Reasoning over SPARQL

Sam Coppens, Miel Vander Sande, Ruben Verborgh, Erik Mannens, Rik Van de Walle

{sam.coppens,miel.vandersande,ruben.verborgh,erik.mannens,rik.vandewalle}@ugent.be

Ghent University - iMinds
Department of Electronics and Information Systems, Multimedia Lab
Gaston Crommenlaan 8 bus 201
B-9050 Ledeberg-Ghent, Belgium

ABSTRACT

Until now, the SPARQL query language was restricted to simple entailment. Now SPARQL is being extended with more expressive entailment regimes. This allows to query over inferred, implicit knowledge. However, in this case the SPARQL endpoint provider decides which inference rules are used for its entailment regimes. In this paper, we propose an extension to the SPARQL query language to support remote reasoning, in which the data consumer can define the inference rules. It will supplement the supported entailment regimes of the SPARQL endpoint provider with an additional reasoning step using the inference rules defined by the data consumer. At the same time, this solution offers possibilities to solve interoperability issues when querying remote SPARQL endpoints, which can support federated querying frameworks. These frameworks can then be extended to provide distributed, remote reasoning.

1. INTRODUCTION

Reasoning is one of main strengths of the Semantic Web. It infers logical consequences from graph structured RDF data based on inference rules. These inference rules are included in ontology languages (e.g., OWL2 [9]) or rule languages (e.g., N3 rules [2]). In order to produce inferred triples, reasoners require, of course, access over the data. In a distributed environment as the Linked Open Data [3] cloud, this raises several concerns: First, all the data needs to be collected from different data sources and stored locally, before one can reason over the data. Second, when dealing with large distributed datasets, as is common in modern web applications, this centralisation of triples becomes problematic, since processor and memory consumption increase with the amount of triples. Third, in this paper, we argue that reasoning over large distributed datasets can be done more effectively if performed in a distributed, atomic way.

On the Semantic Web, RDF data can be accessed through a SPARQL endpoint [5]. Typically, triples can be retrieved by executing a SPARQL query. This process shows some strong similarities with reasoning, which are the following:

- both require the presence of the whole set of triples.
- both benefit from setting conditions and performing filters, to make the operation as precise as possible.
- both return a set of triples. In case of querying, this is the selected subset. In case of reasoning, these are the inferred triples.

We exploit the fact that the SPARQL infrastructure is already well deployed by extending the endpoint's functionality to support reasoning operations. With a single query, inference rules are sent to the endpoint and the inferred triples are returned.

In this paper, we introduce a novel approach to deal with the issues of reasoning over centralised triples, by allowing custom inference rules being executed over data stored on remote systems. We build on existing Semantic Web technology, by adding an extra layer to the SPARQL query language and reusing its endpoint infrastructure. We do not force SPARQL endpoints to support heavy reasoning tasks, but we leave open the possibility. They can decide to not support any reasoning. In this case, the reasoning could be performed by the SPARQL client. But by integrating this reasoning mechanism in the SPARQL protocol and syntax, SPARQL clients can be developed supporting the reasoning. E.g., a SPARQL client that relies on query rewriting for the inference rules supported by the OWL QL query language, and relying on client-side, OWL reasoning for the other inference rules. This kind of SPARQL client wouldn't even need any adaptation to the existing SPARQL endpoints. Another example are SPARQL clients with a MapReduce-infrastructure to support the reasoning. In this paper, we only focus on the extension for the SPARQL syntax and protocol. The SPARQL endpoints are free to decide what reasoning support they give, SPARQL clients are free to decide how to schedule (client vs. server) the reasoning.

This paper is structured as follows. We start by discussing some related work (section 2). Next, we motivate the benefits of remote reasoning (section 3) and discuss how we reused SPARQL to support it (section 4). Then, we describe two future use cases as a proof of concept (section 6). Finally, we conclude with some future work (section 7) and add a final conclusion (section 8).

