Some experiments on the usage of a deductive database for RDFS querying and reasoning

Giovambattista Ianni[1,2], Alessandra Martello[1],
Claudio Panetta[1], and Giorgio Terracina[1]

[1] Dipartimento di Matematica, Università della Calabria,
I-87036 Rende (CS), Italy,
[2] Institut für Informationssysteme 184/3, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{ianni,a.martello,panetta,terracina}@mat.unical.it

Abstract. Ontologies are pervading many areas of knowledge representation and management. To date, most research efforts have been spent on the development of sufficiently expressive languages for the representation and querying of ontologies; however, querying efficiency has received attention only recently, especially for ontologies referring to large amounts of data. In fact, it is still uncertain how reasoning tasks will scale when applied on massive amounts of data. This work is a first step toward this setting: based on a previous result showing that the SPARQL query language can be mapped to a Datalog, we show how efficient querying of big ontologies can be accomplished with a database oriented extension of the well known system DLV, recently developed. We report our initial results and we discuss about benefits of possible alternative data structures for representing RDF graphs in our architecture.

1 Introduction

The *Semantic Web* [4, 11] is an extension of the current Web by standards and technologies that help machines to understand the information on the Web so that they can support richer discovery, data integration, navigation, and automation of tasks. Roughly, the main ideas behind the Semantic Web aim to *(i)* add a machine-readable meaning to Web pages, *(ii)* use ontologies for a precise definition of shared terms in Web resources, *(iii)* make use of KR technology for automated reasoning on Web resources, and *(iv)* apply cooperative agent technology for processing the information of the Web. The development of the Semantic Web proceeds in layers of Web technologies and standards, where every layer stays on top of lower layers.

There is currently much research work on the three consecutive RDF(S), Ontology and Rule (listed from bottom to top) layers. The RDF(S) layer was initially conceived as a basic framework for defining resources available on the Web and their connections. In this vision, RDF(S) should have little or no semantics, and focuses only on the logical format of information, which is based on an encoding of data as a labeled graph (or equivalently, a ternary relation, commonly called RDF *triplestore* or RDF *graph*).

The Ontology layer should be built on top of RDF(S) and should provide the necessary infrastructure for describing knowledge about resources. An ontology can be written using one of the three official flavors of the OWL language [18], currently accepted as a W3C Standard Recommendation. An OWL knowledge base is written in RDF(S), where some of the keywords of the language are now given additional semantics.

OWL is based on decidable flavors of description logics and features rich expressiveness which, unfortunately, introduces high computational costs for many of the reasoning tasks commonly performed over an ontology. Nonetheless, a variety of Web applications require highly scalable processing of data. This puts the focus back to the lower RDF(S) data layer. In this context, RDF(S) should play the role of lightweight ontology language. Indeed, RDF(S) has few and simple descriptive capabilities (mainly, the possibility to describe and reason over monotonic taxonomies of objects). One can thus expect from RDF(S) query systems the ability of querying very large datasets with excellent performance, yet allowing limited reasoning capabilities on the same data.

As a candidate W3C recommendation [8], the SPARQL language is reaching consensus as query language of election for RDF(S) data. In this scenario, an RDF(S) triplestore plays the role of a database, but, as an important difference, a triplestore might contain information not explicitly stored, obtainable by logical inference. Allowed logical inference rules are given by the official RDF(S) semantics specification, whereas SPARQL plays the role of query language.

Although SPARQL-enabled triplestores are many [3, 20, 2, 21] their scalability or querying capabilities are still far from maturity, having one or more of the following drawbacks:

– RDF(S) semantics is implemented by materializing all the inferred data a priori. This latter option can not be adopted in practice if massive amount of data are involved in the inferencing process, since inferred information is usually much bigger in size than explicit information.

– The basic reasoning machinery of RDF(S) prescribes heavy usage of transitive closure (recursive) constructs. Roughly speaking, given a class taxonomy, an individual belonging to a leaf class must be inferred to be member of all the ancestor classes, up to the root class. This prevents a straightforward implementation of RDF(S) over RDBMSs, since RDBMSs usually feature very primitive, inefficient implementations of recursion in their native query languages.

But, interestingly, in Datalog, recursion is a first class citizen. Also, most of the SPARQL features can be mapped to a rule based language with stable model semantics [19]. Intuitively, a large fragment of the RDF(S) semantics can thus be implemented by means of a translation to an equivalent Datalog program.

