Baltic J. Modern Computing, Vol. 8 (2020), No. 4, pp. 719–731
https://doi.org/10.22364/bjmc.2020.8.4.14

Ontology Based Natural Language Queries Transformation into SPARQL Queries

Majid ASKAR[2], Alsayed ALGERGAWY[1], Taysir Hassan A. SOLIMAN[2], Birgitta KÖNIG-RIES[1], Adel A. SEWISY[2]

[1] Heinz Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University of Jena, Germany
[2] Faculty of Computers and Information, Assiut University, Egypt

majid.askar@aun.edu.eg, alsayed.algergawy@uni-jena.de, taysirhs@aun.edu.eg, Birgitta.Koenig-Ries@uni-jena.de, sewisy@aun.edu.eg

Abstract. Ontology-based Data Access (OBDA) enables semantic access to a set of heterogamous data sources, supporting data sharing, exchanging, and integration across these data sources. In the OBDA scheme, normally formal query languages, such as SPARQL, are used to represent the user questions, which limits end users from defining their requests. To cope with this problem a layer that accepts the user request in her own language and transforms it into one of these formal languages has become a necessity. To this end, we introduce a new and interactive method that guides the user during the translation. The proposed approach makes use of the capabilities of natural language processing and the semantic information embedded in the domain ontology. Furthermore, the proposed approach considers user involvement during the translation process. To demonstrate the effectiveness, we implemented the proposed approach and validated it against a query benchmark assessing the query accuracy and efficiency.

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

1 Introduction

Current information systems keep track of a large amount of heterogeneous data. Overseeing and handling these sorts of data and the revelation of valuable data/information are challenging (Antonioli et al., 2013). Numerous upgrades could be applied to the area of knowledge management dependent on ontologies. A lot of research was conducted based on ontologies such as ontology-based recommender systems (Tarus et al., 2018), Ontology-based knowledge representation (Sanfilippo et al., 2019), ontology-based data integration (Buron et al., 2020a), ontology-based RDF integration of heterogeneous data(Buron et al., 2020b), ontologybased messaging service(Elmhadhbi et al.,

2020), and Ontology-Based Data Access (OBDA). Ontology-Based Data Access is concerned with querying data sources in the presence of domain-specific knowledge provided by ontologies (Daraio et al., 2016; Flesca et al., 2018; Rudolph et al., 2013). In OBDA, knowledge resources can be used as a mediator to facilitate 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 (Flesca et al., 2018; Rudolph et al., 2013; Tao et al., 2017; Calvanese et al., 2017). 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 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 a 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's original query. Subsequently, the processed query 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 (Civili et al., 2013) and Ontop (Calvanese et al., 2017). Also, VQS (Soylu et al., 2017) can be used to edit the user query using a visual interface. However, there is some work on natural language to SPARQL translation (Pradel et al., 2014; Cabrio et al., 2012; Lehmann and Bühmann, 2011; Unger et al., 2012; Sander et al., 2014), but the user involvement in the intermediate steps is not considered except in (Sander et al., 2014), and it is not integrated into an existing OBDA system. Also, using natural language techniques such as getting the singular from plural is used, which is not considered by the previous work. Additionally, we used wordnet to get word synonyms to expand the resulting query (in case we did not get an exact match see section 4.2 concept matcher). From the user's point of view, using natural language in queries instead of SPARQL is much better. When using natural language any user can write her own query in her own native tongue without any external help, but with SPARQL only expert users can.

In the oil and gas industry, about 30-70% of engineers time is exhausted in looking for and in evaluating the quality of data (Kharlamov et al., 2015). In a practical way, almost all of this time will be saved if the engineers can fire their queries directly (i.e. without external help or without learning new things). In other words, expressing their queries in their own language (natural language) will be better. In another case in Siemens, it usually takes weeks to generate a new SQL query (Eiter et al., 2012). So, this about generating the query, which indeed costs time (about 2 weeks) and money (IT experts who work on that query). Based on that if we allow the end user to write her

own query in her own terms, sure this will save time which in turn will save money. Two options are available, the user can write her query in SPARQL or in natural language. With SPARQL the user uses her common terms, but, is still restricted by learning the SPARQL structure. On the other hand, it much easier for 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 translates from natural language to SPARQL, and afterward try to integrate it into one or more of the existing systems. The target users of this translator, mainly, will be end users, while expert users are welcomed.

