DIG 2.0 – Towards a Flexible Interface for Description Logic Reasoners

Anni-Yasmin Turhan⋆, Sean Bechhofer†, Alissa Kaplunova‡,
Thorsten Liebig⋆⋆, Marko Luther††, Ralf Möller‡, Olaf Noppens††,
Peter Patel-Schneider‡‡, Boontawee Suntisrivaraporn⋆, Timo Weithöner††

⋆ TU Dresden
† University of Manchester
‡ TU Hamburg-Harburg
⋆⋆ University of Ulm
†† DoCoMo Euro-Labs
‡‡ Bell Labs Research

Abstract. The DIG Interface provides an implementation-neutral mechanism for accessing Description Logic reasoner functionality. At a high level the interface can be realised as XML messages sent to the reasoner over HTTP connections, with the reasoner responding as appropriate. Key changes in the current version (DIG 2.0) include support for OWL 1.1 and well-defined mechanisms for extensions to the basic interface.

1 Introduction

The DL Implementation Group (DIG) is a self-selecting assembly of researchers and developers associated with implementations of Description Logic (DL) systems. One of the main tasks of DIG has been the specification of a standardised interface for Description Logic reasoners: the DIG Interface [1, 2].

The DIG Interface (also known as "DIG") is a lightweight mechanism providing access to reasoning functionality. The specification of DIG 2.0 will be given by an abstract model in UML. This abstract model can be realised in different techniques. One of these techniques is XML messages sent over HTTP connections. To highlight the changes to DIG 1.1 we refer to the realisation in XML in this paper.

The XML statements specify a TELL/ASK interface, which provides DIG applications with the facility to send their knowledge bases to the DIG server by using the TELLS statements and then using the ASKS interface to get inferred information from the reasoner. This approach has several advantages: the issue of implementation language is finessed; the API can be defined in some standard formalism intended for the purpose; a mechanism is provided for applications to communicate with the DL system, either locally or remotely; and alternative DL components can be substituted without affecting the application.

DIG 1.0 and DIG 1.1 specifications provide each an interface for classical DL reasoning services for the DL $\mathcal{SHOIQ}(\mathcal{D})^-$. Thus it supports more language constructs than the web ontology language OWL DL. The existing DIG specifications have had a significant impact in the use of DL reasoners within tools such as ontology editors like OilEd, Protégé and OntoTrack [3]. DIG is not solely intended to support ontology editors, however - middleware and applications making use of DIG include the Instance Store [4], OntoXpl [5] and Jena [6] for example. Key DL reasoner implementations such as CEL [7] FaCT++ [8], KAON2 [9] Pellet [10] and Racer [11] all provide DIG interfaces.

Despite its successful application there are some shortcomings in the DIG 1.1 interface that will be remedied in DIG 2.0. For example, facilities for querying property axioms or equality of individuals were missing [12] and will be added in DIG 2.0. Furthermore, DIG 2.0 introduces a number of enhancements and changes from previous versions. Key changes include support for operators in the proposed OWL 1.1 extensions and well-defined mechanisms for extension to the basic interface along with examples of extensions supporting query for told information, retraction and non-standard inferences. DIG 2.0 will thus provide a flexible mechanism for interfacing with systems that provide reasoning services in Web Ontology Languages, as needed in many Semantic Web applications.

2 DIG 2.0 Overview

In order to address the above mentioned issues, the DIG Working Group proposed a new, standardised Description Logic interface DIG 2.0. DIG provides a basic functionality to allow tools such as ontology editors, middleware, and DL reasoners to interoperate seamlessly in some application system and to enable plug-and-play of a component by another without affecting the application. The DIG interface is composed of a "core DIG interface", which covers the basic services and a number of extensions, which provide additional reasoning service or extend the expressivity of the core DIG.

However, DIG does not intend to provide what we might truly call a reasoning service, but rather helps insulate applications from the location and implementation language of a DL reasoner. The specification does not address issues such as stateful connections, transactions, concurrency, multiple clients and so on. Hereby, the DIG interface makes a number of assumptions.