Thus, one may think to adopt a Datalog based language language for implementing RDF(S). Value invention constructs, as those introduced in [6] (where it is defined a form of Answer Set Programming with external predicates and value invention), allow, in practice, the manipulation of infinite universes of individuals (as in the RDF(S) scenario) in a finite model setting.

Many important efforts in the Semantic Web community aim to integrate Ontologies with Rules under stable model semantics (e.g. [9, 16]), considering both OWL and RDF(S). In this context, the possibility to exploit a Datalog-like language to express both the ontology and the query/rule language would provide important benefits.

However, it is well known in the research community that current (extended) Datalog based systems present important limitations when the amount of data to reason about is large; in fact: *(i)* reasoning is generally carried out in main-memory and, hence, the quantity of data that can be handled simultaneously is limited; *(ii)* the interaction with external (and independent) DBMSs is not trivial and, in several cases, not allowed at all, but in order to effectively share and elaborate large ontologies these must be handled with some database technology; *(iii)* the efficiency of present datalog evaluators is still not sufficient for their utilization in complex reasoning tasks involving large amounts of data.

In the following we refer to a recently proposed database-oriented extension of the well known Answer Set Programming system DLV, named DLV^{DB} [22], which presents the features of a Deductive Database System (DDS) and can do all the reasoning tasks directly in mass-memory; DLV^{DB} does not have, in principle, any practical limitation in the dimension of input data, is capable of exploiting optimization techniques both from the DBMS field (e.g., join ordering techniques [12]) and from the DDS theory (e.g., magic sets [17]), and can easily interact (via ODBC) with external DBMSs.

DLV^{DB} turned out to be particularly effective for reasoning about massive data sets (see benchmark results presented in [22]) and supports a rich query and reasoning language including stratified recursion, true negation, negation as failure, and all built-in and aggregate functions already introduced in DLV [10]. As a consequence, DLV^{DB} seems to be a good candidate also as an ontology querying engine.

To accomplish this goal, several building bricks are missing: *(i)* a mapping from RDF(S) semantics to Datalog; *(ii)* the translation of SPARQL queries in Datalog; *(iii)* the connection of massive RDF(S) data to a suitable system such as DLV^{DB}; *(iv)* the evaluation of queries directly on a given triplestore using DLV^{DB}.

The present paper concentrates on points (iii) and *(iv)*. About point *(i)*, *(ii)* and *(iii)* the reader may refer to [13],[5] and [19]. In particular, it aims to represent a first step toward the reconciliation of expressiveness with scalability for ontology querying, by means of deductive database technology.

The paper is organized as follows. In the next Section we briefly introduce the main peculiarities of the DLV^{DB} system. The Section 3 is devoted to present our experimental results, whereas in the section 4 we discuss about alternative data structure better suited to handling RDF data. Finally, in Section 5 we draw some conclusions.

2 DLV^{DB}

DLV^{DB} [22] is an extension of the well known ASP system DLV [14] designed both to handle input and output data distributed on several databases, and to allow the evaluation of logic programs directly on databases. It combines the expressive power of DLV with the efficient data management features of DBMSs [12].

The detailed description of DLV^{DB} is out of the scope of the present paper; here we briefly outline the main peculiarities which make it a suitable Datalog-based ontology querying engine. The interested reader can find a complete description of DLV^{DB} and its functionalities in [22].
The system, along with documentation and some examples, are available for download at `http://www.mat.unical.it/terracina/dlvdb`.

Generally speaking, DLV^{DB} allows for two typologies of execution: *(i)* direct database execution, which evaluates logic programs directly on database, with a very limited usage of main-memory but with some limitations on the expressiveness of the queries, and *(ii)* main-memory execution, which loads input data from different (possibly distributed) databases and executes the logic program directly in main-memory. In both cases, interoperation with databases is provided by ODBC connections; these allow handling, in a quite simple way, data residing on various databases over the network.

For the purposes of this paper, it is particularly relevant the application of DLV^{DB} in the direct database execution modality for the querying of large ontologies. In fact, usually, the user has his data stored in (possibly distributed) triplestores and wants to carry out some reasoning on them; however the amount of such data can be such that the evaluation of the query can not be carried out in main-memory. Then, it must be evaluated directly in mass-memory.

Moreover, DLV^{DB} turned out to be particularly effective for reasoning about massive data sets (see benchmark results presented in [22]) and supports a sufficiently rich reasoning language for querying ontologies (see also Section 3).

