Sorry, I don't speak SPARQL – Translating SPARQL Queries into Natural Language

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

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

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

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

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

ABSTRACT

Over the past years, Semantic Web and Linked Data technologies have reached the backend of a considerable number of applications. Consequently, large amounts of RDF data are constantly being made available across the planet. While experts can easily gather information from this wealth of data by using the W3C standard query language SPARQL, most lay users lack the expertise necessary to proficiently interact with these applications. Consequently, non-expert users usually have to rely on forms, query builders, question answering or keyword search tools to access RDF data. However, these tools have so far been unable to explicate the queries they generate to lay users, making it difficult for these users to i) assess the correctness of the query generated out of their input, and ii) to adapt their queries or iii) to choose in an informed manner between possible interpretations of their input. This paper addresses this drawback by presenting SPARQL2NL, a generic approach that allows verbalizing SPARQL queries, i.e., converting them into natural language. Our framework can be integrated into applications where lay users are required to understand SPARQL or to generate SPARQL queries in a direct (forms, query builders) or an indirect (keyword search, question answering) manner. We evaluate our approach on the DBpedia question set provided by QALD-2 within a survey setting with both SPARQL experts and lay users. The results of the 115 filled surveys show that SPARQL2NL can generate complete and easily understandable natural language descriptions. In addition, our results suggest that even SPARQL experts can process the natural language representation of SPARQL queries computed by our approach more efficiently than the corresponding SPARQL queries. Moreover, non-experts are enabled to reliably understand the content of SPARQL queries.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Theory

Keywords

Natural language generation, query verbalization, SPARQL

1. INTRODUCTION

An ever-growing number of applications rely on RDF data as well as on the W3C standard SPARQL for querying this data. While SPARQL has proven to be a powerful tool in the hands of experienced users, it remains difficult to fathom for lay users. To address this drawback, approaches such as question answering [28], keyword search [25] and search by example [18] have been developed with the aim of hiding SPARQL and RDF from the user. Still, these approaches internally construct SPARQL queries to address their data backend, without providing lay users with a possibility to check whether the retrieved answers indeed correspond to the intended information need. Consider for example the natural language question What is the birth date of Li Ling?, for which TBSL [28] returns more than 50 possible interpretations, including the birth date of the pole vaulter Li Ling and the age of the sinologist Li Ling. Since each of the interpretations is realized as a SPARQL query, a lay user cannot pinpoint the set of results that correspond to the person he is actually interested in, nor can he easily detect the source of possible errors. Similar problems occur in keyword-based systems. For example, the keywords Jenny Runacre husbands leads to SINA [25] generating queries for the husbands of Jenny Runacre as well as for the role of Jenny Runacre in the movie "The Husbands". We address this drawback by presenting SPARQL2NL[1], a novel approach that can verbalize SPARQL queries and therewith bridge the gap between the query language understood by semantic data backends, i.e. SPARQL, and that of the end users, i.e. natural language. Our approach is tailored towards SPARQL constructs typically used in keyword search and question answering, and it consists of four main steps: a *preprocessing* step which normalizes the query and extracts type information for the occurring variables, a *processing*

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

step during which a generic representation of the query is constructed, a *postprocessing* step which applies reduction and replacement rules in order to improve the legibility of the verbalization, and a *realization* step which generates the final natural language representation of the query. As exemplary use cases, we integrated SPARQL2NL into the user interface for the question answering system TBSL[2] as well as into the BioASQ annotation tool[3]. A demo of SPARQL2NL is available at `http://sparql2nl.aksw.org/demo`.

The rest of this paper is structured as follows: After introducing the notation we employ in this paper, we give an overview of each of the four steps underlying SPARQL2NL. We then evaluate our approach with respect to the *adequacy* and *fluency* [5] of the natural language representations it generates. After a brief review of related work, we conclude with some final remarks. Throughout the rest of the paper, we use the following query shown in Listing 1 as main example[4]. This query retrieves persons that are writers or surfers and were born later than 1950.

```
SELECT DISTINCT ?person ?label
WHERE { ?person rdf:type dbo:Person.
        { ?person dbo:occupation res:Writer. }
        UNION
        { ?person dbo:occupation res:Surfing. }
        ?person dbo:birthDate ?date.
        FILTER(?date > "1950"^^xsd:date) .
        OPTIONAL {?person rdfs:label ?label
        FILTER ( lang(?label) = "en" ) } }
```

Listing 1: Running example.

2. PRELIMINARIES AND NOTATION

The goal of our approach is to generate a complete and correct natural language representation of an arbitrary SPARQL query, where completeness means in this context that we aim to represent all information that is necessary for the user to understand the content of the query. In terms of the standard model of natural language generation proposed by Reiter & Dale [24], our preprocessor and processor steps mainly play the role of the document planner, in particular carrying out the task of document structuring, while the postprocessor step corresponds to the micro-planner, with focus on aggregation operations and the lexicalization of referring expressions. In the following, we give a short overview of the notation used throughout this paper to describe our approach. We begin by giving a brief overview of the most important concepts underlying SPARQL queries. Thereafter, we present our approach to formalizing natural language sentences.

2.1 SPARQL queries and their realization

According to the SPARQL grammar,[5] a SPARQL `SELECT` query can be regarded as consisting of three parts:

1. a *body section* `B`, which describes all data that has to be retrieved,
2. an *optional section* `O`, which describes the data items that can be retrieved by the query if they exist, and
3. a *modifier section* `M`, which describes all solution sequences, modifiers and aggregates that are to be applied to the result of the previous two sections of the query.

Let *Var* be the set of all variables that can be used in a SPARQL query. In addition, let R be the set of all resources, P the set of all properties and L the set of all literals contained in the target knowledge base of the SPARQL queries at hand. We call $x \in \mathit{Var} \cup R \cup P \cup L$ an *atom*. The basic components of the body of a SPARQL query are triple patterns $(\mathtt{s}, \mathtt{p}, \mathtt{o}) \in (\mathit{Var} \cup R) \times (\mathit{Var} \cup P) \times (\mathit{Var} \cup R \cup L)$. Let W be the set of all words in the dictionary of our target language. We define the realization function $\rho : \mathit{Var} \cup R \cup P \cup L \to W^*$ as the function which maps each atom to a word or sequence of words from the dictionary. Formally, the goal of this paper is to devise the extension of ρ to all SPARQL constructs. This extension maps all atoms x to their realization $\rho(x)$ and defines how these atomic realizations are to be combined. We denote the extension of ρ by the same label ρ for the sake of simplicity. We adopt a rule-based approach to achieve this goal, where the rules extending ρ to all valid SPARQL constructs are expressed in a conjunctive manner. This means that for premises $P_1, \ldots, P_n$ and consequences $K_1, \ldots, K_m$ we write $P_1 \wedge \ldots \wedge P_n \Rightarrow K_1 \wedge \ldots \wedge K_m$. The premises and consequences are explicated by using an extension of the Stanford dependencies[6]. We rely especially on the constructs explained in Table 1. For example, a possessive dependency between two phrase elements e_1 and e_2 is represented as $\mathtt{poss}(e_1, e_2)$. For the sake of simplicity, we slightly deviate from the Stanford vocabulary by not treating the copula `to be` as an auxiliary, but denoting it as `BE`. Moreover, we extend the vocabulary by the constructs `conj` and `disj` which denote the conjunction resp. disjunction of two phrase elements. In addition, we sometimes reduce the construct $\mathtt{subj(y,x)} \wedge \mathtt{dobj(y,z)}$ to the triple $\mathtt{(x,y,z)} \in W^3$.

