Ontology-Based Translation of Natural Language Queries to SPARQL

Malte Sander[1,2], **Ulli Waltinger**[1], **Mikhail Roshchin**[1] **and Thomas Runkler**[1,2]
[1]Siemens Corporate Technology, Munich, Germany
{malte.sander, ulli.waltinger, mikhail.roshchin, thomas.runkler}@siemens.com
[2]TU München, Germany

Abstract

We present an implemented approach to transform natural language sentences into SPARQL, using background knowledge from ontologies and lexicons. Therefore, eligible technologies and data storage possibilities are analyzed and evaluated. The contributions of this paper are twofold. Firstly, we describe the motivation and current needs for a natural language access to industry data. We describe several scenarios where the proposed solution is required. Resulting in an architectural approach based on automatic SPARQL query construction for effective natural language queries. Secondly, we analyze the performance of RDBMS, RDF and Triple Stores for the knowledge representation. The proposed approach will be evaluated on the basis of a query catalog by means of query efficiency, accuracy, and data storage performance. The results show, that natural language access to industry data using ontologies and lexicons, is a simple but effective approach to improve the diagnosis process and the data search for a broad range of users. Furthermore, virtual RDF graphs do support the DB-driven knowledge graph representation process, but do not perform efficient under industry conditions in terms of performance and scalability.

Introduction

Processing data in diagnostic or search related purposes in Alarm Management Systems becomes more and more complicated, which is attributed to the following reasons:

The diagnosis process, the search for root causes or the calculation of Key Performance Indicators (KPIs) relies on handling large amounts of data. Archived and streaming data have to be included into the diagnosis to achieve proper results. Nowadays, collected data sums up to hundreds of TB. In addition to the large amounts of data, more and more data is generated every day (Giese et al. 2013). Different vendors of machines or single components, coupled with historical or compatibility reasons, lead to multiple different logical data representations. Providing an unified and efficient access to all the different logical models is complex and sophisticated.

The processed data is heterogeneous and can be divided into two representation forms. While structured data with fixed format or data types represents the minority with only 20%, about 80% of the data is unstructured (Nugent et al. 2013). Modern query languages do not really support the handling of unstructured data, except for storage and manual analysis (Nugent et al. 2013). The manual analysis depends strongly on the available data and is prone to changes. The application of intelligent algorithms is necessary in order to make use and value of both, structured and unstructured data within an unified representation. This results in increasing query complexity to request the desired information contained in the unstructured data.

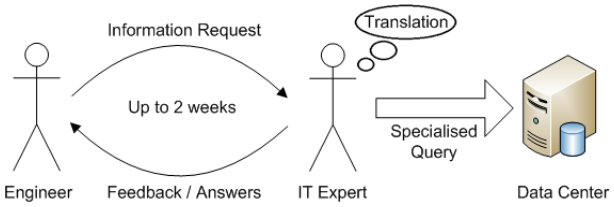

Figure 1: The support of IT Experts is ineffective and time consuming. The iteration process of requesting information and generating feedback lasts up to 2 weeks.

The calculation of KPIs which relate to content in the unstructured data, e.g. the computation of the availability of a plant or machine from text-based events like “start up” and shut down” is cumbersome. The extraction of timestamped events based on unstructured data is prone to errors and differs from domain to domain.

Varying data representations and the difficulties of processing unstructured data, require additional support for engineers through predefined search queries or diagnostic tools. Search queries have to be updated and adapted to different logical representations or upcoming unstructured events regularly. In order to formulate and perform target-oriented search tasks, engineers require additional support from IT Experts more frequently. Requesting the support of IT Experts is ineffective as shown in Figure 1. Making appointments, communicating the problem and desired solutions, as well as necessary training periods cost time and money. In total, the time engineers spend searching for data increases. Faulty or missing information leads to high expenses for companies for several reasons. Bad decisions

based on wrong information cause accidents, resulting in failures and machine damage or even human harm. Additional costs are generated when internal employees are unable to find their required knowledge in time or at all. Consequently, expensive external experts are required (Feldman and Sherman 2001).