2. RELATED WORK

Currently, the W3C SPARQL working group has proposed a recommendation for supporting entailment regimes [7] in SPARQL. The SPARQL 1.1 Query specification defines the evaluation of basic graph pattern by means of subgraph matching. This form of basic graph pattern matching is also called simple entailment. The proposed support of the entailment regimes will allow for retrieving solutions that implicitly follow from the queries graph. The proposed recommendation regimes are: RDF entailment, RDFS entailment, D-entailment, OWL2 RDF-based semantics entailment, OWL2 direct semantics entailment and OWL QL core entailment. This way, a data provider can adapt its basic graph pattern matching algorithm to support one of the entailment regimes to support querying over entailed triples. This recommendation will allow data providers to publish SPARQL endpoints which support one of the proposed entailment regimes. In this solution, the data providers offer the ontology or rule set used for reasoning and the reasoning actually happens on the level of basic graph pattern matching. In our solution, the end-user defines the ontology or rule set used to reason over the queries graph. The reasoning in our solution happens after the basic graph pattern matching. Thus, we have a post-reasoning step, which supplements the pre-reasoning step offered by the entailment regime.

Related to our solution is stream reasoning. Event processing is concerned in timely detecting compound events in streams. Event processing is already capable of doing runtime analysis of the event streams. Stream reasoning on the other hand will allow event processors to combine background knowledge to entail extra knowledge. Event Processing SPARQL (EP-SPARQL, [1]) is a new language for complex event and stream reasoning. Next to the new language for querying streams of events, they also provide an execution model which derives information from streamed RDF events in real-time. The framework is extended based on event-driven backward chaining rules. These are logic rules that can thus be mixed with other background knowledge. Further investigation is needed here to support more expressive formalisms for stream reasoning, such as OWL and its different profiles.

Distributed reasoning is also closely related to our framework. In such frameworks, reasoning is happening with multiple ontologies interrelated with semantic mappings on distributed data. LarKC (Large Knowledge Collider) and DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontology, [11]) are such distributed reasoning frameworks. LarKC performs massive, distributed, and necessarily incomplete reasoning over web-scale knowledge sources. Massive inference is achieved by distributing problems across heterogeneous computing resources. LarKC is based on a pluggable architecture in which it is possible to exploit techniques and heuristics from diverse areas such as databases, machine learning, cognitive science, Semantic Web, and others. In DRAGO, reasoning is the result of a combination of semantic mappings of local reasoning chunks performed in a single OWL ontology. It provides reasoning services for multiple OWL ontologies, interconnected via C-OWL mappings. Future work for DRAGO involves extending the DRAGO framework to cope with distributed T-boxes and to support more expressive ontology mappings.

3. REMOTE REASONING

Remote Reasoning resides in the classic client-server architecture. It is a service provided by an external server, allowing users to reason over its accessible data with a supplied rule set or OWL ontology. When the reasoning process is complete, the server returns the inferred triples. This is demonstrated in fig. 1.

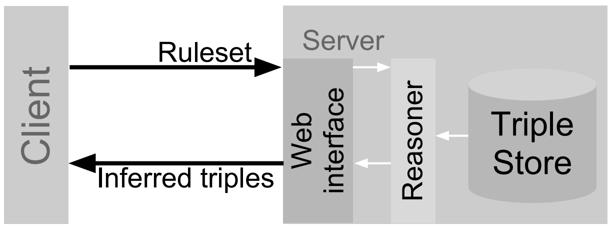

Figure 1: Client-server architecture for remote reasoning

The benefits are three-fold. First, users can benefit from delegating some of the effort of reasoning to the server. Those reasoning tasks performed within the OWL QL space, can easily be performed by the server by query rewriting. This does not only result in faster reasoning times, but is also cost-efficient. Actually, in the case of query rewriting, the reasoning is done by query rewriting, typically at the client, followed by basic graph pattern matching at the server. Second, in many cases, a client only requires the reasoning results, i.e. the inferred triples. Therefore, eliminating the harvesting and storing of data locally, drastically reduces overhead in terms of time and storage consumption. Also, data is not duplicated and synchronizing versions is therefore avoided. Third, the used inference rules are flexible, dynamic and defined by the client, not the server. At the moment SPARQL endpoints can already support some entailment regimes, but here the SPARQL endpoint providers decides the inference rules used for reasoning. In many cases, the data consumer wants to define the inference rules to support its application.

