). Based on that if we allow the end user to write his own query in his own terms, sure this will save time and money. Two options are available, the user can write his query in SPARQL or in natural language. With SPARQL the user uses his common terms, but still restricted by learning the SPARQL structure. On the other hand, it much easier to the user to use the common language. By the existence of NL to SPARQL translator, three gains are achieved: saving time, saving money, and satisfying the user. To this end, there exist some systems that take a SPARQL query and return the results to the user. So, we decide to build a tool that translate form natural language to SPARQL, and afterword try to integrate it to one or more of the existing systems. The target users of this translator, mainly, will be end users, while expert users are welcomed.

1.2 Research Questions

In this study we are going to find answers to some research questions that could help in the improvement of the query translation process. Through the study we will:

- Investigate the role of user involvement in the query translation process, where and how much effort needed?
- Examine accuracy of the translation based on including / excluding some natural language processing operations, which to add/remove and in what order?
- Inspect the translation accuracy based on enhancement on string matching using word synonyms, where, when to use it and could it help?

2 Related Work

Query translation is used in many systems not only for translation but also for question answering. There are some common steps shown in [9], they are:1) matching keywords from the user query with the ontology items or knowledgebase, 2) query building based on step one, 3) ranking the resulting queries, 4) the user selects the suitable query. According to these common steps, our work goes in line with the work presented in [9-13]. Some of these approaches make enhancements as [10,11,12]. QAKiS [10] used an external resource to enhance the matching. It established matching among question parts and relational textual patterns collected from Wikipedia. Autosparql [11] considered the user interaction, but not directly to enhance the translation. It used the user interaction through the user participation in the learning algorithm by answering "yes"," no" questions. SPARQL query construction based on the structure of the user query is shown in [12]. Where the SPARQL query is obtained using the syntactic structure of the natural language question and by a predefined domain independent expression.

However, our approach enhances the generated SPARQL query by involving the user in the query formulation process. Where the user can confirm / edit the matched entity list. This will happen after getting the matched entities from the matching step (see section 4.1, Concept matcher and user interaction). Also using the singular of plural key words besides the plural key words in the matching could enhance the performance of the translator. Where some processing steps could be skipped if the match is found through singular not the plural (see section 4.1, user query handler).

3 Proposed Approach

In this section we present our proposed approach in which we will employ natural language processing techniques and examine the role of user involvement in the query translation process. Also, string-matching algorithms will be used. Initially, we will begin with the processing of the natural language query. Then process mapping between the query entities and corresponding entities in the datasets. Afterwards, The user interaction validating and confirming this mapping should take place. Finally, building the SPARQL query.

For sure, the first step is to apply some natural language processing operations on the given query. The objective of applying such operations is to make some cleaning and identification on the query tokens. So, we can decide which token to be used now, later, or not used at all (i.e. stop word). These common operations are stop word removing, stemming, will be applied to the user query. Also getting the singular from plural is used. Modifiers that is found in the user query is collected to be processed using the query builder. Date and numbers are not handled yet.

The used ontology is flatten to a database to be used for the concept matching. The keywords and their stems found in the user query is matched against the ontology classes and object / data properties. Actually, the matching is done in one of three phases, and if the match is found in one phase then we skip the rest. The first is the exact match,

the second is the wordnet synonym match, the third is a combined similarity match using Jaro Winkler, edit distance, and n-grams with equal share.

After getting the similarity, presenting the matched items to the user to get his feedback is a key step in the whole translation. The user feedback, for sure, enhance the quality of the produced SPARQL query, by including or excluding items from the matched items. This step is somehow kind of confirmation before continue translating (i.e. Do you mean this?). Intermediate user interaction is not considered before.

At the last step, we use the matched items reviewed by the user to build the SPARQL query. Based on these items we have six options to construct the query based on. These six come from the three main items we have class, data property, and object property. We may find the three, two of them, or only one of them.

