SPARTIQULATION: Verbalizing SPARQL Queries

Basil Ell, Denny Vrandečić, and Elena Simperl

KIT, Karlsruhe, Germany
`{basil.ell,denny.vrandecic,elena.simperl}@kit.edu`

Abstract. Much research has been done to combine the fields of Databases and Natural Language Processing. While many works focus on the problem of deriving a structured query for a given natural language question, the problem of query verbalization – translating a structured query into natural language – is less explored. In this work we describe our approach to verbalizing SPARQL queries in order to create natural language expressions that are readable and understandable by the human day-to-day user. These expressions are helpful when having search engines generate SPARQL queries for user-provided natural language questions or keywords and enable the user to check whether the right question has been understood. While our approach enables verbalization of only a subset of SPARQL 1.1, this subset applies to 85% of the 209 queries in our training set. These observations are based on a corpus of SPARQL queries consisting of datasets from the QALD-1 challenge and the ILD2012 challenge.

Keywords: SPARQL, natural language generation, verbalization

1 Introduction

Much research has been done to combine the fields of Databases and Natural Language Processing to provide natural language interfaces to database systems [22]. While many works focus on the problem of deriving a structured query for a given natural language question or a set of keywords [10, 21, 27], the problem of query verbalization – translating a structured query into natural language – is less explored. In this work we describe our approach to verbalizing SPARQL queries in order to create natural language expressions that are readable and understandable by the human day-to-day user. The verbalized form of the generated query is helpful for users since it allows them to understand how

In: C. Unger, P. Cimiano, V. Lopez, E. Motta, P. Buitelaar, R. Cyganiak (eds.): Proceedings of Interacting with Linked Data (ILD 2012), Workshop co-located with the 9th Extended Semantic Web Conference, Heraklion, Greece, 28-05-2012, published at `http://ceur-ws.org`

the results have been retrieved and whether the right question has been asked to the queried knowledge base.

In this paper we describe the current state of our SPARTIQULATION system[1] which allows verbalization of a subset of SPARQL 1.1 SELECT queries.

The remainder of this paper is structured as follows. Section 2 gives an overview of our query verbalization approach in terms of our system's architecture and the tasks that it performs. Section 3 relates it to existing work, and in Section 4 conclusions are drawn and an outlook is provided.

2 Query Verbalization Approach

2.1 Introduction

Our approach is inspired by the pipeline architecture for natural language generation (NLG) systems and the set of seven tasks performed by such systems as introduced by Reiter and Dale [19]. The input to such a system can be described by a four-tuple (k, c, u, d) – where k is a knowledge source (not to be confused with the knowledge base a query is queried against), c the communicative goal, u the user model, and d the discourse history. Since we neither perform user-specific verbalization nor do we perform the verbalization in a dialog-based environment, we omit both the user model and the discourse history. The communicative goal is to communicate the meaning of a given SPARQL query q. However, there are multiple options. Three basic types of linguistic expressions can be used: i) statements that describe the search results where this description is based on the query only and not on the actual results returned by a SPARQL endpoint (e.g. *Bavarian entertainers and where they are born*), ii) a question can be formulated about the existence of knowledge of a specified or unspecified agent (e.g. *Which Bavarian entertainers are known and where are they born?*), and iii) a query can be formulated as a command (e.g. *Show me Bavarian entertainers and where they are born*). In this work we decided to explore how to verbalize queries as statements. Therefore, the communicative goal is to verbalize a query as a statement – more precisely in English.

2.2 Components and Tasks

In this section we present our approach along the seven tasks involved in NLG according to Reiter and Dale [19]. This work is the first step towards the verbalization of SPARQL queries. So far we put a focus on *document structuring*, but not on *lexicalization*, *aggregation*, *referring expression generation*, *linguistic realisation*, and *structure realisation*.

The pipeline architecture is depicted in Figure 1. Within the Document Planner the content determination process creates messages and the document structuring process combines them into a document plan (DP) which is the output of this component and the input to the Microplanner component. Within the

[1] The name is derived from joining *SPARQL* and *articulation*.