Engineers in the oil and gas industry spend about 30% to 70% of their time searching for data and assessing the quality of the data (Alcook 2009). Automating or assisting the search process by offering a general applicable Natural Language (NL) interface, reduces the error-proneness and simplifies the query optimization. Speeding up the response time (Waltinger et al. 2013) and reducing the amount of time spent to adapt queries and to wait for data, leads to great benefits for the engineers and companies itself.

Due to the steady development of new key technologies like the Semantic Web (Berners-Lee et al. 2001) and standards like SPARQL or OWL, new approaches and promising ideas emerge to solve diagnosis and search problems in Alarm Management Systems.

In this paper, we focus on two important aspects with regard to the Semantic Web and its technologies like RDF, SPARQL and OWL. Firstly, we present an approach to realize the SPARQL query generation from natural language based on ontologies and OWL. Secondly, we evaluate the performance of virtual and native Triple Stores representations with respect to a RDBS, in order to verify its applicability under industrial conditions.

Related Work

Natural Language Processing (NLP) in combination with ontologies received a lot of attention recently. The ontology-based understanding of NL and the translation into SPARQL combines the research of several recent publications.

(Tran et al. 2007) propose an approach to ontology-based interpretation of keywords for semantic search. Keywords are mapped to concepts in the ontology, and the graph then is further explored to retrieve available relations between these concepts. These relations most likely are gaps in the query, which are not specified by the user. The user knows these gaps intuitively, e.g. searching for an author name and a book title, the obvious relation would be "authorOf" or the opposite "writtenBy". To avoid exponential search time or dead locks, they define an exploration width to limit the amount of visited nodes.

Another possibility to translate NL to SPARQL has been carried out with AutoSPARQL (Lehmann and Bühmann 2011). They use active supervised machine learning to generate SPARQL queries based on positive examples. Starting with a question and following up with answering, estimating whether an entity belongs to the result set, will continuously improve the algorithm and the results. This approach leads to very good results. One problem is the portability to a different Knowledge Base (KB). Depending on the size, the effort of learning the positive and negative examples will increase drastically with the size of the KB.

SPARK (Zhou et al. 2007) is a prototype for processing NL keyword requests. The output is a ranked list of SPARQL queries. The major process steps are: term mapping, query graph construction and query ranking. The query ranking is a probabilistic model based on the Bayesian Theorem (Smets 1993). The authors claim "encouraging" translation results. The problem here is, that choosing an option out of the ranked query list requires expertise in SPARQL and the underlying KB.

QuestIO (Damljanovic, Tablan, and Bontcheva 2008) is a Question and Answering Interface (QAI) to query ontologies using unconstrained language-based queries. It requires no user training and the integration in other systems is simple. During processing, every single human understandable phrase is extracted from the ontology (classes, properties, instances, relations). Relations are ranked with a similarity score which takes into account the name, the position and distance in the ontology hierarchy. The advantages of this approach is the simple structure, which allows queries with random length and form as well as the slight effort for customization. A problem is the lack of basic NLP operations. No word stemming is used, which conflicts with the statement of "queries with random form", since synonyms or different grammatical forms of a word may lead to no or even wrong results.

FREyA (Damljanovic, Agatonovic, and Cunningham 2012) is a successor tool of QuestIO to interactively query linked data using natural language. In contrast to QuestIO, FREyA utilizes syntactical parsing combined with an ontology-based lookup to interpret a natural language question. If the intention behind the query remains unclear, the system involves the user with a QAI. The answers of the user are used to train the system and improve its performance and accuracy over time. Depending on the knowledge of the user, answering questions of a specialized domain model and vocabulary is difficult or impossible (Unger et al. 2012).

NLPReduce (Kaufmann, Bernstein, and Fischer 2007) is another "naive" approach to introduce a Natural Language Interface (NLI) to query a KB in the semantic web. The natural language processing is limited to stemming and synonym expansion. NLPReduce does not claim to be an intelligent algorithm. It maps domain-independent words and synonyms to their expressions used in the KB.

