SPARQL Query Verbalization for Explaining Semantic Search Engine Queries

Basil Ell[1], Andreas Harth[1], and Elena Simperl[2]

[1] Karlsruhe Institute of Technology (KIT),
Karlsruhe, Germany
{basil.ell, harth}@kit.edu,
[2] University of Southampton,
Southampton, United Kingdom
E.Simperl@soton.ac.uk

Abstract. In this paper we introduce Spartiqulation, a system that translates SPARQL queries into English text. Our aim is to allow casual end users of semantic applications with limited to no expertise in the SPARQL query language to interact with these applications in a more intuitive way. The verbalization approach exploits domain-independent template-based natural language generation techniques, as well as linguistic cues in labels and URIs.

Keywords: #eswc2014Ell, SPARQL verbalization, Natural Language Generation, Natural Language Interfaces

1 Introduction

SPARQL is the W3C Recommendation for querying and accessing RDF data [7]. While it has found broad acceptance among semantic application developers, its usage among those who possess limited to no expertise in semantic technologies remains understandably limited.

A wide variety of systems propose different means for the users to express what they are looking for: (i) keywords, as supported by systems such as SemSearch [11]; (ii) free-text questions, sometimes using a controlled language, such as ORAKEL [1] and FREyA [3], and (iii) pre-defined forms [15]. Independently of how the input is given, SPARQL queries are generated and evaluated.

The system we introduce in this paper, Spartiqulation, is complementary in functionality and purpose to these approaches. More concretely, we address the question of query verbalization, by which the meaning of a query encoded in SPARQL is conveyed to the user via English text. The verbalization allows for the user to observe a potential discrepancy between an intended question and the system-generated query. In addition, Spartiqulation offers an easy-to-use means to gain better insight into the operation of a search system.

We illustrate these aspects via an example. Assume the information need of the user is *The second highest mountain on Earth*; if the system does not know the meaning of the term *second*, it will very likely simply ignore the qualifier and display the highest mountain. Using Spartiqulation, the meaning of the generated SPARQL query can be communicated to the users in a comprehensible way and eventually inform them that the system understood a different question.

Our main contributions are: 1) We introduce a domain-independent SPARQL query verbalization approach based on domain-independent templates. 2) We define the anatomy of a query verbalization. 3) We verbalize a query in a top-down manner by breaking a query into independently verbalizable parts. This allows for a more concise verbalization compared to a bottom-up manner where smallest part of a query are mapped to their realization and then combined.

All verbalizations presented as captions of query listings are generated using Spartiqulation. The work we present here is an extension of our previous work [6] which described the document structuring task. In this paper we provide details on the remaining four tasks that are necessary for the complete system and provide evaluation results. Additional material is available online.[3].

2 The Spartiqulation approach

2.1 Coverage

SPARQL knows the query forms `SELECT`, `CONSTRUCT`, `ASK`, and `DESCRIBE`. Tools such as SemSearch [11], ORAKEL [1], and FREyA [3] that translate user input into SPARQL queries generate `SELECT` queries (e.g., Listing 1) and `ASK` queries (e.g., Listing 2), which are also the query forms that our approach supports. In the current form our approach verbalizes SPARQL 1.1 `SELECT` and `ASK` queries where the `WHERE` clause is a connected basic graph pattern that may contain filter expressions. The aggregation function `COUNT` and the solution modifiers `DISTINCT`, `HAVING`, `LIMIT`, `OFFSET`, and `ORDER BY` may be used. Currently not supported are unconnected basic graph patterns, variables in predicate positions, negations via the language features `EXISTS` and `MINUS`, subqueries, the language features `BIND` and `VALUES`, the solution modifier `REDUCED`, aggregation functions besides `COUNT`, graph names, and path matching.

2.2 Tasks

Our approach is inspired by the pipeline architecture for NLG systems and the tasks these systems commonly perform, as introduced by Reiter and Dale [20]:

T1. Content determination is about deciding which information will be communicated by the text to be generated – see [6] for details.

[3] `http://km.aifb.kit.edu/projects/spartiqulator/ESWC2014SPARQL`

T2. Document structuring refers to the task of constructing independently verbalizable messages from the input query, and deciding upon the order and structure of these messages.

T3. Lexicalization is the task of deciding which specific words to use for conveying the meaning of the original content in text.

T4. Referring expression generation concerns the task that specifies how to refer to an entity that is already introduced in the text.

T5. Aggregation is the task of deciding how to map structures created during content determination and document structuring onto linguistic structures, such as sentences and paragraphs.

T6. Linguistic realization is the task of converting abstract representations of sentences into actual text.

