SweetProlog: A System to Integrate Ontologies and Rules

Loredana Laera[1], Valentina Tamma[1], Trevor Bench-Capon[1], and Giovanni Semeraro[2]

[1] Department of Computer Science, University of Liverpool, L69 3BX, Liverpool, UK
{lori, valli, tbc}@csc.liv.ac.uk
[2] Dipartimento di Informatica, Universita' di Bari, Via Orabona 4,70125, Bari, Italy
semeraro@di.uniba.it

Abstract. This paper describes the design and implementation of SweetProlog, a system for translating Web rules into Prolog. It enables the integration of ontologies and rules on the Semantic Web. This is achieved via a translation of OWL ontologies described in Description Logics and rules expressed in OWLRuleML into a set of facts and rules described in Prolog. Finally, the resulting logic program is interrogated by a Prolog engine to deduce new knowledge.

1 Introduction

Rules play an important role in the Semantic Web, as envisioned by Tim-Berners Lee [BLHL01]. The realization of the rule layer will allow further means to deduce knowledge and combine information. This layer should be built on top of ontology languages, such as OWL, and will offer enhanced representation and reasoning capabilities. The effort to define rule languages and to enable interoperability between major commercial rule systems has been undertaken by the RuleML Initiative[3]. The RuleML Initiative aims to develop a canonical Web language for rules called RuleML [BTW01], which permits both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks.
But there is a crucial problem to solve, how is it possible to enable an OWL ontology to reason over its instances using such rules? One possible solution is to use a Logic Programming Language, such as Prolog, which offers efficient automatic reasoning to accomplish this.
This paper therefore investigates how rules may be used to enhance the content of a ontology and allow the dynamic inclusion of derived facts that are not captured by the ontological taxonomy alone.

2 Representation of rules in OWLRuleML

As a first step we focus attention on how to define the rules and how to link the rules to description logics in OWL [DSHHHMPS03]. For this reason we used

[3] see http://www.dfki.de/ruleml

RuleML as a *lingua franca* to exchange rules between different rule systems and rule components in application software. The rules are therefore specified in OWLRuleML, which is an OWL ontology of the CLP (Courteous Logic Programs) RuleML rule syntax.
Using Description Logics of OWL to specify a meta ontology for RuleML rules, has some advantages:

- arbitrary OWL classes (e.g., descriptions) can be used as predicates in rules;
- rules and ontology axioms can be freely mixed;
- ontology and rules can be parsed in the same way;

OWLRuleML is a language appropriate for specifying derivation rules. It offers, among the others, the following features:

- Support for URI reference;
- Position dependent elements in RuleML;
- Preserved semantics of all constraints in the RuleML;
- Support for strong negation and weak negation;
- Support for prioritised conflict handling via Courteous Logic Programs;

Another advantage of representing rules in OWLRuleML concerns non-monotonic reasoning [Ant02]. OWL is incapable of representing non-monotonicity as is First Order Logic. By using OWLRuleML we can represent non-monotonicity via Courteous Logic Programs, a subset of RuleML. We believe that the non monotonic rules will play an important role in he area of the Semantic Web, where the available information is often incomplete. The next subsection will briefly introduce Courteous Logic Programs.

2.1 CLP

Courteous Logic Programs (CLP) [Gro97] extend expressively Ordinary Logic Programs (OLP), and are tractably compilable to OLP by a Courteous Compiler [Gro99]. CLP provide a method to resolve conflicts between rules using partially prioritized information to guarantee a consistent and unique set of conclusions (answer-set). Partially ordered prioritization among rules is optionally specified via a reserved predicate, called *overrides*. Each rule has an optional label, which has the form of a logical term. The atom *OVERRIDES (label1, label2)* specifies that a rule with *label1* has higher priority than a rule with *label2*. A pairwise exclusion integrity constraint (*mutex*) specifies the scope of conflict, stating that inconsistency for a particular pair of literals can be inferred, given another particular condition. As we mentioned above, CLP can be compiled as ordinary logic programs that are semantically equivalent. More precisely, we use a courteous compiler of IBM CommonRules to transform CLP into Prolog.