Three main features characterize the DLV^{DB} system in the direct database execution modality: *(i)* its ability to evaluate logic programs directly and completely on databases with a very limited usage of main-memory resources, *(ii)* its capability to map program predicates to (possibly complex and distributed) database views, and *(iii)* the possibility to easily specify which data is to be considered as input or as output for the program. In the application context considered in this paper, these characteristics allow the user to have a wide flexibility in querying available ontologies.

In order to properly carry out the evaluation, the system needs to know the mappings between input/output data and program predicates, as well as whether the temporary relations possibly needed for the mass-memory evaluation should be maintained or deleted at the end of the execution. The user can specify this information by some auxiliary directives which must be fed to the system beside the logic program.

3 Experiments

In this section we present the results of our experiments aiming at comparing the performance of DLV^{DB} with several state-of-the-art triplestore. The main goal of our experiments was to evaluate both the scalability and the the query language expressiveness of the tested systems. All tests have been carried out on a Pentium 4 machine with a 3.00 GHz CPU and 1.5 Gbytes of RAM.

3.1 Compared Systems

In our tests we compared DLV^{DB} with three state-of-the-art triplestores, namely: Sesame, ARQ, and Mulgara. The first two systems allow both in-memory and RDBMS storage and, consequently, we tested them on both execution modalities. In the following we shall refer the in-memory version of Sesame (resp., ARQ) as Sesame-Mem (resp. ARQ-Mem) and the RDBMS version as Sesame-DB (resp. ARQ-DB). For each system we used the latest official available release. We next briefly describe them.

Sesame [20] is an open source Java framework with support for storage and querying of RDF(S) data. It offers to developers a flexible access API and several query languages; however, its native language (which is the one adopted in our tests) is SeRQL – Sesame RDF Query Language. In fact, the current stable release of Sesame does not support the SPARQL language yet. Some of the query language's most important features are: *(i)* expressive path expression syntax that match specific paths through an RDF graph, *(ii)* RDF Schema support, *(iii)* string matching. Furthermore, it allows simplified forms of reasoning on RDF and RDFS. In particular, inferences are performed by pre-computing the closure $R(G)$ of the input triplestore G. The latest official release currently available is the version 1.2.7.

ARQ [3] is a query engine implementing SPARQL under the Jena framework[3]. ARQ includes a rule-based inference engine and performs non materialized inference. As for Sesame, ARQ can be executed with data loaded both in-memory and on a RDBMS. We executed SPARQL queries from Java code using the Jena's API (version 2.5) in both execution modalities.

Mulgara [2] is a database system specifically conceived for the storage and retrieval of RDF(S). Mulgara is an Open Source active fork of the Kowari project[4]. The adopted query language is *iTQL* (Interactive Tucana Query Language), a simple SQL-like query language for querying and updating Mulgara databases. A compatibility support with SPARQL is declared, yet not implemented. The Mulgara Store offers native RDF(S) support, multiple databases (models) per server, and full text search functionality. The system has been tested using its internal storage data structures (XA Triplestore). The latest release available for Mulgara is mulgara-1.0.0.

3.2 Benchmark Data Set

We adopted as reference benchmark data the DBLP database [15]. DBLP contains a large number of bibliographic descriptions on major computer science journals and proceedings; the server indexes more than half a million articles and several thousand links to home pages of computer scientists. Recently, an OWL ontology has been developed for DBLP data and the corresponding RDF can be downloaded at the web address `http://sw.deri.org/∼aharth/2004/07/dblp/`. The main classes represented in this ontology are *Author*, *Citation*, *Document*, and *Publisher*, where a *Document* can be one of: Article, Book, Collection, Inproceedings, Mastersthesis, Phdthesis, Proceedings, Series, WWW.

[3] `http://jena.sourceforge.net`

[4] `http://www.kowari.org/`

In order to test the scalability of the various systems we considered several subsets of the entire database, each containing an increasing number of statements and constructed in such a way that the greater sets strictly contain the smaller ones. Generated data sets contain from 50000 to 2000000 RDF statements[5].

3.3 Tested Queries

As previously pointed out, the expressiveness of the query language varies for each tested system. In order to compare both scalability and expressiveness, we designed for kind of queries of increasing complexity, ranging from simple selections to queries requiring different forms of inferences over the data.

In more detail, we selected the following for queries which will be referred to as Q_1, Q_2, Q_3 and Q_4, respectively.