4 Research Methodology

Provided in this section the methodology used in the proposed translator and how we will evaluate it. the proposed translator made up of four parts. We made a survey on some of the related works (see Section 2). Our approach involves the user in the formulation of SPARQL query (see section 4.1, Concept matcher and user interaction). Also using the singular of plural key words besides the plural key words in the (see section 4.1, user query handler). The final judgment on the translator will be conducted based on the precision, the recall and the F-measure (see section 5.2).

5 Architecture

In this section we provide the architecture of the proposed translator and how we will evaluate it. the proposed translator is a combination of four main elements. Where the output of each element is the input of the next one. A benchmark will be used for testing and evaluating the translator.

5.1 Proposed Architecture

The proposed translator consists of the following four components as presented in Fig.1. These four components are the User Query Handler, the Concept Matcher, the User Interaction, and the SPARQL Query Builder.

User Query Handler:
It is responsible for handling the user query by performing multiple operations on the user query. These operations are tokenizing, keeping numbers, keeping modifiers, getting singular from plural, removing stop words, and stemming. Tokenizing is to split the user query into words and store it into a list say L1. The found modifiers (ex. all, min, max, etc.) in the user query is kept in another list say L2 and removed from L1. As the modifiers if there are numbers that they will be kept in another list say L3 and

removed from L1. These modifiers and numbers will be used in the SPARQL query builder. Removing stop words is to remove the stop words (like is, a, in, etc.) from the list L1. Stemming is to get the stem of each word in the list L1 and store these stems into another list say L4 (snowball stemmer is used here).

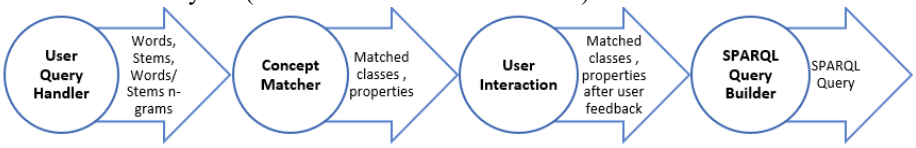

Fig. 1. Proposed query translator architecture.

Concept Matcher:
This component is responsible for two main parts: 1) reading from the ontology the concepts, properties and relations (object properties) and storing these concepts, properties and relations into the database. Note that step (1) is performed only once and could be executed again if we need to change or update the stored ontology. 2) Matching the lists L1, L4 and their n-grams with the concepts, properties and relations. The matching is done in one of three phases. The first is the exact match, where if it is found we skip the next two phases. The second is the wordnet synonym match, where synonyms of the words found in the list L1 inserted to L1 itself and add the stems of these new words to L4. Then again apply exact match using modified lists. If there is a found match, then skip the next phase. The third is a combined similarity match using Jaro Winkler, edit distance, and n-grams with equal share using a given threshold. The matched items from the ontology are stored in a list say L6.

User Interaction:
This component is used to get the user feedback on the resulting list L6. Using this component, the user can determine whether one or more of the items that exists in the list L6 is relevant to his search query or not. Also, the ontology's concepts, data properties and object properties are presented to the user to give him the choice to enhance his query by adding new items to the matched list L6.

SPARQL Query Builder:
The last component in the tools, it is used to generate a SPARQL query based on the final List produced by the user interaction unit. For the items in the list L6, we try to construct the resulting queries using L6 and the previously stored items in the database (see component 2 the concept matcher) following the next cases.

- A complete triple (class, data property, and object property).
- A triple with missing one item (class, and object property), (class, and data property), or (data property, and object property).
- A triple with missing two items (class), (object property), or (data property).

Based on the 1st case we directly build the query. For the 2nd and the 3rd cases we try to constuct the complete triple using the existing ontology. Baesd on the generated triples the SPARQL Query Builder build the SPARQL query. These queries will be

ranked according to the the above three cases. Finally, these queries are presented to the user to select the best one.