In this paper, 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 the 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 the enhancement on string matching using word synonyms, where, when to use it and could it help?

This paper is an extension to the paper published in the DBIS2020 doctoral consortium (Askar, 2020). The current paper provides a general idea about query processing in ODBA systems. Also, we propose an ontology-based query translator. Unlike the existing systems, the user is not only involved in the start and the end of the process, but also in the middle of the process itself. The rest of this paper is organized as follows: in the following section, we discuss the related work. Afterward, in Section 3 we introduce the proposed approach and methodology. Then, we move to explain this proposed architecture in section 4. Results and discussion are presented in Section 5, and Section 6 consecutively.

2 Related work

OBDA is an elegant scheme, where ontologies can be used as mediator to facilitate access to different and multiple data sources. The OBDA system constructs a clear representation of the domain (using ontologies) and connects it in a formal way with existing data sources through a set of mappings. Therefore, in general, OBDA has three main components as shown in Fig. 1 data layer; it contains data sets that the system provides access to them including structured and unstructured data, conceptual layer; it provides the conceptual components where the end user interacts with the system, including the predeveloped ontologies, query interface, and the mapping layer; it maintains the links and mappings between concepts and entities from the conceptual layer and data set attributes. Basically, the OBDA scheme aims to answer user queries by allowing access to data in the data layer. A user query passes through a query processing pipeline, in which the query is processed to get the answer back to the user. One of the key steps in this pipeline is the query translation.

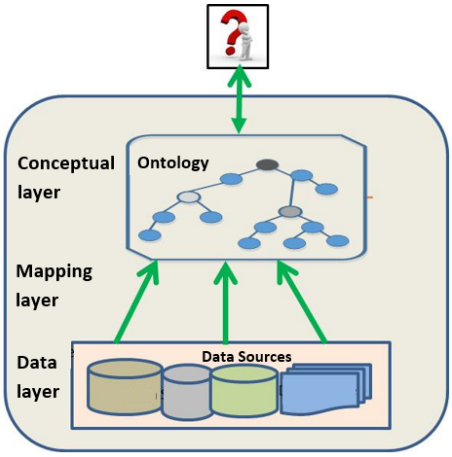

Fig. 1. OBDA system general architecture.

Query translation is used in many systems not only for translation but also for question answering. There are some common steps shown in (Pradel et al., 2014), 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 (Pradel et al., 2014; Cabrio et al., 2012; Lehmann and Bühmann, 2011; Unger et al., 2012; Sander et al., 2014). Some of these approaches make enhancements as (Cabrio et al., 2012; Lehmann and Bühmann, 2011; Unger et al., 2012). QAKiS (Cabrio et al., 2012) used an external resource to enhance the matching. It established matching among question parts and relational textual patterns collected from Wikipedia. Autosparql (Lehmann and Bühmann, 2011) 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 (Unger et al., 2012). Where the SPARQL query is obtained using the syntactic structure of the natural language question and by a predefined domain independent expression. (Jung and Kim, 2020) aimed to enable ontology-based query answering system users to acquire their answers from a domain ontology by inputting their natural language queries. (Jung and Kim, 2020) generate SPARQL queries from Korean natural language queries. The 4th step mentioned at the start of this section is not used by (Jung and Kim, 2020), instead, they internally score query graphs, and the best-rated query graph is converted into SPARQL.

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 en-

tity list. This will happen after getting the matched entities from the matching step (see sections 4.2 and 4.3, Concept matcher and user interaction). Unlike the existing systems, the user is not only involved in the start and the end of the process, but also in the middle of the process itself. Also, using the singular of plural keywords besides the plural keywords in the matching could enhance the performance of the translator. Where, some processing steps could be skipped if the match is found through the singular, not the plural (see section 4.1, user query handler).