3. PREPROCESSING

The goal of the preprocessing step is to normalize the query while extracting central information on projection variables. This is carried out in two steps: *type extraction* and *normalization*.

3.1 Type extraction

Let Q be the input query and C the set of all possible classes from the ontology of the knowledge base to be queried, extended by the classes `Resource`, `Property` and `Value`. The aim of *type extraction* is to assign a combination of types to each projection variable of the query. To this end, we process the query by finding all graph patterns `?x rdf:type C` for each projection variable `?x` (with $\mathtt{C} \in C$). If none of the statements is part of a `UNION` statement, we assign the conjunction of all `C` to `?x`. Otherwise we assign to `?x` the disjunction of all `C` that are such that the `UNION` statements which contain `?x rdf:type C` contain no other statements.

[2]A demo can be found at `http://autosparql-tbsl.dl-learner.org`.
[3]The tool can be accessed at `http://at.bioasq.org`
[4]The following prefixes are used:
`dbo: <http://dbpedia.org/ontology/>`
`res: <http://dbpedia.org/resource/>`
`rdfs: <http://www.w3.org/2000/01/rdf-schema#>`
`rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>`
[5]See `http://www.w3.org/TR/sparql11-query/`.
[6]For a complete description of the vocabulary, see `http://nlp.stanford.edu/software/dependencies_manual.pdf`.

Dependency	Explanation
amod	Represents the *adjectival modifier* dependency. For example amod(ROSE,WHITE) stands for white rose.
cc	Stands for the relation between a conjunct and a given conjunction (in most cases and or or). For example in the sentence John eats an apple and a pear, cc(PEAR,AND) holds. We mainly use this construct to specify reduction and replacement rules.
conj*	Used to build the *conjunction* of two phrase elements, e.g. conj(subj(EAT,JOHN),subj(DRINK,MARY)) stands for John eats and Mary drinks. conj is not to be confused with the logical conjunction $\wedge$, which we use to state that two dependencies hold in the same sentence. For example subj(EAT,JOHN) $\wedge$ dobj(EAT,FISH) is to be read as John eats fish.
disj*	Used to build the *disjunction* of two phrase elements, similarly to conj.
dobj	Dependency between a verb and its *direct object*, for example dobj(EAT,APPLE) expresses to eat an/the apple.
nn	The *noun compound modifier* is used to modify a head noun by the means of another noun. For instance nn(FARMER,JOHN) stands for farmer John.
poss	Expresses a possessive dependency between two lexical items, for example poss(JOHN,DOG) express John's dog.
prep_X	Stands for the preposition X, where X can be any preposition, such as via, of, in and between.
prepc_X	Clausal modifier, used to modify verb or noun phrases by a clause introduced by some preposition X, e.g. prepc_suchthat(PEOPLE,c) represents people such that c, where c is some clause, e.g. their year of birth is 1950.
root	Marks the root of a sentence, e.g. the verb. For example ROOT(EAT) $\wedge$ subj(EAT,JOHN) means John eats. The root of the sentence will not always be stated explicitly in our formalization.
subj	Relation between *subject* and verb, for example subj(BE,JOHN) expresses John is.

Table 1: Dependencies used by SPARQL2NL. The dependencies which are part of our extension of the Stanford dependencies are marked with an asterisk.

Consequently, in Example 1, the type dbo:Person is assigned to the variable ?person.

When no explicit type information for a projection variable ?x is found, we try to detect implicit type information by mapping ?x to Resource if ?x is always the subject or the object of an object property, to Property if ?x is always a property, and to Value in all other cases. Thus, ?label is assigned the type Value, as it occurs as the object of a datatype property. Finally, all explicit information that was used to compute type information is deleted from the input query Q. Overall, this preprocessing step alters Example 1 by removing the triple ?person rdf:type dbo:Person and storing it as type information for the variable ?person. In addition, the variable ?label is assigned the type Value.

3.2 Normalization

One SPARQL feature that often leads to queries that are difficult to understand is the nesting of UNION statements. To ensure that we generate easily legible natural language representations of SPARQL queries, we *normalize* the input queries further by transforming any nesting of disjunctions (i.e. UNION statements) into a disjunctive normal form (DNF). We chose to use DNFs because they allow us to make explicit use of conjunctions binding stronger than disjunctions in English. An obvious drawback of this normalization approach is that it can lead to an exponential growth of the number of terms in a query. Yet, this drawback seems to be of minute relevance for practical applications. For example, no query in the benchmark was verbalized in more than 2s.

4. PROCESSING

The goal of the subsequent processing step is to generate a list of dependency trees for an input query. To achieve this goal (and in accordance with formalization introduced in 2.1), the query is subdivided into the three segments *body* (B), *optional* (O) and *modifier* (M), each of which is assigned its own sentence tree. Since ASK queries only possess a subset of these features, they can also be processed by our approach. Therefore, in the following, we only describe how the representation of each of these segments is generated for SELECT queries. As each of these representations relies on the same processors for processing triple patterns, we begin by presenting the processing of simple graph patterns.

4.1 Processing triple patterns

The realization of a triple pattern s p o depends mostly on the verbalization of the predicate p. If p can be realized as a noun phrase, then a possessive clause can be used to express the semantics of s p o, as shown in 1. For example, if p is a relational noun like author, then the verbalization is ?x's author is ?y. In case p's realization is a verb, then the triple can be verbalized as given in 2. For example, if p is the verb write, then the verbalization is ?x writes ?y.

1. ρ(s p o) $\Rightarrow$ poss(ρ(p),ρ(s)) $\wedge$ subj(BE,ρ(p)) $\wedge$ dobj(BE,ρ(o))
2. ρ(s p o) $\Rightarrow$ subj(ρ(p),ρ(s)) $\wedge$ dobj(ρ(p),ρ(o))

In cases where p is a variable or where our approach fails to recognize the type of the predicate, we rely on the more generic approach in 3 as a fallback, where REL is short for to relate. This representation amounts to s is related to o via p.

