An Integrated Environment for Reasoning over Ontologies via Logic Programming*

Barbara Nardi, Kristian Reale, Francesco Ricca, and Giorgio Terracina

Department of Mathematics, University of Calabria, Italy
{b.nardi,reale,ricca,terracina}@mat.unical.it

Abstract. Ontology-based reasoning is considered a crucial task in the area of knowledge management. In this context, the interest in approaches that resort to Datalog (and its extensions) for implementing various reasoning tasks over ontologies is growing. Nonetheless, looking from the developer point of view, one can notice that the editing environments for ontologies on the one hand and Datalog-like logic programs on the other hand are often developed independently and miss a common perspective. In this paper we face with this issue proposing the integration of two major development environments for Datalog programs and Ontologies, respectively: *ASPIDE* and *protégé*. We extended both systems with specific plugins that enable a synergic interaction between the two development environments. The developer can then handle both ontologies and logic-based reasoning over them by exploiting specific tools integrated to work together.

Introduction

In the area of knowledge management, ontology-based reasoning is becoming more and more a relevant task [1, 2]. New Semantic Web repositories are continuously built either from scratch or by translation of existing data in ontological form and are made publicly available. These repositories are often encoded by using W3C [3] standard languages like RDF(S), and OWL, and query answering on such repositories can be carried out with specific reasoners, supporting SPARQL as the query language.

In this context, the interest in approaches that resort to Datalog (and its extensions) for implementing various reasoning tasks over ontologies is growing. Consider for instance that recent studies have identified large classes of queries over ontologies that can be Datalog-rewritable (see [4] for an overview) or First-Order Rewritable [5]. Approaches dealing with such fragments usually rely on query reformulation, where the original query posed on the ontology is rewritten into an equivalent set of rules/queries that can be evaluated directly on the ontology instances. Many query rewriters that are based on this idea exist [6–10] producing SQL queries or stratified Datalog programs. Moreover, even considering a setting where SPARQL queries are posed on RDF repositories, translations to Datalog with negation as failure were proposed [12] and implemented [13].

* This work has been partially supported by the Calabrian Region under PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project KnowRex POR Calabria FESR 2007-2013 approved in BURC n. 49 s.s. n. 1 16/12/2010.

W. Faber and D. Lembo (Eds.): RR 2013, LNCS 7994, pp. 253–258, 2013.
© Springer-Verlag Berlin Heidelberg 2013

However, if we look at this scenario from a developer point of view, one can notice that different families of tools are required. On the one hand, one needs a good environment for designing and editing ontologies. On the other hand one would like to design, execute and test Datalog programs for ontology reasoning. Unluckily specific tools for these tasks are currently developed independently and miss a common perspective.

In this work we face with this issue proposing the integration of two major development environments for Datalog programs and Ontology editing, respectively: *ASPIDE* [14] and *protégé* [15]. *Protégé* being one of the most diffused environments for the design and the exploitation of ontologies; and *ASPIDE* being the most comprehensive IDE for Datalog extended with non monotonic negation and disjunction under the stable model semantics [16]. In particular, *protégé* is an open source ontology editor and knowledge-base framework, which (i) provides an environment for ontology-based prototyping and application development; (ii) supports several common formats (such as RDF(S), OWL); (iii) supports several querying and reasoning engines; and (iv) can be extended with user-defined plugins. *ASPIDE* supports the entire life-cycle of Datalog programs development, from (assisted) program editing to application deployment. *ASPIDE* combines an advanced editing tool with a collection of user-friendly graphical tools for program composition, debugging, testing, profiling, DBMS access, solver execution configuration and output-handling. Nonetheless, *ASPIDE* can be extended with user-defined plugins [17] to support: (i) new input formats, (ii) program rewritings, and even (iii) the customization of solver results.

In this paper we present an extension of both editors with specific plugins that enable a synergic interaction between them. The user can, thus, handle both ontologies and Datalog-based reasoning by exploiting specific tools integrated to work together. Note that, our solution has to be considered as a first step towards the development of a general platform, which can be personalized and extended (also with the help of the research community) by integrating additional rewriters/reasoners. The aim is to provide an environment for developing, running and testing Datalog-based ontology reasoning tools and their applications.

1 Integrating *ASPIDE* and *protégé*

The integration between *ASPIDE* and *protégé* is obtained by developing two separate plugins respectively extending these systems, see Figure 1. Both plugins are developed according to the following principle: simple modifications to ontologies and logic programs should be possible in both environments, but the user can switch to the most specific editor seamlessly.

