Improving OWL RL reasoning in N3 by using specialized rules

Dörthe Arndt[1], Ben De Meester[1], Pieter Bonte[2], Jeroen Schaballie[2], Jabran Bhatti[3], Wim Dereuddre[3], Ruben Verborgh[1], Femke Ongenae[2], Filip De Turck[2], Rik Van de Walle[1], and Erik Mannens[1]

[1] Ghent University – iMinds – Multimedia Lab, Belgium
{doerthe.arndt, ben.demeester, ruben.verborgh}@ugent.be
[2] IBCN research group, INTEC, Ghent University – iMinds, Belgium
{pieter.bonte, jeroen.schaballie, femke.ongenae}@intec.ugent.be
[3] Televic Healthcare, Belgium
{j.bhatti, w.dereuddre}@televic.com

Abstract. Semantic Web reasoning can be a complex task: depending on the amount of data and the ontologies involved, traditional OWL DL reasoners can be too slow to face problems in real time. An alternative is to use a rule-based reasoner together with the OWL RL/RDF rules as stated in the specification of the OWL 2 language profiles. In most cases this approach actually improves reasoning times, but due to the complexity of the rules, not as much as it could. In this paper we present an improved strategy: based on the TBoxes of the ontologies involved in a reasoning task, we create more specific rules which then can be used for further reasoning. We make use of the EYE reasoner and its logic Notation3. In this logic, rules can be employed to derive new rules which makes the rule creation a reasoning step on its own. We evaluate our implementation on a semantic nurse call system. Our results show that adding a pre-reasoning step to produce specialized rules improves reasoning times by around 75%.

Keywords: Notation3, rule-based reasoning, OWL 2 RL

1 Introduction

With the increasing amount of carefully designed ontologies semantic web reasoning is becoming more popular for industrial applications: ontologies can be employed to solve complex problems in domains like medicine, automotive industry or finance (e.g., [13], [17]). Nevertheless, there are still some obstacles which hinder semantic web reasoning from being fully established. One of these is scalability: depending on the amount of data and the ontologies involved, traditional OWL DL reasoners can be too slow to solve problems in real time. The different OWL 2 profiles [9] provide a solution: by using less expressive but still powerful subsets of OWL DL, reasoning times can be significantly improved.

In this paper we focus on the OWL RL profile which is designed to enable rule-based reasoners to draw the right conclusions from ontology data and concepts. The rules for that, as presented in the specification, are complex in the sense that they rely on rather complicated patterns occurring in both ABox and TBox which have to be found to draw conclusions. We propose to improve OWL RL reasoning performance by adding an extra reasoning step. Based on the ontology's TBox, specialized rules can be automatically produced to be used for further reasoning on the ABox. Due to its expressiveness we use Notation3 Logic [6] to perform this task. The highly performant EYE reasoner [19] is used for reasoning. As our pre-reasoning step has to be executed only once for every TBox, our approach is especially suitable for situations where the same reasoning has to be performed on frequently changing data. We tested our implementation in an event based reasoning set-up: a semantic nurse call system which controls the technical equipment in a hospital and, for example, assigns the most suitable nurse to a patient's call. Our tests showed that our pre-reasoning step reduces reasoning times at about 75% compared to an implementation using the originally proposed rules.

The remainder of this paper is structured as follows: in Section 2 we give an overview of related work. After that, in Section 3, we explain our use case, a semantic nurse call system. Section 4 gives a general introduction to OWL RL in N3. In Section 5 we describe our system in more detail, focusing in particular on the improved rules themselves and the steps which are necessary to produce them. An evaluation of our implementation is given in Section 6. We summarize our main findings and give an outlook to future work in Section 7.

2 Related Work

Traditionally, reasoning over OWL ontologies was performed by description logic based reasoners using (variants of) the tableaux algorithm. Prominent examples of such reasoners are Pellet [16] and HermiT [18]. Both support—as others of their kind—the full OWL DL profile. The expressiveness of this profile and the complexity of the related reasoning, make these reasoners perform rather slow in comparison with, for example, rule-based reasoners. The OWL 2 profiles [9] aim to overcome this gap by defining less expressive but still powerful subsets of OWL DL. One of these profiles is OWL RL, which was designed to enable rule-based reasoners to cope with OWL ontologies. Various implementations make use of the OWL 2 RL/RDF rules as proposed in the specification, among them OWLim [8] and Oracle's RDF Semantic Graph [20]. As most other implementations we are aware of, these reasoners support their own rule format, and optimizations are done internally using the underlying programming language. We propose an optimization which can be done in the logic itself by performing an extra reasoning step. We are thereby independent of a specific reasoner.