Our system is template-based. In particular, inspired by [9] and [21], we have manually collected a series of schema-independent templates, which are used in the verbalization process to generate coherent natural-language text from the different parts of a SPARQL query. These templates could be extended to capture user-, domain- or application-specific aspects to further improve the verbalization results. Our implementation supports such extensions, but already delivers meaningful results in its current generic form. The anatomy of a verbalization consists of up to four parts:

Main entity (ME): `SELECT` and `ASK` queries contain a `WHERE` clause containing a basic graph pattern. Verbalization of a graph or graph pattern requires a starting point. We refer to the node that we chose to begin with as the *main entity.* The main entity is rendered as the subject of the verbalization in singular or plural, with a definite or an indefinite article, and may be counted. Examples are: *Persons* for the `SELECT` query in Listing 1 and *a record* for the `ASK` query in Listing 2.

Constraints (CONS): `CONS` covers restrictions regarding the main entity, such as the relations with other resources and literals. Constraints are rendered in singular or plural, depending on the number of the main entity, and may contain information from the `ORDER BY` and `LIMIT` parts of the query, as well as from `FILTER` expressions. Examples are: *that have a birth place*, *that have "Dana" as an English given name*, *that have an alias that matches "Dana"*.

Requests (REQ): `REQ`, which is only created for `SELECT` queries, includes the projection variables (those that appear in the `SELECT` clause) besides the main entity of a query. Examples of requests are *these birth places* and *if available, their English labels* in Listing 1. Renderings of requests may include information from `FILTER`s and from domain/range information of properties and whether the variable is optional.

Modifiers (MOD): `MOD` is represented using the features `LIMIT`, `OFFSET`, and `ORDER BY` and apply to `SELECT` queries. `MOD` can be partially included within other parts of the verbalization, for example in the ME part (*10 persons*), within constraints (*that has the highest number of languages*), or as an own sentence (*Omit the first 10 results; show not more than 10 results.*).

```
SELECT DISTINCT ?uri ?string ?p WHERE {
  ?uri :birthPlace ?p . ?uri :surname 'Elcar' .
  ?uri rdf:type foaf:Person.
  { ?uri foaf:givenName 'Dana'@en. } UNION {
    ?uri prop:alias ?alias . FILTER regex(?alias,'Dana') .
  }
  OPTIONAL {
    ?uri rdfs:label ?string . FILTER (lang(?string) = 'en')
  }
} LIMIT 10
```

Listing 1. *Persons that have a birth place and that have the surname "Elcar" and that have "Dana" as an English given name or that have an alias that matches "Dana". Show also these birth places and, if available, their English labels.*

```
ASK WHERE {
  ?album rdf:type mo:Record.
  ?album mo:release_status mo:bootleg.
  ?album foaf:maker ?artist.
  ?artist foaf:name 'Pink Floyd'.
}
```

Listing 2. *Is it true that there is a record that has a maker that has the name "Pink Floyd" and that has the release status bootleg?*

```
SELECT ?uri ?string WHERE {
  ?uri rdf:type onto:Country .
  ?uri onto:language ?language.
  OPTIONAL {
    ?uri rdfs:label ?string.
    FILTER (lang(?string) = 'en')
  }
} ORDER BY DESC(COUNT(?language))
LIMIT 1
```

Listing 3. *The country that has the highest number of languages. Show also, if available, its English labels.*

2.3 Document structuring

The system constructs independently verbalizable messages from the input query and determines an appropriate ordering and structure. We first identify the subject of the verbalization and then create and classify the messages.

Main entity selection `SELECT` queries are a means to retrieve bindings for variables. We assume that a user is more interested in variable values than in entities used in the query. *Projection variables* are those variables that appear in the `SELECT` clause and in the `WHERE` clause, such as the variable `?title` in Listing 5. Only bindings retrieved for projection variables are returned as result of a query execution. Therefore, only projection variables are candidates for the main entity selection. Furthermore, only non-optional projection variables come into consideration. A variable is a non-optional variable if within the `WHERE` clause it does not only appear within `OPTIONAL` blocks. For example, in Listing 3 the variable `?uri` is non-optional and the variable `?string` is optional. Due to their nature of being optional and thus less relevant than non-optional variables, only non-optional projection variables come into consideration. The same holds for variables for which a `NOTBOUND` filter (e.g. `! bound(?var)`) is specified.

If a `SELECT` query has multiple non-optional projection variable, a choice is made among these candidates (see [6] for details). For example, given the query shown in Listing 5, selecting `?track` would result in *Things that have a creator that has the title "Petula Clark"*; selecting `?title` would result in *Distinct things that are titles of tracks that have a creator that have the title "Petula Clark"*.

```
SELECT DISTINCT ?uri ?book WHERE {
  ?book rdf:type onto:Book .
  ?book onto:author ?uri .
  ?book rdfs:label 'The Pillars of
    the Earth'@en .
}
```

Listing 4. *Books that have the label "The Pillars of the Earth" in English and that have an author. Show also these books' authors.*

```
SELECT ?track ?title
WHERE {
  ?track rdf:type mm:Track.
  ?track dc:title ?title.
  ?track dc:creator ?artist.
  ?artist dc:title 'Petula Clark'.
}
```

Listing 5. A SELECT query with two non-optional projection variables.