3 Approach and methodology

In this section, we present our proposed approach and methodology. In our approach, 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 be-tween the query entities and corresponding entities in the datasets. Afterward, 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 to 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 are found in the user query are collected to be processed using the query builder. Date and numbers are not handled yet. The used ontology is flattened 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. The tool writes the ontology into the database only once, if any changes appear on the ontology or we want to change the ontology, then we repeat the ontology reading once again (see section 4.1). 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 an equal share.

After getting the similarity, presenting the matched items to the user to get her feedback is a key step in the whole translation. The user feedback, for sure, enhances the quality of the produced SPARQL query. The enhancement is done 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.

Now, we will present our methodology used in the proposed translator and how we will evaluate it. The proposed translator is made up of four parts. We made a survey on some of the related works (see Section 2). So, we figure out the common steps in the query translation process. The user of the translator is involved in the formulation of the SPARQL query, in the beginning, in the middle, and at the end of the translation process (see section 4). Also, using the singular of plural keywords besides the plural keywords in the (see section 4.1, user query handler). The final judgment on the translator, using a benchmark dataset, will be conducted based on the efficiency and accuracy using the precision, the recall, and the F-measure (see section 4.5).

4 Translator 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.

The proposed translator consists of the following four components as presented in Fig. 2. These four components are the User Query Handler, the Concept Matcher, the User Interaction, and the SPARQL Query Builder.

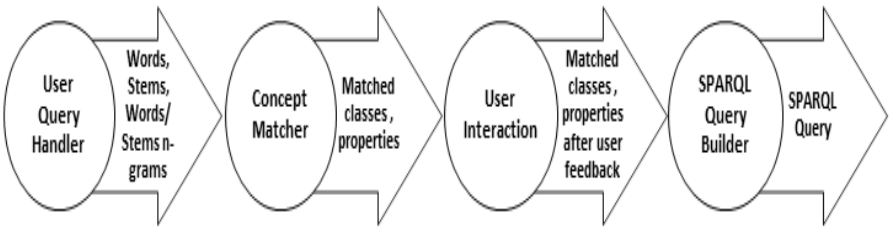

Fig. 2. Proposed query translator architecture.

4.1 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. Tokenization is the first step while processing the user query, in which the user query is divided into words and store those words in a list say L1. If numbers or modifiers exist in the query, the found modifiers (ex. all, min, max, etc.) are kept in another list say L2, and removed from the list L1. As for the modifiers, if there are numbers in the user query, they will be kept in another list say L3, and excluded from the list L1. These modifiers and numbers will be used by the SPARQL query builder. An important step is to remove the stop words. The

existence of these words will cost the translator extra unwanted processing overhead. Stop word removal is to remove the words (like is, a, in, etc.). These stop words are removed from the list L1 (which represents the original query words). Stemming is the process of reducing inflected words to their word stem, base, or root form. We get the stem of each word in the list L1 and store these stems into another list say L4 (snowball stemmer is used here).

4.2 Concept matcher

Directly after processing the user query, We start with the concept matcher. This component is responsible about two main jobs:

1. reading from the ontology the concepts, properties, and relations (object properties) and storing these concepts, properties, and relations into the database (see Fig 3).
2. Matching the lists L1, L4 and their n-grams with the concepts, properties, and relations (see Fig 3).

Note that step (1) is performed only once and could be executed again if we need to change or update the stored ontology. The matching is done in one of three phases:

1. Exact match.
2. Wordnet synonym match.
3. Combined similarity match.

If the exact match is found we skip the next two phases. In the exact match, we try to find the classes and properties that typically be the same as one of the keywords found in the user query. In the second phase the wordnet synonym match, the synonyms of the words found in the list L1 inserted to L1 itself and add the stems of these new words to L4. We typically use wordnet to get these synonyms. Then, again apply exact match using modified lists. If there is a match found, then skip the next phase. At last, We have combined similarity match using Jaro Winkler, edit distance, and n-grams. This combined similarity is calculated with an equal share for each of the three algorithms and applied using a given threshold. The matched items from the ontology are stored in a list say L6.