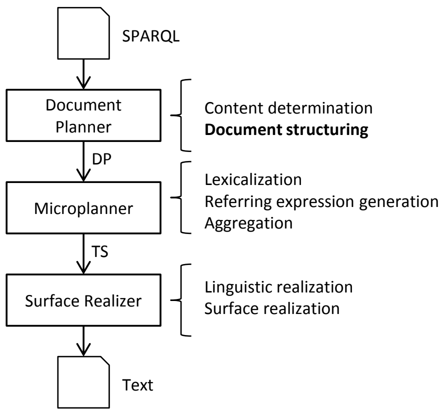


Fig. 1. Pipeline architecture of our NLG system

Microplanner the processes lexicalization, referring expression generation and aggregation take place, which results in a text specification (TS) that is made up of phrase specifications. The Surface Realizer then uses this text specification to create the output text.

Content determination is the task to decide which information to communicate in the text. In the current implementation we decided not to leave this decision to the system. What is communicated is the meaning of the input query without communicating which vocabularies are used to express the query. Therefore, in this task no action is performed.

Document structuring is the task to construct messages from the input query and to decide for their order and structure. These messages are used for representing information in the domain, such as the class to which the selected entities belong to or the number to which the result set is limited. We present the set of message types after introducing the notion of the main entity and the graph transformation. Our observations are based on a corpus of SPARQL queries consisting of datasets from the QALD-1 challenge[2] and the ILD2012 challenge.[3] The full dataset contains 263[4] SPARQL SELECT queries and associated manually created questions. In order to leave parts of this dataset for future evaluation we only regarded 80% of each dataset as training data. Since in our approach we cannot yet handle all features of the SPARQL 1.1 standard, we had to exclude some queries. Within this training set of 209 queries we excluded the queries with the features UNION (22), GROUPBY (7), and those where the triple patterns

[2] `http://www.sc.cit-ec.uni-bielefeld.de/qald-1`

[3] `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/`

[4] For nine questions no query is given since they are out of scope regarding the datasets provided for the challenge. 28 queries are ASK queries.

within the WHERE clause do not form a connected graph (3). This means that this subset covers 85% of the queries within the training set.

We perform a transformation of the query graph, since it reduces the number of necessary message types which are shown in Table 1. Thus it simplifies the verbalization. This transformation is based on the observation that in most queries one variable can be identified that is rendered as the subject of a sentence. For example when querying for mountains (bound to variable `?mountain`) and their elevations (bound to variable `?elevation`), then `?mountain` is verbalized as the subject of the verbalization `mountains and their elevations`. We refer to this variable as the *main entity* of a query. However, for some queries no such element exists. Consider for example the query `SELECT * WHERE { ?a dbpedia:marriedTo ?b .}`. Here a tuple is selected and in a possible verbalization *Tuples of married entities*[5] no single variable appears represented as a subject. In order to identify the main entity we define Algorithm 1 that applies the ordered list of rules shown in Figure 2. These rules propose the exclusion of members from a candidate set. We derived them by looking at queries within the training set having multiple candidates. The candidate set C is initialized with variables that appear in the select clause and the algorithm eliminates candidates step by step. Q denotes the set of triples within the WHERE clause of a query, R_t is the property `rdf:type` and R_l is a labeling property from the set of 36 labeling properties identified by [6]. The application of an exclusion rule R_i to a candidate set C, denoted by $R_i(C)$, results in the removal of the set E proposed by the reduction rule.

Rule 1 $E := \{x \in C \mid \text{"}x\ \textit{appears in OPTIONAL only}\text{"}\}$
Rule 2 $E := \{z \in C \mid \neg\exists(z, R_t, u) \in Q\}$
  if $\exists c_1 \in C : \neg\exists(c_1, R_t, x) \in Q \wedge \exists c_2 \in C : \neg\exists(c_2, R_t, y) \in Q$
Rule 3 $E := \{z \in C \mid \neg\exists(z, R_l, u) \in Q\}$
  if $\exists c_1 \in C : \neg\exists(c_1, R_l, x) \in Q \wedge \exists c_2 \in C : \neg\exists(c_2, R_l, y) \in Q$

Fig. 2. Exclusion rules

The rules can be described as follows where the numbers show how often a rule was successful in reducing the candidate set for the 209 queries within our training set. *Rule 1* (85, 40.67%) proposes removing candidates that appear within the WHERE clause only within OPTIONAL blocks. *Rule 2* (12, 5.74%) proposes removing candidates that represent subjects that are not constrained via `rdf:type` in the case that there are candidates that are constrained via *rdf:type*. *Rule 3* (48, 22.97%) proposes removing candidates for which no label is constrained or requested in the case that there are candidates for which this is the case. In some cases (64, 30.62%) no rule was applied since the candidate set contained only a single variable. For all queries given these rules the main entity has been identified. While our actual list of exclusion rules contained more rules these were never applied for the given training data and thus omitted here.