In order to integrate remote reasoning on the Web, there are a few requirements. First, reasoning happens within the *constraints* of the server. Second, a *syntax* and *communication protocol* need to be defined to assure interoperability. In this paper we focus on this last requirement, which are discussed in the next Section.

4. EXTENDING SPARQL

In general, a reasoning cycle can be divided into five steps, as shown by LarKC [14]:

1. Identification
2. Selection
3. Transformation
4. Reasoning
5. Decision

SPARQL already covers the first three steps (CONSTRUCT queries can be considered as transformations). By extending the SPARQL query language, the fourth step can also be covered by SPARQL, allowing for remote reasoning.

Listing 1: A reason query for executing n3 rules declared in an external file over all triples

```
REASON <http://test.com/rules.n3>
OVER {
  ?s ?p ?o
} WHERE {
  ?s ?p ?o
}
```

4.1 The REASON query

Reasoning can be performed by executing a SPARQL query. Therefore, we introduce a new query form REASON equivalent to SELECT, CONSTRUCT, ASK, and DESCRIBE. The syntax consists of three main parts (as illustrated in listing 1):

- the REASON keyword
- a *rule set* in an ontology or declarative language. This can either be a URL to a rule file, supplied between <>, or inline N3 rules between {}
- the OVER and WHERE keyword combination which defines the triples to reason over. It is equivalent to the CONSTRUCT keyword in combination with its WHERE clause. Its content is placed between {}.

A REASON query returns a valid RDF graph, equivalent to the CONSTRUCT query. This graph includes only the inferred triples.

We chose for the new SPARQL query form because:

- The structure of the REASON query form supports a selection step, as discussed in the context of LarKC [14] and section 4. This way, we can do remote reasoning on a snapshot of the data. By doing this, the server keeps control over the memory consumption a certain reasoning task might require. The SPARQL endpoint provider can, e.g., require the selection step trims down the graph to reason over 10.000 triples or to stop reasoning the moment it has inferred 10.000 triples. Although the results can be incomplete.
- SPARQL allows for subqueries. Subqueries are a way to embed SPARQL queries within other queries, normally to achieve results which cannot otherwise be achieved, such as limiting the number of results from some subexpression within the query. Thus, implementing the remote reasoning over SPARQL via a new query form allows to query the returned entailed triples, as will be shown in the next section 4.1.1.
- The combination of the REASON query form with a SERVICE extension of SPARQL 1.1 provides a way to balance workload with federated querying, as explained in section 4.1.2.

4.1.1 Precise reasoning with nested queries

The SPARQL 1.1 specification allows subqueries and nesting of queries. This can be fully exploited to allow queries on the entailed triples. This enables fine-grained reasoning, which enables optimization of results and execution time. Examples of nested queries are given in listing 2 and listing 3.

Listing 2: Only the inferred child relationship of Jenna is selected.

```
SELECT ?child
WHERE {
  :Jenna :child ?child .
  {
    REASON {
      { ?x :parent ?y } => { ?y :child ?x } .
    }
    OVER {
       ?s :parent ?o .
    }
    WHERE {
    ?s a :Person; :parent ?o .
    }
  }
}
```

Listing 3: The worksWith relationship is inferred only for persons working at UGhent

```
REASON {
  { ?x :knows ?y } => { ?y :knows ?x } .
}
OVER {?s :knows ?o}
WHERE
{
  CONSTRUCT {?s :workedWith ?o}
  WHERE {
     ?s :worksAt :UGhent.
     ?s :participatedInProject ?project.
     ?o :worksAt :UGhent.
     ?o :participatedInProject ?project.
  }
}
```

4.1.2 Balancing workload with Federated Querying

Federated querying is another possibility in SPARQL. It allows triples to be fetched from another endpoint, and include them in your query using the SERVICE element. When we include this into a REASON query, we can reason with one endpoint, over data from another. This is very powerful, because it allows a high performance server to reason over remote data with a single query. Therefore, the biggest workload can be performed on the machine best suited for it. An example query is shown in listing 4.

