A Rule System for Querying Persistent RDFS Data*

Giovambattista Ianni[1], Thomas Krennwallner[2],
Alessandra Martello[1], and Axel Polleres[3]

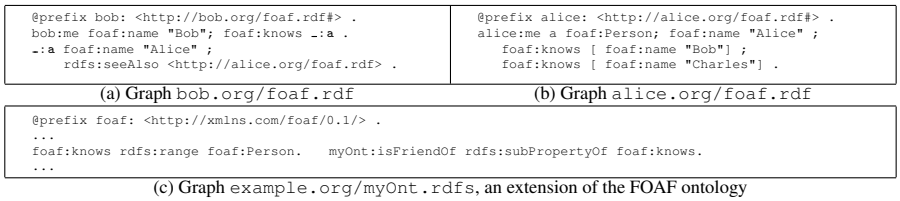
[1] Dipartimento di Matematica, Università della Calabria, I-87036 Rende (CS), Italy
{ianni,a.martello}@mat.unical.it
[2] Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria
tkren@kr.tuwien.ac.at
[3] Digital Enterprise Research Institute, National University of Ireland, Galway
axel.polleres@deri.org

Abstract. We present GiaBATA, a system for storing, aggregating, and querying Semantic Web data, based on declarative logic programming technology, namely on the dlvhex system, which allows us to implement a fully SPARQL compliant semantics, and on DLV^{DB}, which extends the DLV system with persistent storage capabilities. Compared with off-the-shelf RDF stores and SPARQL engines, we offer more flexible support for rule-based RDFS and other higher entailment regimes by enabling custom reasoning via rules, and the possibility to choose the reference ontology on a per query basis. Due to the declarative approach, GiaBATA gains the possibility to apply well-known logic-level optimization features of logic programming (LP) and deductive database systems. Moreover, our architecture allows for extensions of SPARQL by non-standard features such as aggregates, custom built-ins, or arbitrary rulesets. With the resulting system we provide a flexible toolbox that embeds Semantic Web data and ontologies in a fully declarative LP environment.

1 Introduction

Thanks to W3C standards such as RDF(S), OWL, and recently RIF, – see www.w3.org/2001/sw/ for an overview –, an increasing bulk of machine-readable data and knowledge encoded in rules and ontologies are becoming available on the Web. This vast amount of information uses rather lightweight ontologies in contrast to the full power of expressive description logic reasoning, which imposes new demands on systems to deal with Semantic Web data. A Semantic Web system should be able to *handle and evaluate arbitrarily complex queries on large amounts of RDF instance data* that are not manageable in memory. Thus, current RDF stores (see Sec. 2) provide persistent storage facilities and typically support SPARQL [1, 2]. On top of SPARQL, data processing requires *ontological inferences as well as custom, possibly nonmonotonic rules*, which are crucial, e.g., for modelling mappings between vocabularies. Moreover, it should be possible to *query combinations of different data graphs, ontologies, and rules* dynamically harvested from the Web. In this respect, most available RDF databases only offer

* This work has been partially supported by the Italian Research Ministry (MIUR) project Interlink II04CG8AGG, the Austrian Science Fund (FWF) project P20841, by Science Foundation Ireland (SFI) project Líon (SFI/02/CE1/I131), and by the EU FP6 project inContext (IST-034718).



(a) Graph `bob.org/foaf.rdf` (b) Graph `alice.org/foaf.rdf`

(c) Graph `example.org/myOnt.rdfs`, an extension of the FOAF ontology

Fig. 1. Two RDF graphs, and an extract of the FOAF ontology.

limited support for dynamic inference, custom reasoning via rules, or querying upon datasets of choice.

We propose an architecture aiming at closing this gap by providing the GiaBATA system which caters for knowledge-intensive applications on top of Semantic Web data by combining (a) SPARQL querying on varying datasets and varying schemes, (b) rule-based RDFS and (limited) OWL inference, and (c) a persistent storage system based on deductive database technology. Based on systems described in Sec. 2, our proposed RDF store fulfils the above goals and is easily extensible by features such as using arbitrary aggregates and built-in predicates as well as adding custom rules to SPARQL [3].