- Q_1: Select the names of the Authors and the URI of the corresponding Articles they are author of;
- Q_2: Select the names of the Authors which published at least one Article in year 2000;
- Q_3: Select the names of the Authors which are creators of at least one document (i.e. either an Article, or a Book, or a Collection, etc.);
- Q_4: For each Author in the database, select the corresponding name and count the number of Articles he published.

Here, queries Q_1 and Q_2 are simple selections; Q_3 requires a simple form of inference; in fact articles, books, etc. must be abstracted into documents. Query Q_4 requires the capability to aggregate data, which is not provided by all query languages.

It is worth observing that queries Q_1, Q_2, and Q_3 can be executed by all the evaluated systems. As for Q_3, we exploited the Krule engine for Mulgara, the inferencing repository in Sesame and the inferencing Reasoner in ARQ. Note that Sesame-DB materializes the possible inferenced data just during the loading of the RDF dataset in the database; however, in our tests, we measured only the query answering time for it. Query Q_4 can not be evaluated neither by ARQ nor by Sesame because both SPARQL and SeRQL query languages do not support aggregate operators.

Due to space constraints, we can not show here the details of all the queries. Just to show an example, we next present the encodings used for Q_1 in the various systems. Syntax is self-intuitive.

DLV^{DB} encoding for Q_1

$$\begin{aligned} q_1(\mathit{NAME}, \mathit{RES}) \text{ :- } & \mathit{triple}(\mathit{RES}, \text{“}\mathit{rdf{:}type}\text{”}, \text{“}\mathit{Article}\text{”}), \\ & \mathit{triple}(\mathit{RES}, \text{“}\mathit{dc{:}creator}\text{”}, \mathit{PERS}), \\ & \mathit{triple}(\mathit{PERS}, \text{“}\mathit{foaf{:}name}\text{”}, \mathit{NAME}). \end{aligned}$$

[5] An RDF statement is a small cluster of RDF triples usually not larger than 10 within our datasets.

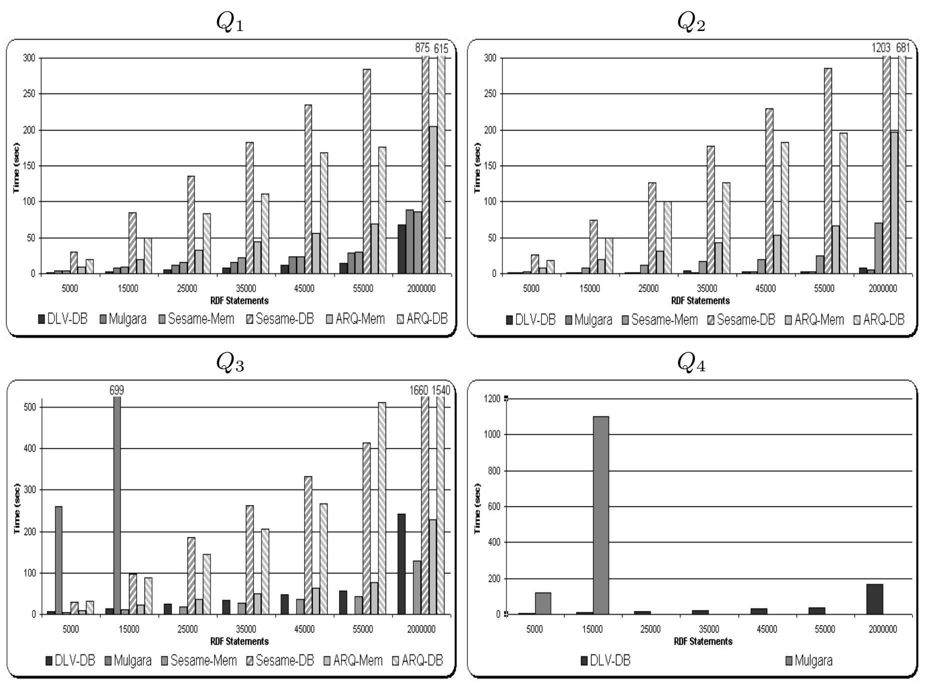

Fig. 1. Results for queries Q_1, Q_2, Q_3, Q_4

Sesame encoding for Q_1

> **select** *name*, *res*
> **from** {*res*} <*rdf:type*> {*type*},
> {*res*} <*dc:creator*> {*pers*}, {*pers*} <*foaf:name*> {*name*},
> **where** *type* =<*Article*>