[5] DBpedia provides no `rdfs:domain` and `rdfs:range` information, such as `foaf:Person` for this property. Therefore here we give a generic verbalization to demonstrate the problem.

Algorithm 1 Applying reduction rules to candidate set.

```
if $|C| = 1$ then
    return $C$
while $|C| > 1$ do
    for all $R_i \in R$ do
        if $|R_i(C)| > 0$ then
            $C \leftarrow R_i(C)$
            if $|C| = 1$ then
                return $C$
return $\emptyset$
```

We transform, as shown in Algorithm 2, queries in a way that the query graph is converted into a graph where the main entity is the root and all edges point away from the root if the query does not come in that shape already. Therefore the algorithm maintains three sets of edges: edges that are already processed (P), edges that need to be followed (F), and edges that need to be transformed (T) which means reversed. An edge is reversed by exchanging subject and object and by marking the property (p) as being reversed (p^r).

Algorithm 2 Graph transformation

```
$P \leftarrow \emptyset,\ F \leftarrow \{(s,p,o) \in Q | s = m\},\ T \leftarrow \{(s,p,o) \in Q | o = m\}$          (init)
while $F \neq \emptyset$ or $T \neq \emptyset$ do
    for all $(s_i, p_i, o_i) \in F$ do
        for all $(s_j, p_j, o_j) \in Q \setminus (P \cup F \cup T)$ do
            if $o_i = s_j$ then
                $F \leftarrow F \cup \{(s_j, p_j, o_j)\}$
            else if $o_i = o_j$ then
                $T \leftarrow T \cup \{(s_j, p_j, o_j)\}$
        Move $(s_i, p_i, o_i)$ from $F$ to $P$
    for all $(s_i, p_i, o_i) \in T$ do
        for all $(s_j, p_j, o_j) \in Q \setminus (P \cup F \cup T)$ do
            if $s_i = s_j$ then
                $F \leftarrow F \cup \{(s_j, p_j, o_j)\}$
            else if $s_i = o_j$ then
                $T \leftarrow T \cup \{(s_j, p_j, o_j)\}$
        $T \leftarrow T \setminus \{(s_i, p_i, o_i)\}$
        $P \leftarrow P \cup \{(o_i, p_i^r, s_i)\}$
return $P$
```

We identified the set of 14 message types (MT), shown in Table 1 that allow us to represent the 209 queries from our training set. The first 9 MTs represent directed paths in the query graph which means that for each directed path that begins at the main entity, we represent this path with a message. Each path starts at the main entity (M) and consists of none to many pairs ($(RV)^*$) of a resource (R) followed by a variable (V). Moreover, they may contain a labeling property (R_l) or the `rdf:type` property (R_t). VAR represent all information about a variable, such as its name, whether it is the main entity, whether it is selected, distinct, optional, counted, or whether any filter is specified for this variable. The MTs $ORDERBY$, $LIMIT$, $OFFSET$ and $HAVING$ represent the respective SPARQL features.

The document plan (DP), which is the output of the Document Planner and input to the Microplanner, structures the content as follows: in the first part, which can later be verbalized as one ore more sentences, the main entity and its constraints are described, followed by a description of the requests (the variables besides the main entity that appear in the select clause) and their constraints. In a second part, if available and not already communicated in the first part, the selection modifiers are verbalized. According to these 3 categories – abbreviated with *cons*, *req*, and *mod* – we classify the MTs as follows. The MTs (1), (2), (4), (6), (7), and (9) from Table 1 belong to the class *cons*, the MTs (3), (5), and (8) belong to the class *req*. MTs (1), (2), (4), (6), (7) and (9) may also belong to class *req* if they contain a variable besides the main entity that appears in the select clause. MTs (10) – (14) belong to the class *mod*. While this set of message types is sufficient for the given training set, which means that all queries can be represented using these message types, we extended this list with 7[6] more types in order to be prepared for queries such as `SELECT ?s ?p ?o WHERE { ?s ?p ?o. }` and `SELECT ?p WHERE { ?s ?p ?o. }` where instead of generating text, canned text is used, such as *All triples in the database* and *Properties used in the database.*

