DLEJena: A Practical Forward-Chaining OWL 2 RL Reasoner Combining Jena and Pellet

Georgios Meditskos [*], Nick Bassiliades

Department of Computer Science, Aristotle University of Thessaloniki, 54124, Thessaloniki, Greece

Abstract

This paper describes DLEJena, a practical reasoner for the OWL 2 RL profile that combines the forward-chaining rule engine of Jena and the Pellet DL reasoner. This combination is based on rule templates, instantiating at run-time a set of ABox OWL 2 RL/RDF Jena rules dedicated to a particular TBox that is handled by Pellet. The goal of DLEJena is to handle efficiently, through instantiated rules, the OWL 2 RL ontologies under direct semantics, where classes and properties cannot be at the same time individuals. The TBox semantics are treated by Pellet, reusing in that way efficient and sophisticated TBox DL reasoning algorithms. The experimental evaluation shows that DLEJena achieves more scalable ABox reasoning than the direct implementation of the OWL 2 RL/RDF rule set in the Jena's production rule engine, which is the main target of the system. DLEJena can be also used as a generic framework for applying an arbitrary number of entailments beyond the OWL 2 RL profile.

Key words: OWL 2 RL, DL reasoner, rule templates, forward-chaining rules

1 Introduction

The OWL 2 language [2] is an extension and revision of OWL that addresses several problems and drawbacks that have been identified throughout the years of the extensive application of OWL in numerous contexts. It provides three profiles with different expressivity, namely OWL 2 EL, OWL 2 QL and OWL 2 RL, each of which targeting at different application scenarios.

[*] Corresponding author.
Email addresses: gmeditsk@csd.auth.gr (Georgios Meditskos), nbassili@csd.auth.gr (Nick Bassiliades).

The OWL 2 RL profile imposes some restrictions on the use of OWL 2 constructs in order for several reasoning tasks to be implemented as a set of rules in a forward-chaining rule engine. It is realized as a partial axiomatization of the OWL 2 semantics in the form of first-order implications inspired by the pD^* semantics [13], called OWL 2 RL/RDF rules [1]. This partial axiomatization mainly ensures that only the explicitly stated individuals would be considered by the reasoning procedure, in a way similar to DLP [3]. For example, the `someValuesFrom` restriction is not allowed to be used on the right side of subclass axioms, since it refers to not explicitly stated individuals.

In this paper we describe DLEJena, a reasoning engine for OWL 2 RL ontologies that combines the inference capabilities of the production rule engine of Jena[1] and the Pellet DL reasoner [12], following the DLE framework [9]. The production rule engine is used in order to execute an instantiated version of the individual-related (ABox) OWL 2 RL/RDF rules that are generated for a particular ontology schema (TBox) after performing TBox reasoning with Pellet. The instantiation of the ABox OWL 2 RL/RDF rules targets at their efficient execution by the Jena's rule engine and it is defined through *template rules* that regulate the number and the type of the rules that are generated based on the TBox. In that way, DLEJena results in more scalable ABox reasoning than the direct implementation of the OWL 2 RL/RDF rule set in the production rule engine of Jena.

The approach of DLEJena to separate the TBox from the ABox reasoning procedure restricts the input graph in order to handle OWL 2 RL ontologies under direct semantics and therefore, the RDF graphs should satisfy the preconditions of Theorem PR1 in [1]. This fact, together with the utilization of a DL reasoner, gives the following characteristics to DLEJena.

(1) DLEJena is not an OWL 2 RL conformant implementation[2], since it cannot handle any RDF graph. Instead, it conforms to the practical subset of OWL 2 RL that separates the schema from the individual vocabulary.
(2) Since DLEJena uses a DL reasoner for TBox reasoning, it may derive more entailments than the ones that follow from the OWL 2 RL/RDF rule set. This depends on the expressivity of the loaded ontologies and it does not invalidate the conformance requirements.

The rest of the paper is structured as follows: in section 2 we present the concept and the application context of the DLE framework which DLEJena is based on. In section 3 we categorize the OWL 2 RL/RDF rules, according to the entailments that derive. In section 4 we describe the architecture of DLEJena, whereas in section 5, we present experimental results. Finally, in section 6, we conclude our work and we present possible extensions to DLEJena.

