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 that generate SPARQL queries for user-provided natural language questions or keywords. Displaying verbalizations of generated queries to a user enables 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 90% 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 [27, 10, 21, 30], 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.

When a system generates SPARQL queries for a given natural language question or a set of keywords, the verbalized form of the generated query is helpful for users, since it allows them to understand whether the right question has been asked to the queried knowledge base and, if the query is executed and results are presented, how the results have been retrieved. Therefore, verbalization of SPARQL queries may improve the experience of users of any such SPARQL

query generating system such as natural language-based question answering systems or keyword-based search systems.

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 in English.

The remainder of this paper is structured as follows. Section 2 gives an overview of the query verbalization approach in terms of the system architecture and the tasks that it performs. Section 3 presents the elements of our approach, Section 4 revisits existing work, and in Section 5 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*). Thus, the communicative goal can be reached in three modes: *statement verbalization*, *question verbalization*, or *command verbalization*. Since the only communicative goal is to communicate the meaning of a query to a user, the various modes the system is built for and the omissions of both the user model and the discourse history, the input to our system is a tuple (k, m) where k is the SPARQL query and $m \in \{statement, question, command\}$ is a mode.

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

[1] The name is derived from joining *SPARQL* and *articulation*. A demo is available at `http://km.aifb.kit.edu/projects/spartiqulator`

not on *lexicalization*, *aggregation*, *referring expression generation*, *linguistic realisation*, and *structure realisation*. Note that the modes in which a communicative goal can be reached are regarded in the task *linguistic realization* only.

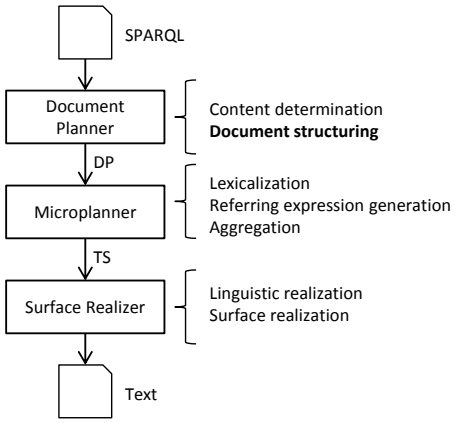

Fig. 1. Pipeline architecture of our NLG system

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 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. For example if *title* occurs in the verbalization and is derived from the label of a property, then it is hidden to the user whether this has been derived from `http://purl.org/dc/elements/1.1/title` or `http://purl.org/rss/1.0/title`.

Document structuring is the task to construct independently verbalizable messages from the input query and to decide for their order and structure. These messages are used for representing information, such as that a variable is selected, the class to which the entities selected by the query belong to or the number to which the result set is limited. The output of this task is a document plan. Our approach to document structuring consists of the following elements:

1. Query graph representation
2. Main entity identification
3. Query graph transformation
4. Message types
5. Document plan

These are the main contributions of this work. We continue this section with an introduction of the remaining tasks. In Chapter 3, each of the elements of our approach are presented in detail.

Lexicalization is the task of deciding which 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. In case of properties, the 7 patterns introduced by Hewlett et al. in [11] are used. For example, Hewlett et al. provide the following pattern:

(is) VP P
- Examples: producedBy, isMadeFrom
- Expansions: X is produced by Y, X is made from Y

The local name *producedBy* of a property *ex:producedBy* is expanded to *produced By* and its constituents are part-of-speech tagged. The expansion rule given for this pattern declares that a triple `ex:X ex:producedBy ex:Y` can be verbalized as *X is produced by Y*.

The main entity[2] is verbalized as *things*. If a constraint for the class of the main entity such as `?m rdf:type yago:AfricanCountries` is given, then it can be verbalized as *African countries*.[3] If the query is limited to a single result using `LIMIT 1` and no sort order is defined using `ORDER BY`, then it can be verbalized as *An African country*. Otherwise, if a sort order is defined such as `ORDER BY DESC(?population)`, then it can be verbalized as *The African country* as in *The African country with the highest population*. Other variables are also verbalized as *things* unless a type is either explicitly given using `rdfs:type` or implicitly given using `rdfs:domain` or `rdfs:range`. For example, this information is regarded when verbalizing the query `SELECT ?states ?uri WHERE { ?states dbo:capital ?uri .}` as *Populated places and their capitals*. Here, the domain of the property *dbo:capital* is defined as *dbpedia-owl:PopulatedPlace*.