The core DIG is agnostic as to multiple client connections. Multi-threaded implementations of a reasoner may be provided, but no guarantees are made as to the semantics when clients attempt to simultaneously update and query. The connection to the reasoner is effectively stateless. There is no identification process between clients and reasoners. Neither the server nor the client has a way of ensuring that the state of its counterpart is the same since its last communication. There is no explicit classification request. The reasoner will decide when it is appropriate to, for example, build a classification hierarchy of concepts. This may happen incrementally after each TELLS request, alternatively it may be done only when necessary or even when there is a lull in traffic. The specification is not intended as a "database system" for knowledge bases. It is simply a protocol that exposes and utilises the reasoning capabilities provided by a DL reasoner. A DIG reasoner or DIG extensions may be used to implement services that provide concurrent access, transactions etc, such functionality is not inherently supported by DIG.

DIG 2.0 is intended to be a *lightweight* mechanism providing access to reasoning functionality, and not to cover everything that reasoning services might need. A DIG reasoner is expected to be a part of a larger architecture. The underlying language for DIG 2.0 is designed to *support OWL 1.1* so that DIG reasoners can fully support applications making use of OWL DL or OWL 1.1. The support also extends to some metalogical features such as annotations, which can be parsed in, but are ignored by the DIG 2.0 reasoner during reasoning.

It is worth noting that the combination of all DIG 2.0 language elements is computationally undecidable. This is, however, not a problem since not all the language elements are to be used simultaneously. In fact, different DL reasoners (applications) support (require) different logics and different reasoning facilities. To partly support these differences and also to cope with undecidability, DIG 2.0 predefines some well-known *DL dialects*, for example, $\mathcal{SHOIQ}$ [13] and $\mathcal{EL}^{++}$ [14]. Additionally, DIG 2.0 is designed to be as *flexible* as possible to embrace upcoming needs, while most common features are standardised and governed by DIG 2.0 core XML Schema and additional functionality can be proposed in terms of *DIG 2.0 extensions* (See Section 3). It should also be possible through extensibility mechanism to define a new DL dialect.

The core DIG can be specified by a XML Schema for a DL concept language, TELL/ASK functionality and a description of a protocol used to communicate these operations. Like many other initiatives, DIG builds a messaging protocol using XML and uses HTTP as the underlying transfer protocol. The messaging protocol essentially consists of request and response. Similar to DIG 1.1, each of these is an XML document with one root element. The server will use the root element of the message to determine the message type, namely, CONFIGURATION, MANAGEMENT and AXIOMS for ASKS and TELLS .

A number of CONFIGURATION requests are available to allow clients to interrogate a reasoner to discover its identity, and adjust settings of the reasoner (such as optimisations). With MANAGEMENT, clients are able to create or release a knowledge base. To support identification of different knowledge bases, URIs are used. When a request for a new knowledge base is made, the reasoner (if successful) will return to the client a URI which the client can then use to identify the knowledge base during TELLS /ASKS requests or to release it.

The central part of DIG 2.0 is, of course, the requests that manipulate and access knowledge bases. TELLS requests, several of which can be bundled into a single message, allow the construction of arbitrary OWL 1.1 (or, equivalently, $\mathcal{SROIQ}(\mathcal{D})^{-}$) knowledge bases. Basically, each TELLS request adds an OWL 1.1 axiom to the current or named knowledge base, in the XML syntax[1] for the forthcoming OWL 1.1 standard. Identifiers can also be supplied for each axiom. Responses to TELLS requests simply acknowledge receipt.

[1] Available under http://dig.cs.manchester.ac.uk/; whilst small differences between the current DIG 2.0 and OWL 1.1 XML syntaxes are being ironed out.

```
<dig>
   ⋮
   <tells>
      <equivalentClasses>
         <class URI="Herbivore"/>
         <intersectionOf>
            <class URI="Animal">
            <allValuesFrom>
               <objectProperty URI="eats"/>
               <class URI="Plant"/>
            </allValuesFrom>
         </intersectionOf>
      </equivalentClasses>
   </tells>
   ⋮
   <asks>
      <isSatisfiable ID="id01">
         <class URI="Herbivore"/>
      </isSatisfiable>
   </asks>
</dig>
```

Fig. 1: DIG 2.0 TELLS/ASKS request

ASKS request statements, which can similarly be bundled into a single DIG 2.0 message, correspond to the standard Description Logic inference services in a knowledge base (either the current knowledge base or the one named), including satisfiability, subsumption, instance checking and finding parents and children in the subsumption hierarchy. Responses to ASKS requests are either simple boolean answers or sets of names from the knowledge base. Identifiers can be supplied for each ASKS request and these identifiers are used in the response statement to match parts of the response to the ASKS request. The specifications of the TELLS /ASKS requests are quite standard and are not given here in detail to save space. Instead, we illustrate TELLS/ASKS statements in Figure 1 by a small example defining the concept "Herbivore" and asking whether its definition is satisfiable.