`ASK` queries are a means to assess whether a query pattern has a solution. If `ASK` queries contain variables then the selection procedure is the same as for projection variables in `SELECT` queries. Otherwise, we select the triple's subject.

Message creation and classification After the main entity identification the graph is transformed so that the main entity is the root and all edges point away from the root. For details we refer to [6]. During this transformation edges may become reversed, which means that subject and object of the underlying triple are exchanged and the property is marked with a prepended `-`.

The query graph is split into independently verbalizable messages which represent paths that start at the main entity and consist of sets of triple patterns. For the query in Listing 4, where the variable `?uri` is selected as the main entity, two path messages are created. The first message represents the path `(?uri -onto:author ?book. ?book rdf:type onto:Book.)`.[4] The second message represents the path `(?uri -onto:author ?book. ?book rdfs: label 'The Pillars of the Earth'@en.)`. Further messages are created for variables and contain information about filters, as well as information related to the SPARQL features `HAVING`, `LIMIT`, `OFFSET`, `OPTIONAL`, `ORDER BY`, and `UNION`.

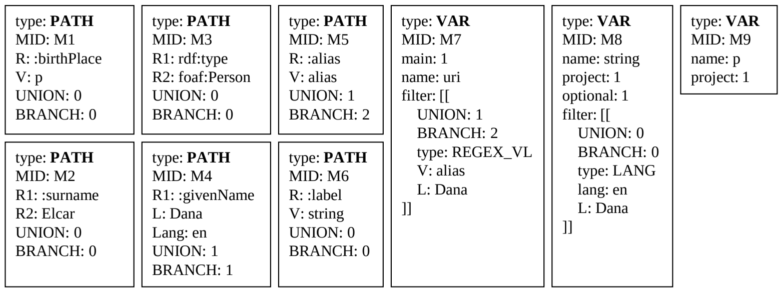
The messages that represent the `SELECT` query from Listing 1 are depicted in Fig. 1. The query is represented using 6 path-representing messages and 3 variable-representing messages. The main entity is represented with message id (MID) M6. The query contains one `UNION` with 2 branches. Note that the `REGEX_VL` filter related to the main entity is specific to branch 2 in `UNION` 1.

Messages that represent a path are classified as `CONS`. In case of a `SELECT` query if the path contains a projection variable besides the main entity, then the path-representing message is also classified as `REQ`. In Fig. 1, the path-representing messages M1 and M6 are classified as `REQ` messages.

2.4 Lexicalization

During lexicalization the system determines the actual wording to denominate an entity; in our case, such entities are RDF resources represented by URIs or variables. For each entity, the URI that represents the entity is dereferenced to retrieve a label from the resulting RDF data using one of the 36 labeling properties defined in [5]. Should no label be available for a given entity, the system

[4] Note that the minus symbol in `-onto:author` indicates that the property is reversed.



```
type: PATH        type: PATH        type: PATH      type: VAR          type: VAR         type: VAR
MID: M1           MID: M3           MID: M5         MID: M7            MID: M8           MID: M9
R: :birthPlace    R1: rdf:type      R: :alias       main: 1            name: string      name: p
V: p              R2: foaf:Person   V: alias        name: uri          project: 1        project: 1
UNION: 0          UNION: 0          UNION: 1        filter: [[         optional: 1
BRANCH: 0         BRANCH: 0         BRANCH: 2         UNION: 1         filter: [[
                                                      BRANCH: 2          UNION: 0
type: PATH        type: PATH        type: PATH        type: REGEX_VL     BRANCH: 0
MID: M2           MID: M4           MID: M6           V: alias           type: LANG
R1: :surname      R1: :givenName    R: :label         L: Dana            lang: en
R2: Elcar         L: Dana           V: string       ]]                   L: Dana
UNION: 0          Lang: en          UNION: 0                           ]]
BRANCH: 0         UNION: 1          BRANCH: 0
                  BRANCH: 1
```

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

derives one from the local name of the URI string. For instance, the local name of the URI `http://purl.org/ontology/mo/MusicGroup` is `MusicGroup`, which can be de-camelcased and lowercased to *music group*. Variables are expressed in natural language either using their type constraints or, if absent, using the term *thing*. A variable can be explicitly constrained by its type such as the variable `?v` in `?v rdf:type ex:Actor`. In this case, we lexicalize the variable using the label of the typing class, that is *Actor*. Such typing information can also be determined taking into account the domain and range of a property. For example, in the triple pattern `?var1 dbo:capital ?var2` with domain of `dbo:capital` defined as *PopulatedPlace*, we can derive `?var1` as *populated place*.

We lexicalize properties as shown in Table 1. We use the Stanford parser [10] to identify the part of speech of a property's label or local name to chose the most appropriate lexicalization. These property patterns are based on the work by Hewlett et al. [9].