4.2 Ontology classification

The inference rules used for reasoning can come from OWL ontologies or declarative rules. Classification of the used inference rules will allow to select the appropriate reasoner for doing the inferencing. Today, highly optimised reasoners such as EYE[4], Pellet [12], FaCT++ [13], RacerPro [8] and HermiT [6] are able to classify many ontologies used in applications. The optimisations employed by these reasoners aim not only to improve performance on individual subsumption tests, but also to reduce the number of tests performed when classifying a given ontology. Such an ontology classification can also be a means to enforce a constraint of the server. The server can decide to only support OWL QL inference rules, because in this case the query can be rewritten and SPARQL can be exploited to the fullest, without actual generating inferred triples.

Listing 4: Example query showing how the data is first fetched from two endpoints and then used for reasoning

```
REASON {
  { ?x foaf:knows ?y } => { ?y foaf:knows ?x } .
}
OVER {
  :Jenna foaf:knows ?person .
         }
WHERE{
  {
  SERVICE <http://example.org/sparql> {
    :Jenna foaf:knows ?person .
  } } UNION {
  SERVICE <http://example2.org/sparql> {
    :Jenna foaf:knows ?person .
  } }
}
```

Listing 5: Without the presence of foaf:knows (not selected in the over clause) no triples will be inferred.

```
REASON {
  { ?x foaf:knows ?y }
    => { ?y foaf:knows ?x } .
}
OVER {
  :Jenna a foaf:Person .
}
WHERE {
  :Jenna a Foaf:Person; ?p ?o
}
```

4.3 Handling triple selection validation for reasoning

When executing a REASON query, the triples to reason over are selected in the OVER clause. This principle holds a potential pitfall. When the selection lacks the necessary triples for the given inference rule set, the result will be empty. An example is given in listing 5. Although this can not be considered an error, it needs to be discussed in the query specification. In the end, the user is made responsible for the triple selection in the OVER clause. When a result set turns up empty, it can two reasons (i) the inference rules lead to no inferred triples, or (ii) the selection of the triples to reason over was incomplete. Automatic validation is also possible using the aforementioned reasoners, but in combination with the entailment regimes, this becomes hard. A query can at first sight lead to no results and invalidate, but can rely on the supported entailment regime of the SPARQL endpoint provider to provide the results.

5. A SPARQL REASONING ENDPOINT

For the proof-of-concept, we have extended Apache Jena ARQ[1] query engine to support remote reasoning. This engine is very popular and is often used for implementing SPARQL endpoints. By extending the ARQ library to support remote reasoning, existing SPARQL endpoints can be extended easily to support this remote reasoning. As explained in section 4, we introduced a new SPARQL query form: the REASON query form. In our first prototype, this new SPARQL query form supports N3 rules. This way, we are able to offer already reasoning on top of SPARQL. For the future, special attention will go to OWL QL inference rules, because these can be rewritten. This way, the server impact can remain minimal. OWL QL seems to be the fit for empowering SPARQL with reasoning capabilities.

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

Listing 6: A query overcoming interoperability issues

```
SELECT ?name
WHERE {
  ?s a :Artist .
  {
    REASON {
      { ?x :role 'artist' } => { ?x a :Artist} .
    }
    OVER {
      ?s :role ?o
    }
    WHERE {
      ?s a foaf:Person; :role ?o
  }
}
```

6. USE CASES

In this section, we discuss two possible use cases for remote reasoning: ontology interoperability and distributed reasoning. In both cases, the remote reasoning on top of SPARQL offers some possibilities, discussed in the next sections.

6.1 Ontology interoperability

Interoperability between different ontologies can be simplified by supplying mapping rules at query time similar to the approach followed by [10]. This allows a user to use a custom vocabulary in a SPARQL query that differs from the one used in the endpoint. This is typically a problem for query federation frameworks, where different parts of a query are evaluated by different SPARQL endpoints. These SPARQL endpoints often use different vocabularies to describe their data. By including mapping rules, interoperability issues are avoided when querying. An example of a federated query, supporting ontology interoperability is shown in listing 6.