4.3 User interaction

Now, the user query is processed by natural language techniques, and We get the matched items through the concept matcher. Moving to the current component, 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 exist in the list L6 is relevant to her search query or not. Also, the ontologys concepts, data properties, and object properties are presented to the user to give him the choice to enhance her query by adding new items to the matched list L6. As shown in Fig. 3 two lists are presented to the user one for the ontology items, and one for the matched items. the user now decides what to keep/remove from the matched list, and what to add from the ontology list to the matched list.

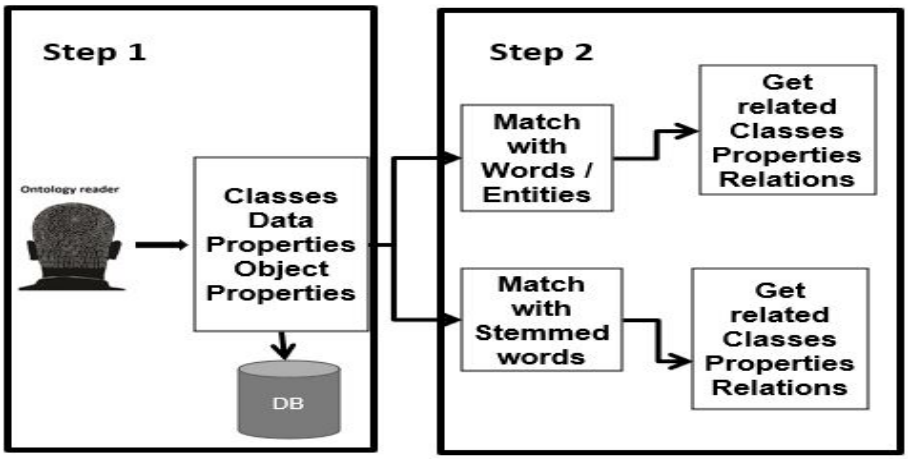

Fig. 3. Ontology matcher.

4.4 SPARQL query builder

Till now every component is done. The last component in the tools is the query builder, 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) see Fig. 4.
- 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 first case, we directly build the query. Because the three main components that We need to build the SPARQL query are ready. For the 2nd and the 3rd cases, we try to construct the complete triple using the existing ontology. Thus, in the second case, We need to get only one missing item out of three. Subsequent to finding the missing item We move back to case one. Regarding case three, it is a little bit harder than case two. In the third case, we need to discover the two missing items of the triple. Again, when the missing parts are here, We go back to case one. Based on the generated triples the SPARQL Query Builder builds the SPARQL query. These queries will be ranked according to the above three cases. The highest rank goes for the queries constructed according to the first case. Queries built according to the second case got the 2nd highest rank. Queries constructed upon the third case are ranked in the third place after the queries from the first two cases. This ranking was introduced based on the size of guessing while getting the triples. When the guessing grows the rank goes down. Finally, these queries are presented to the user to select the best one.

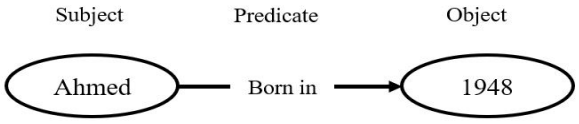

Fig. 4. Complete Triple.

4.5 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, as shown in Fig. 5, and Fig. 6, will give us an indicator of 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. As for now, A single user query will be translated into multiple SPARQL queries. The results of the execution of these queries are combined together. The precision, the recall, and the F-measure are calculated against the combined results.