(Estival and others 2004) carry out research for ontology-based approaches to process NL. They state two options for the facilitation of NLP with the assistance of ontologies. Firstly, using an ontology to build a lexicon and defining terms (concepts and relations). Secondly, an ontology represents a KB in a formal language and provides further knowledge to conduct more complex language processing. Both insights are valuable with regard to this paper. Building up a lexicon and integrating additional knowledge, that is not represented in the base data, supports the formulation of search queries from NL input.

A lot of research has already been carried out for ontology-based translation of keywords or natural language, QAI to query ontologies and machine learning approaches to generate SPARQL queries. But the combination of domain specific knowledge in ontologies, lexicons and rules for a straight forward creation and extension of SPARQL queries is a new approach towards the automatic query generation in order to solve emerging diagnostic and search problems.

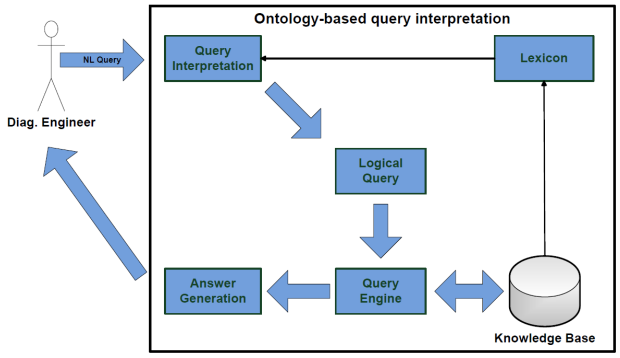

Figure 2: A general approach to ontology-based query interpretation. The NL query is interpreted and transformed into a logical query under consideration of a lexicon and an underlying KB. The query engine transforms the logical query into a query language, under consideration of the relations and the topology in the KB. Afterwards, the query is executed in a specified database. The last step, the answer generation filters and returns the requested data to the user.

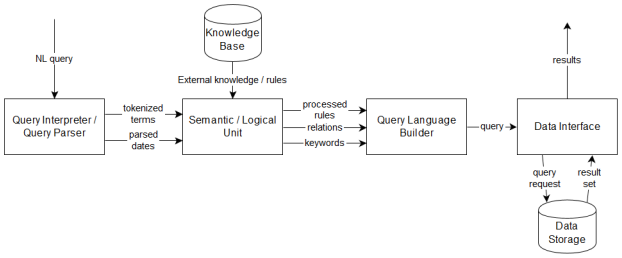

Figure 3: The process to interpret and understand NL queries. The NL input is parsed syntactically to extract meaningful terms and dates for the time reference. The tokens are checked against the KB to identify concepts, relations, individuals or rules from the ontology and to extract synonyms or other keywords from the lexicon; The result is a logical query representation which is mapped into a SPARQL query. The answer generation consists of filtering and returning the requested data from a Triple Store to the user.

Architecture proposal for ontology-based NLP

In this section, we propose an architecture to put ontology-based NLP into practice, in order to produce accurate and efficient SPARQL queries. The goal of this approach is the reduction of the time and money spent by engineers and other users of different domains, to search or request important data. A general approach to ontology-based NL interpretation is depicted in Figure 2. A more detailed approach is illustrated in Figure 3. The NL input is interpreted under consideration of domain specific concepts, individuals, relations and restrictions in the ontology, as well as additional knowledge e.g. keywords, synonyms or KPIs, which are stored in a Simple Knowledge Organization System (SKOS) lexicon (Miles and Bechhofer 2009). SKOS provides possibilities to interpret context sensitive synonyms and predefined keywords. Additionally SKOS supports existing standards like thesauri, which reduce the necessary creation effort in well-known domains.

After the interpretation or extraction of required information to parse a NL query successfully, the input is mapped to predefined SPARQL Inferencing Notation (SPIN) (Knublauch, Hendler, and Idehen 2009) rules which provide the skeleton for the required SPARQL query. Therefore reducing the complexity of the extraction of the query focus (e.g. the variables behind the *SELECT* clause) and the manual creation of complex query structures. Afterwards, the existing query is enriched with additional information from the NL query to specify and limit the required query results as depicted in Figure 4.

The required filter variable names like *appliance* or *timestamp* are extracted from the domain and range values of the object and data properties, belonging to the selected concept or individual. The same applies to the extension of an existing rule with additional triples.

