Augmenting the expressivity of the Ontology Pre-Processor Language

Luigi Iannone, Mikel Egaña, Alan Rector, and Robert Stevens

School of Computer Science
University of Manchester
Manchester
M13 9PL UK
{iannone,mikel.eganaaranguren,rector,robert.stevens}@cs.manchester.ac.uk

Abstract. We present the latest release of the Ontology Pre-Processor Language, a language for querying and modifying Description Logic knowledge bases expressed in OWL. We briefly describe its renewed syntax and decidability. We compare the proposed language expressivity with the state of the art.

1 Motivation

The version of the Ontology Pre-Processor Language (OPPL) described here is the successor of the initial effort presented in [1]. The need for this kind of language is particularly evident in bio-ontologies, where knowledge models originally consist of initial skeletons that are open to enrichment in a semi-automatic fashion [2]. To do this, ontology engineers need the means of expressing declaratively the set of axioms to be added (edited) along with the circumstances under which these additions (editing) should be performed. Although the initial effort on OPPL proved to be a successful first step towards this objective, it was deemed necessary to partially rework it aiming at:

– Reach full compatibility with the current OWL-DL specification,
– Push the initial OPPL expressivity from a single variable language to a language with an arbitrary number of variables.

As we shall detail further at the end of this paper, to the best of our knowledge, currently there is no single formalism, allowing for expressing both queries and atomic operations on a knowledge base using the correct language abstraction (OWL-DL). There are indeed query languages for Description Logic (DL) knowledge bases, as well as Application Programming Interfaces (APIs) for carrying out operations on such knowledge bases, and both the languages and the APIs provide support for reasoning. The query languages do not, however, provide declarative support for specifying operations to be executed on query results and all the APIs we surveyed need to individuate the subjects of their operations programmatically. Hence, our primary motivation has been to investigate whether and how such synthesis was possible. In the following we briefly present our results for such investigation, namely:

– A language that responds to our use case and its syntax;2
– Some consideration upon its decidability together with brief comparative survey of the state of the art (Section 3);
– And, at the end, some conclusions and the future directions we intend to pursue.

2 The augmented Ontology Pre-Processor Language

A generic OPPL statement in the new syntax will look as follows:

```
SELECT Axiom,....,Axiom
BEGIN
ADD | REMOVE Axiom
...
ADD | REMOVE Axiom
END
```

This will not only harmonize the `SELECT` clauses with the correct level of abstraction offered by OWL-DL, but also achieve uniformity between the query and the action section. The most important objective for this new release of OPPL is the explicit introduction of variables. The first release of OPPL could be regarded as a weakly[1] typed single variable language. This limitation left out of the language a range of required query types. For example, it was impossible either to select all the classes that are subclasses of a class that, in its turn, is disjoint with a given class; or selecting all the classes equivalent to an existential restriction on a variable property with a given filler. Instead, we wanted to provide the possibility of having an arbitrary number of variables spanning across the query section as well as the action section. Therefore, one can now select a set of axioms built upon some fixed and variable parts (query) and then add or remove other axioms depending on a subset of the variables introduced in the previous select section.

An OPPL statement is decomposable in the following sections:

1. Variable definition (before `SELECT`)
2. Selection (between `SELECT` and `BEGIN`)
3. Actions (between `BEGIN` and `END`)

The sections described above can be further broken down as in Figure 1 where we report the entire OPPL syntax grammar. As we already mentioned above, OPPL syntax is built upon the Manchester OWL Syntax. To improve readability for those already familiar with it, we did not expand the production rules for the non terminal `Axiom` in Figure 1. Readers can find the details for such production at `http://www.cs.man.ac.uk/~iannonel/oppl/grammar.html`.

We report an example of a statement in the new OPPL syntax in Figure 2. In general, variables can have the following types:

[1] Weak here means that the type of the sole variable is inferred from the whole OPPL statement rather than declared as shown in the examples above.

1. CLASS;
2. OBJECTPROPERTY;
3. DATAPROPERTY;
4. INDIVIDUAL;
5. CONSTANT.

Each variable type covers a possible category of entities in the OWL specification. An entity here, is a named object or a constant. Therefore, an anonymous class description **is not** an acceptable substitution for any variable type, including CLASS. This limitation has important bearings on the possibility of building complete algorithms to execute an arbitrary query in OPPL.

```
OPPL Statement  ::=  ( VariableDeclaration )? ( Query )? ( Actions )? ";"
VariableDeclaration  ::=  VariableDefinition ( "," VariableDefinition )*
Actions  ::=  "BEGIN" Action ( "," Action )* "END;"
VariableDefinition  ::=  <IDENTIFIER> ":" variableType
Constraint  ::=  <IDENTIFIER> <NEQ> OWLExpression
variableType  ::=  "CLASS" | "OBJECTPROPERTY |
"DATAPROPERTY" | "INDIVIDUAL" | "CONSTANT"
Query  ::=  "SELECT" Axiom ( "," Axiom )*
( <WHERE> Constraint ( <COMMA> Constraint )* )?
Action  ::=  "ADD" | "REMOVE" Axiom
Axiom  ::=  An axiom in Manchester OWL Syntax
(possibly containing variables)
IDENTIFIER::= "?"<LETTER> (<LETTER>|<DIGIT>)*
LETTER ::= ["_","a"-"z","A"-"Z"]
DIGIT ::= ["0"-"9"]
OWLExpression  ::=  An OWL entity in Manchester OWL Syntax
   (possibly containing variables)
```