Notation3 Logic (N_3) was introduced in 2008 by Tim Berners-Lee et al. [6]. It forms a superset of RDF and extends the RDF data model by formulas (graphs), functional predicates, universal variables and logical operators, in particular the

implication operator. Rules in N3 can not only be applied to derive new RDF triples, it is also possible to write and apply rules with new rules in their consequence, and thus to derive new rules. It is exactly this property which made us opt for using N3 instead of other rule formats like, e.g., swrl [14].

There are several reasoners supporting N3: FuXi [1] is a forward-chaining production system for Notation3 whose reasoning is based on the RETE algorithm. The forward-chaining cwm [4] reasoner is a general-purpose data processing tool which can be used for querying, checking, transforming and filtering information. EYE [19] is a reasoner which is enhanced with Euler path detection. It supports backward and forward reasoning and also a user-defined mixture of both. Amongst its numerous features are the option to skolemize blank nodes and the possibility to produce and reuse proofs for further reasoning. The reason why we use EYE in our implementation is its high performance. Existing benchmarks and results are listed in the above mentioned paper [19] and on the EYE website [11].

3 Use Case

Our use case is a nurse call system in a hospital. The system is aware of certain details about personnel and patients represented in an OWL ontology. Such information can include: personal skills of a staff member, staff competences, patient information, special patient needs, and/or the personal relationship between staff members and patients. Furthermore, there is dynamic information available, for example, the current location of staff members and their status (busy or free). When a call is made, the nurse call system should be able to assign the best staff member to answer that call. The definition of this "best" person varies between hospitals and can be quite complex. The system additionally controls different devices. If for example staff members enter a room with a patient, a light should be switched on; if they log into the room's terminal, they should have access to the medical lockers in the room. The event-driven reasoning system for this use case has to fulfill certain requirements.

scalability It should cope with data sets ranging from 1000 to 100,000 relevant triples (i.e., triples necessary to be included for the reasoning to be correct). Especially in bigger hospitals the number of staff members and patients and thereby also the amount of available information about those can be quite big. It is not always possible to divide this knowledge into smaller independent chunks as this data is normally full of mutual dependencies.

functional complexity It should implement deterministic decision trees with varying complexities. The reasons to assign a nurse to a certain patient can be as manifold as the data. Previous work has shown that this complexity is not only theoretically possible but also desired by the parties interested in such a semantic system [15].

configuration It should support the ability to change these decision trees at configuration time. Different hospitals have different requirements and even in one single hospital those requirements can easily change due to e.g., an

```
1 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
3
4 {?C rdfs:subClassOf ?D. ?X a ?C} => {?X a ?D}.
```

Listing 1: OWL RL rule for `rdfs:subClassOf` class axiom in N3.

increase of available information or a simple change in the hospital's organizational concepts or philosophy.

real-time It should return a response within 5 seconds to any given event. Especially in such a delicate sector as patient care, seconds can make a difference. Even though a semantic nurse call system will not typically be employed to assign urgent emergency calls through complex decision trees, a patient should not wait too long till his possibly pressing request is answered.

The functional complexity requirement together with the configuration constraint motivate the choice of a reasoning system which supports rules as these can be seen as the most natural way to express decision trees. Even though a numerous amount of OWL DL reasoners support at least one rule format, their reasoning is too slow to meet the scalability and the real-time constraint [2]. Therefore, we chose a rule-based solution.

4 OWL RL in N3

In a first attempt to solve the above mentioned problem we used a direct translation of OWL 2 RL/RDF rules as listed on the corresponding website [9]. Where possible, we made use of existing N3-translations of these rules as provided by EYE [12]. Missing concepts were added. The data was represented using the ACCIO ontology [15] which will be further described in section 6.1. The results of this implementation were already promising [2], but for larger data sets the reasoning took multiple minutes and, thus, did not meet the requirements claimed above.

