SPARQL2NL – Verbalizing SPARQL queries

Axel-Cyrille Ngonga Ngomo
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
ngonga@informatik.uni-leipzig.de

Lorenz Bühmann
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
buehmann@informatik.uni-leipzig.de

Christina Unger
Bielefeld University, CITEC
Universitätsstraße 21–23,
33615 Bielefeld
cunger@cit-ec.uni-bielefeld.de

Jens Lehmann
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
lehmann@informatik.uni-leipzig.de

Daniel Gerber
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
dgerber@informatik.uni-leipzig.de

ABSTRACT

Linked Data technologies are now being employed by a large number of applications. While experts can query the backend of these applications using the standard query language SPARQL, most lay users lack the expertise necessary to proficiently interact with these applications. Consequently, non-expert users usually have to rely on forms, query builders, question answering or keyword search tools to access RDF data. Yet, these tools are usually unable to make the meaning of the queries they generate plain to lay users, making it difficult for these users to i) assess the correctness of the query generated out of their input, and ii) to adapt their queries or iii) to choose in an informed manner between possible interpretations of their input.

We present SPARQL2NL, a generic approach that allows verbalizing SPARQL queries, i.e., converting them into natural language. In addition to generating verbalizations, our approach can also explain the output of queries by providing a natural-language description of the reasons that led to each element of the result set being selected. Our evaluation of SPARQL2NL within a large-scale user survey shows that SPARQL2NL generates complete and easily understandable natural language descriptions. In addition, our results suggest that even SPARQL experts can process the natural language representation of SPARQL queries computed by our approach more efficiently than the corresponding SPARQL queries. Moreover, non-experts are enabled to reliably understand the content of SPARQL queries. Within the demo, we present the results generated by our approach on arbitrary questions to the DBpedia and MusicBrainz datasets. Moreover, we present how our framework can be used to explain results of SPARQL queries in natural language.

Categories and Subject Descriptors

H.5.2 [**Information systems**]: User Interfaces—*Natural language, Theory and methods*

General Terms

Algorithms, Experimentation, Theory

Keywords

Natural language generation, query verbalization, SPARQL

1. INTRODUCTION

Most Semantic Web applications rely on RDF data as well as on the W3C standard SPARQL for querying this data. While SPARQL has proven to be a powerful tool in the hands of experienced users, it remains difficult to fathom for lay users. Approaches such as question answering [5], keyword search [4] and search by example [2] aim to hide SPARQL and RDF from the user. Yet, these approaches still have constructed SPARQL queries to address their data backend, without providing lay users with a possibility to check whether the retrieved answers indeed correspond to the intended information need. Consider for example the natural language question What is the birth date of Li Ling?, for which TBSL [5] returns more than 50 possible interpretations, including the birth date of the pole vaulter Li Ling and the age of the sinologist Li Ling. Since each of the interpretations is realized as a SPARQL query, a lay user cannot pinpoint the set of results that correspond to the person he is actually interested in, nor can he easily detect the source of possible errors. Similar problems occur in keyword-based systems. For example, the keywords Jenny Runacre husbands leads to SINA [4] generating queries for the husbands of Jenny Runacre as well as for the role of Jenny Runacre in the movie "The Husbands".

The rationale behind SPARQL2NL[1] is to verbalize SPARQL queries and therewith bridge the gap between the query language understood by semantic data backends, i.e., SPARQL,

[1] `http://aksw.org/projects/SPARQL2NL` - an open source implementation is available at `https://github.com/AKSW/SPARQL2NL`

and that of the lay users, i.e., natural language. Our approach is tailored towards SPARQL constructs typically used in keyword search and question answering, and it consists of four main steps: a *preprocessing* step which normalizes the query and extracts type information for the occurring variables, a *processing* step during which a generic representation of the query is generated, a *postprocessing* step which applies reduction and replacement rules in order to improve the legibility of the verbalization, and a *realization* step which generates the final natural language representation of the query. As an exemplary use case, we integrated SPARQL2NL into a user interface for the question answering system TBSL to enable users to read and disambiguate the different SPARQL queries generated when processing a question.[2] The demo of the SPARQL2NL is available online at `http://sparql2nl.aksw.org/demo`. A technical description of the approach can be found in [3].

The rest of this paper is structured as follows: We first give an overview of the SPARQL2NL pipeline. Then, we give some insights into how well our approach performs with respect to the *adequacy* and *fluency* [1] of the natural language representations it generates. We finally give an overview of the demo we aim to present and conclude.