5.2 Proposed Translator Evaluation

To test and evaluate the proposed tool, we will start with simple queries as a test for the prototype. Translating these simple queries will give us an indicator about the efficiency of the proposed translator. Then, as a complete evaluation, the Query Answering over Linked Data (QALD) benchmark will be used. Using this benchmark and based on the precision, the recall and the F-measure final judgment on the translator will be conducted.

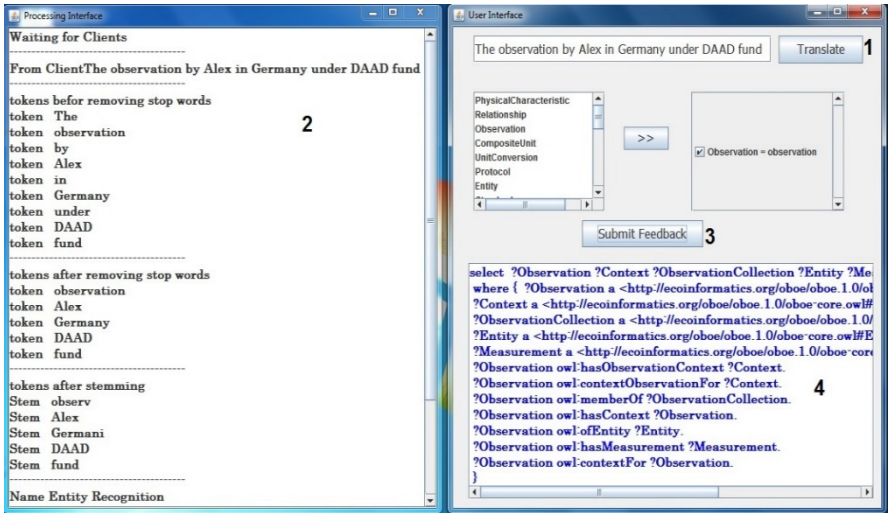

Fig. 2. Proposed tool snapshot.

6 Results

Presented in this section a demonstration for the proposed tool. It is clear from Fig. 2 that the user interface is split into two interfaces (the user and processing interface). This is done just to show the results of the intermediate steps, until the final result (SPARQL query) is presented on the user interface (numbered 4 on the figure). As a first step, the user start using the tool by writing his query in natural language and press the button translate as in the figure (numbered 1 in Fig.2). Then the output of the intermediate steps (see user query handler, and ontology reader and concept matcher) is shown on the processing interface as in Fig.2 (numbered 2). Now it is the turn of the user to give his feedback on the matched concepts (see concept matcher). This is done by navigating through the two lists on the user interface part and then choosing/adding the relative concepts to the user query, and the final step in the user feedback component is done when the user presses the submit feedback button as in Fig.2 (numbered 3).

Finally, based on all of the previous operations the SPARQL query is presented to the user as shown in Fig.2 (numbered 4).

7 Discussion

To allow access and discover knowledge for a set of different data sources, the OBDA scheme is introduced as an elegant solution. It allows the end user interacting with the system through the conceptual layer to get access to data sets. This necessitates the need an effective and efficient query translation in the context of OBDA. Query translation in OBDA is very important nowadays in many application domains. In this paper, we discussed an ontology-based approach for translating NL queries into SPARQL queries. To the best of our knowledge, the proposed tool is the first tool that augment the user interaction, as a key element not as a helper one, in the intermediate translation steps. We faced some difficulties such as choosing stemming algorithm, choosing string similarity algorithm, and the choice between building multiple queries or single query. Until now we are using the snowball stemmer. For the string similarity, a combination of the three methods with equal share is used. Multiple queries are produced by the suggested translator, because producing one query is not the best option. Producing one query could give a negative effect. This occurs when the output of the generated query is very narrow or very large. Based on the involvement of the user and using the singular and the plural of key words, the preliminary results demonstrate the effectiveness and the quality of the proposed tool.