We explain the idea behind these OWL RL rules in N3 and how they can be improved by an example: Listing 1 shows the class axiom rule[4] which is needed to deal with the rdfs concept `subclassOf`. For convenience we omit the prefixes in the formulas below. The empty prefix refers to the ACCIO ontology, `rdf` and `rdfs` have the same meaning as in Listing 1. Consider that we have the following TBox triple stating that the class `:Call` is a subclass of the class `:Task`:

```
:Call rdfs:subClassOf :Task.        (1)
```

If the ABox contains an individual which is member of the class `:Call`

```
:call1 a :Call.                     (2)
```

[4] The rule is the N3 version of the cax-sco rule in Table 7 on the OWL 2 Profiles website [9].

an OWL DL reasoner would make the conclusion that the individual also belongs to the class `Task`:

```
:call1 a :Task.                                    (3)
```

Our rule in Listing 1 does exactly the same: as Formula 1 and Formula 2 can be unified with the antecedent of the rule, a reasoner derives the triple in Formula 3. But this unification is rather expensive: if we take a closer look to the antecedent we see that it contains three different variables occurring in two different triples which have to be instantiated with the data of the ontology. In our use case information as stated in Formula 2 can change—patients will make new calls—but statements as Formula 1 can be considered as fixed: the terminology does not change during the reasoning process, calls are tasks for our ontology. Our solution makes use of this observation: what is valid for the triple in Formula 1 also counts for other TBox-triples. We consider the TBox as static knowledge which can be used for pre-processing. The idea of our solution is to do as much unification as possible before dealing with (possibly) dynamic data. We produce more specialized rules, in the case mentioned above, for example the rule

```
{?X a :Call.} => {?X a :Task.}.                    (4)
```

which will derive for every new call, that it is also a task, just as the rule in Listing 1 does.

5 Producing TBox-rules

In order to achieve the goal explained in the last section, producing specialized rules based on the concepts present in the ontology's TBox, we use the EYE reasoner. Reasoning in EYE can be considered as a single process, having as input all necessary files representing the knowledge (i.e., the necessary ontologies, data, and rule-files), and a query-file that filters the output of the reasoning result. We have to perform two steps:

1. Produce a grounded copy of the TBox.
2. Use rules to translate the grounded TBox into specialized rules.

The need of the first step has to do with the fact that an ontology can contain anonymous classes represented by blank nodes. Used in rules, these blank node class names have, due to the semantics of N3, a limited scope. It is therefore difficult to use them to reference the same class in different rules. We will give a more elaborate explanation in the next section. After that we will describe the translation step in more detail.

5.1 Grounding the Ontology

Before translating the TBox into rules we have to replace all blank nodes by URIs or literals. To understand the reason for this skolemization step, consider the example in Listing 2. The example contains triples which further describe

```
@prefix : <http://ontology/Accio.owl#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

:Call  rdfs:subClassOf [
                         rdf:type owl:Class ;
                         owl:unionOf (
                                       :PlannedTask
                                       :UnplannedTask
                                     )
                         ].
```

Listing 2: ACCIO example: a call can be a planned or an unplanned task.

the class `:Call` from Formula 2. A call can be a planned or an unplanned task, or to be more specific: the class `:Call` is subclass of an anonymous class which is the union of the classes `:PlannedTask` and `:UnplannedTask`. Even though N3 supports rules which contain blank nodes, it is exactly this anonymous class which causes problems. Being unlabeled, the blank node can be referred by an arbitrary new blank node name. A translation as done in Formula 4 would result in a rule like:

```
{ ?X a :Call. } => { ?X a _:newblank.}.
```
(5)

This rule means, that every instance of the class `:Call` is also instance of *some* other class. This knowledge can already be gained by Formula 4 and does not have much influence on further reasoning. And even if the blank node in Listing 2 would be labeled by for example `_:union1` a new rule

```
{ ?X a :Call. } => { ?X a _:union1.}.
```
(6)

would have no other meaning than Formula 4 as in N3 the scope of a blank node is always only the graph, i.e. the curly brackets `{ }`, it occurs in [3,5]. The consequence of the rule would not refer to our union of planned and unplanned tasks.

We perform the grounding step by using the EYE reasoner. The reasoner provides the option to obtain a skolemized version of any input N3 file(s). The switch `--no-qvars` replaces every blank node by a unique skolem IRI following the naming convention as described in the RDF specification [10]. It additionally makes sure that equally named blank nodes only get assigned the same skolem IRI if they actually refer to the same thing. Producing a grounded version of the ontology enables us in further reasoning steps to use the new identifiers for (formally) anonymous classes in different rules.