3. ρ(s p o) $\Rightarrow$ subj(REL,ρ(s)) $\wedge$ dobj(REL,ρ(o)) $\wedge$ prep_via(REL,ρ(p))

In our running example, verbalizing ?person dbo:birthDate ?date would thus lead to ?person's birth date is ?date, as birth date is a noun.

4.2 Generating the body section

The main effort during the processing step is concerned with representing the body of the query, i.e. the content of the WHERE clause. Our approach begins by transforming the type information retrieved by the preprocessing step described in 3.1 above (either an atomic type y, or a conjunction $\mathtt{y} \wedge \mathtt{z}$ or disjunction $\mathtt{y} \vee \mathtt{z}$) into a coordinated phrase element CPE_T, relying on the following rules:

4. ρ(?x rdf:type y) $\Rightarrow$ nn(ρ(y),?x)
5. ρ(y $\wedge$ z) $\Rightarrow$ conj(ρ(y),ρ(z))
6. ρ(y $\vee$ z) $\Rightarrow$ disj(ρ(y),ρ(z))

For Example 1, nn(dbo:Person,?person) is generated. DISTINCT is considered an adjective while COUNT is mapped to a noun phrase:

7. ρ(DISTINCT X) $\Rightarrow$ amod(ρ(X),DISTINCT) (i.e., distinct X)
8. ρ(COUNT X) $\Rightarrow$ prep_of(NUMBER,ρ(X)) (i.e., number of X)

The processing then continues by converting the content of the WHERE clause into a second coordinated phrase element CPE_W. Within this clause, only group graph patterns GP (i.e., combinations of conjunctions, UNIONs and FILTERs) can be used. The following set of rules deal with conjunctions and disjunctions:

9. ρ(GP$_1$,GP$_2$) $\Rightarrow$ conj(ρ(GP$_1$),ρ(GP$_2$))
10. ρ(UNION(GP$_1$,GP$_2$)) $\Rightarrow$ disj(ρ(GP$_1$),ρ(GP$_2$))

Processing FILTER is more intricate due to the large number of operators and functions that can be used in this construct. Therefore, FILTER leads to a more significant number of rules. In general, most operators OP that can be used in a filter can be expressed by a verbal clause ρ(OP). Binary operators can be translated by the following rule:

11. ρ(OP(x,y)) $\Rightarrow$ subj(ρ(OP),x) $\wedge$ dobj(ρ(OP),y)

Functional filters (i.e., filters of the form f(x)=y) can be translated into equivalent operators f(x,y) and verbalized as above. The body section B is finally generated by joining CPE_T and CPE_W as follows:

12. ρ(B) $\Rightarrow$ subj(RETRIEVE,QUERY) $\wedge$ dobj(RETRIEVE,ρ(CPE_T)) $\wedge$ prepc_suchthat(ρ(CPE_T),ρ(CPE_W))

This leads to query verbalizations of the general form This query retrieves... such that..., e.g. This query retrieves distinct values ?x such that ?x is Abraham Lincoln's death date.

4.3 Processing the optional section

The SPARQL constructs that can be used in the OPTIONAL section of the query are a subset of the constructs that can be used in the query body. For constructing the representation O of the optional section of the query (if such a section exists) we therefore reuse the same set of rules as those used to generate the body B.

4.4 Processing solution modifiers and aggregates

Solution modifiers alter the order in which the results of the SPARQL query are presented to the user. They include ORDER BY, LIMIT and OFFSET constructs. Translating ORDER BY OG(?x) (where OG is either the empty string, ASC or DESC) follows the rule given in 13, e.g. yielding The results are in descending order.

13. ρ(ORDER BY OG(?x)) $\Rightarrow$ subj(BE,RESULTS) $\wedge$ prep_in(BE,ORDER) $\wedge$ amod(ORDER,ρ(OG))

If no ordering is specified, we assume OG=ASC according to the SPARQL specification. For LIMIT and OFFSET, we use the generic rule in 14, e.g. yielding The query returns results between number 2 and 5.

14. ρ(OFFSET n LIMIT m) $\Rightarrow$ subj(RETURN,QUERY) $\wedge$ dobj(RETURN,RESULTS) $\wedge$ prep_between(RESULTS, conj(ρ(n+1),ρ(n+m)))

The sections of this rule are altered depending on whether LIMIT is used without OFFSET and in case the difference between the argument of LIMIT and OFFSET is 1, e.g. in order not to construct The query returns results between number 1 and 1 but rather The query returns the first result. The aggregation constructs are dealt with in a similar fashion.

After the processing step, our Example 1 would be verbalized as follows:

15. This query retrieves distinct people ?person such that ?person's occupation is Surfing or ?person's occupation is Writer, ?person's birth date is ?date and ?date is later than 1950. Additionally, it retrieves distinct values ?string such that ?person's label is ?string and ?string is in English if such exist.

5. POSTPROCESSING

Although the natural language output of the verbalization step just described is a correct description of the content of the SPARQL query, it often sounds very artificial. The general goal of the subsequent postprocessing step is thus to transform the generated description such that it sounds more natural. To this end, we focus on two types of transformation rules (cf. [3]): *aggregation* and *referencing*. Aggregation serves to remove redundancies and collapse information that is too verbose otherwise, for example:

?place is Shakespeare's birth place or ?place is Shakespeare's death place.

$\Rightarrow$?place is Shakespeare's birth or death place.

Referencing aims at achieving a natural verbalization of noun phrases, in particular avoiding variables wherever possible. For example:

This query retrieves values ?height such that ?height is Claudia Schiffer's height.

$\Rightarrow$ This query retrieves Claudia Schiffer's height.

The input to the postprocessor is the output of the preceding processing step described in Section 4 above, i.e., a set of variables with types (the select clause) and a list of dependency trees describing these variables. In the following, we describe the transformation rules we employ in more detail. The order in which they are applied is: clustering and ordering (5.1), aggregation (5.3), grouping (5.2), referencing (5.4).

5.1 Clustering and ordering rules

The very first aggregation step serves to cluster and order the input sentences. To this end, the variables occurring in the query are ordered with respect to the number of their occurrences, distinguishing projection variables, i.e. variables that occur in the SELECT clause, from all others, and assigning them those input sentences that mention them. In case the most frequent variable is the object of the sentence, the sentence is passivized (presupposed the verb is not an auxiliary or copulative verb such as is or has), in order to maximize the effect of aggregation later on. If, for example, the variable is ?river, then a sentence Brooklyn Bridge crosses ?river is transformed into its passive counterpart ?river is crossed by Brooklyn Bridge. We process the input trees in descending order with respect to the frequency of the variables they contain, starting with the projection variables and only after that turning to other variables. As an example, consider the following query retrieving the youngest player in the Premier League:

```
SELECT DISTINCT ?person WHERE {
 ?person dbo:team ?sportsTeam .
 ?sportsTeam dbo:league res:Premier_League .
 ?person dbo:birthDate ?date .
}
ORDER BY DESC(?date) OFFSET 0 LIMIT 1
```

The only projection variable is ?person (two occurrences), other variables are ?sportsTeam and ?date (one occurrence each). The three triple patterns are verbalized as given in 16a–16c. Clustering and ordering now first takes all sentences containing the primary variable, i.e. 16a and 16b, which are ordered such that copulative sentences (such as ?person is a person) come before other sentences, and then takes all sentences containing the remaining variable ?sportsTeam in 16c (the only occurrence of ?date is already settled with 16b), resulting in a sequence of sentences as in 17.