Currently the knowledge and data management are improved by ontology-based data access. Query processing is a key element in OBDA. One of the most valuable components of query processing is query translation. We built a prototype for translating natural language query into SPARQL query. The user is involved in the translating process, this involvement improves that query translation. The initial results show the effectiveness and the quality of the proposed tool.

We are planning to enhance the current prototype to build a complete tool that can be used alone or as a part of an existing system. First, we have to enhance the matching step using key word synonyms (see concept matcher). Until now the prototype supports only one ontology. Support multiple ontologies will be taken in consideration. The user interface is one of the important aspects. Enabling the autocomplete feature in the user interface could be an option. After establishing and testing the tool, integration is a good choice. Integrate the proposed translator to one or more the existing OBDA systems could be possible.

Acknowledgements. Many thanks for Birgitta König-Ries, Alsayed Algergawy, Taysir Hassan A. Soliman, and Adel A. Sewisy for their guidance and care. This work has been partially funded by the DAAD funding through the BioDialog project.

References

1. Cinzia Daraio, Maurizio Lenzerini, Claudio Leporelli, Paolo Naggar, Andrea Bonaccorsi, and Alessandro Bartolucci. The advantages of an ontology-based data management approach: openness, interoperability and data quality. Scientometrics, 108(1):441-455, 2016.
2. Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. Using ontologies for semantic data integration. In A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years, pages 187-202. Springer, 2018.
3. Roman Kontchakov, Mariano Rodriguez-Muro, and Michael Zakharyaschev. Ontology-based data access with databases: A short course. In Reasoning web. semantic technologies for intelligent data access, pages 194-229. Springer, 2013
4. Ming Tao, Kaoru Ota, and Mianxiong Dong. Ontology-based data semantic management and application in iot-and cloud-enabled smart homes. Future Generation Computer Systems, 76:528-539, 2017.
5. Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. Semantic Web, 8(3):471-487, 2017.
6. Cristina Civili, Marco Console, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Lorenzo Lepore, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, et al. Mastro studio: managing ontology-based data access applications. Proceedings of the VLDB Endowment, 6(12):1314-1317, 2013.
7. Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. Semantic Web, 8(3):471-487, 2017.
8. Ahmet Soylu, Martin Giese, Ernesto Jimenez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, and Ian Horrocks. Ontology-based end-user visual query formulation: Why, what, who, how, and which? Universal Access in the Information Society, 16(2):435-467, 2017.
9. Pradel, Camille, Ollivier Haemmerlé, and Nathalie Hernandez. "Swip: a natural language to SPARQL interface implemented with SPARQL." In International Conference on Conceptual Structures, pp. 260-274. Springer, Cham, 2014.
10. Cabrio, E., Cojan, J., Aprosio, A., Magnini, B., Lavelli, A., Gandon, F.: Qakis: an open domain qa system based on relational patterns. In: International Semantic Web Conference (Posters & Demos) (2012)
11. Lehmann, J., B¨ uhmann, L.: Autosparql: Let users query your knowledge base. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 63–79. Springer, Heidelberg (2011)
12. Unger, C., B¨ uhmann, L., Lehmann, J., Ngonga Ngomo, A., Gerber, D., Cimiano, P.: Template-based question answering over rdf data. In: Proceedings of the 21st International Conference on World Wide Web, pp. 639–648. ACM (2012)
13. Sander M, Waltinger U, Roshchin M, Runkler T. Ontology-based translation of natural language queries to SPARQL. InNLABD: AAAI Fall Symposium 2014 Sep 24.
14. Evgeny Kharlamov, Dag Hovland, Ernesto Jiménez -Ruiz, Davide Lanti, Hallstein Lie, Christoph Pinkel, Martin Rezk, Martin G Skjæveland, Evgenij Thorstensen, Guohui Xiao, et al. Ontology based access to exploration data at statoil. In International Semantic Web Conference, pages 93-112. Springer, 2015.
15. T. Eiter, M. Ortiz, M. Simkus, T. Tran, and G. Xiao. Query rewriting for hornshiq plus rules. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012