ARQ encoding for Q_1 (SPARQL syntax)

> **select** *?name ?res*
> **where** {*?res* <*rdf:type*><*Article*> .
> *?res* <*dc:creator*> *?pers.* *?pers* <*foaf:name*> *?name*}

Mulgara encoding for Q_1

> **select** *$name*, *$res*
> **from** <*rmi://localhost/server1#triple*>
> **where** *$res* <*rdf:type*><*Article*> **and**
> *$res* <*dc:creator*> *$pers* **and** *$pers* <*foaf:name*> *$name*;

3.4 Results and Discussion

Figure 1 shows the results we have obtained for the five queries described above. In the figure, the chart of a system is absent whenever it has not been able to solve the query due to some system's fault or if its response time was greater than 3600 seconds (1 hour). Moreover, if a system's query language was not sufficiently expressive to answer a certain query, it has not been included in the graph. From the analysis of the figure we can draw the following observations.

Mulgara has, after DLV^{DB}, the more expressive query language and, for the simple queries Q_1 and Q_2 the best performance along DLV^{DB} and, in some cases, Sesame-Mem. However, when the queries involve the more advanced parts of the language, the efficiency of Mulgara quickly drops; in fact, both in query Q_3 and in query Q_4 its response time exceeded the limit set in our tests already after 15000 RDF statements.

Sesame-Mem turned out to be competitive in all considered data sets only for queries Q_1 and Q_3; in fact, it has not been able to solve query Q_4 due to lack of expressiveness in the query language; moreover, in query Q_2, its performance degraded when the input data sets increased. Sesame-DB always had significantly worse performance than Sesame-Mem.

ARQ always presented the worst performances (except in one case); moreover, as occurred for Sesame, also the database version of ARQ revealed worse performance than its in-memory version. The expressiveness of ARQ's query language prevented to encode queries Q_4.

Finally, DLV^{DB} revealed both the best performance (in almost all the data sets and queries) and the highest expressiveness of the query language, thus demonstrating its good potential to be exploited as ontology querying engine.

It is worth pointing out that both Sesame and ARQ performance are negatively influenced by the usage of a DB (see, in particular, results of queries Q_1 and Q_2); this can be probably motivated by the fact that they carry out (at least parts of) their computations in main memory anyway and, consequently, transferring data from disk to memory produces just overhead. On the contrary, DLV^{DB} and Mulgara exploit the database technology directly for their reasoning tasks and, consequently, are more effective.

4 Experiments with alternative data structures

In the context of real-world applications it becomes crucial the choice of a data schema for the relational database handling RDF data model, since this has a direct impact on the performance and scalability issues. The discussed solution assumes to store the RDF(s) graph at hand, using a straightforward representation, where a single 3-columns table contains one row for each statement of the form $\langle subject, predicate, object\rangle$. This representation, though flexible, is not efficient when several self-joins are required to sweep over this single large table. A first step in order to improve the performance of the database, maintaining this simple schema, is to reduce the execution time required by the join's operations. A solution largely adopted in similar applications and discussed in [1], is to avoid to store explicitly string values referring to URIs and literals in the main table, replacing them with an hash value. Indeed, integer matching is intuitively much faster than string matching. Each URIs/literal string is mapped to and integer: the main tables stores triples in form of integer values, while additional tables store the association from URI/Literals to integer, which is used for a post-normalization. Several experiments that we reproduced on this new configuration show the validity of this approach with respect to the first one.
Finally, we have considered other suggestions for alternative data structures better suited for handling RDF data, called property tables technique ([1],[23]). These aim at denormalizing RDF tables by storing them in a flattened representation, trying to encode triples according to the hidden "schema" of RDF data,