16. (a) ?person's team is ?sportsTeam.
    (b) ?person's birth date is ?date.
    (c) ?sportsTeam's league is Premier League.
17. ?person's team is ?sportsTeam, ?person's birth date is ?date, and ?sportsTeam's league is Premier League.

5.2 Grouping

Grouping is described by Dalianis & Hovy [3] as a process "collecting clauses with common elements and then collapsing the common elements". The common elements are usually subject noun phrases and verb phrases (verbs together with object noun phrases), leading to subject grouping and object grouping. In order to maximize the grouping effects, we additionally collapse common prefixes and suffixes of sentences, irrespective of whether they are full subject noun phrases or complete verb phrases. In the following we use $X_1, X_2, \dots$ as variables for the root nodes of the input sentences and Y as variable for the root node of the output sentence. Furthermore, we abbreviate a subject $\texttt{subj}(X_i, s_i)$ as $\texttt{s}_i$, an object $\texttt{dobj}(X_i, o_i)$ as $\texttt{o}_i$, and a verb $\texttt{root}(ROOT_i, v_i)$ as $\texttt{v}_i$.

Object grouping collapses the subjects of two sentences if the realizations of the verbs and objects of the sentences are the same, where the *coord* $\in \{\mathsf{and}, \mathsf{or}\}$ is the coordination combining the input sentences X_1 and X_2, and $\texttt{coord} \in \{\texttt{conj}, \texttt{disj}\}$ is the corresponding coordination combining the subjects.

18. $\rho(\texttt{o}_1) = \rho(\texttt{o}_2) \land \rho(\texttt{v}_1) = \rho(\texttt{v}_2) \land \texttt{cc}(\texttt{v}_1, \textit{coord})$
    $\Rightarrow \texttt{root}(Y, \texttt{PLURAL}(\texttt{v}_1)) \land \texttt{subj}(\texttt{v}_1, \texttt{coord}(\texttt{s}_1, \texttt{s}_2)) \land$
    $\texttt{dobj}(\texttt{v}_1, \texttt{o}_1)$

For example, the sentences in 19 share their verb and object, thus they can be collapsed into a single sentence. Note that to this end the singular auxiliary was needs to be transformed into its plural form were. In case the subjects themselves share common elements, the subjects are collapsed as well, as in 20.

19. Benjamin Franklin was born in Boston and Leonard Nimoy was born in Boston. $\Rightarrow$ Benjamin Franklin and Leonard Nimoy were born in Boston.
20. Abraham Lincoln's birth place is Washington or Abraham Lincoln's death place is Washington. $\Rightarrow$ Abraham Lincoln's birth place or death place is Washington.

In addition, we remove repetitions that arise when triple pattern verbalizations lead to the same natural language representation. Due to space restrictions, we leave out a presentation of subject grouping, as it works analogously.

A further aggregation rule that removes redundant mentions of variables collapses more generally common suffixes and prefixes, i.e. sentences of form $\texttt{subj}_1$ $\texttt{verb}_1$ $\texttt{dobj}_1$ with sentences of form $\texttt{subj}_2$ $\texttt{verb}_2$ $\texttt{dobj}_2$ in case $\texttt{dobj}_1$ is the same as $\texttt{subj}_2$ (and if its is a variable, does not occur anywhere else) and either $\texttt{verb}_1$ or $\texttt{verb}_2$ is a form of to be. An example is given in 22.

21. (a) $\rho(\texttt{o}_1) = \rho(\texttt{s}_2) \wedge \rho(\texttt{v}_1) = \texttt{BE}$
    $\Rightarrow \texttt{subj}(Y, \texttt{s}_1) \wedge \texttt{dobj}(Y, \texttt{o}_2) \wedge \texttt{root}(Y, \texttt{v}_2)$
    (b) $\rho(\texttt{o}_1) = \rho(\texttt{s}_2) \wedge \rho(\texttt{v}_2) = \texttt{BE}$
    $\Rightarrow \texttt{subj}(Y, \texttt{s}_1) \wedge \texttt{dobj}(Y, \texttt{o}_2) \wedge \texttt{root}(Y, \texttt{v}_1)$
    If $\texttt{o}_1/\texttt{s}_2$ is not a variable occurring anywhere else.

22. ?w's year is ?x. ?x is greater than or equal to 2007.
    $\Rightarrow$?w's year is greater than or equal to 2007.

5.3 Aggregating filters and optional information

The verbalization of filters can be quite verbose, although they often express a simple constraint on a value. Postprocessing thus attaches filter information to the expression they constrain. For example a filter like in 23 is verbalized and then collapsed as in 24.

23. `?person rdfs:label ?name.`
    `FILTER(regex(?name,'Michelle'))`

24. ?person's label is ?name. ?name matches "Michelle".
    $\Rightarrow$?person has the label ?name matching "Michelle".
    $\Rightarrow$?person has a label matching "Michelle".
    (if ?name does not occur anywhere else)

For every sentence, the filter linearizations are checked whether they contain either the subject or object of the sentence, and if they do, they are attached to the it either using a gerund or a relative clause, depending on the filter. A filter construct that is particularly difficult to handle are `!BOUND` filters as in 25. Postprocessing transforms statements of form X does not exist into the negation of some statement containing X, either negating the verb phrase, as in 26, or by adding the quantifier no, as in This query asks whether there is no entity such that. . . .

25. `res:Frank_Herbert dbo:deathDate ?date . FILTER (!BOUND(?date))`

26. This query asks whether Frank Herbert's death date is ?date and ?date does not exist. $\Rightarrow$ This query asks whether Frank Herbert's death date does not exist.

An extensive and careful treatment of all kinds of `BOUND` filters in `SELECT` and `ASK` queries is subject of future work.

Information about the label of a variable and its type (if it is not already part of the `SELECT` clause) may be part of the filters and is verbalized by the processing step for example as in 27a. In order to collapse these information into something less verbose, the postprocessor collects such sentences based on simple string matching and transforms them into a single sentence as in 27b. In case only the label information is expressed, the result is as in 27c; in case the variable is part of the `SELECT` clause, the type and label information is attached there, as in 27d.