3 Current DIG 2.0 Extensions

Different DL reasoners support different DLs or reasoning facilities and possibly provide other services which are beyond standard DIG 2.0. One can think of particular query languages, non-standard inference services, or specific KB management as extra functionality. Even if those services or languages are not in the common intersection of all DL systems, it is a declared goal of DIG 2.0 to provide a mechanism which allows system independent access to such functionality. Therefore, DIG 2.0 provides an extensibility mechanism in order to define interfaces which provide additional service features or languages, which might be needed in a Semantic Web application using a Web Ontology Language similar to OWL. This utilises the DIG communication channel for almost any kind of extended functionality without being forced to establish a proprietary link-up for commands which are not within the least common service set.

Technically an extension consists of two documents namely an XML Schema document defining the syntax of messages and an associated HTML document providing a detailed account of the extension. The first specifies the offered service syntactically. The latter should, at a minimum, describe the service results as well as error messages of the extension and provide meaning for all the messages supported in the extension to a level of detail sufficient to support implementation and usage of the extension. Both documents should be available on the web, and, by convention, their URIs must differ only in their extension (i.e. XSD resp. HTML extension). Note that it depends on the extension whether it either defines additional ASKS operators (such as a query language), TELLS operators (such as data processable by a specialised calculus), management operations (e.g. to temporarily dump the knowledge base in a reasoner-dependent format or a transaction model), or if the extension provides further reasoning services (such as explaining, debugging, etc.).

The DL Implementation Group has developed a selection of extensions, which provide interfaces to some functionality the DL community has frequently been asked for in the past. We therefore expect these extensions to become a first standard set of extensions supported by many DIG servers. Note however, that there is no official distinction between standardised and non-standardised extensions.

In the following we will take a deeper look at some of the standard extensions developed by the working group. Here, we will refer to the extension-free functionality offered by DIG 2.0 as the core DIG.

3.1 Told Information Access Extension

In many applications it is useful to be able to access the unprocessed information sent to a Description Logic reasoner. Examples of such applications are adjunct reasoning components which debug KBs or explain inferences as well as many other non-standard reasoning services. In addition, providing access to previously submitted information by a DIG application would allow a third component to readout axioms and assertions for back up purposes or visualisation in parallel.

To this end, the DIG 2.0 told extension has been defined which provides the ability to retrieve the information that has been explicitly given to the reasoner as axioms or assertions. This allows clients to distinguish between explicitly given and inferred information.

The told extension comes with two levels: a set of canned queries at level 1 and freely definable queries at level 2. Level 1 is intended to provide the basic set of queries sufficient for most told-aware applications and offers the essential part of the told interface. It contains queries in order to retrieve axioms about classes, properties, assertions about individuals as well as general concept inclusion axioms (GCIs). In addition, there are some more sophisticated told ABox queries for retrieving property fillers, related individuals, etc. All of these queries retrieve told information on top-level granularity. This means that they only address entire axioms rather than fractions. The returned axioms are structurally exactly as given in previous DIG 2.0 TELLS messages following an axiom preservation approach.

Level 2 is a superset of level 1 providing unrestricted access to any portions of previous told KB data for all those applications which need more flexibility than available within level 1.

3.2 Retraction Mechanism

In dynamic applications, ABox assertions will inherently change over time. One example here are so-called "context- and situation-aware" applications in the mobile world. Here, the user's context is used to reason upon his constantly changing situation. As a consequence, the corresponding ABox needs to be frequently updated in terms of additions and retractions of assertions. Beyond that, in the area of ontology authoring the retraction of TBox definitions has been considered as useful.

This does not only apply to ontology editing tools which provide instant-reasoning feedback [3] but also to non-standard inference services such as diagnosing inconsistencies using a "black-box" approach in which the reasoner is frequently used as an oracle for a certain set of inferences about a decreasing set of KB axioms [15].