2.5 Referring expression generation

Referring expression generation is the task of deciding how to refer to a previously introduced entity. For example, the query in Listing 4 asks for books that have a certain label and known authors. The variable `?uri` is introduced and rendered as *authors* within the `CONS` part *have authors*. Since the `SELECT` clause contains a second variable `?uri` besides the main entity `?book`, the verbalization needs to communicate that entities that can be bound to `?uri` need to be part of the query result. This is achieved by adding the phrase *Show also these books' authors* to the verbalization, where *these books' authors* is the referring expression corresponding to the second variable `?uri`.

On the one hand, a referring expressions needs to unambiguously refer to an entity. On the other hand, a referring expression should be concise to aid the understandability of a verbalization. For example, for the `SELECT` query in Listing 1, the projection variable can be verbalized as *these birth places* or as *these persons' birth places*. Since within the query *birth place* does not occur multiple times, it is sufficient to generate *these birth places*, which means that from the `REQ` message only the projection variable itself needs to be verbalized. If this leads to an ambiguity, which means that two referring expressions are

$A \hat{=}$ The DISTINCT modifier is used
$B \hat{=}$ Main entity is counted as in `SELECT(COUNT ?main)`
$C \hat{=}$ Result set is limited with LIMIT as in `LIMIT 10`
$D \hat{=}$ ME needs to be verbalized in singular (LIMIT = 1)
$E \hat{=}$ Main entity is ordered as in `ORDER BY DESC(?main)`
$F \hat{=}$ Main entity has multiple types as in
`?main rdf:type ex:A. ?main rdf:type ex:B.`

Fig. 2. The set of facts that control within the verbalization template how the main entity is verbalized as shown in Fig. 3.

identical or substring of each other, then the part of the `REQ` message that is actually verbalized needs to be extended step by step until the full path between that variable and the main entity is verbalized. We perform this extension until the set of all referring expressions is free of ambiguities or if the full paths have already been verbalized which means that in this case we cannot generate nonambiguous referring expressions.

2.6 Linguistic realization

Abstract representations of sentences are translated into actual text. Verbalization of the `MOD` part of a query is straightforward; the verbalization of `REQ` messages is similar to the verbalization of `CONS` messages. Therefore, the remainder of this section will focus on the realization of the messages corresponding to the main entity and the `CONS` part.

A set of 6 boolean variables, shown in Fig. 2, is necessary to fully capture the necessary variations for the main entity template. Fig. 3 shows an excerpt of this template.[5] Besides these boolean variables the template is provided with a hash map `$D` that contains strings that are either literals appearing in the query or labels of resources. For example, `$D{TPL}` is the variable's type in plural,[6] `$D{TSI}` is the variable's type in singular, and `$D{L}` is a limit value as specified using the SPARQL `LIMIT` feature. The strings, such as `Abcdef`, indicate which of the boolean variables is true. Capital characters indicate a true value, lowercase characters indicate a false value.

`CONS` messages are verbalized in smaller parts, each with the help of a template, which are then joined together. Consider for example a `CONS` message that represents a path consisting of two triples `?main :prop1 ?v1. ?v1 rdfs:label "x"`. This path is split into the parts (i) `:prop1 ?v1`, and (ii) `rdfs:label "x"`. For the first part, that consists of a resource (R) and a variable (V), a specific `RV` template is selected based on linguistic properties of that resource. For example, if the property label is a noun, then the template `CONS-RV-C1` is selected. Similar to the main entity verbalization, the specific verbalization procedure is

[5] Given the 6 boolean variables, the template could define up to 64 different verbalizations. However, the variables are not independent. Therefore, the number of different slots that need to be filled in this template is reduced from 64 to 17.
[6] We use the Perl module Lingua::EN::Inflect in order to convert a word to plural.

`Abcdef`	'Distinct ' . \$D{TPL} → Distinct scientists
`aBcdef`	'Number of ' . \$D{TPL} → Number of scientists
`AbcdEf`	'The distinct ' . \$D{TPL} → The distinct scientists

`abCdef`	'Not more than ' . \$D{L}
	.' '.\$D{TPL}
	→ Not more than 10 scientists

`abcDEf`	'The ' . \$D{TSI} → The scientist

Fig. 3. Excerpt of the ME template.

`abcdefghIJ`	'that has no ' . \$D{PSI} → that has no email
`abcdEFGHij`	'that have the highest number of '.\$D{PPL} → that have the highest number of languages
`abcDefghIj`	'that is not '.\$D{A} .' '.\$D{PSI}.' of a thing' → that is not a holder of a thing

Fig. 4. Excerpt of the CONS-RV template for properties of class 1.

controlled by a set of facts as shown in Fig. 5. Given these facts the template produces verbalizations as shown in Fig. 4.