5 Results

Presented in this section a demonstration of the proposed tool. It is clear from Fig. 5, and Fig. 6 that the user interface is split into two interfaces (the user and processing interface). The user interface is used by the user to type her query, send her feedback, and get the results. While the processing interface 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 Fig. 5). As a first step, the user starts using the tool by writing her query in natural language and press the button named translate as in the figure (numbered 1 in Fig. 5). At this point, the user query handler starts working followed by the concept matcher. Then the output of the intermediate steps (see section 4.1, section 4.2) is shown on the processing interface as in Fig. 6 (numbered 2). Now it is the turn of the user to give her feedback on the matched concepts (see concept matcher). Here We are passing through the user interaction part (see section 4.3). 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. 5 (numbered 3). At the moment, the SPARQL query builder plays its role (see section 4.4). Based on all of the previous operations the SPARQL query is presented to the user as shown in Fig. 5 (numbered 4). Enhancing the results could be possible by disclosing the semantic relations between keywords. Lodifier by (Augenstein et al., 2012) automatically extract information from a text and transform it into a formal description. Lodifier can be used as an addition and as an improvement to the proposed translator.

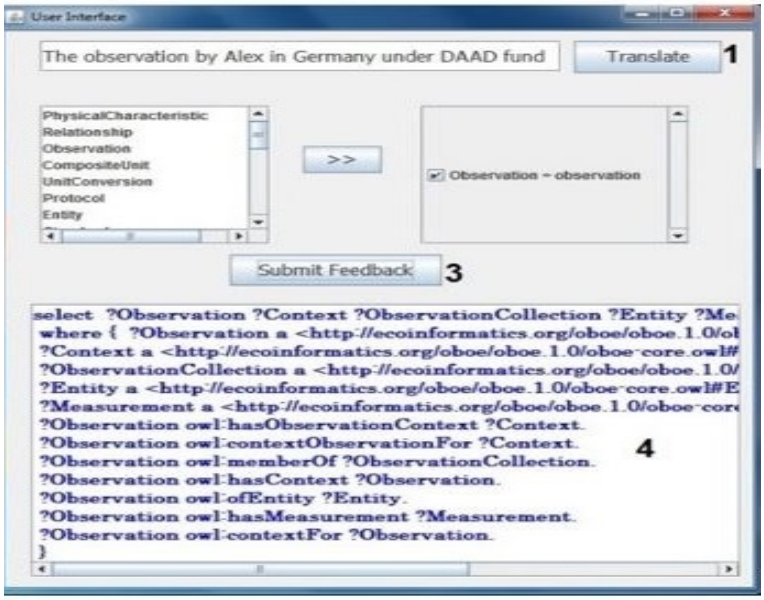

Fig. 5. Proposed tool: User Interface.

Fig. 6. Proposed tool: Processing Interface 2.

6 Discussion

To allow access and discovery knowledge for a set of different data sources, the OBDA scheme is introduced as an elegant solution. It allows the end user to interact with the system through the conceptual layer to get access to data sets. This necessitates the need for 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 augments the user interaction, as a key element not as a helper one, in the intermediate translation steps. We faced some difficulties such as choosing a stemming algorithm, choosing a string similarity algorithm, and the choice between building multiple queries or a 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 keywords, the preliminary results demonstrate the effectiveness and the quality of the proposed tool.

Currently, 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 the semantic relations between keywords (see concept matcher and user query handler). Until now the prototype supports only one ontology. Support multiple ontologies will be taken into 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 of the existing OBDA systems could be possible.

7 Conclusion

Ontology Based Data Access (OBDA) improves knowledge sharing and reuse and grants a set of options to enhance the power of knowledge management. For users to express queries in their own languages and terms, a transformation from a natural language to SPARQL is needed. We propose a Translator, which is composed of four main components. These four parts are the User Query Handler, the Concept Matcher, the User Interaction, and the SPARQL Query Builder. A benchmark will be used for testing and evaluating the translator. We intend to improve the suggested translator using semantic relations between keywords. Also, augmenting the user interface with more options(autocomplete for example) is considered.

8 Acknowledgements

This work has been partially funded by the DAAD funding through the BioDialog project.

References

Antonioli, N., Castano, F., Civili, C., Coletta, S., Grossi, S., Lembo, D., Lenzerini, M., Poggi, A., Savo, D. F., Virardi, E. (2013). Ontology-based data access: the experience at the italian department of treasury.

Askar, M. (2020). Towards transforming natural language queries into sparql queries, *14th International Baltic Conference on Databases and Information Systems*, pp. 65–72.