1. The NL query input: Complete sentences or single keyword input as well as logical terms like *AND*, *OR* or *greater equal*, *equal* et cetera are possible.
2. The input is separated into syntactic markers and processable (meaningful) tokens. Syntactic markers are not considered at this level of NLP, and generally offer limited information. The processable tokens need at least one recognizable term which is matched against the ontology or the lexicon to retrieve a valid rule.
3. Token stemming (reducing tokens to their root form; treating words with the same root stem as synonyms. Therefore providing the possibility for a simple comparison of two words with different grammatical occurrence, which increases the chances of positive matches in the ontology), extract NL dates and remove punctuation.
4. The preprocessed or stemmed tokens are separated from the date indicator. The original terms are still stored in a map in the case that individuals like *appliance1* are accidentally stemmed and can not be recognized in the ontology later.
5. The remaining tokens are checked against the ontology. Concepts, individuals, relations and properties are matched: the appliance labels are extracted. In case of a concept the underlaying individuals are extracted, since concepts only exist in the TBox knowledge and are unavailable at the data level.
6. Tokens without corresponding entries in the ontology are checked against the lexicon. These are either KPIs or keywords, which do not belong into the ontology.
7. If the token is not contained in the lexicon, the SKOS information is requested to identify possible synonyms. These synonyms are rechecked against the lexicon again: *disposability* is a synonym for *availability*.
8. After identifying the tokens in the ontology or lexicon, the corresponding SPIN rule is extracted. A similarity score based on the Levenshtein distance (Yujian and Bo 2007)

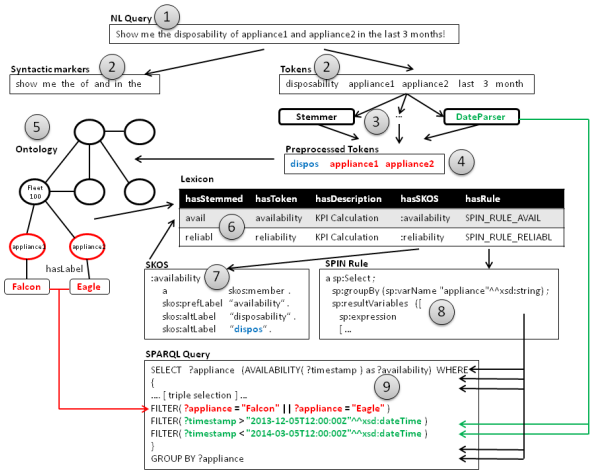

Figure 4: Ontology-based translation of NL queries using SPIN rules and SKOS lexicon

is calculated against predefined recognizable tokens in the spin rule, to select the best fitting or the most specific rule.

9. The last step merges the defined SPARQL body from the SPIN rule with the additional information from the ontology and the extracted dates.

The usage of SKOS and SPIN reduces the complexity and error-proneness of the manually generated queries and offers powerful extension possibilities. The considered queries have low complexity and specifying or reducing the amount of results of a *SUBSELECT* is not possible yet. This requires a higher level of query analysis which is not part of this paper. Nevertheless, complex SPIN rules using *SUBSELECT* and other recursive structures are possible and can be limited at the top level with the current approach.

Query Evaluation

The evaluation is split into two parts. The first part evaluates several data storage possibilities in order to select the best storage type for the requested data. The second part evaluates the generated queries based on a query catalog against manually optimized queries, in order to retrieve the performance loss of automatic generated queries. For evaluation and demonstration purposes, a simple Java prototype based on Apache Jena[1] was implemented. Jena is an open source framework for the Semantic Web technology. It contains an internal Triple Store, an ontology API and of course SPARQL support.

Query Catalog

The Query Catalog is a collection of predefined NL queries, which have to be transformed into SPARQL. A small excerpt of queries is listed below. Deviations from these queries, for example with respect to a selected number of appliances or specified keywords and individuals are possible.