№	Examples	Expansions	№	Examples	Expansions
1	email, hasColor	X has an email Y X has a color Y R: Y is an email of X R: Y is a color of X	5	collaboratesWith	X collaborates with Y R: Y collaborates with X
2	knows	X knows Y R: Y is known by X	6	visiting R: Y is visited by X	X is visiting Y
3	brotherOf, isBrotherOf	X is brother of Y R: Y has a brother X	7	locatedInArea	X is located in area Y in which X is located
4	producedBy, isMadeFrom	X is produced by Y X is made from Y R: Y produces X R: Y is used to make X	8	marriedTo	X is married to Y R: Y is married to X

Table 1. Classes of properties, expansions, and expansions of the reverted properties.

A ≙	the variable is the first optional variable and this variable is not NOTBOUND	F ≙	plural form is required
B ≙	the variable has exactly one type	G ≙	ordered by variable
C ≙	the variable has more than one type	H ≙	descending order
D ≙	the property is reversed	I ≙	the variable is OPTIONAL and NOTBOUND
E ≙	the variable is counted	J ≙	the property is numeric

Fig. 5. The set of facts that control how a CONS part is verbalized as shown in Fig. 4.

2.7 Aggregation

Aggregation maps the structures created so far onto linguistic structures such as sentences and paragraphs. Verbalization consists of at least one and no more than three sentences in case of `SELECT` queries and of one sentence in case of `ASK` queries. The first sentence begins with the main entity (ME) followed by *that is* or *that are* followed by the verbalized and joined `CONS` messages. The second sentence begins with *Show also* followed by the verbalized and joined `REQ` messages. If there are no `REQ` messages then we omit the sentence. The third sentence verbalizes all `MOD` messages if they have not yet been verbalized as part of the main entity verbalization. Verbalizations of `ASK` queries consist of exactly one sentence which verbalizes the main entity and the `CONS` messages. They begin with *Is it true that* followed by *there is* or *there are* followed by the verbalized and joined `CONS` messages.

`UNION`s are dealt with in the aggregation step as follows. For the set of `CONS` messages that do not belong to any `UNION`, their verbalizations are joined with `and`. For the query in Listing 1, verbalization of the `CONS` messages M1 and M2 results in the string *that have a birth place and that have the surname "Elcar"*. For a set of `CONS` messages that belong to the same branch of a `UNION`, the `CONS` verbalizations are joined by *and* to create the branch verbalization. Each `UNION` is verbalized by joining the branch verbalizations by *or* to create the `UNION` verbalization, for example resulting in *that have "Dana" as an English given name or that have an alias that matches "Dana"*.

Another aspect of the aggregation is the inclusion of the `LIMIT`, `OFFSET` and `ORDER BY` modifiers into the main entity verbalization, as shown in Section 2.4. This form of aggregation allows for more concise verbalizations compared to the more wordy alternative where a dedicated sentence is created.

3 Evaluation

3.1 Overview

According to Mellish and Dale [14], evaluation in the context of an NLG system can be carried out for the purpose of assessing (i) the properties of a theory; (ii) the properties of a system; and (iii) the application potential. We focused on the second aspect: on the quality of our system in terms of a set of specific dimensions. We performed (i) a comparative evaluation where we compared Spartiqulation against the (to the best of our knowledge) only other available alternative, SPARQL2NL [18]; and (ii) assessed the performance of our system according to these dimensions.

3.2 Dimensions

The evaluation dimensions, inspired by [12, 14, 19], are defined as follows:

Coverage: the ratio of SPARQL `SELECT` queries the system accepts. This dimension can be measured automatically.

Accuracy: the degree to which the verbalization conveys the meaning of the SPARQL query. This quality is measured through human judgement using a 4-point scale adapted from [17]: (1) The meaning of the verbalization does neither leave out any aspect of the query, nor does it add something. (2) The meaning of the query is not adequately conveyed to the verbalization. Some aspects are missing. (3) The meaning of the query is not adequately conveyed to the verbalization. Most aspects are missing. (4) The meaning of the query is not conveyed to the verbalization.

Syntactic correctness: the degree to which the verbalization is syntactically correct, in particular whether it adheres to English grammar: (1) The verbalization is completely syntactically correct. (2) The verbalization is almost syntactically correct. (3) The verbalization presents some syntactical errors. (4) The verbalization is strongly syntactically incorrect.

Understandability: The level of understandability of the verbalization, adapted from [17]: (1) The meaning of the verbalization is clear. (2) The meaning of the verbalization is clear, but there are some problems in word usage, and/or style. (3) The basic thrust of the verbalization is clear, but the evaluator is not sure of some detailed parts because of word usage problems. (4) The verbalization contains many word usage problems, and the evaluator can only guess at the meaning. (5) The verbalization cannot be understood at all.

Adequacy and Efficiency According to Dale [2], criteria relevant for referring expressions are *adequacy* and *efficiency*. A referring expression is adequate if it allows to unambiguously identify the referent. It can be measured as the ratio of expressions for variables within the REQ part that unambiguously identify a variable. In addition, a referring expression is said to be efficient if it is perceived to not contain more information than necessary. Since these criteria are clearly defined, they are manually evaluated by the authors.