The core DIG 2.0 specification defines a communication interface which reflects some kind of batch-oriented reasoning procedure that builds up a knowledge base monotonically. Note that it is always possible to interleave queries and new definitions and assertions. However, the only procedure for removing information is by starting over from an empty knowledge base and retransmitting the non-changed axioms and facts. As a matter of course, such a procedure can only be considered as a work-around because the ratio between retracted information and the non-changed part of the ontology usually tends to be very low. In addition, in the case of large ontologies such an approach is very time-consuming with respect to the serialisation, transmission, and un-serialisation work flow. The DIG 2.0 retraction extension provides a simple method for retracting information from a KB. Even if current reasoner implementations need to perform a complete reclassification regardless of the amount or number of retractions, this extension at least minimises the communication overhead.

The retraction extension is a straightforward mechanism for withdrawing previously added axioms and assertions from the KB. It only allows to retract top-level statements syntactically rather than semantical entities such as classes or individuals in order to avoid side-effects. For instance, the consequences of retracting the class B as a semantical entity in the presents of the axioms $A \sqsubseteq B$ and $B \sqsubseteq C$ is not obvious and requires a formal update semantics. Replacing B by the most general class $\top$ would result in an equivalence of $\top$ and C whereas one very likely would expect the result $A \sqsubseteq C$.

Therefore, retraction is restricted to the entire removal of top-level axioms and assertions with the semantics of deleting this statement syntactically from the set of previously told statements. For this reason the retraction extension is build upon the told extension. If a client wants to retract an axiom or fact, the retraction mechanism checks whether the expression actually has been previously told and in case of a match it is deleted from the KB.

3.3 Extensions providing Non-standard Inferences

The so-called non-standard inferences (NSI) are a collection of relatively new inferences, that are mainly applied to extend and maintain DL knowledge bases.

There are currently two proposals for DIG 2.0 NSI extensions – one for Explanations and one for a collection of different NSIs containing *least common subsumer* (LCS), *approximation*, *rewriting* and *matching* [16].

The explanation extension offers an interface to inference services of explaining certain logical entailments, e.g., concept unsatisfiability. The current proposal covers only explanation by means of axiom pinpointing, in which the response to an explanation query is essentially a minimal set of told axioms (or their IDs) that are responsible for the logical entailment in question.

The other NSI extension offers for example computing the LCS of a set of concepts, which yields a concept that is a generalisation of all input concepts. Thus for ontologies with a flat concept hierarchy computing the LCS can provide concepts that augment the concept hierarchy by an intermediate level which helps browsing the ontology. Obviously, this non-standard inferences extension adds new ASKS statements for each NSI. Besides this the core DIG has to be extended by a mechanism to declare *concept patterns* and *variables*, which are the input for the matching reasoning service. Most NSIs from this extension return complex concept descriptions and not Boolean values as most standard inferences do. These NSI take into account the "target DL" for which they are applied. For example concept approximation "translates" concept descriptions from one DL L_1 to another DL L_2. Thus the extension must provide means to specify a target DL.

While methods for Explanation often need to be integrated in a DL standard reasoner, since they access the internal data structures, the other NSIs can be realised as a stand-alone application. However, they require access to told information and to the DL standard reasoner. Thus such NSI extensions will be realised as a DIG server and a DIG client at the same time.

The first system that offers this collection of NSI services is Sonic [17]. Besides the DIG server, Sonic implements also a DIG 2.0 NSI extension client by providing a plug-in for ontology editors Protégé and OilEd.

3.4 Query Interface Extension

In many practical applications based on DLs, a powerful ABox query language is one of the main requirements. Such a query language is provided in DIG 2.0. A query consists of a *head* and a *body*. The head lists variables for which the user would like to compute bindings. The body consists of query atoms (see below) in which all variables from the head must be mentioned. If the body contains additional variables, they are seen as existentially quantified. A query answer is a set of tuples representing bindings for variables mentioned in the head.

Query atoms can be *concept* query atoms, *role* query atoms, *same-as* query atoms as well as *concrete domain* query atoms. The latter are introduced to provide support for querying the concrete domain part of a knowledge base. Complex queries are built from query atoms using boolean constructs for conjunction, union and negation (for the latter, for instance, negation as failure semantics is assumed). In addition, a *projection* operator is supported in order to reduce the dimensionality of an intermediate tuple set. This operator is particularly important in combination with negation (complement).