2. SPARQL2NL IN A NUTSHELL

The basic processing pipeline of SPARQL2NL consists of four main steps: pre-processing, processing, post-processing and verbalization. In the following, we give a brief overview of each of these steps and explain them by using the query shown in Listing 1. For a technical description of the steps, please see [3].

```
SELECT DISTINCT ?person
WHERE { ?person a dbo:Person.
        { ?person dbo:occupation res:Writer. }
        UNION
        { ?person dbo:occupation res:Surfing. }
        ?person dbo:birthDate ?date.
        FILTER(?date > "1950"^^xsd:date) .
}
```

Listing 1: Running example SPARQL query.

2.1 Pre-processing

The pre-processing aims to normalize the query for further processing (*normalization*) and to collect information on the type of projection variables (*type extraction*). To ensure that we generate easily legible natural language representations of SPARQL queries, we *normalize* the input queries further by transforming any nesting of disjunctions, i.e. `UNION` statements, and conjunctions into a disjunctive normal form (DNF). We chose to use DNFs as this yields representations that are faithful to the disjunctive character of unions.

The type extraction is achieved by processing the query and finding all graph patterns `?x rdf:type C` for each projection variable `?x`. If none of the statements is part of a `UNION` statement, we assign the conjunction of all `C` to `?x`. Otherwise we assign to `?x` the disjunction of all `C` that are such that the `UNION` statements which contain `?x rdf:type C` contain no other statements. Consequently, in our example, the type `dbo:Person` is assigned to the variable `?person`.

[2]A demo can be found at `http://autosparql-tbsl.dl-learner.org`.

2.2 Processing

The goal of the *processing* step is to generate a list of dependency trees for an input query. To achieve this goal, the query is subdivided into the three segments *body*, *optional* and *modifier*, each of which is assigned its own sentence tree. Since `ASK` queries only possess a subset of these features, they can also be processed by our approach. Therefore, in the following, we only describe how the representation of each of these segments is generated for `SELECT` queries.

2.2.1 Processing triple patterns

The realization of a triple pattern `s p o` depends mostly on the verbalization of the predicate `p`. If `p` can be realized as a noun phrase, then a possessive clause can be used to express the semantics of `s p o`. For example, the property `occupation` in our example leads to the verbalization ?person's occupation is Surfing. In case `p`'s realization is a verb, then the triple can be verbalized as a verbal phrase. For example, if `p` is the verb `write`, then the verbalization is ?x writes ?y. As fallback, i.e., when our approach cannot determine whether a property is a nominal or a verbal phrase, we generate `s` is related to `o` via `p`.

2.2.2 Generating the segments of SPARQL queries

The main effort during the processing step is concerned with representing the body of the query, i.e. the content of the `WHERE` clause. Our approach begins by transforming the type information retrieved by the pre-processing into a coordinated phrase element. The processing then continues by converting the content of the `WHERE` clause into a second coordinated phrase element by making use of the fact that only group graph patterns *GP* (i.e., combinations of conjunctions, `UNION`s and `FILTER`s) can be used within this clause. The approach also provides means for processing `FILTER`s. The `OPTIONAL` section is processed in a way similar to the body and lead to another sentence while the solution modifiers (i.e., `ORDER BY`, `LIMIT` and `OFFSET`) are compiled to yet another sentence. In our example, the processing of the body leads to?person's birth date is later than 1950 and person's occupation is Writer or person's occupation is Surfing.

2.3 Post-processing

The general goal of the post-processing step is to transform the generated description such that it sounds more natural. To achieve this goal, we use a rule-based approach. The aggregation rules serve to cluster and order the input sentences. To this end, the variables occurring in the query are ordered with respect to the number of their occurrences, distinguishing projection variables, i.e. variables that occur in the `SELECT` clause, from all others, and assigning them those input sentences that mention them. We process the input trees in descending order with respect to the frequency of the variables they contain, starting with the projection variables and only after that turning to other variables. In our example, the post-processing of the query leads to the output shown in Listing 2.

```
This query retrieves distinct people such that
    their birth date is later than 1950 and
    their occupation is Writer or Surfing.
```

Listing 2: Verbalization of the example query.

2.4 Verbalization

The verbalization is the final step of SPARQL2NL. The goal of this step is to transform all the information generated in the previous steps and to generate natural language. *Classes and resources* are verbalized by using their label. If no label is available, then we use the local name of the resource or class as label. *Literals* are verbalized in accordance with their type. For example, while `"Albert Einstein"@en` is verbalized as `Albert Einstein`, we verbalize the literal `"123"^^<http://dbpedia.org/datatype/squareKilometre>` as `123 square kilometres`. Properties are more tedious to verbalize as they can be either nominal or verbal phrases. Here we rely on WordNet synsets to derive the correct verbalization.

3. USER STUDY

3.1 Experimental Setup

In our user study, we evaluated the whole SPARQL2NL pipeline, in order to clarify the following two questions:

1. Are the SPARQL2NL verbalizations correct, and are they easy to understand?
2. Do the verbalizations help users that are not familiar with SPARQL, i.e. can they use the verbalizations efficiently and effectively?