3.3 Data set

We created a corpus of SPARQL queries using data from the QALD-1[7] and the ILD2012 challenges.[8] The aim of these challenges was to answer natural language questions against data from DBpedia and MusicBrainz. The organizers provide a training set encompassing questions and the corresponding SPARQL queries. Our corpus refers to 291 of a total of 300 queries, more concretely `SELECT` and `ASK` queries. Nine of the questions in the original data set were eliminated since no SPARQL equivalent was specified. We randomly split the data into a training set (251 queries) and an evaluation set (40 queries) as follows:

Training data set: 44 queries from *2011-dbpedia-train*, 44 queries from *2011-musicbrainz-train*, 79 queries from *2012-dbpedia-train*, and 84 queries from *2012-musicbrainz-train.* The set contained 251 queries (228 SELECT queries, 23 ASK queries) which is about 86% of the full corpus.

Evaluation data set: 6 queries from *2011-dbpedia-train*, 6 queries from *2011-musicbrainz-train*, 14 queries from *2012-dbpedia-train*, and 14 queries from *2012-musicbrainz-train.* The set contained 40 queries (35 SELECT queries and 5 ASK queries) which is about 14% of the full corpus.

3.4 Comparative evaluation

We compared our work with SPARQL2NL in a blind setting with the help of six evaluators who had experiences with writing SPARQL queries. The evaluators were not aware of the fact that the verbalizations were generated by different systems. In cases where both systems successfully verbalized a query (38/40 queries), their task was, given a query and two verbalizations, to compare the

[7] `http://www.sc.cit-ec.uni-bielefeld.de/qald-1`
[8] `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/`

first verbalization with the second regarding accuracy, syntactic correctness, and understandability. For example, we asked *Compare the two verbalizations regarding accuracy* where we provided the options (i) *The first one is more accurate*, (ii) *They are equally accurate*, (iii) *The first one is less accurate*, and (iv) *Not applicable (Please explain)*.

3.5 Non-comparative evaluation

After the comparative evaluation we asked the same group to evaluate the *accuracy*, *syntactical correctness*, and *intelligibility* of Spartiqulation. Given the set of 40 successfully verbalized queries, we asked the evaluators to assess the verbalizations along these three criteria.

Coverage was measured as – per definition – the ratio of queries accepted by our system both in the training and the evaluation data set. Adequacy and efficiency of referring expressions were evaluated manually by the authors. Since these criteria, unlike accuracy, syntactic correctness, and understandability, are unambiguously defined in the literature, we evaluated them ourselves without diminishing the objectivity of the remaining findings.

3.6 Results

Comparative evaluation. Fig. 6 shows the results of the comparative evaluation. 38 verbalizations were assessed by the 6 evaluators, leading to a total number of 114 assessments. Higher accuracy was reported in 43 cases (37.72%), equal accuracy was reported in 66 cases (57.89%), higher syntactical correctness was reported in 52 cases (45.61%), equal syntactical correctness was reported in 45 cases (39.47%), higher understandability was reported in 74 cases (64.91%), and equal understandability was reported in 16 cases (14.04%). We used Krippendorff's alpha [8] to measure inter-rater agreement regarding whether our results are comparable or. The results are $\alpha = 0.56$ for accuracy, $\alpha = 0.726$ for syntactical correctness and $\alpha = 0.676$ for understandability.

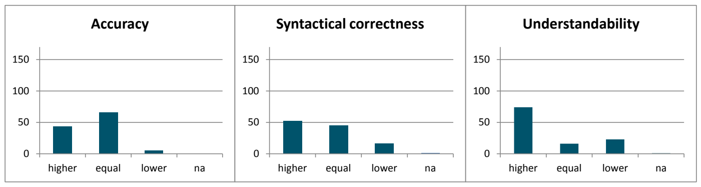


Fig. 6. Results regarding accuracy, syntactical correctness, and understandability from the comparative evaluation

Non-comparative evaluation. Coverage: We counted how many queries contain features that our system does not support using the data set described in

Section 3.3. Within the evaluation set of 40 queries, every query was verbalizable. Within the training set of 251 queries, four queries did not meet the limitations described in Section 2.1. Each of these queries contained two disconnected basic graph patterns that were only connected via FILTER expressions, such as `FILTER (?large = ?capital)`. This means a total coverage of 287/291 (98.6%).

Accuracy, syntactical correctness, and understandability: Fig. 7 shows the results regarding the 40 evaluation queries that were were assessed by the 6 evaluators, leading to a total number of 120 assessments. The numbers on the x-axis correlate with the scales introduced in Section 3.2. Verbalizations show a high accuracy, high syntactical correctness, and good understandability. 106 out of 120 times the evaluators attested the best accuracy score (88.33%). Regarding syntactical correctness, and understandability there is room for improvement. 70 out of 120 times the evaluators attested the best accuracy score (58.33%), 47 out of 120 times the evaluators attested the best understandability score (39.16%).