27. (a) ?uri is of type film. ?uri has label ?string. ?string is in English.
    (b) ?uri is a film with the English label ?string
    (c) ?uri has the English label ?string
    (d) This query retrieves films ?uri and their English label ?string such that. . .

Content in `OPTIONAL` statements is expressed as an additional sentence of the form Additionally, it retrieves V such that O if such exists, for some selected entities V and triple patterns O. Postprocessing integrates the information in O into the main body of the natural language description, marking the statements with the modal may, e.g. transforming 27c above into ?uri may have the English label ?string.

5.4 Referencing

Referencing refers to the process of deciding how to verbalize each occurrence of an entity (e.g., as a singular or plural noun phrase or a pronoun). The ultimate aim of the postprocessing step is to collapse and substitute all variable occurrences such that the final output does not contain any variables at all. This goal is achieved for almost all sentences, but proves hard in the case of very complex queries with lots of variables. An easy case is the following: If the input sentence with root X is of the form This query retrieves o such that B, the body B contains a copula statement Y whose subject (resp. object) has the same realization as o, and o does not occur anywhere else, then it is safe to collapse the input sentence into This query retrieves x such that B', where x ist the object (resp. subject) of Y, and B' is B without Y. An example is the following, where the keyword distinct is dropped, as this seems more natural, especially for laymen, although it could be argued that it should be kept for experts:

28. This query retrieves distinct entities ?string such that Angela Merkel's birth name is ?string. $\Rightarrow$ This query retrieves Angela Merkel's birth name.

In case the verb of the body statement is not a copula, as in the following example, the information is added to the input sentence in form of a relative clause:

29. This query retrieves entities such that ?river is crossed by Brooklyn Bridge. $\Rightarrow$ This query retrieves entities that are crossed by Brooklyn Bridge.

In addition, the postprocessing step replaces all occurrences of a projection variable, i.e. a variable which occurs in the `SELECT` clause, by pronouns if this is the only projection variable, e.g. transforming 30 into 31. In case of more variables, this would lead to ambiguities. Also, the first occurrence of remaining non-projecting variables, i.e. those variables which do not occur in the `SELECT` clause, are replaced by an indefinite (choosing their type as description, if given, e.g. some mountain, or entity otherwise), and all further occurences are replaced by a definite, e.g. further transforming 31 into 32.

30. This query retrieves distinct entities ?uri such that ?uri is ?x's b-side and ?x's musical artist is Ramones.

31. This query retrieves distinct entities such that they are ?x's b-side and ?x's musical artist is Ramones.

32. This query retrieves distinct entities such that they are some entity's b-side and this entity's musical artist is Ramones.

The output of the postprocessor for Example 1 is the following:

33. This query retrieves distinct people and their English label (if it exists) such that their birth date is later than 1950 and their occupation is Writer or Surfing.

Full-fledged referencing still remains a challenging task, especially if a query contains a range of different entities with several occurrences, also because it requires some knowledge about whether it is semantically singular or plural, in order to choose the correct description, e.g. the members of Prodigy (plural) and the father of Queen Elizabeth II (singular), or to decide whether to use the concept name label or name, and whether the entity is animated or not, in order to correctly choose the correct pronoun in the singular case (he/she or it).

6. REALIZATION

The realization of atoms must be able to deal with resources, classes, properties and literals.

6.1 Classes and resources

In general, the realization of classes and resources is carried out as follows: Given a URI u we ask for the English label of u using a SPARQL query.[7] If such a label does not exist, we use either the fragment of u (the string after `#`) if it exists, else the string after the last occurrence of `/`. Finally this natural language representation is realized as a noun phrase, and in the case of classes is also pluralized. In our running example, `dbo:Person` is realized as people (its label).

6.2 Properties

The realization of properties relies on the insight that most property labels are either nouns or verbs. While the mapping of a particular property p can be unambiguous, some property labels are not as easy to categorize. For examples, the label crosses can either be the plural form of the noun cross or the third person singular present form of the verb to cross. In order to automatically determine which realization to use, we relied on the insight that the first and last word of a property label are often the key to determining the type of the property: properties whose label begins with a verb (resp. noun or gerund) are most to be realized as verbs (resp. nouns). We devised a set of rules to capture this behavior, which we omit due to space restrictions. In some cases (such as crosses) none of the rules applied. In these cases, we compare the probability of $P(p|\texttt{noun})$ and $P(p|\texttt{verb})$ by measuring

$$P(p|X) = \frac{\sum\limits_{t \in synset(p|X)} \log_2(f(t))}{\sum\limits_{t' \in synset(p)} \log_2(f(t'))}, \qquad (1)$$

where $synset(p)$ is the set of all synsets of p, $synset(p|X)$ is the set of all synsets of p that are of the syntactic class $X \in \{\texttt{noun}, \texttt{verb}\}$ and $f(t)$ is the frequency of use of p in the sense of the synset t according to WordNet. For

$$\frac{P(p|\texttt{verb})}{P(p|\texttt{noun})} \geq \theta, \qquad (2)$$

we choose to realize p as a noun; else we realized it as a verb. For $\theta = 1$, for example, dbo:crosses is realized as a verb.

6.3 Literals

The realization of *literals* is carried out by differentiating between plain and typed literals. For plain literals we simply use the lexical form, i.e. omit language tags if they exist. For example, `"Albert Einstein"@en` is realized as Albert Einstein. For typed literals we further differentiate between built-in and user-defined datatypes. For the former we also use the lexical form, e.g. `"123"^^xsd:int` ⇒ 123. The latter were processed by using the literal value together with the (pluralized) natural language representation of the datatype URI, similarly to the case of classes and resources. Thus, we realize `"123"^^<http://dbpedia.org/datatype/squareKilometre>` as `123 square kilometres`.

7. EXPERIMENTS AND EVALUATION RESULTS

We evaluated SPARQL2NL with respect to i) the realization of atomic graph patterns, ii) the verbalization of whole SPARQL queries, and iii) other approaches, all using the QALD-2 benchmark[8] as basis.

7.1 Realization of atomic graph patterns

Given that the choice for the realization of atomic graph patterns depends on whether the predicate is classified as being a noun phrase or a verb phrase, we measured the accuracy (i.e., the percentage of right classifications) of our approach by realizing all properties occurring in the QALD-2 benchmark. As baseline, we used a simple classifier that classifies every property as a noun. We preferred this classifier over one that classifies every property as a verb, simply because it has a higher accuracy than both the verb classifier and a random classifier. The evaluation was carried out manually by two annotators who assessed the `rdfs:label` of every property in the QALD-2 benchmark regarding whether it is a noun or a verb. All mismatches were resolved by the same annotators. Note that in rare cases, some properties from the `property` namespace are used ambiguously within DBpedia. For example, `property:design` is used to mean designed as, designed in and even designed by. In these cases the annotators evaluated whether our realization mapped the intention of the query as specified in the benchmark.