In the literature, two different semantics for these kinds of queries are discussed. In *standard* conjunctive queries, variables are bound to (possibly anonymous) domain objects. In so-called *grounded* conjunctive queries, variables are bound to named domain objects (object constants). However, in grounded conjunctive queries the standard semantics can be obtained for so-called tree-shape queries by using existential restrictions in query atoms. For an ABox query language as part of DIG 2.0 we do not commit to a certain semantics. The semantics depends on the reasoner. The standard semantics is implemented in the system QuOnto [18] (for a Description Logic of the DL-Lite family) whereas the grounded semantics is implemented in RacerPro [19] and also KAON2 [9]. For all systems, an interface corresponding to the DIG 2.0 standard will be developed. A possibility to identify the fragment of the query language supported by a reasoner is foreseen in the DIG 2.0 protocol.

In case a reasoner has to deal large result sets for queries, iterative query answering can help to improve performance. Therefore, in DIG 2.0, result sets can be retrieved iteratively using small chunks of tuples. A reasoner can indicate if resource consumption is likely to increase if further tuples are retrieved. In addition, DIG 2.0 supports instructions to let a query answering engine compute results proactively to support faster retrieval of subsequent chunks of triples.

4 Reference Middleware

With the new DIG specification system developers face the challenge of adapting their implementations to the new interface. To support this transition the DL Implementation Group decided to develop a reference middleware, rather than reference implementations of DIG 2.0 clients and servers. The purpose of the middleware is to lower the burden of adapting the new specification, to provide the developers a reliable tool for testing, and finally to constitute a proof of concept for the new specification.

The DIG 2.0 Reference Middleware is able to extend a DIG 1.1 reasoner or client with a DIG 2.0 interface. Therefore it mediates between a DIG client and a DIG server regardless of the version they implement. Internally, a Told Knowledge Base (ToldKB) keeps track of previously sent axioms, able to process told queries and retraction requests. This way, the middleware also provides an implementation of the DIG 2.0 told and retraction extensions. To summarise, the DIG 2.0 Reference Middleware offers the following functionalities:

– Mediation between DIG 1.1 and DIG 2.0 components
– Query processing for told information
– Realization of a naive retraction mechanism

The processing steps of the middleware (depicted in Figure 2) are as follows: Received DIG requests (1) are translated into the internal DIG 2.0 representation (if necessary). Axioms are copied into the ToldKB and told queries as well as

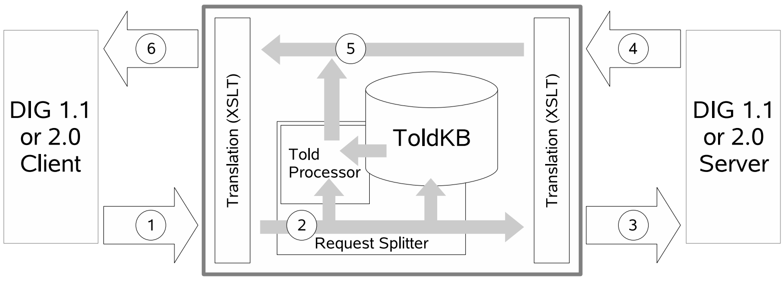

Fig. 2. DIG 2.0 Reference Middleware Architecture

retraction requests are processed (2). The reminder of the request is sent to the reasoner (3). The answer received from the reasoner (4) is merged with the results of the internal processing (5), translated (if necessary) and sent back to the DIG client (6).

The ToldKB is used to answer told requests, in case the reasoner itself is unable to do so. If a told query is identified within a DIG message, the ToldKB is scanned for matching axioms which are then returned.

The ToldKB also enables retraction, if the connected reasoner does not support this extension natively. Retraction requests are applied to the ToldKB. Once all corresponding axioms have been removed, the resulting ToldKB is sent to the reasoner, replacing the previously loaded knowledge base.

5 Open Issues and Future Work

The on-going work for the DIG interface regards the expressivity of the DLs supported by the interface as well as general structural issues. The core DIG is to be extended to include also "extralogical" language features that are covered by the OWL 1.1 standard, such as annotations and deprecations. Furthermore, there is a proposal for concrete domains extension.