[1] `http://jena.sourceforge.net`
[2] `http://www.w3.org/TR/owl2-test`

2 The DLE Framework

In many practical applications, there is usually a shared terminological (TBox) vocabulary on which the ontology individuals (ABox) are defined. The TBox is not (or rarely) modified by applications at run-time, whereas the extensional part is continuously modified or enriched with new individual assertions. For example, the European and the International Semantic Web conferences publish metadata [3] that cover information relevant to papers, schedules, attendees, etc. These metadata are defined upon shared terminological vocabularies, such as the SWRC [4] ontology, and every year new individuals are added.

In such scenarios, the ontology vocabulary is partitioned, that is, the vocabulary of classes and properties is separated from the individuals vocabulary, since the classes and properties are not used at the same time as individuals. Furthermore, the extensional knowledge usually involves large number of individual assertions and therefore, there is a need for scalable implementations. The DLE framework targets at the efficient execution of the individual-related RDF triple-based inference rules in such application domains by defining a framework for the combination of a DL reasoner and a production rule engine. The motivation of such a combination is summarized in the following.

- **Degree of TBox completeness.** Many practical ontology reasoners [4],[5], [8], [6] implement forward-chaining inference rules in the form of RDF triple-based rules [13] for TBox and ABox reasoning, following the RDF Model Theory [11]. These approaches, like DLP, have limited TBox reasoning capabilities, since they are based on rules. For example, we have observed that they are unable to derive that two properties p and q are equivalent if p and q are both the inverse properties of g, because they do not implement the corresponding entailment rule [5] . The DLE framework, instead of using rules for TBox reasoning, it uses a DL reasoner for TBox reasoning completeness.
- **ABox reasoning performance.** The ABox reasoning procedure in DLE is based on instantiated individual-related inference rules. The experimental evaluation of DLE with a number of rule engines [9] has shown that the instantiated rules are executed faster with less memory requirements, compared to the execution of predefined, generic rules in the same rule engine.

The present paper describes the implementation of the DLE framework for the OWL 2 RL profile, using the Pellet DL reasoner and the forward-chaining rule engine of Jena. The choice of these two systems is justified by the convenient API that exists for their integration that provides the necessary communication infrastructure between the two systems.

[3] `http://data.semanticweb.org/`
[4] `http://ontoware.org/projects/swrc/`
[5] This entailment is neither supported by the OWL 2 RL/RDF rule set

DLEJena combines the reasoning paradigm that is based on the execution of RDF triple-based entailment rules in a forward-chaining rule engine [10] and the transformation paradigm of an ontology into a set of instantiated instance-related entailment rules. It should be noticed that the instantiation of these rules is based on TBox reasoning in order to allow DJEJena to handle an arbitrary number of entailments beyond the OWL 2 RL profile.

3 OWL 2 RL/RDF Rule Classification

DLEJena is based on the classification of the OWL 2 RL/RDF rules in three categories, according to the semantic conditions that are involved in each rule. This classification targets at the identification of the schema-related rules, whose semantics are implemented by Pellet, and the individual-related rules that are implemented as templates or ordinary Jena rules. In this section, we elaborate briefly on the characteristics of each category.

- **Terminological Rules.** The rules that deduce class and property relationships are referred to as *terminological.* These rules are not implemented in DLEJena and their semantics are handled by Pellet instead. For example, the rule that handles the subclass transitivity is a terminological rule, referred to as `scm-sco` in [1], which is expressed in Jena's rule syntax as:

```
[scm-sco: (?c1 rdfs:subClassOf ?c2)
          (?c2 rdfs:subClassOf ?c3)
          -> (?c1 rdfs:subClassOf ?c3)]
```

- **Hybrid Rules.** These rules deduce individual relationships by matching both TBox and ABox information in their body. As we explain in section 4.3.1, these rules are defined through templates in DLEJena that generate instantiated ABox rules. For example, the rule that defines the inverse property relationship, which is depicted below, is a hybrid rule.

```
[prp-inv1: (?p owl:inverseOf ?q)
           (?x ?p ?y)
           -> (?y ?q ?x)]
```

- **Exceptional Rules.** Finally, the rules that deduce ABox relationships by matching only ABox information in their body are referred to as *exceptional.* These rules are expressed in DLEJena directly as Jena rules, since they cannot be further instantiated, such as the `eq-sym` rule:

```
[eq-sym: (?x owl:sameAs ?y)
         -> (?y owl:sameAs ?x)]
```

The terminological rules are also referred as *TBox rules*, whereas the other two types constitute the *ABox rules* of DLEJena.