We used the 200 DBpedia queries provided by the QALD-2 benchmark, all of which our approach was able to translate into natural language. We ran both a controlled and an uncontrolled survey. The survey consists of three different tasks with 10 randomly selected queries each. At the start of the survey users can indicate whether or not they are SPARQL experts. If not, only Task 3 was presented, otherwise they were asked to complete all three tasks. In Task 1, the survey participant is presented a SPARQL query and its SPARQL2NL verbalization, and is asked to judge the verbalization regarding fluency and adequacy [1]. In Task 2, the participant is presented a SPARQL query as well as five different possible answers and has to select the correct one. Task 3 is similar to Task 2, with the difference that the natural language verbalizations of the SPARQL query and a verbalization of the triples were presented.

3.2 Results

The controlled survey phase was carried out by 10 persons. As these participants were monitored by one of the authors, we used it for time measurements on the three different tasks. The maximum (minimum) time required was 17 (7) minutes, 13 (6) minutes and 12 (4) minutes for Tasks 1, 2 and 3, respectively. We then ran a public survey with 115 participants of which 39 stated they were experts in SPARQL. We used our initial time measurements to filter out those survey participants in the public evaluation. To this end, we decided to admit a time window of 5-18 minutes for Task 1 and 3-15 minutes for Tasks 2 and 3.

The results of the first task showed the fluency of the natural language descriptions to be 4.56 ± 1.29, where in expressions of the form $x \pm y$, x denotes the average value and y denotes the standard deviation. The majority of natural

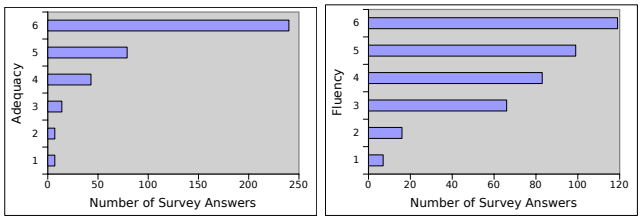

Figure 1: Adequacy and fluency results in survey

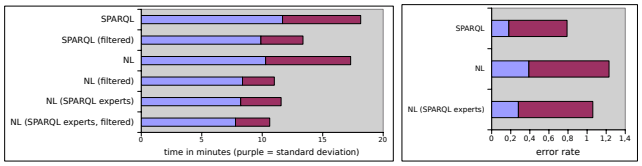

Figure 2: Time and error rate analysis

language descriptions were understandable, where 94.1% of the cases achieved a rating of 3 or higher. The adequacy of the verbalizations was judged to be 5.31 ± 1.08, which we consider a positive result. Details for the results of Task 1 are depicted in Figure 1. For Tasks 2 and 3, our main goal was to directly compare the results of users dealing with SPARQL queries against the results of users dealing with natural language descriptions. Regarding efficiency, participants required 11.68 ± 6.46 minutes to complete Task 2. Applying the time window mentioned above, the required time drops to 9.89 ± 3.48. For Task 3 we obtained execution times of 10.28 ± 7.03 without filtering, and 8.37 ± 2.63 with time filtering. Using a paired t-test with 95% confidence interval, the difference between the time required for Tasks 2 and 3 is statistically significant. Note that even the SPARQL experts were faster when being presented the natural language description, with 8.22 ± 3.34 minutes without time window filter and 7.79 ± 2.83 with time window filter.

Finally we also compared the error rates of participants in Tasks 2 and 3, i.e. the number of incorrect answers per questions, see Figure 2. It turned out that almost all errors occurred with two specific queries, both due to bugs in the implementation of SPARQL2NL. We fixed these issues in an updated version of SPARQL2NL and ran an internal evaluation again using the new verbalizations. 13 participants from the AKSW and CITEC research groups, excluding the authors, took part in this validation phase. The error rate for the natural language expressions (0.12 ± 0.35) is only slightly higher (+0.05) compared to the SPARQL expressions for SPARQL experts (0.07 ± 0.25). Moreover, SPARQL2NL achieves a fluency of 5.05 ± 1.01 and adequacy of 5.60 ± 0.85. We therefore conclude that the SPARQL2NL translations can be read efficiently and understood by both experts and non-experts.