5.2 Translation Step

As explained above the next step after having produced a grounded version of the ontology's TBox is to produce the new specialized rules. Here, we make use

```
1 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
3
4 {?C rdfs:subClassOf ?D.} => {{?X a ?C.}=>{?X a ?D.}.}.
```

Listing 3: Rule producing new rule for every occurrence of `rdfs:subClassOf`; based on the `rdfs:subClassOf` class axiom of Listing 1.

of a property of Notation3: rules can not only be applied to derive new triples but also to derive new rules. To illustrate that we consider a simple rule:

$$\{\underbrace{\texttt{:Call rdfs:subClassOf :Task.}}_{\text{satisfied ontology triple(s)}}\}=>\{\underbrace{\texttt{\{?X a :Call\}=>\{?X a :Task.\}.}}_{\text{produced new rule(s)}}\}.$$

Just as simple rules enable the reasoner to derive new triples from the fact that its antecedent is fulfilled, the rule above, applied on Formula 1, derives a new rule, namely Formula 4. Nevertheless, the rule as stated above is too specific to be used for our purpose: if we already knew that the ontology contained the triple in Formula 1 we could also write the rule in Formula 4 directly instead or writing a rule which will surely produce it. Our rule needs to be more general as we want to handle all `owl:subclassOf` triples in that same way and always produce a rule similar to the rule expressed in Formula 4. This more general rule can be found in Listing 3. Applied on Formula 1 the variable `?C` gets unified with the URI `:Call` and the variable `?D` gets unified with `:Task`, thus, Rule 4 can be derived. Similarly, an application of the rule in Listing 3 on triple

```
:PlannedTask rdfs:subClassOf :Task.
```

results in a new rule

```
{?X a :PlannedTask.} => {?X a :Task.}.
```

The same principle can be applied for other OWL concepts. Listing 4 shows a rule[5] which handles the concept `owl:unionOf`. Note that this rule uses a built-in predicate of Notation3, `list:in`. A triple using `list:in` as a predicate is true if and only if the object is a list and the subject is an entry of that list. If we apply this rule to the (now skolemized) union expressed in Listing 2

```
:UnionClass1 owl:unionOf ( :PlannedTask :UnplannedTask ).
```

two rules will be produced by that:

```
{?x a :PlannedTask} => {?x a :UnionClass1.}.
```

and

[5] The rule is motivated by the cls-uni rule in Table 6 on [9].

```
@prefix list: <http://www.w3.org/2000/10/swap/list#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

{?C owl:unionOf ?L. ?D list:in ?L.} =>
                                  {{?X a ?D.}=>{?X a ?C.}.}.
```

Listing 4: Rule-producing rule for `owl:unionOf`.

```
{?x a :UnplannedTask} => {?x a :UnionClass1.}.
```

The above example illustrates another useful property of Notation3: Notation3 treats lists themselves, not only their reified version, as elements of the language. There are many built-in predicates which enable the user to write clear rules regarding lists and to refer to all elements of a given list. For working with OWL ontologies this is a real advantage as lists are normally used together with many OWL concepts like the above `owl:unionOf` or for example `intersectionOf`.

To produce new rules by applying the rules described above, the rule producing rules have to be applied as filter rules for the reasoner. Notation3 reasoners normally take one ore more input files—consisting of rules and facts—and a query file containing rules into account. Based in the input files the reasoner outputs the logical consequences of the filter rules. In our present case these are the specialized rules. The rules produced by the two described steps do now replace the TBox of the ontology and can be used for further reasoning.

6 Evaluation

The aforementioned methodology replaces generic and complex constructs in the TBox by specialized rules that provide the same functionality. To test how much performance we gain by using this pre-processing step we tested a scenario of our use case with two rule sets: the first traditional rule set processes the triples of the original TBox while reasoning and acts on top of those together with the actual ABox data, the second precomputed rule set contains the specialized rules which already take all TBox triples into account, therefore in this case the original TBox is not needed for further reasoning. All experiments were run on the same technology stack[6].

6.1 Ontology and Data

To represent the data as described above we make use on the ACCIO ontology which was designed to represent all aspects of patient care in a hospital. The ontology contains ca. 3,500 triples (414 named classes, 157 object properties, 38 data type properties). A full description is given by Ongenae et al. [15].

[6] Hardware: Intel(R) Xeon(R) E5620@2.40GHz CPU with 12 GB RAM. Software: Debian "Wheezy", EYE-Summer15 edition 2015-07-29T18:01:28Z and SWI-Prolog 6.6.6

This ontology was filled with data describing wards in a hospital. This data was simulated, based on real-life situations, as deducted from user studies [15]. The data was scaled by increasing the amount of wards from 1 to 10 to fill the ABox with more data. The description of such a ward contains approximately 1,000 static triples. Additionally, there was dynamic data such as for example the location of nurses or the status of calls taken into account.