[1]http://jena.apache.org/index.html

Statistic Queries

1. Show the TOP 10 events occurring within time interval T before the shutdown of an appliance
2. Show the TOP 5 event categories within an appliance

Event Occurrence Queries

3. Show all reoccurring error events and the number of occurrences X within time interval T
4. Show the first occurrence of an event X within time interval T

Turbine Background Queries

5. Show all appliances with power output $> X$ Mega Watt
6. Show all appliances with availability $< Y$ AND power output $> X$ Mega Watt

KPI Queries

7. Show the availability distribution of all appliances in fleet XYZ
8. Show the reliability of appliance A and appliance B

The queries above are general templates for the kind of NL queries that occur and need to be processed. In other words, these queries have to be translated into SPARQL and are the basis for the following evaluations.

Data Storage Evaluation

Triple Stores and SPARQL are built to handle undetermined and varying or growing data representations. In contrast, RDBS and SQL require a well designed and complete database layout. Varying or additional relations, that require subsequent changes to the database layout are costly. Furthermore, SPARQL naturally interacts better with ontologies than SQL, since the graph based topology of ontologies is similar to the data representation of many Triple Stores (Kumar, Kumar, and Kumar 2011).

Although Triple Stores are a better fit than RDBMS, the latter are wider spread and usually contain more data. Virtual RDF graphs like D2RQ[2] translate SPARQL queries into SQL queries and return results from a RDBMS in RDF format. The main reason for this evaluation is a performance and scalability test for D2RQ under industrial conditions.

Query efficiency and performance evaluation

In this evaluation, the proposed queries from the query catalog are passed on to the implemented prototype and compared to manually written and optimized queries. This emphasizes the performance loss that occurs during the ontology, lexicon and rule lookup as well as the query analysis and extension.

Results

Data store comparison

The results for the data store comparison are shown in Table 1 for a 40 MB data set and in Table 2 for a 400 MB data set. The experiments were conducted on an Intel Q6600 with 4x2400GHz and 16 GB RAM.

[2]http://d2rq.org/

QueryID	PostgreSQL	Jena TDB	Virtuoso	Sesame	D2RQ
1	36 ms	107 ms	98 ms	86 ms	7201 ms
2	313 ms	263 ms	238 ms	214 ms	15628 ms
3	54 ms	67 ms	69 ms	58 ms	9311 ms
4	37 ms	51 ms	52 ms	45 ms	9272 ms
5	14 ms	23 ms	22 ms	22 ms	207 ms
6	592 ms	814 ms	799 ms	715 ms	32981 ms
7	601 ms	876 ms	814 ms	718 ms	34409 ms
8	579 ms	798 ms	803 ms	687 ms	34012 ms

Table 1: Query response time of the RDBMS PostgreSQL, various native Triple Stores and the virtual Triple Store D2RQ with a 40 MB data set

QueryID	PostgreSQL	Jena TDB	Virtuoso	Sesame	D2RQ
1	528 ms	4732 ms	4960 ms	4389 ms	8m43s
2	4290 ms	8561 ms	8801 ms	7837 ms	28m11s
3	745 ms	6015 ms	5832 ms	5628 ms	10m29s
4	253 ms	2799 ms	2783 ms	2506 ms	16788 ms
5	14 ms	115 ms	127 ms	98 ms	207 ms
6	8023 ms	10512 ms	11172 ms	9848 ms	33m25s
7	7707 ms	10350 ms	10709 ms	10026 ms	32m56s
8	7943 ms	10378 ms	10931 ms	10105 ms	31m59s

Table 2: Query response time of the RDBMS PostgreSQL, various native Triple Stores and the virtual Triple Store D2RQ with a 400 MB data set

The results show, that the virtual Triple Store solution D2RQ is not competitive yet. The majority of data is still stored in a RDBMS and the ability to query this RDBMS via SPARQL would have spared the effort to convert and duplicate the data into a Triple Store. But the results are obvious concerning D2RQ. Whenever a full data search (e.g. no limitations) was performed, D2RQ had a relatively high but stable query response. The D2RQ query response time exceeded the 30 minute mark regularly on the data set 2 (400MB), which is highly impractical. The processed data easily reaches several GB of data, which decreases the query response time of D2RQ further. On the contrary, the native Triple Stores show stable and with respect to time acceptable results. Sesame performs best with regard to the chosen queries. Followed by Virtuoso and Jena TDB, which alternate the second place depending on the query.