Referring expression generation is the task of deciding how to refer to an entity that is already introduced. Consider the following two example verbalizations:

1. *Albums of things named Last Christmas and where available their labels.*

[2] The main entity is rendered as the subject of the verbalization.
[3] *African countries* is the rdfs:label of yago:AfricanCountries.

2. *Albums of things named Last Christmas and where available the labels of these albums.*

In the beginning of the verbalizations the entities albums and things are introduced. At the end labels are requested. In the first verbalization it is not clear whether the labels of the albums or the labels of the things are requested, whereas in the second verbalization it is clear that the labels of the albums are requested.

Aggregation is the task to decide how to map structures created within the document planner onto linguistic structures such as sentences and paragraphs. For example, without aggregation a query such as `SELECT ?m WHERE {?m dbo: starring res:Julia_Roberts . ?m dbo:starring res:Richard_Gere . }` would be verbalized as *Things that are starring Julia Roberts and that are starring Richard Gere*. With aggregation the result is more concise: *Things that are starring Julia Roberts as well as Richard Gere.*

Linguistic realization is the task of converting abstract representations of sentences into real text. Thereby the modes *statement*, *question*, and *command* are regarded. As introduced in the next chapter, chunks of content of a SPARQL query are represented as messages given the list of message types (MT) from Figure 5. For each of the message types (1)-(9) 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 named "Hal Roach"`. The function `article` choses an appropriate article (`a` or `an`) depending on the lexicalization `lex(prop1)` of the property.

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. Bontcheva [2] points out that hypertext usability studies [18] have shown that formatting is important since it improves readability. Indenting complex verbalizations, adding bullet points and changing the font size can help to communicate the meaning of a query.

3 Document structuring

The elements our approach consists of can be summarized as follows. We transform textual SPARQL SELECT queries into a graphical representation – the query graph – which is suitable for performing traversal and transformational actions. Within a query graph we identify the main entity which is a variable that is rendered as subject of a verbalization. After a main entity is identified the graph is transformed into a graph where the main entity is the root. Then the

graph is split into independently verbalizable parts called messages. We define a set of message types that allow to represent a query graph using messages. Message types are classified due to their role within the verbalization. The document plan is presented which orders the messages according to their classes and is the output of the Document Planner – the first component in our NLG pipeline.

Some observations in this chapter, namely regarding the main entity identification in Section 3.2 and the message types in Section 3.4, are based on a training set. This training set is derived from a corpus of SPARQL queries consisting of datasets from the QALD-1 challenge[4] and the ILD2012 challenge.[5] The full dataset contains 263[6] SPARQL SELECT queries and associated manually created questions. In order to derive a training set we used 80% of each dataset as training data – in total 209 SELECT queries.[7] 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 UNION feature (20 queries) and those that were not parsable (1 query). This means that this subset – 188 queries in total – covers 90% of the queries within the training set.

3.1 Query graph representation

We parse a SPARQL query into a query graph since this allows for easier manipulation of the query compared to its textual representation. Thereby each subject S and object O of a triple pattern $< S, P, O >$ within the query is represented by a node in the query graph. The nodes are connected by edges labeled with P. Since $< S, P, O >$ is a triple pattern and not an RDF triple, this means that each element can be a variable. Unless the subject or object is a variable, for each subject or object an own node is created. Therefore multiple non-variable nodes with the same label may exist. For every variable that appears within the query in subject or object position only one node is created.

As an example regard the SPARQL query in textual representation in Listing 1 and the visual representation of the qurery graph in Figure 2. In this visual representation,[8] nodes are filled if they represent resources and not filled if they represent variables. Nodes are labeled with the name of the resource, variable, literal or blank node respectively. Labels of variables begin with a question mark. Labels of variables that appear in the SELECT clause of a query are underlined. Literal values are quoted. Edges are labeled with the name of the property and point from subject to object. Filters are attached to their respective variable(s)

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

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

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

[7] 37 queries from `2011-dbpedia-train`, 36 queries from `2011-musicbrainz-train`, 68 queries from `2012-dbpedia-train`, and 68 queries from `2012-musicbrainz-train`.