6.2 Test scenario

We compared the reasoning times of the two rule sets by running a scenario, based on a real-life situation. This scenario consists of a sequence of events, which we list below, where the expected outcome of the reasoning is indicated in brackets.

1. A patient launches a call (*assign nurse and update call status*)
2. The assigned nurse indicates that she is busy (*assign other nurse*)
3. The newly assigned nurse accepts the call task (*update call status*)
4. The nurse moves to the corridor (*update location*)
5. The nurse arrives at the patients' room (*update location, turn on lights and update nurse status*)
6. The nurse logs into the room's terminal (*update status call and nurse, open lockers*)
7. The nurse logs out again (*update status call and nurse, close lockers*)
8. The nurse leaves the room (*update location and nurse status and turn off lights*)

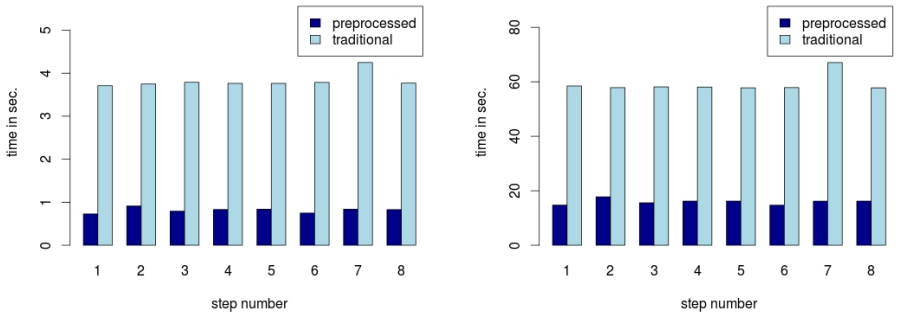
6.3 Results

The aforementioned scenario was run 35 times, consisting of 3 warm-up runs and 2 cool-down runs, for 1 ward and 10 wards, for both rule sets. By averaging the 30 remaining reasoning times per amount of wards and per rule set, we provide the results as shown as a table in Figure 1, and depicted in Figure 2.

wards	**1 ward**								**10 wards**							
event	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
traditional	3.7	3.7	3.8	3.8	3.8	3.8	4.2	3.8	58.5	57.9	58.1	58.1	57.8	57.9	67.1	57.8
preprocessed	0.7	0.9	0.8	0.8	0.8	0.7	0.8	0.8	14.7	17.8	15.6	16.2	16.2	14.7	16.2	16.2

Fig. 1: Reasoning times using traditional rules and preprocessed rules in seconds. Preprocessing significantly reasoning times.

The figures show how preprocessing the rules improves reasoning times significantly, consistently requiring only a quarter of the reasoning time. This trend



(a) 1 ward, reasoning time per event. (b) 10 wards, reasoning time per event.

Fig. 2: Comparison of reasoning times using preprocessed and traditional rules. The preprocessing step improves reasoning times.

manifests itself regardless of the amount of dynamic data involved. Whereas the traditional rule set can no longer be used in a hospital with 10 wards—as its reasoning times are in the order of minutes—the preprocessed rule set still provides reasonable reasoning times.

7 Conclusion and Future Work

In this paper we have shown that precomputing and using specialized rules based on the ontology's TBox improve reasoning times of OWL RL reasoning by about 75%. The main cause for that is that the newly computed rules are less complex—in terms of the variables which have to be unified during reasoning—than the original version of the rules taken from the OWL RL profile description. Another aspect which makes reasoning faster in our set up is the fact that rules for concepts which are not even present in the ontology's TBox will not get produced by the preprocessing step. If for example the rather expensive concept `owl:sameAs` does not occur in any triple of the considered ontology, no specialized rules will be produced for this concept.

The presented preprocessing step consists of two simple reasoning runs which can be performed before dealing with additional input data. Using the EYE reasoner this preprocessing normally takes only a few seconds. In set ups where the TBox does not change during run time the produced rules can be used whenever the data to reason on changes as in the example introduced in this paper. Our approach is independent of the reasoning done on top of the TBox by additional rules. This makes the rule version of the ontology's TBox even more suitable for reuse.