Overall, the virtual Triple Stores are not yet applicable under industrial perspectives. The query response time is too high to present a scalable and efficient solution with D2RQ. Native Triple Stores are mature enough to compete with a RDBMS and still scale well on larger data sets. Moreover, RDF and Triple Stores are perfectly suited for the combination with ontologies and OWL respectively.

Query efficiency and accuracy

The query efficiency or the query response time effected by the number of results, depends strongly on the available SPIN rules. These rules provide a fixed skeleton for a certain type of SPARQL query. The query efficiency relates to the quality of the SPARQL query that is measured in the amount of requested triples, used aggregate functions and more complex constructs like SUBSELECT, OPTIONAL or UNION. The query accuracy or the ratio of expected triples and returned triples depends on the ability of the prototype to understand the NL input. Choosing the corresponding SPIN rule and identifying concepts, relations or individuals correctly are crucial in order to add the sufficient amount of external information from the ontology with functions like FILTER (Unger et al. 2012).

The following examples illustrate the difference between a manually written and automatically generated query. Starting of with the NL query as basis for the manual and automatic query:

Show me the top 10 events from appliance1?

Manual query:

```
PREFIX data: <http://www.example.org/data#>

SELECT ((COUNT(?eventtext)) as ?count) ?eventtext
WHERE
{
  ?s a data:Message .
  ?s data:eventtext ?eventtext .
  ?s data:appliance <http://www.example.org/data#appliance1>
}
GROUP BY ?eventtext
LIMIT 10
```

Automatic query:

```
PREFIX data: <http://www.example.org/data#>

SELECT ((COUNT(?eventtext)) as ?count) ?eventtext
WHERE
{
  ?s a data:Message .
  ?s data:eventtext ?eventtext .
  ?s data:appliance ?appliance .
  FILTER(?appliance = <http://www.example.org/data#appliance1>)
}
GROUP BY ?eventtext
LIMIT 10
```

Comparing both queries above, a slight difference is observed. While the manual query refers directly to the URI of *appliance1* in the triple selection, the automatic query uses the *FILTER* function provided by SPARQL to reduce the result set to the requested information. The manual query requests only triples that contain data of *appliance1*. In contrary, the automatic query requests all triples of type *data:Message*. In other words, triples that contain data about *appliance2*, *appliance3* et cetera. The larger result set is filtered afterwards. Replacing *FILTER* functions in this way is a common optimization process (Bernstein, Kiefer, and Stocker 2007) and improves the query response time.

In this case, the manual written query is well optimized to the specific NL query, and therefore performs better than the automatic query, due to a smaller returned and processed result set.

The automatic query is created using ontologies and generalized rules. Adding variable data like the focus on an appliance is simplest achieved via *FILTER*. Manual optimization must always reach at least the response time of the automatic generation, because there is always a trade-off between generalization and performance.

The second example illustrates the advancing complexity of the optimization for the automatic query generation. The example query is almost equal to the first one, with the exception that another appliance is included.

Show me the top 10 events from appliance1 and appliance2?

Manual query:

```
PREFIX data: <http://www.example.org/data#>

SELECT ((COUNT(?eventtext)) as ?count) ?eventtext
WHERE
{
 {
  ?s a data:Message .
  ?s data:eventtext ?eventtext .
  ?s data:appliance <http://www.example.org/data#appliance1>
 } UNION
 {
  ?s a data:Message .
  ?s data:eventtext ?eventtext .
  ?s data:appliance <http://www.example.org/data#appliance2>
 }
}
GROUP BY ?eventtext
LIMIT 10
```

Automatic query:

```
PREFIX data: <http://www.example.org/data#>

SELECT ((COUNT(?eventtext)) as ?count) ?eventtext
WHERE
{
  ?s a data:Message .
  ?s data:eventtext ?eventtext .
  ?s data:appliance ?appliance .
  FILTER(?appliance = <http://www.example.org/data#appliance1> ||
         ?appliance = <http://www.example.org/data#appliance2>)
}
GROUP BY ?eventtext
LIMIT 10
```

Based on the assumption that there exist more than two appliances in the data, the manual query is better optimized again. The *UNION* exactly merges both result sets from *appliance1* and *appliance2*. The automatic query simply adds another *FILTER* condition. These two examples illustrate the complexity of the (manual) query optimization. Depending on the complexity of the NL queries, the ideal solution is hard to find and prone to errors. The automatic query is generated by given rules and fills in conditions to reduce the dataset. This results in much simpler queries, which return the exact same results. The only disadvantage is the loss of performance.