nr name	nr name	nr name
(1) $M(RV)^*RR$	(2) $M(RV)^*RL$	(3) $M(RV)^*RV$
(4) $M(RV)^*R_lR$	(5) $M(RV)^*R_lV$	(6) $M(RV)^*R_lL$
(7) $M(RV)^*R_tR$	(8) $M(RV)^*R_tV$	(9) $M(RV)^*R_tL$
(10) VAR	(11) $ORDERBY$	(12) $LIMIT$
(13) $OFFSET$	(14) $HAVING$	

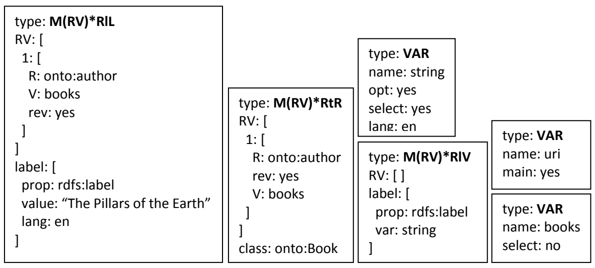
Table 1. Message types

As an example the SPARQL query in Listing 1 is represented using the 6 messages shown in Figure 3. Note that due to the graph transformation the property `onto:author` is reversed which is denoted by `rev: yes` within the data structure stored in the messages. This query can be verbalized as: *Authors of books with English name "The Pillars of the Earth" and if available their English names.* Note that plural (*authors*, *books* and *names* instead of *author*, *book*, and *name*) is used per default. The filter for English labels is stored within the message representing the variable `string`.

[6] Given that all three variables can either be selected or not selected and at least one variable needs to be selected, this results in 7 combinations.

```
SELECT ?uri ?string WHERE {
   ?books rdf:type onto:Book .
   ?books onto:author ?uri .
   ?books rdfs:label "The␣Pillars␣of␣the␣Earth"@en .
   OPTIONAL {
      ?uri rdfs:label ?string .
      FILTER (lang(?string) = 'en')
   }
}
```

Listing 1. Who wrote the book The pillars of the Earth? – SPARQL query



```
type: M(RV)*RIL
RV: [
  1: [
    R: onto:author
    V: books
    rev: yes
  ]
]
label: [
  prop: rdfs:label
  value: "The Pillars of the Earth"
  lang: en
]
```

```
type: M(RV)*RtR
RV: [
  1: [
    R: onto:author
    rev: yes
    V: books
  ]
]
class: onto:Book
```

```
type: VAR
name: string
opt: yes
select: yes
lang: en
```

```
type: M(RV)*RIV
RV: [ ]
label: [
  prop: rdfs:label
  var: string
]
```

```
type: VAR
name: uri
main: yes
```

```
type: VAR
name: books
select: no
```

Fig. 3. Messages for the SPARQL query in Listing 1.

Lexicalization is the task of deciding what specific words to use for expressing the content. For each entity we dereference its URI and in case that RDF data is returned, we check if an English label is provided using one of the 36 labeling properties defined in [6]. Otherwise, we derive a label from the URI's local name using the patterns introduced by Hewlett et al. in [11].

Referring expression generation is the task of deciding how to refer to an entity. Considering the example *Entertainers born in Bavaria and where* they *are born*. Here, *they* is the expression that refers to the Bavarian entertainers.

Aggregation is the task to decide how to map structures created within the document planner onto linguistic structures such as sentences and paragraphs. For messages of type *cons* and *req* sentence parts are created that are joined into a single sentence. Messages of type *mod* are verbalized in further sentences. Aggregation is indispensable for concise verbalization. Since we split a query graph into (overlapping) paths where each path is represented by a message, aggregation would exploit these overlappings.