[8] Note that this not a complete visual representation of SPARQL SELECT queries and only solves for the purpose of visually representing the query graph examples.

and parts of the graph that appear only within an OPTIONAL clause are marked as such.

```
SELECT DISTINCT ?uri ?string WHERE {
  ?states rdf:type yago:AfricanCountries .
  ?states dbo:capital ?uri .
  ?uri dbp:population ?population .
  FILTER ( ?population < 1000000 ) .
  OPTIONAL {
    ?uri rdfs:label ?string.
    FILTER (lang(?string) = 'en')
  }
}
```

Listing 1. Example SPARQL query

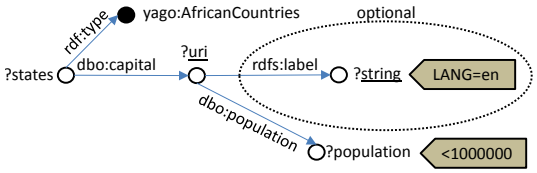

Fig. 2. Example query graph

3.2 Main entity identification

We perform a transformation of the query graph, since it reduces the number of message types that are necessary to represent information contained in the query graph thus simplifying the verbalization process. 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*[9] 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 3. These rules propose the exclusion of members from a candidate set. We derived

[9] 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 effect of missing domain and range information.

them by looking at queries within the training set having multiple candidates. The candidate set C for a given query is initialized with variables that appear in the SELECT clause[10] 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 which is proposed by the exclusion rule.

We identified the ordered list of exclusion rules shown in Figure 3. The numbers show how often a rule was successful in reducing the candidate set for the 188 queries within our training set. In some cases (61, 32.45%) no rule was applied since the candidate set contained only a single variable. In the case that given the rules above the algorithm does not manage to reduce the candidate set to a single variable (21, 11.17%), the first variable in lexicographic order is selected.

Rule 1 $E := \{x \in C \mid$ "x *appears in OPTIONAL only*"$\}$
Rule 1 (71, 37.77%) proposes removing candidates that appear within the WHERE clause only within OPTIONAL blocks. For example in a query `SELECT ?a WHERE { ex:R1 ex:R2 ?a . OPTIONAL { ?a ex:R3 ?b . } }` the variable `b` is removed from the candidate set which contains `a` and `b`.

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 2 (10, 5.32%) 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*. For example in a query `SELECT ?a ?b WHERE { ?a rdf:type ex:R1 . ?b ex:R2 ex:R3 . }` with `ex:R2` $\neq$ `rdf:type`, the variable `b` is removed from the candidate set which contains `a` and `b`.

Rule 3 $E := \{z \in C \mid \neg\exists(z, R_l, u) \in Q\}$
if $\exists c_1 \in C : \neg\exists(c_1, R_l, x) \in Q \wedge \exists c_2 \in C : \neg\exists(c_2, R_l, y) \in Q$
Rule 3 (25, 13.3%) proposes removing candidates for which the existence of a label is not constrained or requested in the case that there are candidates for which the existence of a label is constrained or a label is requested. For example in a query `SELECT ?a ?b WHERE { ?a <`R_l`> ?l . ?b ex:R2 ex:R3 . }` where R_l is a labelling property, the variable `b` is removed from the candidate set which contains `a` and `b`.

Fig. 3. Exclusion rules

As an example regard the SPARQL query in Listing 1 which is visually presented in Figure 2. The candidate set is initialized as $\{uri, string\}$. Rule 1 proposes to remove the variable *string* since it appears only within an OPTIONAL clause. Since the candidate set is reduced to $\{uri\}$ containing a single entity, this entity is the main entity.

[10] In case of a `SELECT *` query, all variables within the WHERE clause are candidates.

Algorithm 1 Applying exclusion rules to candidate set.

```
if |C| = 1 then
    return C
while |C| > 1 do
    for all R_i ∈ R do
        if |R_i(C)| > 0 then
            C ← R_i(C)
            if |C| = 1 then
                return C
return ∅
```

3.3 Query graph transformation

Algorithm 2 transforms a query graph into a graph for which the main entity is the root and all edges point away from the root. 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

The query graph shown in Figure 2 is transformed into the query graph shown in Figure 4 where the main entity – the variable *uri* – is highlighted. Compared to the graph before the transformation, the edge *dbo:capital* was reversed. Therefore this edge now points away from the main entity and is marked as being reversed by the minus in superscript.