Finally, the provided possibilities in the Query Catalog combined with the proposed variations achieve a promising accuracy on the data set. Meaning the automatic generated SPARQL queries return the same results as the manually defined SPARQL queries.

As stated above, the efficiency suffers under the trade-off between generalization and performance. But the fact, that some SPARQL engines are able to rewrite and optimize certain queries, benefit the automatically generated queries (Schmidt, Meier, and Lausen 2010). The *FILTER* function in the first example could be transformed into the more efficient solution from the manual written query and therefore reduce the performance loss. Of course the targeted data set must be large enough, that the query transformation does not consume more time than the actual query response time.

Query Response Time

The final evaluation aims at the query response time of the prototype in contrast to a manually defined and optimized query. This results in the consumed computation time the prototype requires to parse the NL input, translate it into a SPARQL query and the probable performance loss due to less efficient queries.

Query ID	Manual query	Automatic query	Δ
1	4732 ms	4291 ms	-441 ms
2	8561 ms	7923 ms	-638 ms
3	6015 ms	5478 ms	-537 ms
4	2799 ms	2376 ms	-423 ms
5	115 ms	197 ms	82 ms
6	10512 ms	11388 ms	876 ms
7	10350 ms	10965 ms	615 ms
8	10378 ms	11115	737 ms

Table 3: Query Response Time of manually optimized SPARQL queries and generated queries based on the presented approach; **Δ** denotes the time difference between the manually optimized queries and the generated queries.

The comparison in Table 3 is created based on the 400 MB data set. The interesting fact, that the automatic query outperforms the optimized query in row 1 to 4, is attributed to the Lucene index, which belongs to the Apache Jena framework and speeds up the performance when querying unstructured event data.

The queries 6 to 8 show, that the computation time of the prototype to translate the NL input to a SPARQL query is less than one second. This difference is explained due to the better performance of the optimized query, as well as the lost computation time, when accessing the KB for external keywords and SKOS data. These are not stored in the memory yet and have to be requested and loaded every time. The ontology data on the other hand is kept in memory. This is illustrated by query 5, whereby only ontology data is requested and the difference is less than 100 ms.

All in all, the prototype and the automatically generated queries perform well on the given data sets. The loss of performance is kept within limits. Due to technologies like Lucene[3] which is a high-performance, full-featured text search engine library that speeds up several types of queries, the generated queries almost match up to the manually written and well optimized queries.

Conclusion

This paper showed that the ontology-based translation of NL queries into SPARQL is a legitimate approach to provide intuitive and general applicable semantic search functionality to a wide range of domains and users. The demand for query formulation or query adaptation, the associated susceptibility to errors and the large time expenses are eliminated by the automated query generation. This results in a reduction of the time consuming and expensive support of (external) IT Experts, who have to create and adapt queries to changed

[3] http://lucene.apache.org/core/

or new requirements. Due to the intuitive NL input, the required knowledge and training period is decreased, while the overall usability is increased. The freed-up time can be invested to assess the quality of the data or to conduct a more detailed analysis in order to retrieve better diagnostic results. Three essential steps are taken into account to concern the problem of ontology-based translation of NL sentences.

- Concepts, individuals, relations, axioms and restrictions contained in the ontology.
- Synonym handling and further information encoded in the SKOS data.
- SPIN rules containing the requested SPARQL skeleton, which is enhanced with additional information from the ontology

The interaction of these steps lead to a portable and generic solution, which can be adapted to other domains by exchanging the data and knowledge foundation. In other words, switching the ontology, the SKOS data and the SPIN rules is sufficient to adapt the software prototype to a new domain. The evaluation of the D2RQ showed that the performance on large scale data representations is not yet applicable under industrial conditions. While the approach offers the combination of the advantages from the both data tabular and triple / graph data representations, the algorithms and implementations require improvements to be considered in the industry.