Linguistic realization is the task of converting abstract representations of sentences into real text. Text parts are generated for each of the message types (1) – (9) from Table 1. For each such type a rule is invoked that produces a sentence fragment, for example for the MT $MRVR_lL$ –which is an instance of the MT $M(RV)^*R_lL$ – the rule `article(lex(prop1)) + lex(prop1) + L` produces for two triples `?uri dbpedia:producer ?producer` and `?producer rdfs:label "Hal Roach"` the text `a producer Hal Roach`. The function `article` choses an appropriate article (`a` or `an`) depending on the lexicalization `lex(prop1)` of the property. This fragment is added to the part of the verbalization where the constraints for the main entity are described and may be joined by the word `and` with other constraint fragments.

Structure realization is the task to add markup such as HTML code to the generated text in order to be interpreted by the presentation system, such as a web browser. While this could be helpful to enhance the readability of a complex verbalization, which is the case in [2], we do not currently exploit this opportunity.

3 Related Work

While to the best of our knowledge no work is published on the verbalization of SPARQL queries, related work comes from three areas: verbalization of RDF data [5, 16, 24, 25, 29], verbalization of OWL ontologies [1, 3, 4, 7–9, 11, 12, 14, 20, 23, 26, 28], and verbalization of SQL queries [13, 17, 18]. Although the first two fields provide techniques that we can apply to improve the lexicalization and aggregation tasks, such as the template-based approach presented in [5], the document structuring task, on which we focus here, is rarely explored. Compared to the SQL verbalization work by Minock [17, 18], where they focus on tuple relational queries, our problem of verbalizing SPARQL queries is different in the sense that we strive for having a generic approach that can be applied to any datasource without being tied to any schema. Patterns need to be manually created to cover all possible combinations for each relation in the schema whereas in our work we defined a set of message types that are schema-agnostic. Koutrika et al. [13] annotate query graphs with template labels and explore multiple graph traversal strategies. Moreover, they identify a main entity (the *query subject*), perform graph traversal starting from that entity, and distinguish between *cons* (*subject qualifications*) and *req* (*information*).

4 Conclusions and Outlook

For the task of verbalizing SPARQL queries we focused on a subset of the SPARQL 1.1 standard which covers 88% of the queries in a corpus of 209 SPARQL queries. Evaluation will have to show the representativeness of this corpus compared to real-life queries and the qualities of the verbalizations generated using our SPARTIQULATION system. While in our architecture 6 tasks are needed to generate verbalizations, our main focus has been the task of *document structuring* which we described in this work. In order to realize the full verbalization pipeline, 5 other tasks need to be explored in future work. Since the current approach is mostly schema-agnostic – only terms from the RDFS vocabulary are regarded during document structuring – we believe that this approach is generic in terms of being applicable to queries for RDF datasources using any vocabularies. However, in the future the tasks of lexicalization can be improved by regarding schemas such as FOAF since persons are treated differently in verbalizations then non-persons, genders can be regarded etc. Having message types designed for specific vocabularies allows to tailor the verbalization to a specific use case and may lead to more concise verbalizations. In the current implementation message types are hard-coded thus limiting the flexibility of the approach. Having the possibility to load a set of message types into the system would add the possibility to integrate automatically learned or application-specific message types.

Acknowledgements

The work presented in this paper is supported by the European Union's 7th Framework Programme (FP7/2007-2013) under Grant Agreement 257790.

References

1. Aguado, G., Bañón, A., Bateman, J. A., Bernardos, S., Fernández, M., Gómez-Pérez, A., Nieto, E., Olalla, A., Plaza, R., Sánchez, A.: ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation. In: Proc. of the Workshop on Applications of Ontologies and Problem Solving Methods, ECAI 1998
2. Bontcheva, K.: Generating tailored textual summaries from ontologies. In: Proc. of ESWC 2005 531–545
3. Bontcheva, K., Wilks, Y.: Automatic Report Generation from Ontologies: the MIAKT approach. In: Proc. of NLDB 2004
4. Cregan, A., Schwitter, R., Meyer, T.: Sydney OWL Syntax - towards a Controlled Natural Language Syntax for OWL 1.1. In: Proc. of OWLED 2007
5. Davis, B., Iqbal, A., Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Handschuh, S.: RoundTrip Ontology Authoring. In: Proc. of The Semantic Web, ISWC 2008 50–65
6. Ell, B., Vrandečić, D., Simperl, E.: Labels in the Web of Data. In: Proc. of ISWC 2011
7. Fliedl, G., Kop, C., Vöhringer, J.: Guideline based evaluation and verbalization of OWL class and property labels. Data Knowl. Eng. **69**(4) (2010) 331–342
8. Galanis, D., Androutsopoulos, I.: Generating multilingual descriptions from linguistically annotated OWL ontologies: the NaturalOWL system. In: Proc. of the Eleventh European Workshop on Natural Language Generation (2007) 143–146