Fig. 1. OPPL EBNF Grammar Specification

```
?x:CLASS SELECT ?x equivalentTo participates_in
only (intellectual_dinner and party)
BEGIN
REMOVE ?x subClassOf lives_on only (not campus);
END;
```

Fig. 2. Selects all the classes equivalent to the all value restriction participates_ in only(intellectual_dinner and party) and removes the axiom, subClassOf lives_on only (not campus) when asserted on the matching ones

3 Discussion

OPPL can work both in asserted-only or in inferred mode. This means that queries can be executed respectively on the asserted model only or considering also the inferred axioms. In the former case no interaction with the reasoner is required and query-matching is at worst linear in the number of axioms contained in the ontology. The latter case is computationally more complex. The limitation for `CLASS` ensures the query language decidability. We can imagine a very simple, brute-force. algorithm that replaces every variable with every entity in the ontology and, then, asks the reasoner if the resulting instantiated axiom holds in the model. Such algorithm terminates in finite time as the number of variables and entities are both finite, and returns the correct results for the queries. Nevertheless, such simplistic approach is not scalable and real implementations need optimisations that are currently being explored and evaluated.

OPPL is non-monotonic on account of `REMOVE` actions, therefore the execution order of OPPL statement matters as they are not commutative. Moreover, in the case inference is on, `REMOVE` actions will affect only asserted axioms.

To the best of our knowledge, OPPL is unique in combining the possibility of querying and performing actions on the matched entities. The only language with comparable functionalities is SPARQL[2]. It crucially differs, however, from OPPL on its level of abstraction (RDF triples versus OPPL's OWL axioms). Its non- monotonic `REMOVE` feature makes it significantly different from the Semantic Web Rule Language and its DL-safe fragment [3]. If, however, we restrict to single functionalities there is some related work which is worth mentioning briefly.

For queries, SPARQL-DL [4] is the language whose expressivity is the closest to OPPL. The main difference between SPARQL-DL and OPPL is that SPARQL-DL queries select n-uples of variables. OPPL, on the contrary, focusses on axioms. Moreover, at least with respect to the abstract syntax mentioned in [4], OPPL offers a greater expressivity, e.g.: nesting variables into complex class descriptions which seems not to be possible using SPARQL-DL.

For actions, there are several different APIs for dealing with OWL (see [5] and [6] for the two most comprehensive frameworks) that provide developers with similar or more ample set of possible operations than OPPL. Such APIs, however, present procedural details to which their users must pay attention, in order to accomplish the equivalent simple OPPL action. For example, adding the following subclass axiom:

```
A subClassOf C and B
```

Corresponds, using the Manchester OWL API, to executing the following set of instructions:

```
OWLClass a = dataFactory.getOWLClass("A");
OWLClass b = dataFactory.getOWLClass("B");
OWLClass c = dataFactory.getOWLClass("C");
```

[2] SPARQL Update: `http://jena.hpl.hp.com/~afs/SPARQL-Update.html`

```
Set<OWLClass> set = new HashSet<OWLClass>(2);
set.add(b);
set.add(c);
OWLDescription intersection =
dataFactory.createOWLObjectIntersection(set);
OWLAxiom axiom = dataFactory.getOWLSubClassAxiom(a,intersection);
ontologyManager.applyChange(new AddAxiom(ontology,axiom));
```

which, apart from its length, requires that an ontology engineer knows the details of the API infrastructure (data factories, difference between `OWLDescription` and `OWLClass`, etc.).

The following issues need to be tackled in the future:

1. Complete the transition from the old syntax to the new one, providing support for querying non logical axioms (i.e: annotations).
2. Empirically evaluate the optimisations for the querying algorithm in presence of inference.

Furthermore, it is equally interesting the potential integration of OPPL with the *Lint* framework developed at Manchester University[3]. The notion of Lint comes from programming, where it represents generic bad practices in code writing that could potentially lead to defects in the software (e.g: declare variables and not use them). We ported this notion into OWL ontology engineering and implemented a detection engine framework into which users could plug-in their own Lint checks[4]. At the moment, user-defined Lints must be developed in Java using the OWL API mentioned above, but, after the integration with OPPL, users will be able to specify declaratively (by means of a query) the potential lint as well as the actions to undertake to fix the problem.

4 Acknowledgements

Mikel Egaña is funded by the University of Manchester and EPSRC.

References

1. Egaña, M., Antezana, E., Stevens, R.: Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In: Proceedigns of OWLED 2008 DC OWL: Experiences and Directions. (2008)
2. Egaña, M., Rector, A., Stevens, R., Antezana, E.: Applying Ontology Design Patterns in bio-ontologies. In: To appear in proceedings of EKAW 2008. (2008)
3. Motik, B., Sattler, U., Studer, R.: Query answering for owl-dl with rules. J. Web Sem. **3** (2005) 41–60

[3] Available at `http://www.cs.man.ac.uk/~iannonel/lintRoll/`

[4] A similar effort appeared recently as a Pellet add-on - `http://pellet.owldl.com/pellint`, although it is specifically aimed at flagging and (optionally) repairing "modeling constructs that are known to cause performance problems".

4. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: OWLED 2007 OWL: Experiences and Directions 2007. (2007)
5. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 Touch Paper: The OWL API. In Golbreich, C., Kalyanpur, A., Parsia, B., eds.: OWLED 2007 OWL: Experiences and Directions 2007. Volume 258 of Workshop Proceedings., CEUR (2007)
6. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In Feldman, S.I., Uretsky, M., Najork, M., Wills, C.E., eds.: WWW (Alternate Track Papers & Posters), ACM (2004) 74–83