Adequacy and minimality of referring expressions: Among the 40 queries in the evaluation set, for 25 queries referring expressions (RE) had to be generated. In 24 cases the RE were nonambiguous which means an adequacy of 96%. In 3 out of 25 cases the RE was not efficient. For example, a query was verbalized as *Distinct things that are presidents of the united states or that are presidents that have President of the United States as a title. Show also, if available, these things' (that are presidents of the united states or presidents) English labels.* Here, the RE *these things' (that are presidents of the united states or presidents) English labels* could by reduced to *these things' English labels.*

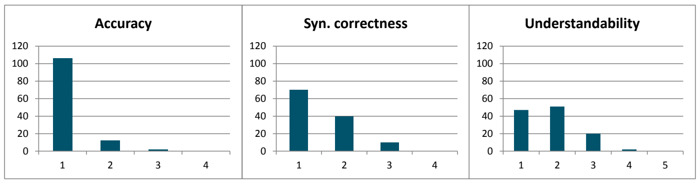


Fig. 7. Results regarding accuracy, syntactical correctness, and understandability from the non-comparative evaluation

3.7 Discussion

What makes the verbalization generated by the SPARQL2NL system less understandable for complex queries is the fact that variable names are included in the verbalization such as in *"This query retrieves distinct countries ?uri and distinct values ?string such that ?string is in English. Moreover, it returns exclusively results that the number of ?uri's official languages is greater than 2."* In comparison, our system verbalizes the same query as *"Distinct countries that have more than 2 official languages. Show also, if available, these countries' English labels."* In some cases experts tended to prefer the variant with variable names since this seems to be more natural to them. However, the experts pointed out their belief that verbalizations containing variables will be hard to understand by non-experts. We claim that as long as experts are not the only intended users of semantic search engines this verbalization style is inappropriate. Especially, variable names in automatically generated queries may carry little information. An interesting feature is that datatypes are utilized as in the verbalization of the literal `?date <= '2010-12-31'^^xsd:date` to *?date is greater than or equal to January 31, 2010*. SPARQL2NL was capable of verbalizing the 4 queries that were rejected by our system.

Problems both systems were facing arise from four areas: 1) Missing labels for entities. Relying on local names leads to results such as *that have the release type type soundtrack* for the triple pattern `?album mm:releaseType mm:TypeSoundtrack`. 2) The way the data is modeled complicates the verbalization as in *Things that are begin dates of things that have to artists that have the title "Slayer"*. Here, the verbalization of the property in `?bandinstance ar:toArtist ?band` as *to artist* is currently the best guess the system can make. 3) Missing linguistic information about properties. The property `dbo:crosses` in the triple pattern `res:Brooklyn_Bridge dbo:crosses ?uri` can be interpreted as the plural of the noun cross or an inflected form of the verb to cross. Here, verbalization could be improved if linguistic information was attached to the vocabulary, for example using the *lemon model* [13]. 4) Problems with pluralization and capitalization exist and could also be fixed given a lexicon.

4 Related work

To the best of our knowledge SPARQL2NL [18] is the only approach which is similar to ours in scope and functionality. One part of our evaluation was dedicated to the comparison along a number of quality dimensions established in the field of Natural Language Generation. The evaluation revealed that both approaches have advantages and disadvantages, whereas Spartiqulation seems to perform better for cases dealing with complex queries with multiple variables.

Both approaches are geared towards a similar subset of SPARQL. How a triple is verbalized depends on linguistic cues found in property labels and type information extracted from the queries. The main differences are two-fold: 1) SPARQL2NL maps each atom (variable, property, resource, literal) to their realization, stores these realizations as dependency trees and aggregate these trees in a later step. In our approach, we fragment the query graph into independently verbalizable parts (messages). This higher level of granularity has positive influences on conciseness of the resulting verbalization. 2) SPARQL2NL does not map variables to realizations, which means that names of variables appear in verbalizations as they are in the query. Even in a scenario where variable names are meaningful this is expected to have a negative impact on the understandability of the verbalization for non-experts. Moreover, meaningful variable names cannot be guaranteed if queries are generated by a natural language interface. In cases where variables have been introduced due to filters, SPARQL2NL is capable of removing those variable names from the verbalizations using aggregation.

Further related work comes from three areas: verbalization of RDF data [21], verbalization of OWL ontologies [22], and verbalization of SQL queries. 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 [4]. The main difference between Spartiqulation and the work by Minock [16], which focuses on relational queries over tuples, is that our approach is schema-agnostic. Besides some dependencies to RDF(S) and OWL, our system does not require any information about the domain of the application scenario, and about the vocabularies used to encode knowledge about this domain. By contrast, the verbalizer by Minock foresees manually created patterns covering all possible combinations of relations in the schema.