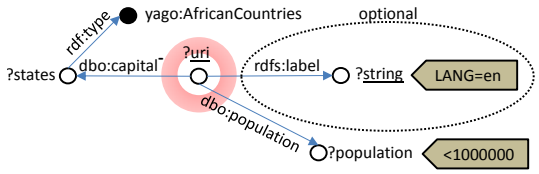

Fig. 4. Example query graph after transformation

3.4 Message types

We identified the set of 14 message types (MT), shown in Figure 5 that allow us to represent the 209 queries from our training set. Here, $M(RV)*$ denotes a path beginning at the main entity via an arbitrary number of property-variable pairs such as `?M <ex:R1> ?V1 . ?V1 <ex:R2> ?V2 .` 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. R_l denotes a labeling property and R_t the property `rdf:type`. The MTs $ORDERBY$, $LIMIT$, $OFFSET$ and $HAVING$ represent the respective SPARQL features.

As an example the SPARQL query in Listing 1 is represented using the 7 messages shown in Figure 6. Note that due to the graph transformation the property `dbo:capital` is reversed which is denoted by `REV: 1` in message 2. This query can be verbalized as: *English names of African countries having capitals which have a population of less than 1000000 and the English names of these capitals.* Note that the plural form *capitals* instead of *capital* is used per default since no information is available that a country has exactly one capital. The filter for English labels is stored within message 6 representing the variable `string` as `lang: en`.

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[11] 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.*

3.5 Document plan

The document plan (DP), which is the output of the Document Planner and input to the Microplanner, contains the set of messages and defines their order. The verbalization consists of two parts. In the first part 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

[11] 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.

(1) $\boldsymbol{M(RV)^*RR}$ Messages of this type represent (RV)*-paths in the query that end with a resource in property position and a resource in object position, for example the path `?main ex:starring ex:Richard_Gere`.
(2) $\boldsymbol{M(RV)^*RL}$ Example (RV)*-path: `?main ex:date "1960-12-29"^^xsd:dateTime`.
(3) $\boldsymbol{M(RV)^*RV}$ Example (RV)*-path: `?main ex:duration ?d`.
(4) $\boldsymbol{M(RV)^*R_lR}$ While a query such as `SELECT * WHERE { ?m rdfs:label ex:R . }` is syntactically correct, semantically it is not valid since the range of the property *rdfs:label* is *rdfs:Literal* and not *rdfs:Resource*. We introduce this message type since we cannot assume that every query processed by our verbalizer is semantically correct.
(5) $\boldsymbol{M(RV)^*R_lV}$ Example (RV)*-path: `?main ex:label ?l`.
(6) $\boldsymbol{M(RV)^*R_lL}$ Example (RV)*-path: `?main ex:label "John Cage"@en`.
(7) $\boldsymbol{M(RV)^*R_tR}$ Example (RV)*-path: `?main rdf:type ex:SoloMusicArtist`.
(8) $\boldsymbol{M(RV)^*R_tV}$ Example (RV)*-path: `?main rdf:type ?t`.
(9) $\boldsymbol{M(RV)^*R_tL}$ Similarly to message type 4, a query such as `SELECT * WHERE { ?m rdf:type "L" . }` is semantically not valid since the range of the property *rdf:type* is *rdfs:Class* and not *rdfs:Literal*.
(10) $\boldsymbol{VAR}$ A message of this type stores the name of the variable, whether it appears within the select clause, whether it is the main entity, whether it appears only within optional clauses, whether it is counted as in `SELECT COUNT(DISTINCT ?m)`, and the list of filters based on this variable.
(11) $\boldsymbol{ORDERBY}$ A message of this type stores the order of the solution sequence as specified by the ORDER BY clause, such as `ODER BY DESC(?main)`.
(12) $\boldsymbol{LIMIT}$ If a limit is specified using `LIMIT 10` then this integer value is stored.
(13) $\boldsymbol{OFFSET}$ If an offset is specified, for example using `OFFSET 10`, then this integer value is stored.
(14) $\boldsymbol{HAVING}$ A message of this type stores the order of the solution sequence as specified by the HAVING clause, such as in `GROUP BY ?x HAVING(AVG(?size) > 10)`.