4 The DLEJena Architecture

The architecture of DLEJena is depicted in Fig. 1 and comprises four modules: the *Ontology Loader*, the *TBox Reasoner*, the *Template Processor* and the *ABox Reasoner*. DLEJena makes use of the Jena API and therefore, it supports all the Jena-specific interfaces for conducting queries, e.g. SPARQL queries.

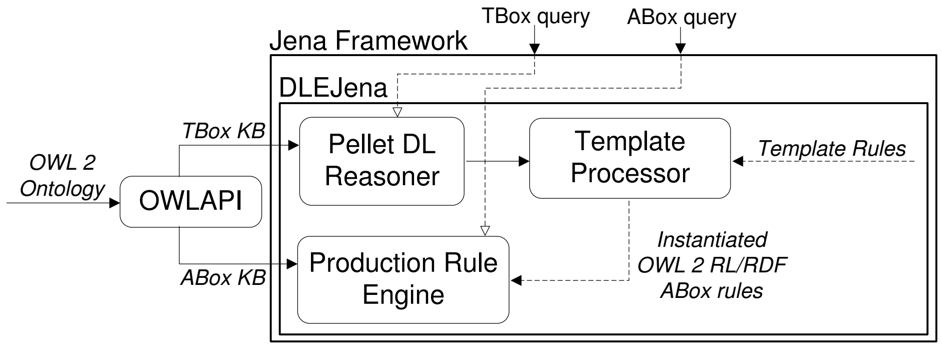

Fig. 1. The DLEJena architecture.

4.1 The Ontology Loader

The Ontology Loader is the module responsible for the separation of the TBox from the ABox axioms of the loaded ontology, since DLEJena applies different TBox and ABox ontology reasoning procedures. To achieve this, the Ontology Loader makes use of the OWLAPI [6], creating two Jena models for the asserted TBox and ABox triples, respectively. These two models serve as the base assertions for the two separated ontology reasoning procedures.

4.2 The TBox Reasoner

In contrast to existing rule-based implementations that use inference rules for TBox reasoning, such as the systems mentioned in section 2, the TBox reasoning in DLEJena is performed by the Pellet DL reasoner, without implementing any terminological OWL 2 RL/RDF rule (see section 3). The intuition behind

[6] `http://owlapi.sourceforge.net/`

this approach is that the TBox reasoning can be done efficiently, seamlessly and with greater degree of completeness by DL reasoners than by rules. In that way, Pellet reasons over the asserted TBox model of the imported ontologies that is created by the Ontology Loader, performing TBox consistency checking, computing the subsumption hierarchy and the property-related semantics. Note that the use of a DL reasoner may produce more entailments than the OWL 2 RL/RDF rules impose, as we have mentioned in the introduction.

4.3 The ABox Reasoner

The ABox reasoning procedure of DLEJena is defined upon the Jena's RETE-based rule engine and it is based on the hybrid and exceptional ABox rules mentioned in section 3 in order to achieve the scalability where the OWL 2 RL profile aims at. In the following, we describe the approach that DLEJena follows in order to implement the rules of each category.

4.3.1 Hybrid rules

DLEJena implements the hybrid rules following an approach based on *template rules.* Each hybrid rule is expressed as a template rule that generates instantiated rules by removing all the TBox references from the condition of the hybrid rule and grounding the remaining unbound variables with actual TBox values. In that way, (a) more than one rule may be generated for a particular hybrid rule, and (b) the rule base of DLEJena is not predefined and it is formed at run-time, according to the schema constructs of the loaded ontology. A template rule is of the form (using the rule syntax of Jena):