6.2 Distributed reasoning

Many federated query frameworks rely on SPARQL to federate sub queries (parts of the incoming query) to the appropriate SPARQL endpoints and to aggregate all the results. These frameworks can already benefit from remote reasoning to solve interoperability issues between the contacted SPARQL endpoints. These federated querying frameworks could be extended with this SPARQL extension to become federated reasoning frameworks. The example shown in listing 7 shows how this could work.

Listing 7: A subset from the reason results can be extracted using sparql features

```
SELECT ?artist
WHERE {
  ?artist a :Artist .
  { REASON {
      {?y dbpprop:artist ?x.}
      =>
      {?x a :Artist} .
    }
   OVER {
      ?artwork dbpprop:artist ?person .
    }
   WHERE {
      SERVICE <http://example.org/sparql> {
      ?person a foaf:Person.
      ?artwork dbpprop:artist ?person .
    }
  }
}
```

7. FUTURE WORK

Future work will first focus on the following things:

- We want to add support for other rule languages besides N3. Special focus will go to inference rules belonging to the OWL QL profile.
- For the OWL QL inference rules, reasoning can be replaced by query rewriting. These conversions will be investigated for integration into our solution.
- For other inference rules, dedicated reasoners can be selected. Here, the classification comes into play. Dedicated reasoners perform much faster then general purpose reasoners. Thus a good classification can increase performance by selecting dedicated reasoners or a combination of dedicated reasoners.

A second objective for the future will be the integration of remote reasoning via SPARQL into federated querying frameworks to build a distributed reasoner, as shown in section 6.

8. CONCLUSION

Distributed reasoning is becoming more important nowadays, because of the distributed characteristic of Linked Open Data. In this paper, we have proposed an extension to the SPARQL query language to support remote reasoning. This extension will support two use cases: ontology interoperability, e.g., when federating queries, and distributed reasoning. In the future, reasoning on distributed data sources will become more important and this extension to SPARQL can be a building block to facilitate distributed reasoning. By extending SPARQL, the selection of data to reason over is incorporated into the remote reasoning framework we propose. This is a very important feature to restrict the amount of data to reason over, because it allows to reason on certain snapshots of distributed data, e.g., versions.

9. ACKNOWLEDGEMENTS

The research activities that have been described in this paper were funded by Ghent University, iMinds, the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), and the European Union.

10. REFERENCES

[1] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 635–644, New York, NY, USA, 2011. ACM.

[2] Berners-Lee, T. and Connolly, D. Notation3, 2006. Available at `http://www.w3.org/DesignIssues/Notation3`.

[3] Bizer, C. and Heath, T. and Idehen, K. and Berners-Lee, T. Linked Data on the Web. In *Proceedings of the 17th International World Wide Web Conference - LDOW Workshop*, pages 1265–1266, Beijing, China, April 2008.

[4] De Roo J. Eye reasoner.

[5] S. H. Garlik, A. Seaborne, and E. Prud'hommeaux. *SPARQL 1.1 Query Language*. World Wide Web Consortium, 2013.

[6] B. Glimm, I. Horrocks, B. Motik, R. Shearer, and G. Stoilos. A novel approach to ontology classification. *J. Web Sem.*, 14:84–101, 2012.

[7] Glimm, B. and Ogbuji, C., editor. *SPARQL 1.1 Entailment Regimes*. W3C Recommendation. World Wide Web Consortium, January 2013. Available at `http://www.w3.org/TR/sparql11-entailment/`.

[8] V. Haarslev, K. Hidde, R. Möller, and M. Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012.

[9] Motik, B. and Patel-Schneider, P. F., and Parsia, Bijan, editor. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. W3C Recommendation. World Wide Web Consortium, December 2012. Available at `http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`.

[10] A. Polleres, F. Scharffe, and R. Schindlauer. Sparql++ for mapping between rdf vocabularies. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 878–896. Springer, 2007.

[11] L. Serafini and A. Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *ESWC*, pages 361–376. Springer, 2005.

[12] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics*, 5(2):51–53, 2007.

[13] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006*, pages 292–297. Springer, 2006.

[14] L. Ying and J. Shujuan. International Workshop on LarKC - a Platform for Massive Distributed Incomplete Reasoning. *Digital Library Forum, China Science and Technology Information Institute, Beijing*, April 2011.