Motivating Example. Take the two data graphs about our well-known friends Alice, Bob, and Charles in Fig. 1(a)+(b). Both use vocabulary defined in an extended FOAF ontology (`http://xmlns.com/foaf/spec/`); the relevant part is in Fig. 1(c).

For instance, a SPARQL query could be used to extract names of persons mentioned in graphs that belong to friends of Bob. We assume that Bob provides links to the graphs associated to the persons he knows using `rdfs:seeAlso` statements:

```
select ?N from <http://example.org/myOnt.rdfs> from <http://bob.org/foaf.rdf>
       from named <http://alice.org/foaf.rdf>
where { bob:me myOnt:isFriendOf ?X . ?X foaf:seeAlso ?G .
        graph  ?G { ?P a foaf:Person; foaf:name ?N } }
```
(1)

The *dataset* $DS = (G,N)$ of a SPARQL query is defined by the *default graph* G obtained from the RDF merge [4] (denoted $\uplus$) of all graphs mentioned in **`from`** clauses plus a set of named graphs $N = \{(g_1, G_1), \ldots, (g_k, G_k)\}$, i.e., pairs of IRIs given in **`from named`** clauses and their corresponding graphs. In our example, the dataset is being specified explicitly by a set of **`from`** and **`from named`** clauses, i.e., the dataset of query (1) would be $DS_1 = (\, G_{\texttt{myOnt}} \uplus G_{\texttt{bob.org}}, \{(\texttt{<http://alice.org/>}, G_{\texttt{alice.org}})\})$.

Since neither of the three source graphs contain a triple matching the pattern `bob:me myOnt:isFriendOf ?X` in the **`where`** clause, most existing RDF stores with SPARQL support would return an empty result on (1). This is because typically SPARQL engines only take simple RDF entailment into account when answering queries, which boils down to basic graph pattern matching. When taking ontological inference by the statements of the `myOnt` ontology into account, however, one would expect to obtain three matchings for the variable `?N`: {`?N`/`"Alice"`, `?N`/`"Bob"`, `?N`/`"Charles"`}.

In order to obtain this expected answer, SPARQL's basic graph pattern matching needs to be extended, see [1, Section 12.6]. In principle, this means that the graph pattern in the **`where`** clause needs to be matched against a graph enlarged by applying inferences

prescribed by the given entailment regime.[1] SPARQL extensions providing entailment regimes other than simple RDF entailment are still an open research problem.[2]

Approximations of entailment regimes such as RDFS and OWL [5–7], however, can be modeled by finite Datalog rulesets that cover sound but not necessarily complete RDFS and OWL inferences. For instance, the RDF Semantics document contains an informative set of entailment rules, a subset of which is implemented by most available RDF stores. However, these rules could be dynamically evaluated upon query time, or inferences could be materialized at graph loading time, producing a persistent extended graph which queries are issued on. Materialization upon loading time has drawbacks though: when the dataset includes a number of different graphs, it is not clear upfront (i) which ontology should be taken into account for which data graph, and (ii) to which graph the inferred triples "belong," which in turn complicates querying named graphs.

As for (i), assume that a user agent wants to issue another query on graph `bob.org` with the original FOAF ontology instead of our modified version within the default graph. Since the property `myOnt:isFriendOf` is stated to be a subproperty of `foaf:knows`, the triple `bob:me myOnt:isFriendOf _:a`, which could be inferred from $G_{\texttt{myOnt}} \uplus G_{\texttt{bob.org}}$, would most probably contribute in an undesired way to such a different query. Current RDF stores however prevent to parametrize inference with an ontology of choice, since typically inferences are computed upon loading time *once and for all*.