$$\begin{array}{l}
\texttt{[name: (}s_1\ p_1\ o_1\texttt{)}_{\mathrm{T}_1} \ldots \texttt{(}s_n\ p_n\ o_n\texttt{)}_{\mathrm{T}_n} \\
\texttt{-> [name}'\texttt{: (}s'_1\ p'_1\ o'_1\texttt{)}_{\mathrm{A}_1} \ldots \texttt{(}s'_m\ p'_m\ o'_m\texttt{)}_{\mathrm{A}_m} \\
\qquad\qquad \texttt{-> (}s''_1\ p''_1\ o''_1\texttt{)}_{\mathrm{A}_1} \ldots \texttt{(}s''_k\ p''_k\ o''_k\texttt{)}_{\mathrm{A}_k}\texttt{]],}
\end{array}$$

where $\texttt{(s p o)}_{\mathrm{T_i}}$ is a triple that matches TBox information, $\texttt{(s p o)}_{\mathrm{A_i}}$ is a triple that matches ABox information and the s_i, s'_i and s''_i elements can be rule variables, nodes or blank nodes. In that way, the rule variables of the body range over TBox information and they are used in order to ground the corresponding rule variables of the head. Intuitively, a template rule can be viewed as a production rule that asserts dynamically other production rules. It is worth mentioning that the instantiated rules are in fact exceptional rules, since they do not refer to TBox information in their body. Therefore, a hybrid rule is substituted by one or more exceptional rules.

The Template Processor is responsible for executing the template rules against the TBox that is obtained from Pellet after reasoning and for collecting the instantiated ABox rules of their head. To achieve this, it makes use of the *hybrid rule engine* of Jena. The default reasoning capability of Jena involves backward-chaining execution of entailment rules in the hybrid rule engine that actually combines a RETE and an LP engine. The RETE engine can be used in order to instantiate only backward-chaining rules in the LP engine. DLEJena is based on the hybrid rule engine of Jena and extends it, allowing the RETE engine to derive also forward-chaining rules. The functionality of the Template Processor involves the following steps:

- The TBox is loaded in DLEJena in order to generate the Jena ontology model that has the Pellet as the underlying reasoner.
- Based on Pellet, DLEJena performs TBox reasoning and the ontology model is checked for TBox inconsistencies.
- The template rules are applied over the Jena ontology model, incorporating seamlessly any additional inferences that are computed by Pellet.

Example. Consider the following ontology in the OWL Functional Syntax that defines the inverse `hasParent` and `hasChild` object properties.

```
Declaration(ObjectProperty(hasParent))
Declaration(ObjectProperty(hasChild))
InverseObjectProperties(hasParent hasChild)
```

These are all TBox axioms and therefore, they will be handled by Pellet that applies the symmetry in the `inverseOf` axiom. In order to enable the processing of the inverse property semantics at the individual level, the `prp-inv1` hybrid rule of section 3 should be defined in the form of a template rule as:

```
[prp-inv1: (?p owl:inverseOf ?q)
-> [D_prp-inv1:(?x ?p ?y)
               -> (?y ?q ?x)]]
```

More specifically, the template rule matches the `inverseOf` property relationship in its body, generating the instantiated rules in its head. By executing the template rule in DLEJena, the Template Processor retrieves the following two (exceptional) forward-chaining ABox rules:

```
[D_prp-inv1:(?x hasParent ?y)
            -> (?y hasChild ?x)]

[D_prp-inv1:(?x hasChild ?y)
            -> (?y hasParent ?x)]
```

4.3.2 Exceptional Rules

The exceptional rules are the actual ABox inference rules that are executed by the forward-chaining rule engine of DLEJena. These are the rules that have been generated by the Template Processor after the rule instantiation procedure, together with the set of the native exceptional OWL 2 RL/RDF rules that we have described in section 3. Note that an exceptional rule is actually a template rule with an empty body, since none of its rule variables range over TBox information. In that way, the exceptional rules can be also represented as templates. For example, the exceptional `eq-sym` rule of section 3 can be defined in the form of a template rule as:

```
[eq-sym:
->[eq-sym:(?x owl:sameAs ?y)
          ->(?y owl:sameAs ?x)]]
```

5 Experiments

The intention of DLEJena is to be used as an alternative reasoning module inside the Jena framework in order to provide efficient memory-based forward-chaining implementation of the OWL 2 RL semantics through instantiated entailment rules. To this end, we performed a number of experiments with different ontologies, comparing DLEJena to the direct implementation of the OWL 2 RL/RDF rule set in the forward-chaining rule engine of Jena (Jena$_{fc\_rules}$). We used synthetically generated extensional datasets of the UOBM [7], Vicodi[7], Wine[8] and Semintec[9] ontologies. The experiments ran on a Linux machine with a Core 2 Quad processor at 2.50 GHz and 2 GB main memory.