As far as *ASPIDE* is concerned, we developed an input plugin [17] that recognizes and takes care of the ontology file types. The *ASPIDE* plugin offers two editing modalities for ontology files, which can be selected by clicking on the standard "switch button" of *ASPIDE*. The first modality opens a simple text editor embedded in *ASPIDE*, the second modality automatically opens the selected file in *protégé*. The plugin can also associate ontology files with some specific query rewriter, which is available in *ASPIDE* in the form of a rewriting plugin. Clearly new rewriters can be added to the system by developing additional rewriting plugins. *ASPIDE* is equipped with run

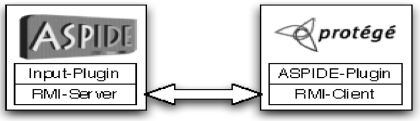


Fig. 1. Integration of *ASPIDE* and *protégé*

configurations. A run configuration allows to setup a Datalog engine with its invocation options, to select input files and to (possibly) specify the associated rewriters.

Concerning the *protégé* side, we developed a plugin for *protégé* that displays the currently open *ASPIDE* workspace[1] in the usual tree-like structure. The idea is that the *ASPIDE* workspace acts as a common repository for Datalog programs and ontologies. The user can browse, add, remove or modify files in the workspace from *protégé*. Datalog programs can be modified in *protégé* by using a simple text editor in the plugin panel, whereas ontology files in the workspace are open and displayed in *protégé* as usual. The user can also: require to open specific files in *ASPIDE*, execute one of the available rewriters, and invoke a Datalog reasoning engine by setting up and executing an *ASPIDE* run configuration.

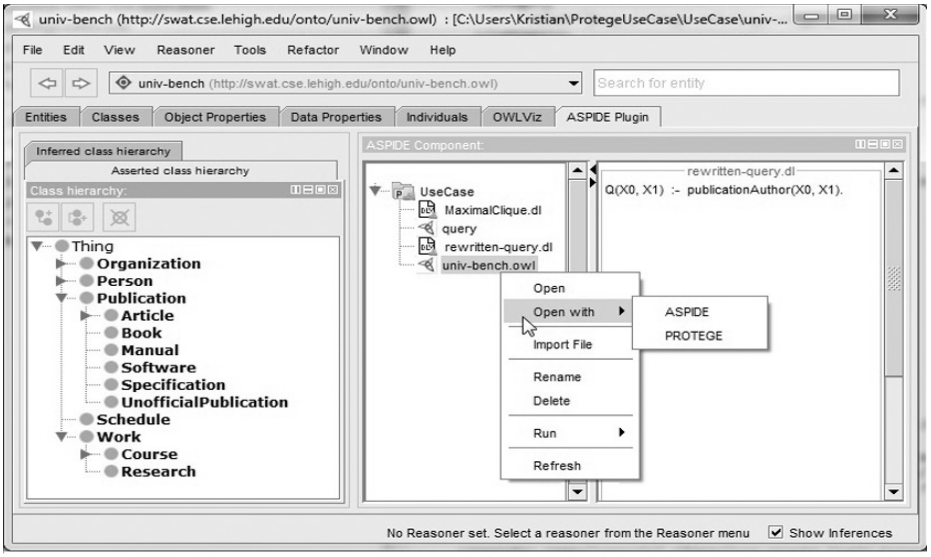
The two plugins are connected each other via Java RMI. In particular, the *ASPIDE* plugin acts as a server. It publishes a remote interface that allows the *protégé* plugin to access the *ASPIDE* workspace and to require the execution of available commands. The same remote interface is exploited by *ASPIDE* to open ontologies in *protégé*.