similarly to a traditional relational schemas. The idea is to define a set of property tables containing (cluster of) properties that tend to be defined together (and then storing the triples from the RDF dataset whose properties belong to the selected attributes), or to cluster similar sets of subjects together, grouping them in a property-class table. There is a variety of storage schemes and several variations of these which have also been implemented in existing RDF stores, using hybrid representation that combine features of both. The most important advantage of these choices is the possibility of accessing directly all the triples having the same property value. However, these configurations can be extremely sparse (by the presence of NULL values in the table) and not well suited for supporting multi-valued attributes. Thus, while such techniques usually improve performance of queries involving a single property table, it is required to properly cluster the property values occurring in the dataset.
Inspired by these considerations we extended our representation schema implementing a fully decomposed storage model in which the triples table is rewritten into n two column tables where n is the number of unique properties in the dataset. This approach (discussed in [1], [23]) support succinct representation of multi-valued attributes and heterogeneous records (subjects not defining a particular property). Moreover, this data scheme allows to access directly assertions related to the same property value. Unfortunately, for a query which quantifies over property values, several tables have to be merged. This overhead seems reasonable (as we verified testing performance on query ranging on variable predicates). Furthermore, insert and update operations can be slower, since for operation on statements related to the same subject, more tables need to be accessed.
We carried out several experiments on these new data structures to compare the execution time of the queries for the same dataset used in previous test. Clearly, this implies the translation of queries to queries over the new representations. The results obtained shows that the solution using triples table storing identifiers instead of strings performs better than the simple one, but the best choice (actually) is to use a fully decomposed storage schema. Especially, this seems to give more benefit as the number of triples grows. For example, the query Q_1 (run on the biggest dataset) takes 74 seconds running on the single table representation, 46 seconds running using hash representation of URIs/literals and 28 seconds running on the fully decomposed schema configuration with hash representation.

5 Conclusions

In this paper we presented a first step toward efficient and reliable ASP-based querying of ontologies. We experimentally proven that our solution, based on a database oriented implementation of ASP, improves both scalability and expressivity of several state-of-the-art systems. Although, currently, RDF data are stored in the standard triple format, the first experiments with alternative data structures are very promising. The representation of data in some more structured form (as already some of the tested systems do) could significantly improve performance. Another promising research line consists in using database integration techniques in the ontology context such as in [7].

References

1. D. J. Abadi, A.Marcus, S. Madden, and K. J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422, 2007.
2. T. Adams, G. Noble, P. Gearon, and D. Wood. MULGARA homepage. http://www.mulgara.org/ , since 2006.
3. ARQ homepage. http://jena.sourceforge.net/ARQ/, since 2004.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
5. J. de Bruijn and S. Heymans. RDF and logic: Reasoning and extension. In *Proceedings of the 6th WebS, in conjunction with the 18th DEXA*, Regensburg, Germany, September 3–7 2007.
6. F. Calimeri, S. Cozza, and G. Ianni. External sources of knowledge and value invention in logic programming. *Ann. Math. Artif. Intell.*, 50(3-4):333–361, 2007.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. Efficient integration of relational data through dl ontologies. CEUR Electronic Workshop Proceedings, 2007.
8. A. Seaborne E. Prud'hommeaux. Sparql query language for rdf. w3c candidate recommendation, 14 june 2007. `http://www.w3.org/tr/rdf-sparql-query/`.
9. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *(IJCAI) 2005*, pages 90–96, Edinburgh, UK, August 2005.
10. W. Faber and G. Pfeifer. `DLV` homepage, since 1996. http://www.dlvsystem.com/.
11. D. Fensel, W. Wahlster, H. Lieberman, and J. Hendler, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* MIT Press, 2002.
12. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation.* Prentice Hall, 2000.
13. G. Ianni, A. Martello, C. Panetta, and G. Terracina. Faithful and effective querying of RDF ontologies using DLVDB.
14. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, July 2006.
15. Michael Ley. Digital bibliography and library project http://dblp.uni-trier.de/.
16. B. Motik and R. Rosati. A faithful integration of description logics with logic programming. In *IJCAI*, pages 477–482, 2007.
17. I.S. Mumick, S.J. Finkelstein, H. Pirahesh, and R. Ramakrishnan. Magic conditions. *ACM Trans. Database Systems*, 21(1):107–155, 1996.
18. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Owl web ontology language semantics and abstract syntax. w3c recommendation, 10 february 2004.
19. A. Polleres. From sparql to rules (and back). In *In Proceedings of the 16th World Wide Web Conference (WWW2007), Banff, Canada*, 2007. Extended technical report version available at `http://www.polleres.net/publications/GIA-TR-2006-11-28.pdf`.
20. SESAME homepage. http://www.openrdf.org/, since 2002.
21. Sparql implementations. `http://esw.w3.org/topic/sparqlimplementations`.
22. G. Terracina, N. Leone, V. Lio, and C. Panetta. Experimenting with recursive queries in database and logic programming systems. *Theory and Practice of Logic Programming (TPLP). Available on-line at http://arxiv.org/abs/0704.3157*, 2007. Forthcoming.
23. Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *International Semantic Web Conference*, pages 685–701, 2005.