Table 1 depicts the TBox reasoning time, that is, the time needed to derive all the schema-related inferences. In the case of DLEJena, this time involves the application of Pellet for TBox reasoning, whereas in the case of Jena$_{fc\_rules}$, this time involves the execution of the schema-related OWL 2 RL/RDF rules. Furthermore, Table 1 depicts the time that DLEJena needs to apply the template rules. As far as TBox reasoning is concerned, the two systems have similar performance, except for the Wine ontology where DLEJena terminates the TBox reasoning procedure considerably faster than Jena$_{fc\_rules}$. It is worth mentioning that DLEJena employs complete TBox reasoning, through the use of Pellet, whereas Jena$_{fc\_rules}$ applies only the schema-related semantics that follow from the OWL 2 RL/RDF rule set.

[7] `http://www.vicodi.org`

[8] `http://www.w3.org/TR/owl-guide/wine.rdf`

[9] `http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`

Table 1
The TBox reasoning times of the two implementations (in seconds)

		UOBM	**Vicodi**	**Wine**	**Semintec**
DLEJena	TBox	0.22	0.15	0.86	0.1
	Templates	0.81	0.67	1.9	0.5
	Total	1.03	0.82	2.8	0.6
Jena$_{fc\_rules}$	TBox	0.15	0.38	8.2	0.05

As far as the execution of the template rules is concerned, it introduces an extra overhead to the performance of DLEJena, which is not present in Jena$_{fc\_rules}$. However, this is an insignificant overhead compared to the difference in the ABox reasoning performance between the two implementations that we analyze in the following.

Fig. 2 depicts the ABox reasoning time of the two implementations, that is, the time needed to derive all the ABox inferences. In the case of DLEJena, this time involves the execution of the instantiated entailments, whereas in the case of Jena$_{fc\_rules}$, this time involves the execution of the predefined ABox OWL 2 RL/RDF rules. Note that both implementations use forward-chaining rules, following a complete materialization approach of the semantics.

DLEJena achieves a considerably faster ABox reasoning performance in all ontologies. For the UOBM ontology, DLEJena managed to reason on ∼993,000 triples using all the available main memory, whereas Jena$_{fc\_rules}$ reasoned on ∼200,000 with the same memory utilization. For the Wine ontology, Jena$_{fc\_rules}$ reasoned on ∼25,000 triples before reaching the memory limit, whereas DLEJena managed to reason on a dataset of ∼138,000 triples using 1 GB of main memory. For the Vicodi ontology, DLEJena managed to reason on ∼161,000 triples before reaching the memory limit while processing a dataset of ∼215,000 triples, whereas Jena$_{fc\_rules}$ was able to reason only on the dataset of ∼54,000 triples. Finally, for the Semintec ontology, DLEJena managed to reason on a dataset twice the size of the one processed by Jena$_{fc\_rules}$. To conclude, the same datasets were processed faster by DLEJena than by Jena$_{fc\_rules}$ with less memory requirements, enabling DLEJena to reason on more triples than Jena$_{fc\_rules}$ before reaching the memory limit.

For the completeness of the presentation, we present also in Figure 3 the ABox reasoning times of Bossam [4], OWLIM [5] and BaseVisor [8] that are based totally on rules. OWLIM is the fastest RDF triple-based rule reasoner using the highly scalable TRREE engine, whereas Bossam presents the worst performance. For the UOBM ontology, BaseVisor requires more time to reason than Jena$_{fc\_rules}$, in contrast to the Vicodi and Semintec ontologies, showing that