Fig. 5. Message types

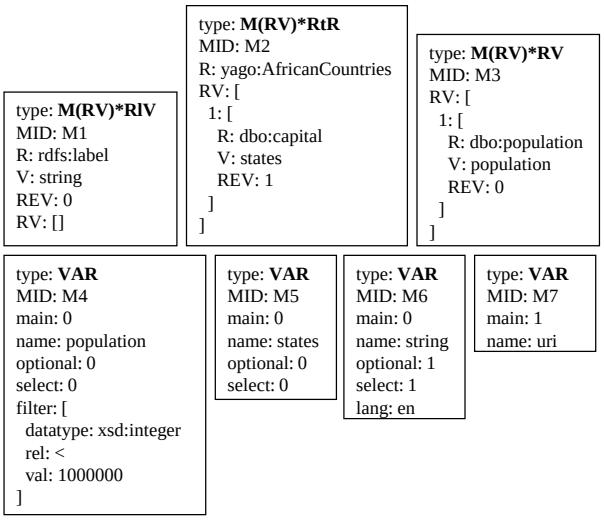

Fig. 6. Messages representing the SPARQL query in Listing 1.

selection modifiers are verbalized. According to these 3 categories – abbreviated with *cons*, *req*, and *mod* – we classify the message types (MT) as follows. The MTs (1), (2), (4), (6), (7), and (9) from Figure 5 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 (11) – (14) belong to the class *mod*. The VAR message is not classified since its only purpose is to store information about variables and variables are verbalized when verbalizing other messages. For the example query in Listing 1, which is represented using the messages shown in Figure 6, the messages M1, M2, and M3 are classified as *cons*, the message M1 is classified as *req* and no message is classified as *mod*.

4 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 [25, 15, 5, 24, 29], verbalization of OWL ontologies [1, 28, 23, 3, 11, 12, 8, 20, 7, 11, 4, 26, 9, 14], and verbalization of SQL queries [16, 17, 13]. 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 [16, 17], 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*).

5 Conclusions and Outlook

For the task of verbalizing SPARQL queries we focused on a subset of the SPARQL 1.1 standard which covers 90% of the queries in a corpus of 209 SPARQL SELECT 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 vocabularies RDFa and RDFS as well as a list of labeling properties from various vocabularies are regarded – 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 and OWL. FOAF is interesting since if an entity is a *foaf:Person*, it can be treated differently. For example the person's gender can be regarded. OWL is interesting since if it is known that a property is functional, then the singular form can be used instead of, as per default, the plural form.[12] 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 here was partially supported by the project RENDER under the grant number FP7257790 funded by the Seventh Framework Program of the European Commission. We would like to thank the attendees at the ILD workshop at ESWC 2012 in Heraklion for their valuable feedback.

[12] For example the query `SELECT ?m WHERE { ex:PersonA ex:wife ?m . }` can then be verbalized as *The wife of PersonA* instead of *The wives of PersonA*.

References

1. G. Aguado, A. Bañón, J. A. Bateman, S. Bernardos, M. Fernández, A. Gómez-Pérez, E. Nieto, A. Olalla, R. Plaza, and A. Sánchez. ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation. In *Workshop on Applications of Ontologies and Problem Solving Methods, ECAI'98*, 1998.
2. K. Bontcheva. Generating tailored textual summaries from ontologies. In *Proceedings of the Second European conference on The Semantic Web: research and Applications*, ESWC'05, pages 531–545, Berlin, Heidelberg, 2005. Springer-Verlag.
3. K. Bontcheva and Y. Wilks. Automatic Report Generation from Ontologies: the MIAKT approach. In *Nineth International Conference on Applications of Natural Language to Information Systems (NLDB'2004)*, 2004.
4. A. Cregan, R. Schwitter, and T. Meyer. Sydney OWL Syntax - towards a Controlled Natural Language Syntax for OWL 1.1. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
5. B. Davis, A. Iqbal, A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, and S. Handschuh. RoundTrip Ontology Authoring. pages 50–65. 2008.
6. B. Ell, D. Vrandečić, and E. Simperl. Labels in the Web of Data. In N. et al., editor, *Proceedings of the 10th International Semantic Web Conference (ISWC2011)*. Springer, Oktober 2011.
7. G. Fliedl, C. Kop, and J. Vöhringer. Guideline based evaluation and verbalization of OWL class and property labels. *Data Knowl. Eng.*, 69:331–342, April 2010.
8. D. Galanis and I. Androutsopoulos. Generating multilingual descriptions from linguistically annotated OWL ontologies: the NaturalOWL system. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, ENLG '07, pages 143–146, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
9. L. Gareva-Takasmanov and I. Sakellariou. OWL for the Masses: From Structured OWL to Unstructured Technically-Neutral Natural Language. In P. Kefalas, D. Stamatis, and C. Douligeris, editors, *BCI*, pages 260–265. IEEE Computer Society, 2009.
10. P. Haase, D. M. Herzig, M. Musen, and D. T. Tran. Semantic Wiki Search. In L. A. P. et al., editor, *6th Annual European Semantic Web Conference, ESWC2009, Heraklion, Crete, Greece*, volume 5554 of *LNCS*, pages 445–460. Springer Verlag, Juni 2009.
11. D. Hewlett, A. Kalyanpur, V. Kolovski, and C. Halaschek-Wiener. Effective NL Paraphrasing of Ontologies on the Semantic Web. In *End User Semantic Web Interaction Workshop at the 4th International Semantic Web Conference*, 2005.
12. K. Kaljurand and N. E. Fuchs. Verbalizing OWL in Attempto Controlled English. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
13. G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining Structured Queries in Natural Language. *ICDE '10*, 2010.
14. S. F. Liang, R. Stevens, and A. Rector. OntoVerbal-M: a Multilingual Verbaliser for SNOMED CT. In E. Montiel-Ponsoda, J. McCrae, P. Buitelaar, and P. Cimiano, editors, *Multilingual Semantic Web*, volume 775 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
15. C. Mellish and X. Sun. The semantic web as a Linguistic resource: Opportunities for natural language generation. *Knowl.-Based Syst.*, 19(5):298–303, 2006.