Augenstein, I., Padó, S., Rudolph, S. (2012). Lodifier: Generating linked data from unstructured text, *Extended Semantic Web Conference*, Springer, pp. 210–224.

Buron, M., Goasdoué, F., Manolescu, I., Mugnier, M.-L. (2020a). Obi-wan: ontology-based rdf integration of heterogeneous data, *Proceedings of the VLDB Endowment* **13**(12), 2933–2936.

Buron, M., Goasdoué, F., Manolescu, I., Mugnier, M.-L. (2020b). Obi-wan: ontology-based rdf integration of heterogeneous data, *Proceedings of the VLDB Endowment* **13**(12), 2933–2936.

Cabrio, E., Cojan, J., Aprosio, A. P., Magnini, B., Lavelli, A., Gandon, F. (2012). Qakis: an open domain qa system based on relational patterns.

Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G. (2017). Ontop: Answering sparql queries over relational databases, *Semantic Web* **8**(3), 471–487.

Civili, C., Console, M., De Giacomo, G., Lembo, D., Lenzerini, M., Lepore, L., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M. et al. (2013). Mastro studio: managing ontology-based data access applications, *Proceedings of the VLDB Endowment* **6**(12), 1314–1317.

Daraio, C., Lenzerini, M., Leporelli, C., Naggar, P., Bonaccorsi, A., Bartolucci, A. (2016). The advantages of an ontology-based data management approach: openness, interoperability and data quality, *Scientometrics* **108**(1), 441–455.

Eiter, T., Ortiz, M., Simkus, M., Tran, T.-K., Xiao, G. (2012). Query rewriting for horn-shiq plus rules, *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Elmhadhbi, L., Karray, M.-H., Archimède, B., Otte, J. N., Smith, B. (2020). Promes: An ontology-based messaging service for semantically interoperable information exchange during disaster response, *Journal of Contingencies and Crisis Management* **28**(3), 324–338.

Flesca, S., Greco, S., Masciari, E., Saccà, D. (2018). *A comprehensive guide through the italian database research over the last 25 years*, Springer.

Jung, H., Kim, W. (2020). Automated conversion from natural language query to sparql query, *Journal of Intelligent Information Systems* pp. 1–20.

Kharlamov, E., Hovland, D., Jiménez-Ruiz, E., Lanti, D., Lie, H., Pinkel, C., Rezk, M., Skjæveland, M. G., Thorstensen, E., Xiao, G. et al. (2015). Ontology based access to exploration data at statoil, *International Semantic Web Conference*, Springer, pp. 93–112.

Lehmann, J., Bühmann, L. (2011). Autosparql: Let users query your knowledge base, *Extended semantic web conference*, Springer, pp. 63–79.

Pradel, C., Haemmerlé, O., Hernandez, N. (2014). Swip: a natural language to sparql interface implemented with sparql, *International Conference on Conceptual Structures*, Springer, pp. 260–274.

Rudolph, S., Gottlob, G., Horrocks, I., Van Harmelen, F. (2013). *Reasoning Web. Semantic Technologies for Intelligent Data Access: 9th International Summer School 2013, Mannheim, Germany, July 30–August 2, 2013. Proceedings*, Vol. 8067, Springer.

Sander, M., Waltinger, U., Roshchin, M., Runkler, T. (2014). Ontology-based translation of natural language queries to sparql, *2014 AAAI fall symposium series*, Citeseer.

Sanfilippo, E. M., Belkadi, F., Bernard, A. (2019). Ontology-based knowledge representation for additive manufacturing, *Computers in Industry* **109**, 182–194.

Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I. (2017). Ontology-based end-user visual query formulation: Why, what, who, how, and which?, *Universal Access in the Information Society* **16**(2), 435–467.

Tao, M., Ota, K., Dong, M. (2017). Ontology-based data semantic management and application in iot-and cloud-enabled smart homes, *Future generation computer systems* **76**, 528–539.

Tarus, J. K., Niu, Z., Mustafa, G. (2018). Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning, *Artificial intelligence review* **50**(1), 21–48.

Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.-C., 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.

Received December 10, 2020 , accepted December 12, 2020