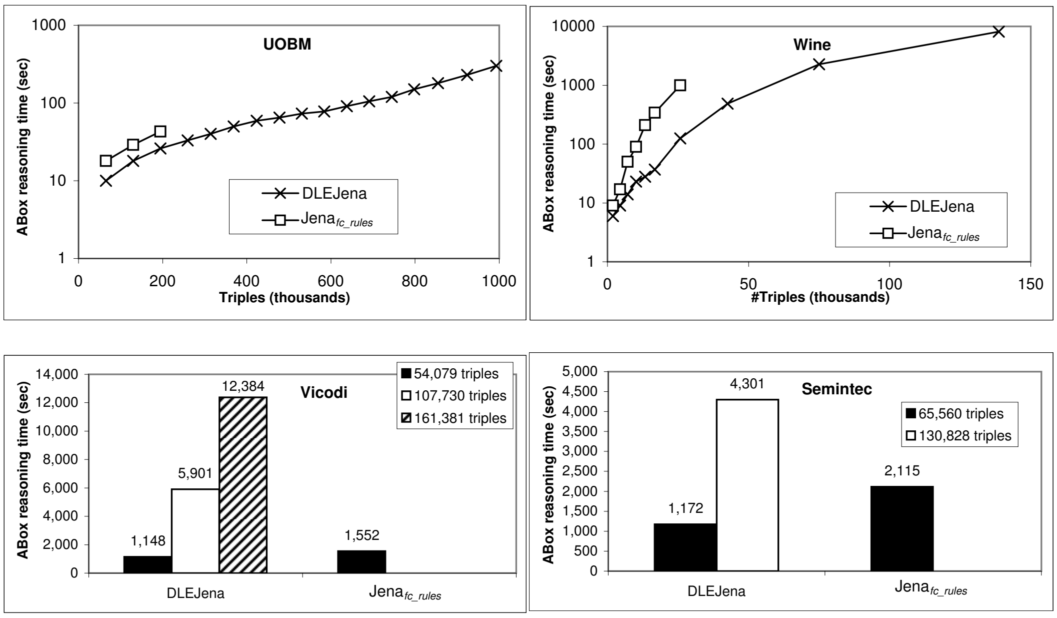

Fig. 2. ABox reasoning times (complete materialization of semantics).

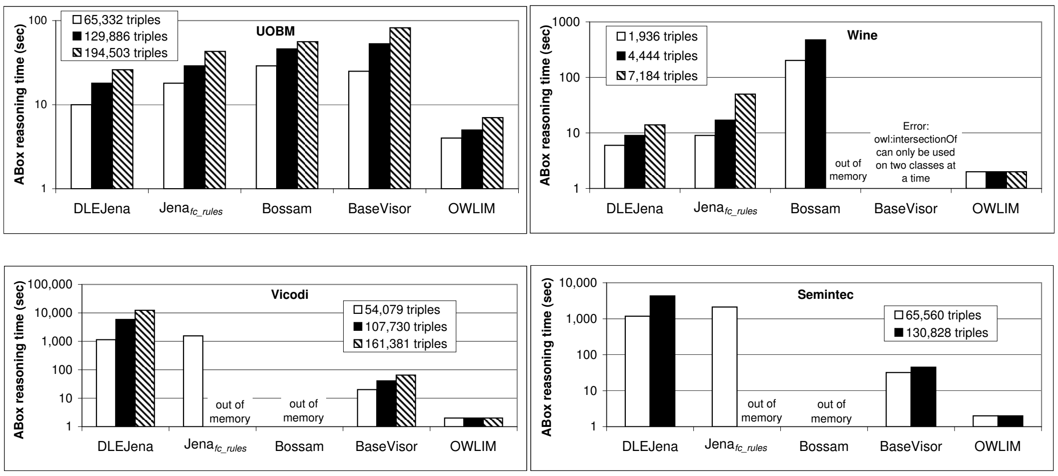

Fig. 3. Short comparison of the Jena-based systems to other implementations.

the rule engine of BaseVisor can handle more efficiently ontologies with simple TBox and large ABox than the rule engine of Jena, such as the Semintec and Vicodi ontologies. This behavior of the rule engine of Jena is inherited to DLEJena and therefore, the performance of DLEJena is bounded by the performance of Jena's rule engine. However, DLEJena has better performance than Jena$_{fc\_rules}$, which is the actual goal of DLEJena. It should be noted that our intention is not to compare the efficiency of different production rule engines, for example, to compare the Jena's rule engine against the BaseVisor's rule engine or the rule engine of OWLIM, since each implementation has different capabilities and features. To conclude, the idea behind DLEJena is to propose a practical framework in order to increase the reasoning performance of the typical implementation of the OWL 2 RL/RDF rules in Jena (Jena$_{fc\_rules}$) that DLEJena achieves for all the ontologies. Similar improvements were also observed for other implementations of the DLE framework in different rule engines (e.g. in Bossam) [9].