As for (ii), queries upon datasets including named graphs are even more problematic. Even if inference was supported properly, the answer to (1) over dataset DS_1 is just $\{\texttt{?N}/\texttt{"Alice"}\}$. The `myOnt` ontology is only merged into the default graph, but not into the named graph. Thus, there is no way to infer that `"Bob"` and `"Charles"` are actually names of `foaf:Persons` within the named graph $G_{\texttt{alice.org}}$. Indeed, SPARQL does not allowed to explicitly specify a dataset to be merged with named graphs.

In order to overcome both point (i) and (ii) above, the current nonintuitive semantics of SPARQL with RDFS can be mitigated by slightly extending SPARQL's syntax. We give the possibility of specifying ontologies which are used for the entire dataset, including named graphs, by means of the new directive **`using ontology`**. For instance, to specify the dataset $DS_2 = (G_{\texttt{myOnt}} \uplus G_{\texttt{bob.org}}, \{(\texttt{<http://alice.org/>}, G_{\texttt{myOnt}} \uplus G_{\texttt{alice.org}})\})$, we allow the following syntax replacing the first line of query (1):

```
select ?N using ontology <http://example.org/myOnt.rdfs> ...
```

Hence, the **`using ontology`** construct allows for coupling the given dataset with the proper class/property hierarchy of the `myOnt` data schema. The possibility of query dependent and dynamic ontology inference shows the effectiveness of this approach.

2 System Description and Evaluation

GiaBATA is based on a combination of the DLV^{DB} [8] and dlvhex [9] systems, and caters for persistent storage of both data and ontology graphs. The former system is a variant of the DLV system with built-in database support. The latter is a solver for the so called HEX-programs [9]. dlvhex features an extensible plugin system which we used for developing a rewriter-plugin able to translate SPARQL queries to HEX-programs. Our prototype supports the standard SPARQL specifications extended with

[1] The entailment rules for getting the expected results for (1) are RDFS3 and RDFS7 from [4].

[2] cf. `http://www.polleres.net/sparqltutorial/`, Unit 5b, for details

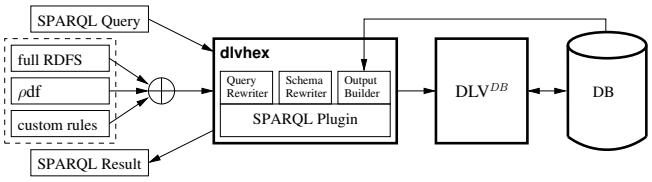


Fig. 2. The GiaBATA system

the syntax outlined above, arbitrary entailment regimes (currently, we support RDFS), and the addition of custom rules. We provide a SPARQL-protocol compliant Web service interface. Fig. 2 shows a high-level view of our system architecture. It consists of four main components: (i) a SPARQL to datalog rewriter as part of a plugin for dlvhex (see [10, 11]), (ii) a schema rewriter which manipulates the rewritten datalog rules, adds auxiliary definitions in order to match the underlying database schema, and implements the chosen semantics by adding rules to the input of the module in (iii) the DLV^{DB} datalog engine, which rewrites the input program to native SQL queries, and thereby accesses triples persistently stored in (iv) a DBMS of choice storing the RDF data according to a particular storage schema. This architecture provides a fully declarative implementation of SPARQL and brings support for further (deductive) database and LP optimization and extensibility.

By and large, the chosen database schema for the triples exploits a main table storing quadruples composed of subject, predicate, object, and the source graph of each triple. This representation has been improved [12] by additional relations mapping URIs/literal string to integer values, which avoids string matching in favour of integer comparison.

We have investigated the persistent storage facilities as well as SPARQL features of some of the state-of-the-art RDF stores, namely, Sesame RDF database 2.0, Owlim v3.0b8, AllegroGraph 3.0.1, and Jena/Joseki 3.2.[3] All systems support persistent storage, either based on RDBMS or on other dedicated backends, and deal with RDFS inferencing and partial or non-standard OWL fragments. While a persistence strategy is provided, both reasoning and query evaluation are usually performed in main memory, adopting a materialization strategy. In our tests we focused on querying under ρDF inference regime [5], requiring the capability to match patterns against named graphs. The investigation revealed that combining RDFS and named graph queries lead to unexpected behavior. While inference is properly handled as long as the query ranges over the whole dataset, RDFS reasoning fails in case of queries using explicit default or named graphs. That leaves dynamic named graph queries with inference infeasible in current systems.