2 Some Use Cases

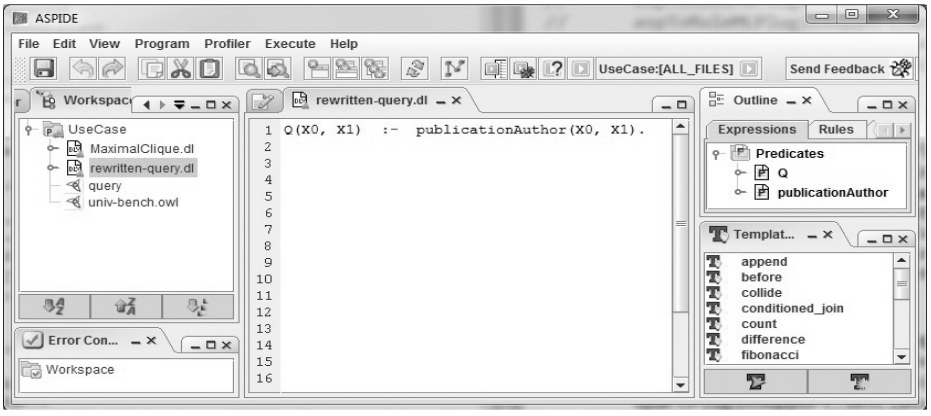
In this section we consider some use cases possibly involving three kind of users interacting with our platform, namely: (i) an ontology engineer, (ii) an engineer of query rewritings for ontologies, and (iii) a Datalog specialist using ontologies for reasoning tasks. In our description we refer to the well known Datalog-based rewriters, Requiem [8] and Presto [9]. The first has been already integrated in our platform as a rewriting plugins of *ASPIDE*, and we are already working to integrate also Presto. Moreover, for the sake of presentation, we refer to some instantiation of the well known LUBM ontology. First of all, consider an ontology engineer, whose main objective is the design/update of ontologies. In this case the user starts his session from *protégé* by opening the ontology and modifying it with standard *protégé* tools. Then, in order to check the result of some reasoning task on the ontology, to be carried out with Requiem or Presto, the *ASPIDE* plugin can be opened inside *protégé* and used to select a run configuration choosing the desired rewriter and Datalog engine. The plugin allows also to inspect the produced rewriting before/after the execution, as well as the query result in the output panel of *ASPIDE*. Figure 2(a) shows LUBM opened in *protégé* and a Requiem query rewriting shown in the ASPIDE panel inside *protégé*.

Let us now consider a rewriting engineer, whose main objective is the design/improvement of Datalog-based rewritings for ontology reasoning. In this case, his session

[1] The workspace of *ASPIDE* is a directory collecting programs and files organized in projects.



(a) ASPIDE plugin for Protégé.



(b) Protégé input plugin for ASPIDE, and Requiem rewriting.

Fig. 2. *ASPIDE* and *protégé* plugins at work

can start either from *protégé* or from *ASPIDE*. In fact, the *ASPIDE* plugin for *protégé* allows to open ontology files directly in *ASPIDE*, in order to perform basic inspection/modification tasks. Now, provided that an *ASPIDE* rewriting plugin is available, the user can activate the corresponding algorithm, inspect the resulting Datalog program, possibly modify it and run the Datalog engine to check the result. As a simple example, one could be interested in studying performance improvements possibly obtained on query answering by the application of magic-sets or unfolding strategies[2] applied in cascade, as a post-processing step, to a Requiem output program. This kind

[2] These are standard query optimization strategies for Datalog programs.

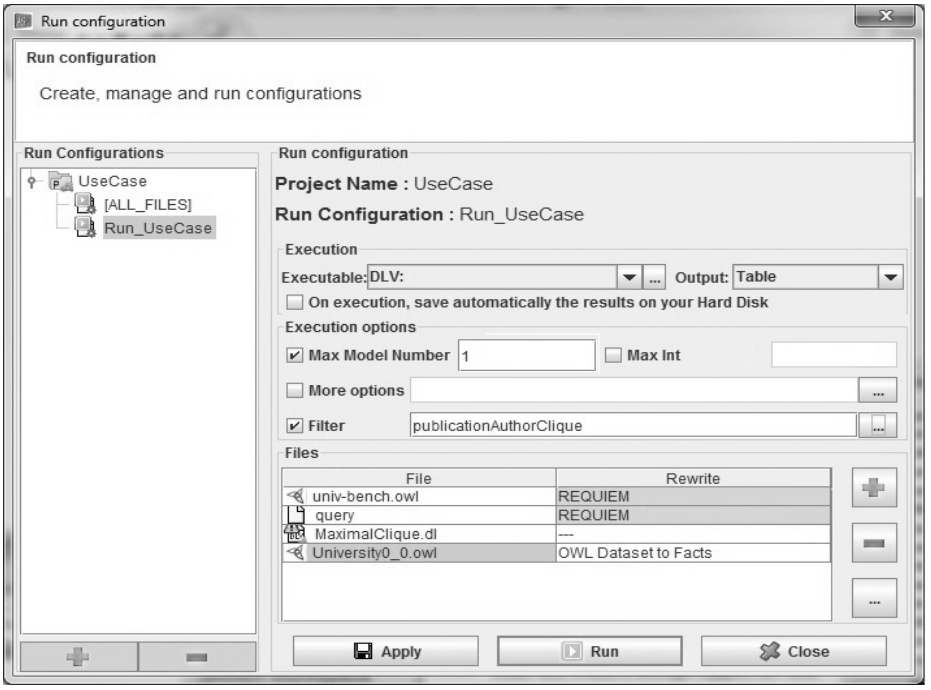


Fig. 3. Setting up a run configuration in *ASPIDE*

of analysis can be easily carried out with the proposed platform by properly setting two run configurations, one including and one excluding the post-processing step. As another example the rewriting engineer can be interested in checking the correctness and the performance of his brand new rewriting w.r.t. existing ones; again, this can be simply done by setting up different run configurations working on the same ontology.