We evaluated our approach with $\theta = 1$ and $\theta = 2$ by using the accuracy measure, which states the percentage of cases

[7]Note that it could be any property which returns a natural language representation of the given URI, see [6].

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

Dataset	Namespace	Frequency	#Verbs	#Nouns	Accuracy in %		
					$\theta = 1$	$\theta = 2$	Baseline
DBpedia-test	`property`	40	8	25	87.50	**90.00**	75.00
	`ontology`	97	7	48	91.75	**94.85**	86.60
	Other	99	2	1	**98.99**	**98.99**	32.32
	Overall	236	17	74	94.07	**95.76**	61.86
DBpedia-train	`property`	41	1	26	**100.00**	**100.00**	80.25
	`ontology`	81	5	43	95.06	**100.00**	85.37
	Other	135	3	2	**98.51**	**98.51**	42.96
	Overall	257	9	71	97.67	**99.22**	61.48

Table 2: Accuracy of realization of atomic graph patterns. Namespace stands for the namespace of the properties used in a SPARQL query. Frequency denotes the number of times that a property from a given namespace was used, for example `property` (which stands for `http://dbpedia.org/property`) or `ontology` (`http://dbpedia.org/ontology`). #Verbs (resp. #Nouns) is the number of properties that were classified as verbs (resp. nouns).

in which the correct classification was achieved across the queries in the benchmark. The results are shown in Table 2. We clearly outperform the baseline in all cases. Especially, setting $\theta = 2$ achieves an overall accuracy of 99.22% and a perfect score on the property from DBpedia namespace contained in the training dataset of QALD-2. Experiments with other values of θ did not lead to better results.

7.2 SPARQL2NL survey

In our second series of experiments, we evaluated the whole SPARQL2NL pipeline, in order to clarify the following two questions:

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

7.2.1 *Experimental setup*

We performed a user study in order to evaluate our verbalizations of SPARQL queries, using the 200 DBpedia queries provided by the QALD-2 benchmark, all of which our approach was able to translate into natural language. We ran a two-phase survey[9]: In the first phase, users were stripped from any communication devices, required to focus on the task at hand and complete the survey swiftly. In the second phase, we ran uncontrolled experiments with users from Semantic Web and NLP mailing lists as well as smaller non-research communities. The survey consists of three different tasks with 10 randomly selected queries each. At the start of the survey users can indicate whether or not they are SPARQL experts. If not, only Task 3 was presented, otherwise they were asked to complete all three tasks.

Task 1 (only for experts):. In this task, the survey participant is presented a SPARQL query and its SPARQL2NL verbalization, and is asked to judge the verbalization regarding fluency and adequacy, following the machine translation standard presented in [5]. Adequacy captures how well the verbalization captures the meaning of the SPARQL query, according to the following six ratings: (6) Perfect. (5) Mostly correct, maybe some expressions don't match the concepts very well. (4) Close, but some information is missing or incorrect. (3) There is significant information missing or incorrect. (2) NL description and SPARQL query are only loosely connected. (1) NL description and SPARQL query are in no conceivable way related. Fluency, on the other hand, captures how good the natural language description is in terms of comprehensibility and readability, according to the following six ratings: (6) Perfectly clear and natural. (5) Sounds a bit artificial, but is clearly comprehensible. (May contain minor grammatical flaws.) (4) Sounds very artificial, but is understandable. (May contain significant grammatical flaws.) (3) Barely comprehensible, but can be understood with some effort. (2) Only a loose and incomplete understanding of the meaning can be obtained. (1) Completely not understandable at all.

Task 2 (only for experts):. In this task, the participant is presented a SPARQL query as well as five different possible answers (variable bindings). From these answers, those which would actually be returned by the query had to be selected. To this end, for each answer a set of triples was offered as explanation, which should be used to judge whether the answer is correct. These triples were generated as follows: We first executed the SPARQL query and randomly selected up to five results from the query answer. For each correct answer, we replaced the return variable (`?uri` in the case of the QALD-2 `SELECT` queries) by the URI of the answer, and replaced all other URIs occurring in the query by variables, in order to retrieve all triples relevant for answering the query[10]. For each incorrect answer, we first generalised the SPARQL query by removing a triple pattern, or by replacing a URI by a variable. This procedure is repeated until the query returns results not returned by the original query. This ensures that the incorrect answers are similar to the correct answers in the sense that they are results of a similar SPARQL queries. The formal details of the procedure follow [18].

[9]The survey interface can be accessed at `http://sparql2nl.aksw.org/eval`.

[10]This simple technique does not cover all cases, but we refrain from a full explanation, since it is not necessary to understand the survey.

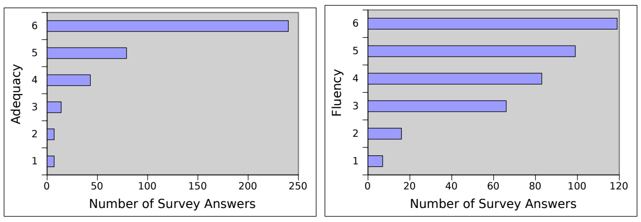

Figure 1: Adequacy and fluency results in survey

Task 3 (experts and non-experts):. This task is similar to Task 2, with the difference that the natural language verbalizations of the SPARQL query and a verbalization of the triples were presented, instead of the query and the triples themselves. We also ensured that the queries used were different from those used in Task 2, in order to avoid training effects on particular questions.

7.2.2 *Results*

The first survey phase was carried out by 10 members of the AKSW and CITEC research groups. As these participants were monitored by one of the authors, we used it for time measurements on the three different tasks. The maximum (minimum) time required was 17 (7) minutes, 13 (6) minutes and 12 (4) minutes for Tasks 1, 2 and 3, respectively. We then ran a public survey, that was announced on Semantic Web and NLP mailing lists as well as to other non-research communities, collecting 115 participants, of which 39 stated they were experts in SPARQL. We used our initial time measurements to filter out those survey participants in the public evaluation who are unlikely to have thoroughly executed the survey or who were likely distracted while executing it. To this end, we decided to admit a time window of 5-18 minutes for Task 1 and 3-15 minutes for Tasks 2 and 3.[11] Although this cannot eliminate all side effects, it reduces the effect of outliers, e.g. people leaving the computer for a long period of time.

The results of the first task showed the fluency of the natural language descriptions to be 4.56 ± 1.29, where in expressions of the form $x \pm y$, x denotes the average value and y denotes the standard deviation. The majority of natural language descriptions were understandable, where 94.1% of the cases achieved a rating of 3 or higher. The adequacy of the verbalizations was judged to be 5.31 ± 1.08, which we consider a positive result. 62% of all verbalizations were judged to be perfectly adequate. Details for the results of Task 1 are depicted in Figure 1.