4. DEMO DESCRIPTION

In the demonstration, we aim to highlight the contribution of SPARQL2NL, i.e.,

1. the verbalization of SPARQL queries of different complexity,

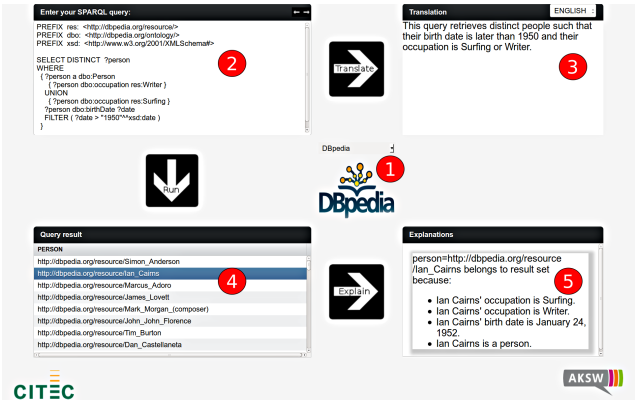

Figure 3: Screenshot of the SPARQL2NL online demo at `http://sparql2nl.aksw.org/demo`.

2. the domain-independence of the approach,
3. the use of SPARQL2NL for explaining the results of SPARQL queries and
4. the integration of SPARQL2NL in question answering tools.

Consequently, we will demo SPARQL2NL both as in a standalone demo as well as integrated in a question answering tool.

The standalone part of the demo consists of the interface shown in Figure 3. The user begins by selecting the dataset against which he wants the SPARQL query to be ran (see ①). In our demo, we show results on the DBpedia and MusicBrainz Datasets. Now in ②, the user can give in a SPARQL query for which a verbalization is required. For example, he might choose to give in the query shown in Listing 1.

Clicking on the "Translate" button leads to the SPARQL2NL being ran and the query being verbalized. The verbalization of query is shown in ③. In our example, SPARQL2NL returns the verbalization shown in Listing 2.

In addition to viewing the verbalization of the query, the user can choose to run the query by clicking on the "Run" button. This leads to the results of the query (as far as some exist) being displayed in tabular format in ④. Each of the rows of the table is clickable. Upon a selection of a result and a click on the "Explain" button, the RDF statements that led to the row being included in the result set are retrieved from the endpoint selected by the user. These results are verbalized by SPARQL2NL. Verbalizing RDF triples makes use of the fact that each RDF statement can be regarded as a variable-free triple pattern. The verbalized RDF triples are finally displayed in the panel marked with ⑤. In our example, the verbalization of the results for Ian Cairns leads to the following explanatory statements:

- `Ian Cairns' occupation is Surfing.`
- `Ian Cairns' occupation is Writer.`
- `Ian Cairns' birth date is January 24, 1952.`
- `Ian Cairns is a person.`

In addition to the standalone demo, we will present the benefits of SPARQL2NL by showing its integration into TBSL [5] as shown in Figure 4. Here the user can give in a natural-language question such as for example `Books written by Dan Brown` in the search field (①). TBSL then generate possible interpretations of user query in the form of several SPARQL queries. In the case of our example query, TBSL generate semantically very different SPARQL queries due to `written by` and `Dan Brown` each matching several resources from DBpedia. TBSL uses SPARQL2NL to verbalize each of the interpretations it generates and displays as well as tries out the highest scored interpretation first (see ②). If the interpretation is incorrect, the user can choose to click on the "Wrong!" button to see alternative interpretations of his natural-language query, which are displayed in the "Did you mean?" box (see ③). He can then select the natural-language representation that is most accurate and run this query without ever having to deal with SPARQL or RDF.

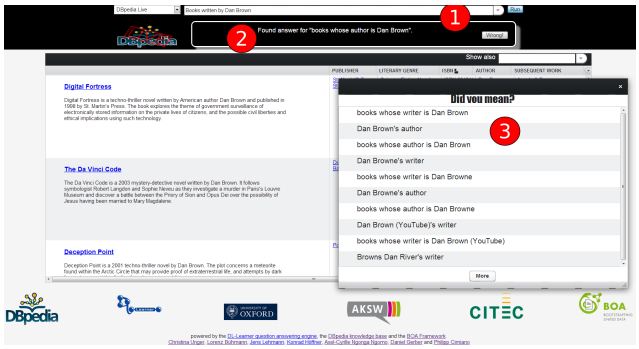

Figure 4: Screenshot of the TBSL online demo at `http://autosparql-tbsl.dl-learner.org/`.

5. REFERENCES

[1] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of HLT*, pages 138–145, 2002.

[2] Jens Lehmann and Lorenz Bühmann. Autosparql: Let users query your knowledge base. In *Proceedings of ESWC 2011*, 2011.

[3] Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. Sorry, i don't speak sparql – translating sparql queries into natural language. In *Proceedings of WWW*, 2013.

[4] Saeedeh Shekarpour, Sören Auer, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, Sebastian Hellmann, and Claus Stadler. Keyword-driven sparql query generation leveraging background knowledge. In *ACM/IEEE WI*, 2011.

[5] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over RDF data. In *Proceedings of WWW*, 2012.