3 Related Work and Outlook of the Demo

There are earlier approaches to integrate ontological inference in a logic programming environment. As opposed to SWI Prolog's Semantic Web library [13] which also offers SPARQL support, we support Answer Set Programming as underlying LP paradigm,

[3] `http://www.openrdf.org/`, `http://www.ontotext.com/owlim/`, `http://jena.hpl.hp.com/`, and `http://agraph.franz.com/`

instead of Prolog. In contrast to OntoDLV [14], which supports proprietary ontology and query languages, the focus of GiaBATA is on compliance with Web standards such as SPARQL, RDF(S), OWL, and RIF. We currently work on the efficiency of FILTER expressions by rewriting them to SQL queries over the underlying database, see [15, 16]. Moreover, one of the next steps includes to support arbitrary database schemes. Applying optimization techniques from both the DB and LP area should boost evaluation performance. Extensions of SPARQL by aggregates and custom built-ins presented in earlier works [11] carry over to our persistent storage version with minor modifications. We are going to add support for extended graphs, that is SPARQL "views" as defined also in [11].

The live discussion will provide a better insight into GiaBATA showing use cases that cover the key features of the system in accessing, storing and querying Semantic Web data. For features explanation we will refer to largely adopted benchmark suites,[4] while ad hoc showcases will help to understand the advantages of our proposed **`using ontology`** extension. Finally, the users will get an overview of GiaBATA's capabilities and will learn how to interact with it.

References

1. Prud'hommeaux, E., (eds.), A.S.: SPARQL Query Language for RDF (January 2008) W3C Recommendation.
2. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ISWC 2006. 30–43
3. Polleres, A., Scharffe, F., Schindlauer, R.: SPARQL++ for mapping between RDF vocabularies. In: ODBASE 2007. Springer (2007) 878–896
4. Hayes, P.: RDF semantics. (February 2004) W3C Recommendation.
5. Muñoz, S., Pérez, J., Gutiérrez, C.: Minimal deductive systems for RDF. ESWC 2007. 53–67
6. ter Horst, H.J.: Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In: ISWC 2005. Springer 668–684
7. Ianni, G., Martello, A., Panetta, C., Terracina, G.: Efficiently querying RDF(S) ontologies with Answer Set Programming. J Logic Computation (August 2008).
8. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theor Pract Log Prog **8**(2) (2008) 129–165
9. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective Integration of Declarative Rules with External Evaluations for Semantic Web Reasoning. In: ESWC 2006. Springer 273–287
10. Polleres, A.: From SPARQL to rules (and back). In: WWW2007. ACM (2007) 787–796
11. Polleres, A., Schindlauer, R.: dlvhex-sparql: A SPARQL-compliant query engine based on dlvhex. In: ALPSWS 2007. Vol. 287 of CEUR WS Proceedings. (September 2007) 3–12
12. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: VLDB 2007. ACM (2007) 411–422
13. Wielemaker, J., Hildebrand, M., van Ossenbruggen, J.: Prolog as the Fundament for Applications on the Semantic Web. In: ALPSWS 2007.
14. Ricca, F., Gallucci, L., Schindlauer, R., Dellarmi, T., Grasso, G., Leone, N.: OntoDLV: An ASP-based System for Enterprise Ontologies. J Logic Computation (August 2008).
15. Lu, J., Cao, F., Ma, L., Yu, Y., Pan, Y.: An Effective SPARQL Support over Relational Databases. In SWDB-ODBIS07, co-located with VLDB 2007. (2007) 57–76
16. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. In: ISWC 2005. Springer (2005) 685–701

[4] LUBM Benchmark available at `http://swat.cse.lehigh.edu/projects/lubm/`