Figure 2(b) illustrates the *ASPIDE* plugin at work on the LUBM ontology, with the Datalog program resulting from the activation of a rewriting plugin.

Finally, consider a Datalog specialist that needs to carry out complex reasoning tasks over ontologies. As an example, assume that the user is interested in identifying maximal cliques of coauthors in LUBM. LUBM provides both the *Person* and *Publication* concepts and the *publicationAuthor* role, which specifies the Authors (i.e. *Persons*) of each *Publication*. Finding a maximal clique of Authors is something that can be easily expressed with disjunctive Datalog extended with weak constraints,[3] provided the output of an ontology reasoner querying the ontology. Observe that, without our platform, the user should first infer authors and co-authorship relations from LUBM, then he should translate obtained results in a Datalog compliant format, and finally he should run a maximal clique encoding based on Datalog. To the contrary, by using the *protégé* plugin inside *ASPIDE* and a single *ASPIDE* run configuration it is sufficient to specify: the input ontology file, the queries needed to infer data of interest, the rewriting plugin to activate, the program for computing maximal cliques, and the Datalog

[3] We refer the reader to [11, 18] for more details on these extensions of Datalog.

evaluation engine supporting needed language extensions. Figure 3 shows the run configuration implementing the above example, where the DLV system [18] is selected as Datalog engine.

Implementations Availability. The two plugins are available in beta version for *ASPIDE* v. 1.35 and *protégé* v. 4.3. They can be dowloaded from the *ASPIDE* website `http://www.mat.unical.it/ricca/aspide`, and installed acting on the *ASPIDE* menu "File→Plugins→Manage Plugins".

References

1. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. In: Proc. of PODS 2009, pp. 77–86. ACM (2009)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The dl-lite family. Journal of Automated Reasoning 39(3), 385–429 (2007)
3. W3C: The World Wide Web Consortium (2012), `http://www.w3.org/`
4. Heymans, S., Eiter, T., Xiao, G.: Tractable reasoning with dl-programs over datalog-rewritable description logics. In: Proc. of ECAI 2010, pp. 35–40. IOS Press (2010)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. Artif. Intell. 195, 335–360 (2013)
6. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting for OWL 2 QL. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 192–206. Springer, Heidelberg (2011)
7. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QUONTO: querying ontologies. In: Proc. of AAAI 2005, vol. 4, pp. 1670–1671. AAAI Press (2005)
8. Pérez-Urbina, H., Motik, B., Horrocks, I.: A comparison of query rewriting techniques for dl-lite. In: Proceedings of the 22st International Workshop on Description Logics, DL 2009, vol. 477. CEUR-WS.org (2009)
9. Rosati, R., Almatelli, A.: Improving Query Answering over DL-Lite Ontologies. In: Proc. of KR 2010, pp. 290–300. AAAI Press (2010)
10. Stocker, M., Smith, M.: Owlgres: A scalable owl reasoner. In: Dolbear, C., Ruttenberg, A., Sattler, U. (eds.) OWLED 2008, vol. 432. CEUR-WS.org (2008)
11. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
12. Polleres, A.: From sparql to rules (and back). In: Proc. of WWW 2007, pp. 787–796. ACM (2007)
13. Ianni, G., Krennwallner, T., Martello, A., Polleres, A.: A rule system for querying persistent RDFS data. In: Aroyo, L., et al. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 857–862. Springer, Heidelberg (2009)
14. Febbraro, O., Reale, K., Ricca, F.: ASPIDE: Integrated development environment for answer set programming. In: Delgrande, J., Faber, W. (eds.) LPNMR 2011. LNCS, vol. 6645, pp. 317–330. Springer, Heidelberg (2011)
15. Stanford University: The Protégé Ontology Editor and Knowledge Acquisition System (2012), `http://protege.stanford.edu`
16. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)
17. Febbraro, O., Leone, N., Reale, K., Ricca, F.: Extending aspide with user-defined plugins. In: CILC, vol. 857, pp. 236–240. CEUR-WS.org (2012)
18. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)