For Tasks 2 and 3, our main goal was to directly compare the results of users dealing with SPARQL queries against the results of users dealing with natural language descriptions. Here we consider the time required to answer a question as an indicator for efficiency, and the error rate of a user as an indicator for effectiveness. Regarding efficiency, participants required 11.68 ± 6.46 minutes to complete Task 2. Applying the time window mentioned above, the required time drops to 9.89 ± 3.48. For Task 3 we obtained execution times of 10.28 ± 7.03 without filtering, and 8.37 ± 2.63 with time

[11]Task 2 and 3 require the same limits in order to avoid a bias when doing cross comparisons between Task 2 and 3.

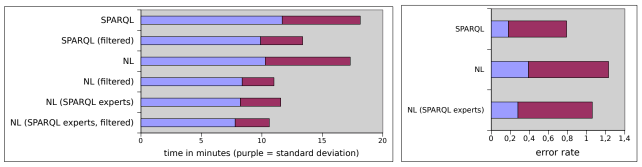

Figure 2: Time and error rate analysis

filtering. In general, execution times were in line with what we expected after the first evaluation phase, i.e. most participants swiftly completed the survey. Using a paired t-test with 95% confidence interval, the difference between the time required for Tasks 2 and 3 is statistically significant. Hence, we conclude that the natural language descriptions generated by our approach can be more efficiently read and understood. Note that even the SPARQL experts were faster when being presented the natural language description, with 8.22 ± 3.34 minutes without time window filter and 7.79 ± 2.83 with time window filter. We therefore conclude that the SPARQL2NL translations can be processed efficiently by both experts and non-experts.

Finally we also compared the error rates of participants in Tasks 2 and 3, i.e. the number of incorrect answers per questions, see Figure 2. The error rate in Task 2, i.e. when displaying SPARQL queries and RDF triples, was 0.18 ± 0.61, in contrast to 0.39 ± 0.84 in Task 3, i.e. when displaying natural language descriptions. If we consider only experts in Task 3, i.e. the group of people who did both Task 2 (displaying SPARQL and RDF) and Task 3 (displaying natural language), the error rate was 0.28 ± 0.78. This is a negative result for SPARQL2NL as it appears that non-expert participants made more errors than expert participants; although the overall rate still seems reasonably low. Upon a deeper investigation of this issue, it turned out that almost all errors occurred with two specific queries, both due to bugs in the implementation of SPARQL2NL: one query translation used a passive form incorrectly, and the other one lacked the keyword also indicating that two criteria had to be satisfied.

We fixed these issues in an updated version of SPARQL2NL and ran an internal evaluation again using the new verbalizations. 13 participants from the AKSW and CITEC research groups, excluding the authors, took part in this validation phase. It turned out that the error rate for the natural language expressions in that case is only slightly higher (+0.05) compared to the SPARQL expressions for SPARQL experts. Both error rates were lower than in the public evaluation with 0.12 ± 0.35 and 0.07 ± 0.25 for the natural language and SPARQL part, respectively. The improvements based on the evaluation also led to improved fluency (increased by 0.51 to 5.05 ± 1.01) and adequacy (increased by 0.29 to 5.60 ± 0.85) results. These results strengthen the conclusions made above with respect to the fluency and adequacy of our verbalizations.

7.3 Comparison with other approaches

To the best of our knowledge, SPARTIQULATION [7] is the only other approach that verbalizes SPARQL queries. It relies on detecting a main entity, which is used to subdivide the query graph into subgraphs, that are ordered and matched

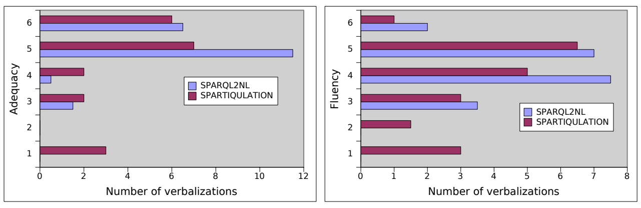

Figure 3: Average adequacy and fluency results for comparison of SPARQL2NL and SPARTIQULATION

with pre-defined message types.

We compared SPARQL2NL with SPARTIQULATION on a random sample of 20 queries retrieved from the QALD-2 benchmark within a blind survey: We asked two SPARQL experts to evaluate the adequacy and fluency of the verbalizations achieved by the two approaches. The experts were not involved in the development of any of the two tools and were not aware of which tool produces which verbalization. Of the 20 queries, SPARTIQULATION could only verbalize 17 while SPARQL2NL was able to verbalize all queries. This difference is due to SPARTIQULATION being currently limited to `SELECT` queries and not covering some important SPARQL features such as `UNION` and `GROUP BY` constructs, features which we can deal with as shown above. The results of our comparison show that in average there is a difference of 0.24 resp. 0.27 with respect to adequacy resp. fluency between the two approaches (5.24 resp. 4.41 in adequacy resp. fluency for SPARQL2NL versus 5.0 resp. 4.15 for SPARTIQULATION) when only taking SPARQL queries into consideration queries that SPARTIQULATION was able to process. Note that even in this setting, SPARQL2NL outperforms SPARTIQULATION. When considering all queries in our sample, counting the adequacy and fluency for a query that could not be translated as 1, our approach clearly outperforms SPARTIQULATION, as shown in Figure 3. Most of the verbalizations of SPARQL2NL are scored with 6 or 5 with respect to adequacy by the experts, while a larger amount of verbalizations from SPARTIQULATION are scored with 1–4. Moreover, our verbalizations are most frequently assigned fluency scores between 4 and 5. Overall, SPARQL2NL achieved an average adequacy resp. fluency of 5.15 resp. 4.38 while SPARTIQULATION achieved 4.40 resp. 3.68.

8. RELATED WORK

Although there is a substantial amount of work on translating natural language into database elements or queries (see, e.g., [23]) or even SPARQL [25, 28], the other direction, i.e. verbalizing databases and queries, has started to receive attention only recently [15]. Most of the papers related to this work have been on ontology and RDF verbalization or on reusing such data for the purpose of verbalization. [1] for example combines linguistic and domain-specific ontologies to generate natural-language representations of portions of ontologies required by users. One of the results of this work is that even graphical representations of ontologies are of little help for lay users. This result is the premise for the work by Wilcock [29], who presents a more generic approach for the purpose of verbalizing OWL and DAML+OIL. [14] present an approach for generating paraphrases of the content of OWL ontologies that combines natural-language patterns for expressing the structure of property labels and a verbalization approach for OWL class expression. Works such as [16, 13, 27] use controlled fragments of natural language such as English and Baltic languages to generate textual representation of OWL ontologies. Other works on verbalizing OWL ontologies include [2, 4, 8] and [10].