5 Conclusions

In this paper we presented our approach to SPARQL query verbalization. Our aim is to create natural-language descriptions of queries to communicate the meaning of these queries to users who are not familiar with the intricacies of the query language. To do so, we realized a system which implements an established natural language generation pipeline complemented by templates manually derived through the analysis of a representative set of SPARQL queries used by the community in related research challenges.

The results of our work could be beneficial for a variety of Linked-Data-related scenarios in which SPARQL cannot be assumed as the most appropriate form of interaction between an application and its users. In particular, verbalization could complement the functionality of information retrieval systems, which transform unstructured or semi-structured user queries into SPARQL, as the availability of human-readable renderings of SPARQL queries allows users to gain a better understanding of the way the information retrieval system matches their information needs to sources, and of the reasons why certain sources are included in the result set.

Acknowledgements

The authors acknowledge the support of the European Commission's Seventh Framework Programme FP7/2007-2013 (PlanetData, Grant 257641) and FP7-ICT-2011-7 (XLike, Grant 288342).

References

1. P. Cimiano, P. Haase, J. Heizmann, M. Mantel, and R. Studer. Towards portable natural language interfaces to knowledge bases–The case of the ORAKEL system. *Data & Knowledge Engineering*, 65(2):325–354, 2008.
2. R. Dale. Cooking up referring expressions. In *Proceedings of the 27th annual meeting on Association for Computational Linguistics*, ACL '89, pages 68–75, Stroudsburg, PA, USA, 1989. Association for Computational Linguistics.

3. D. Damljanovic, M. Agatonovic, and H. Cunningham. FREyA: An interactive way of querying Linked Data using natural language. In *The Semantic Web: ESWC 2011 Workshops*, pages 125–138. Springer, 2012.
4. B. Davis, A. Iqbal, A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, and S. Handschuh. RoundTrip Ontology Authoring. In *ISWC 2008*, pages 50–65. 2008.
5. B. Ell, D. Vrandečić, and E. Simperl. Labels in the Web of Data. In N. et al., editor, *ISWC 2011*. Springer, October 2011.
6. B. Ell, D. Vrandečić, and E. Simperl. SPARTIQULATION: Verbalizing SPARQL queries. In *Proceedings of the Interacting with Linked Data (ILD) Workshop at ESWC 2012*, Heraklion, Greece, 5 2012.
7. S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, `http://www.w3.org/TR/sparql11-query/` (URL last accessed 2013-10-19), October 2010.
8. A. F. Hayes and K. Krippendorff. Answering the call for a standard reliability measure for coding data. *Communication methods and measures*, 1(1):77–89, 2007.
9. 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 ESWC'05*, 2005.
10. D. Klein and C. D. Manning. Accurate Unlexicalized Parsing. In E. Hinrichs and D. Roth, editors, *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 2003.
11. Y. Lei, V. Uren, and E. Motta. SemSearch: A search engine for the semantic web. *EWAW 2006*, pages 238–245, 2006.
12. J. Lester and B. Porter. Developing and empirically evaluating robust explanation generators: The KNIGHT experiments. *Comp. Linguistics*, 23(1):65–101, 1997.
13. J. McCrae, D. Spohr, and P. Cimiano. Linking Lexical Resources and Ontologies on the Semantic Web with Lemon. In *ESWC 2011*, volume 6643 of *LNCS*, pages 245–259. Springer, 2011.
14. C. Mellish and R. Dale. Evaluation in the conext of natural language generation. *Computer Speech and language*, 12(4):349–374, 1998.
15. P. Mendes, B. McKnight, A. Sheth, and J. Kissinger. TcruziKB: Enabling Complex Queries for Genomic Data Exploration. In *Semantic Computing, 2008*, pages 432 –439, aug. 2008.
16. M. Minock. C-Phrase: A system for building robust natural language interfaces to databases. *Data Knowl. Eng.*, 69(3):290–302, Mar. 2010.
17. M. Nagao, J.-i. Tsujii, and J.-i. Nakamura. The Japanese government project for machine translation. *Comput. Linguist.*, 11(2-3):91–110, Apr. 1985.
18. A.-C. Ngonga Ngomo, L. Bühmann, C. Unger, J. Lehmann, and D. Gerber. SPARQL2NL: Verbalizing Sparql Queries. pages 329–332. International World Wide Web Conferences Steering Committee, 2013.
19. E. Reiter and A. Belz. An investigation into the validity of some metrics for automatically evaluating natural language generation systems. *Computational Linguistics*, 35(4):529–558, 2009.
20. E. Reiter and R. Dale. *Building Natural Language Generation Systems*. Natural Language Processing. Cambridge University Press, 2000.
21. X. Sun and C. Mellish. An experiment on "free generation" from single RDF triples. ENLG '07, pages 105–108, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
22. A. Third, S. Williams, and R. Power. OWL to English: a tool for generating organised easily-navigated hypertexts from ontologies. In *ISWC 2011*, 2011.