6 Discussion

In this paper we presented DLEJena, a proof of concept implementation of the DLE framework for the OWL 2 RL profile, using the Pellet DL reasoner and the production rule engine of Jena. The goal is to exploit the TBox reasoning capabilities of Pellet and to instantiate at run-time the ABox-related OWL 2 RL/RDF rules in order to result in a more scalable implementation in the Jena's production rule engine, than of applying directly the entailment rules.

Regarding the rule instantiation procedure of DLEJena, we have mentioned that it is defined upon the TBox model that is inferred by Pellet and not directly on the asserted knowledge. In that way, DLEJena can be used as a generic reasoning framework able to implement any type of entailments beyond the OWL 2 RL profile. However, the approach of DLEJena to apply the template rules after TBox reasoning, results in some redundant instance-related rules, which may affect the performance (not the correctness), but it still results in considerably better performance than the direct application of predefined entailments, as the experimental evaluation shows.

DLEJena is available[10] with a set of 36 preregistered template rules as a useful starting point for practical implementations. This set can be extended with more rules beyond the OWL 2 RL profile, such as with rules that infer the existence of individuals not present in the KB. We plan also to develop a version of DLEJena that would use a database for handling very large ontologies, similar to the approach of [14].

Acknowledgements

This work was partially supported by a PENED program (EPAN M.8.3.1, No. 03EΔ73), jointly funded by EU and the Greek Ministry of Development-GSRT

[10] `http://lpis.csd.auth.gr/systems/DLEJena/`

References

[1] OWL 2: Profiles, `http://www.w3.org/TR/owl2-profiles` (2009).

[2] B. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: The Next Step for OWL, Web Semantics: Science, Services and Agents on the World Wide Web 6 (4) (2008) 309–322.

[3] B. N. Grosof, I. Horrocks, R. Volz, S. Decker, Description Logic Programs: Combining Logic Programs with Description Logic, in: Proceedings of the 12th International Conference on World Wide Web, ACM, 2003.

[4] M. Jang, J.-C. Sohn, Bossam: An Extended Rule Engine for OWL Inferencing, in: G. Antoniou, H. Boley (eds.), RuleML, 2004.

[5] A. Kiryakov, D. Ognyanov, D. Manov, OWLIM - A Pragmatic Semantic Repository for OWL, in: WISE Workshops, 2005.

[6] J. Kopena, W. C. Regli, DAMLJessKB: A Tool for Reasoning with the Semantic Web, IEEE Intelligent Systems 18 (3) (2003) 74–77.

[7] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, S. Liu, Towards a Complete OWL Ontology Benchmark, in: European Semantic Web Conference, 2006.

[8] C. J. Matheus, K. Baclawski, M. M. Kokar, BaseVISor: A Triples-Based Inference Engine Outfitted to Process RuleML and R-Entailment Rules, in: RuleML, 2006.

[9] G. Meditskos, N. Bassiliades, Combining a DL Reasoner and a Rule Engine for Improving Entailment-Based OWL Reasoning, in: 7^{th} International Semantic Web Conference (ISWC 2008), Karlsruhe, Germany, 2008.

[10] G. Meditskos, N. Bassiliades, Rule-based OWL Reasoning Systems: Implementations, Strengths and Weaknesses, Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Global, ISBN Number 978-1-60566-402-6, 2009.

[11] J. Z. Pan, I. Horrocks, RDFS(FA): Connecting RDF(S) and OWL DL, IEEE Trans. on Knowl. and Data Eng. 19 (2) (2007) 192–206.

[12] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A Practical OWL-DL Reasoner, Web Semantics: Science, Services and Agents on the World Wide Web 5 (2) (2007) 51–53.

[13] H. J. ter Horst, Completeness, Decidability and Complexity of Entailment for RDFSchema and a Semantic Extension Involving the OWL Vocabulary, Web Semantics: Science, Services and Agents on the World Wide Web 3 (2-3) (2005) 79–115.

[14] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, J. Srinivasan, Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in ORACLE, in: 24th International Conference on Data Engineering (ICDE 2008), IEEE, 2008.