In addition to the work on OWL, research on textual descriptions of RDF triples is also gaining momentum. For example, [22] elaborates on an approach for transforming RDF into Polish. The authors of [20] argue for relying on the Linked Data Web being created by using to reverse engineer structured data into natural language. The same authors show in [26] how this approach can be used for generating text out of RDF. In newer work, [19] generated natural language out of RDF by relying on the BOA framework [12, 11] with the aim of computing the trustworthiness of RDF triples by using the Web as background knowledge. Other approaches and concepts for verbalizing RDF include [21] and [30]. Moreover, approaches to verbalizing first-order logics [9] are currently being devised.

An approach for translating database queries into natural language text has been provided by, e.g., Koutrika et al. [17], focusing on SQL queries but noting that the same need arises for SPARQL queries. A noteworthy approach is that presented in [15], where the authors apply graph algorithms to an efficient partition and realization of SQL queries. The only work we are aware of that verbalizes SPARQL queries is the aforementioned recent approach by Ell et al. [7].

9. CONCLUSION

In this paper, we presented SPARQL2NL, an approach for verbalizing SPARQL queries. It produces both a direct, literal verbalization of the content of the query and a more natural, aggregated version of the same content. We presented the key steps of our approach and evaluated it with a user survey. Our evaluation showed that the verbalizations generated by our approach are both complete and easily understandable. In addition, our approach allows users not familiar with SPARQL to understand the content of SPARQL queries and also accelerates the understanding of queries by SPARQL experts. Still, our evaluation showed that the legibility of our approach is worse when the queries get more complex. In future work, we will thus improve upon our referencing algorithm so as to further increase the fluency of our approach. Moreover, we will devise a consistency checking algorithm to improve upon the correctness of the natural language generated by our approach. Finally, we will integrate paraphrasing approaches into our system to augment the variety of the formulations used by our system and thus improve the quality of the interaction with the end users. SPARQL2NL represents the first step towards semantic applications that enable lay users to understand the behavior of the applications without any need for technical knowledge. We hope that it will facilitate the acceptance of Semantic Web technologies across domains of application.

Acknowledgement

Parts of the work presented herein were financed by the FP7 Support Action BioASQ under grant agreement no. 318652.

10. REFERENCES

[1] G. Aguado, A. Bañón, John 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] Kalina Bontcheva and Yorick Wilks. Automatic report generation from ontologies: The miakt approach. In *NLDB*, pages 324–335, 2004.

[3] H. Dalianis and E.H. Hovy. Aggregation in natural language generation. In G. Adorni and M. Zock, editors, *Trends in natural language generation: an artificial intelligence perspective*, volume 1036 of *Lecture Notes in Artificial Intelligence*, pages 88–105. Springer, 1996.

[4] Brian Davis, Ahmad Iqbal, Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, and Siegfried Handschuh. Roundtrip ontology authoring. In *ISWC*, pages 50–65, 2008.

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

[6] Basil Ell, Denny Vrandecic, and Elena Paslaru Bontas Simperl. Labels in the web of data. In *Proceedings of ISWC*, volume 7031, pages 162–176. Springer, 2011.

[7] Basil Ell, Denny Vrandečić, and Elena Simperl. SPARTIQULATION – Verbalizing SPARQL queries. In *Proceedings of ILD Workshop, ESWC2012*, 2012.

[8] Günther Fliedl, Christian Kop, and Jürgen Vöhringer. Guideline based evaluation and verbalization of owl class and property labels. *Data Knowl. Eng.*, 69(4), 2010.

[9] Norbert E. Fuchs. First-order reasoning for attempto controlled english. In *CNL*, pages 73–94, 2010.

[10] Dimitrios Galanis and Ion 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.

[11] Daniel Gerber and Axel-Cyrille Ngonga Ngomo. Extracting multilingual natural-language patterns for rdf predicates. In *EKAW*, pages 87–96, 2012.

[12] Daniel Gerber and Axel-Cyrille Ngonga Ngomo. Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction @ ISWC 2011*, 2011.

[13] Normunds Gruzitis, Gunta Nespore, and Baiba Saulite. Verbalizing ontologies in controlled baltic languages. In Inguna Skadina and Andrejs Vasiljevs, editors, *Baltic HLT*, volume 219 of *Frontiers in Artificial Intelligence and Applications*, pages 187–194. IOS Press, 2010.

[14] Daniel Hewlett, Aditya Kalyanpur, Vladimir Kolovski, and Chris Halaschek-Wiener. Effective natural language paraphrasing of ontologies on the semantic web. In *Proceedings of the End User Semantic Web Interaction Workshop (ISWC 2005)*, 2005.

[15] Yannis Ioannidis. From databases to natural language: The unusual direction. In E. Kapetanios, V. Sugumaran, and M. Spiliopoulou, editors, *Natural Language and Information Systems*, volume 5039 of *LNCS*, pages 12–16, 2008.

[16] Kaarel Kaljurand and Norbert E. Fuchs. Verbalizing OWL in Attempto Controlled English. In *Proceedings of Third International Workshop on OWL: Experiences and Directions, Innsbruck, Austria (6th–7th June 2007)*, volume 258, 2007.

[17] G. Koutrika, A. Simitsis, and Y.E. Ioannidis. Explaining structured queries in natural language. In *Proceedings of the 26th International Conference on Data Engineering (ICDE)*, pages 333–344, 2010.

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

[19] Jens Lehmann, Daniel Gerber, Mohamed Morsey, and Axel-Cyrille Ngonga Ngomo. Defacto - deep fact validation. In *ISWC*, 2012.

[20] Chris Mellish and Xiantang Sun. The semantic web as a linguistic resource: opportunities for natural language generation. In *Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 2006.

[21] H. Piccinini, M. A. Casanova, A. L. Furtado, and B. P. Nunes. Verbalization of rdf triples with applications. In *ISWC - Outrageous Ideas track*, 2011.

[22] Aleksander Pohl. The polish interface for linked open data. In *Proceedings of the ISWC 2010 Posters & Demonstrations Track*, pages 165–168, 2011.

[23] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, IUI '03, pages 149–157, 2003.

[24] Ehud Reiter and Robert Dale. *Building natural language generation systems.* Cambridge University Press, New York, NY, USA, 2000.

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

[26] Xiantang Sun and Chris 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.

[27] Allan Third, Sandra Williams, and Richard Power. Owl to english: a tool for generating organised easily-navigated hypertexts from ontologies. In *ISWC Poster and Demo Track*, 2011.

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

[29] Graham Wilcock. Talking OWLs: Towards an Ontology Verbalizer. In *Human Language Technology for the Semantic Web and Web Services, Workshop at ISWC 2003*, pages 109–112, 2003.

[30] Graham Wilcock and Kristiina Jokinen. Generating Responses and Explanations from RDF/XML and DAML+OIL, 2003.