16. M. Minock. A Phrasal Approach to Natural Language Interfaces over Databases. In A. Montoyo, R. Muoz, and E. Mtais, editors, *NLDB*, volume 3513 of *Lecture Notes in Computer Science*, pages 333–336. Springer, 2005.
17. M. Minock. C-Phrase: A system for building robust natural language interfaces to databases. *Data Knowl. Eng.*, 69(3):290–302, Mar. 2010.
18. J. Nielsen. *Designing Web Usability: The Practice of Simplicity.* New Riders Publishing, Thousand Oaks, CA, USA, 1999.
19. E. Reiter and R. Dale. *Building Natural Language Generation Systems.* Natural Language Processing. Cambridge University Press, 2000.
20. N. Schütte. Generating natural language descriptions of ontology concepts. In *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, pages 106–109, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
21. S. Shekarpour, S. Auer, A.-C. Ngonga Ngomo, D. Gerber, S. Hellmann, and C. Stadler. Keyword-driven SPARQL Query Generation Leveraging Background Knowledge. In *International Conference on Web Intelligence*, 2011.
22. A. Simitsis and Y. E. Ioannidis. DBMSs Should Talk Back Too. *CoRR*, abs/0909.1786, 2009.
23. R. Stevens, J. Malone, S. Williams, and R. Power. Automating class definitions from OWL to English. In *Proceedings of Bio-Ontologies 2010: Semantic Applications in Life Sciences SIG at the 18th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB 2010)*, July?Summer 2010.
24. X. Sun and C. Mellish. Domain Independent Sentence Generation from RDF Representations for the Semantic Web. In C. Callaway, A. Corradini, J. Kreutel, J. Moore, and M. Stede, editors, *Proc. of Combined Workshop on Language-Enabled Educational Technology and Development and Evaluation of Robust Spoken Dialogue Systems (part of ECAI 2006)*, 2006.
25. X. Sun and C. Mellish. An experiment on "free generation" from single RDF triples. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, ENLG '07, pages 105–108, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
26. A. Third, S. Williams, and R. Power. OWL to English: a tool for generating organised easily-navigated hypertexts from ontologies. 2011.
27. D. T. Tran, H. Wang, and P. Haase. Hermes: Data Web search on a pay-as-you-go integration infrastructure. *Journal of Web Semantics*, 7(3), 2009.
28. G. Wilcock. Talking OWLs: Towards an Ontology Verbalizer. In *Human Language Technology for the Semantic Web and Web Services, ISWC'03*, pages 109–112, Sanibel Island, Florida, 2003.
29. G. Wilcock and K. Jokinen. Generating Responses and Explanations from RDF/XML and DAML+OIL, 2003.
30. M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Deep answers for naturally asked questions on the web of data. In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 445–449, New York, NY, USA, 2012. ACM.