One of the open issues for DIG 2.0 is the choice of the underlying protocol. It is planned to offer different realisations (such as for example WSDL or even in process communication) for that. For an overview of the recent discussion around the DIG 2.0 interface please refer to the DIG document index at http://dig.cs.manchester.ac.uk/.

Acknowledgements

We would like to thank the following DIG working group members for their contribution: Yusri Bong, Sebastian Böhm, Matthew Horridge, Ian Horrocks and Boris Motik. This work was partially supported by EU FET Project Thinking ONtologiES –TONES (grant IST-2005-7603).

References

1. Bechhofer, S., Möller, R., Crowther, P.: The DIG Description Logic Interface. In: Proc. of the 2003 Description Logic Workshop (DL'03), Rome, Italy (2003)
2. Bechhofer, S.: The DIG Description Logic interface: DIG/1.1. Available from http://dl-web.man.ac.uk/dig/2003/02/interface.pdf (2003)
3. Liebig, T., Noppens, O.: OntoTrack: A semantic approach for ontology authoring. Journal of Web Semantics **3**(2-3) (2005) 116–131 OntoTrack download page: http://www.informatik.uni-ulm.de/ki/ontotrack/.
4. Bechhofer, S., Horrocks, I., Turi, D.: The OWL instance store: System description. In Proc. of the 22th Conf. on Automated Deduction (CADE-22). LNCS, Springer (2005) 177–181 Instance Store download page: http://instancestore.man.ac.uk/.
5. Haarslev, V., Lu, Y., Shiri, N.: OntoXpl - intelligent exploration of OWL ontologies. In: Proc. of the IEEE/WIC/ACM Int. Conf. on Web Intelligence (WI'04). (2004)
6. McBride, B.: Jena: A semantic web toolkit. IEEE Internet Computing **6**(6) (2002)
7. Baader, F., Lutz, C., Suntisrivaraporn, B.: CEL—a polynomial-time reasoner for life science ontologies. In Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06). Springer (2006) CEL download page: http://lat.inf.tu-dresden.de/systems/cel/.
8. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06). (2006). FaCT++ download page: http://owl.man.ac.uk/factplusplus/.
9. Motik, B., Sattler, U.: A Comparison of Techniques for Querying Large Description Logic ABoxes. In : Proc. of the 13th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06). Springer (2006) To appear. KAON2 download page: http://kaon2.semanticweb.org/.
10. Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In Proc. of the 2004 Description Logic Workshop (DL'04). (2004) Pellet download page http://www.mindswap.org/2003/pellet/.
11. Haarslev, V., Möller, R.: RACER system description. In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR'01). LNCS, Springer (2001)
12. Dickinson, I.: Implementation experience with the DIG 1.1 specification. Tech. Report HPL-2004-85, Hewlett Packard, Digital Media Sys. Labs, Bristol (2004).
13. Horrocks, I., Sattler, U.: A tableaux decision procedure for $\mathcal{SHOIQ}$. In: Proc. of the 19th Int. Joint Conf. on Art. Intelligence (IJCAI'05), Morgan Kaufm. (2005)
14. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05), Morgan Kaufmann (2005)
15. Parsia, B., Sirin, E., Kalyanpur, A.: Debuggin OWL Ontologies. In: Proc. of the 14th Int. World Wide Web Conference (WWW'05), Chiba, Japan (2005)
16. Brandt, S., Turhan, A.Y.: Using non-standard inferences in description logics — what does it buy me? In Proc. of the 2001 Applications of DL Workshop (ADL 2001). CEUR Workshop (2001)
17. Turhan, A.-Y.: Pushing the SONIC border – SONIC 1.0. In Proc. of 5th Int. Workshop on First-Order Theorem Proving (FTP'05), Tech. Rep. Univ. of Koblenz (2005), SONIC download page: http://lat.inf.tu-dresden.de/systems/sonic.
18. Acciarri, A., Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: Querying ontologies. In Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI'05). (2005) QuOnto web page http://www.dis.uniroma1.it/~quonto/index.htm.
19. Wessel, M., Möller, R.: A High Performance Semantic Web Query Answering Engine. In Proc. of the 2005 Description Logics Workshop (DL'05). (2005)