Our approach makes use of the special properties of Notation3 Logic. By providing the option of using rules to produce new rules this logic is particularly suitable for our purposes. Furthermore Notation3 offers multiple predicates to act on lists as for example the function `list:in`. This eases the implementation of rule producing rules based on OWL predicates as there are many OWL constructs which are normally stated with lists in their object position.

Notation3 Logic posses other interesting properties which we are planning to apply in future work: in N3, rules can have existential variables in their consequence. Using this particular property it will be possible to also cover OWL EL concepts which are not present in OWL RL. Similarly as done in for example OWLim [7] we are planning to include OWL EL in our implementation. We furthermore want to investigate the actual costs of processing the different OWL concepts by our newly produced rules. This will enable us to recommend the exclusion of particular concepts if not really needed.

Acknowledgements The research activities described in this paper were funded by Ghent University, iMinds, the IWT Flanders, the FWO-Flanders, and the European Union, in the context of the project "ORCA", which is a collaboration of Televic Healthcare, Internet-Based Communication Networks and Services (IBCN), and Multimedia Lab (MMLab).

References

1. FuXi 1.4: A Python-based, bi-directional logical reasoning system for the semantic web, `http://code.google.com/p/fuxi/`
2. Arndt, D., De Meester, B., Bonte, P., Schaballie, J., Bhatti, J., Dereuddre, W., Verborgh, R., Ongenae, F., De Turck, F., Van de Walle, R., Mannens, E.: Ontology reasoning using rules in an ehealth context. In: Proceedings of the 9th International Web Rule Symposium: Industry Track (Aug 2015)
3. Arndt, D., Verborgh, R., De Roo, J., Sun, H., Mannens, E., Van de Walle, R.: Semantics of Notation3 logic: A solution for implicit quantification. In: Proceedings of the 9th International Web Rule Symposium (Aug 2015)
4. Berners-Lee, T.: *cwm* (2000–2009), `http://www.w3.org/2000/10/swap/doc/cwm.html`
5. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C Team Submission (Mar 2011), `http://www.w3.org/TeamSubmission/n3/`
6. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008)
7. Bishop, B., Bojanov, S.: Implementing owl 2 rl and owl 2 ql rule-sets for owlim. In: OWLED. vol. 796 (2011)
8. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: Owlim: A family of scalable semantic repositories. Semantic Web 2(1), 33–42 (2011)
9. Calvanese, D., Carroll, J., Di Giacomo, G., Hendler, J., Herman, I., Parsia, B., Patel-Schneider, P.F., Ruttenberg, A., Sattler, U., Schneider, M.: OWL 2 Web Ontology Language Profiles (second edition). W3C Recommendation (Dec 2012), `www.w3.org/TR/owl2-profiles/`

10. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and Abstract Syntax. W3C Recommendation (Feb 2014), http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
11. De Roo, J.: Euler yet another proof engine (1999–2015), http://eulersharp.sourceforge.net/
12. De Roo, J.: EYE and OWL 2 (1999–2015), http://eulersharp.sourceforge.net/2003/03swap/eye-owl2.html
13. Declerck, T., Krieger, H.U.: Translating xbrl into description logic. An approach using Protege, Sesame & OWL. In: BIS. pp. 455–467 (2006)
14. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission (21 May 2004), http://www.w3.org/Submission/SWRL/, available at http://www.w3.org/Submission/SWRL/
15. Ongenae, F., Bleumes, L., Sulmon, N., Verstraete, M., Van Gils, M., Jacobs, A., De Zutter, S., Verhoeve, P., Ackaert, A., De Turck, F.: Participatory design of a continuous care ontology: towards a user-driven ontology engineering methodology. In: Knowledge Engineering and Ontology, Proceedings. pp. 81–90 (2011)
16. Parsia, B., Sirin, E.: Pellet: An OWL DL reasoner. In: Proceedings of the Third International Semantic Web Conference (2004)
17. Patel, C., Cimino, J., Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: Matching patient records to clinical trials using ontologies. Springer (2007)
18. Shearer, R., Motik, B., Horrocks, I.: Hermit: A highly-efficient OWL reasoner. In: OWLED. vol. 432, p. 91 (2008)
19. Verborgh, R., De Roo, J.: Drawing conclusions from linked data on the web. IEEE Software 32(5) (May 2015), http://online.qmags.com/ISW0515?cid=3244717&eid=19361&pg=25
20. Wu, Z., Eadon, G., Das, S., Chong, E.I., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle. In: Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. pp. 1239–1248. IEEE (2008)