9. Gareva-Takasmanov, L., Sakellariou, I.: OWL for the Masses: From Structured OWL to Unstructured Technically-Neutral Natural Language. In: Proc. of BCI 2009 260–265
10. Haase, P., Herzig, D., Musen, M., Tran, D. T.: Semantic Wiki Search. In: Proc. of ESWC 2009 445–460
11. Hewlett, D., Kalyanpur, A., Kolovski, V., Halaschek-Wiener, C.: Effective NL Paraphrasing of Ontologies on the Semantic Web. In: Proc. of the End User Semantic Web Interaction Workshop at the 4th International Semantic Web Conference (2005)
12. Kaljurand, K., Fuchs, N. E.: Verbalizing OWL in Attempto Controlled English. In: Proc. of OLWED 2007
13. Koutrika, G., Simitsis, A., Ioannidis, Y. E.: Explaining Structured Queries in Natural Language. In: Proc. of ICDE'10 (2010)
14. Liang, S. F., Stevens, R., Rector, A.: OntoVerbal-M: a Multilingual Verbaliser for SNOMED CT. In: Proc. of Multilingual Semantic Web (2011)
15. McKay, B. D.: Practical graph isomorphism. Congressus Numerantium **30** (1981) 45–87
16. Mellish, C., Sun, X.: The semantic web as a Linguistic resource: Opportunities for natural language generation. Knowl.-Based Syst. **19**(5) (2006) 298–303
17. Minock, M.: A Phrasal Approach to Natural Language Interfaces over Databases. In: Proc. of NLDB 2005 333–336
18. Minock, M.: C-Phrase: A system for building robust natural language interfaces to databases. Data Knowl. Eng. **69**(3) (2010) 290–302
19. Reiter, E., Dale, R.: Building Natural Language Generation Systems. Cambridge University Press (2000)
20. Schütte, N.: Generating natural language descriptions of ontology concepts. In: Proc. of the 12th European Workshop on Natural Language Generation (2009) 106–109
21. Shekarpour, S., Auer, S., Ngonga Ngomo, A.-C., Gerber, D., Hellmann, S., Stadler, C.: Keyword-driven SPARQL Query Generation Leveraging Background Knowledge. In: Proc. of International Conference on Web Intelligence (2011)
22. Simitsis, A., Ioannidis, Y. E.: DBMSs Should Talk Back Too. In: Proc. of CIDR 2009
23. Stevens, R., Malone, J., Williams, S., Power, R.: Automating class definitions from OWL to English. In: Proc. of Bio-Ontologies 2010: Semantic Applications in Life Sciences SIG at the 18th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB 2010)
24. Sun, X., Mellish, C.: Domain Independent Sentence Generation from RDF Representations for the Semantic Web. In: Proc. of Combined Workshop on Language-Enabled Educational Technology and Development and Evaluation of Robust Spoken Dialogue Systems, ECAI 2006
25. Sun, X., Mellish, C.: An experiment on "free generation" from single RDF triples. In: Proc. of the Eleventh European Workshop on Natural Language Generation (2007) 105–108
26. Third, A., Williams, S., Power, R.: OWL to English: a tool for generating organised easily-navigated hypertexts from ontologies. In: Proc. of ISWC 2011
27. Tran, D. T., Wang, H., Haase, P.: Hermes: Data Web search on a pay-as-you-go integration infrastructure. Journal of Web Semantics **7**(3) (2009)
28. Wilcock, G.: Talking OWLs: Towards an Ontology Verbalizer. In: Proc. of Human Language Technology for the Semantic Web and Web Services, ISWC 2003 109–112

29. Wilcock, G., Jokinen, K.: Generating Responses and Explanations from RDF/XML and DAML+OIL. In: Proc. of IJCAI 2003