References

Alcook, P. 2009. R. crompton (2008), class and stratification, 3rd edition. cambridge. *Journal of Social Policy* 38.

Berners-Lee, T.; Hendler, J.; Lassila, O.; et al. 2001. The semantic web. *Scientific american* 284(5):28–37.

Bernstein, A.; Kiefer, C.; and Stocker, M. 2007. *OptARQ: A SPARQL optimization approach based on triple pattern selectivity estimation*. Citeseer.

Damljanovic, D.; Agatonovic, M.; and Cunningham, H. 2012. Freya: An interactive way of querying linked data using natural language. In *The Semantic Web: ESWC 2011 Workshops*, 125–138. Springer.

Damljanovic, D.; Tablan, V.; and Bontcheva, K. 2008. A text-based query interface to OWL ontologies. In *LREC*.

Estival, D., et al. 2004. Towards ontology-based natural language processing. In *Proceeedings of the Workshop on NLP and XML (NLPXML-2004): RDF/RDFS and OWL in Language Technology*, 59–66. Association for Computational Linguistics.

Feldman, S., and Sherman, C. 2001. The high cost of not finding information. *IDC Whitepaper*.

Giese, M.; Calvanese, D.; Haase, P.; Horrocks, I.; Ioannidis, Y.; Kllapi, H.; Koubarakis, M.; Lenzerini, M.; Mller, R.; Rodriguez-Muro, M.; zcep, .; Rosati, R.; Schlatte, R.; Schmidt, M.; Soylu, A.; and Waaler, A. 2013. Scalable end-user access to big data. In Akerkar, R., ed., *Big Data Computing*. CRC Press.

Kaufmann, E.; Bernstein, A.; and Fischer, L. 2007. NLP-Reduce: A naive but Domain-independent Natural Language Interface for Querying Ontologies. ESWC.

Knublauch, H.; Hendler, J.; and Idehen, K. 2009. SPIN–SPARQL inferencing notation.

Kumar, A. P.; Kumar, A.; and Kumar, V. N. 2011. A comprehensive comparative study of SPARQL and SQL. *International Journal of Computer Science and Information Technologies* 2(4):1706–1710.

Lehmann, J., and Bühmann, L. 2011. Autosparql: Let users query your knowledge base. In *The Semantic Web: Research and Applications*. Springer. 63–79.

Miles, A., and Bechhofer, S. 2009. SKOS simple knowledge organization system reference. Technical report, W3C.

Nugent, A.; Halper, F.; Kaufman, M.; et al. 2013. *Big Data For Dummies*. John Wiley & Sons.

Schmidt, M.; Meier, M.; and Lausen, G. 2010. Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory*, 4–33. ACM.

Smets, P. 1993. Belief functions: the disjunctive rule of combination and the generalized bayesian theorem. *International Journal of approximate reasoning* 9(1):1–35.

Tran, T.; Cimiano, P.; Rudolph, S.; and Studer, R. 2007. Ontology-based interpretation of keywords for semantic search. In *The Semantic Web*. Springer. 523–536.

Unger, C.; Bühmann, L.; Lehmann, J.; Ngonga Ngomo, A.-C.; Gerber, D.; and Cimiano, P. 2012. Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web*, 639–648. ACM.

Waltinger, U.; Tecuci, D.; Olteanu, M.; Mocanu, V.; and Sullivan, S. 2013. Usi answers: Natural language question answering over (semi-) structured industry data. In Muñoz-Avila, H., and Stracuzzi, D. J., eds., *Proceedings of the Twenty-Fifth Innovative Applications of Artificial Intelligence Conference, IAAI 2013, July 14-18, 2013, Bellevue, Washington, USA*.

Yujian, L., and Bo, L. 2007. A normalized levenshtein distance metric. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29(6):1091–1095.

Zhou, Q.; Wang, C.; Xiong, M.; Wang, H.; and Yu, Y. 2007. SPARK: adapting keyword query to semantic search. In *The Semantic Web*. Springer. 694–707.