3 OWLRuleML rules on top of OWL ontologies

The next step is to find a possible way to integrate rules with OWL in logic programming environments such as Prolog. We follow the approach proposed in

[Gro03], which is to *build rules on top of ontologies*. This enables rules to have access to the Description Logic ontological definitions of the vocabulary primitives used by the rules. In this way OWLRuleML rules can reference OWL ontologies. The names of predicates in the OWLRuleML rules are URI's that link to classes and properties in an OWL ontology. The OWL ontology that is referenced forms a background theory for the rulebase. In this approach, the unary and binary

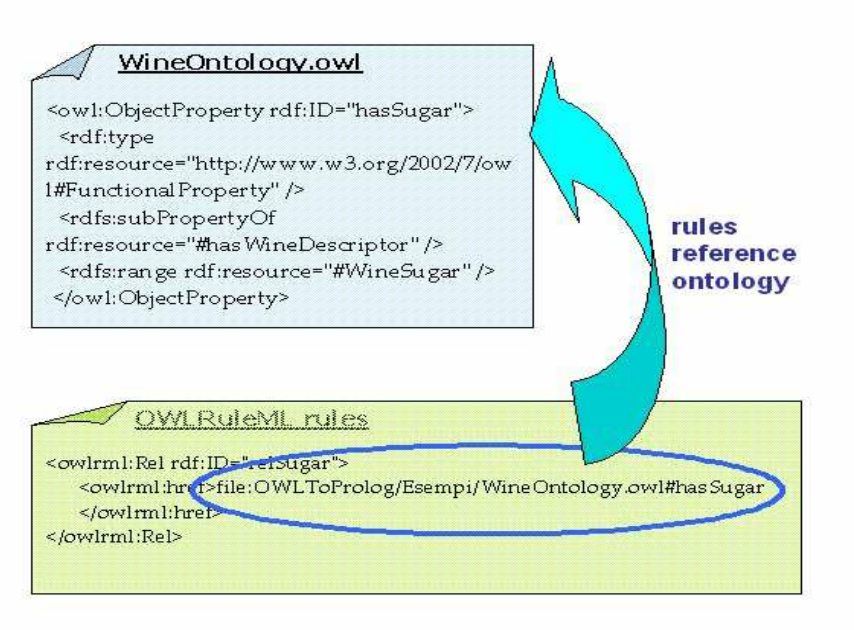

Fig. 1. Rules on top of Ontologies

predicates represent, respectively, OWL classes and OWL properties. Likewise, assertions about instances of class and properties are viewed as facts.
To enable such integration it was necessary to define a intermediate knowledge representation contained within the intersection of Description Logics and Prolog. This intersection, called Description Logic Programs (DLP) [GHVD03], enables ontological definitions to be supplemented by rules. DLP provide a significant degree of expressiveness, substantially greater than the RDFS fragment of Description Logics. DLP also capture a substantial fragment of OWL, including the whole of the OWL fragment of RDFS, simple frame axioms and more expressive property axioms.

4 Representation of OWL into PROLOG

The subset of OWL implemented in Prolog includes:

– RDF Schema features of OWL Lite, which provide the following primitives: *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain* and *rdfs:range* (T-box axioms);
– Transitivity of *rdfs:subclass* and *rdfs:subProperties* relationships;
– Equality of classes, properties and individuals of OWL: *owl:equivalentClass*, *owl:equivalentProperty*, *owl:sameIndividualAs* and *owl:sameAs*;
– All property characteristics;
– Cardinality constraints with minimum cardinality 0 and maximum cardinality 1. This is equivalent to defining a property as functional;
– Range restrictions on properties: *owl:allValuesFrom* and *owl:hasValue*;
– Conjunction of classes;
– Construction of classes by enumeration;

5 Implementation

SweetProlog, which stands for *Semantic WEb Enabling Technologies for Prolog*, is a system for the interoperability of rules between RuleML and Prolog. The idea of SweetProlog follows the approach of another existent project SweetJess [GGF03], which it is a system for interoperability of rules between RuleML and Jess.
SweetProlog is implemented in Java and makes use of three languages: Prolog as a rule engine, OWL as a ontology language and OWLRuleML as a rule language that reasons over the data model of OWL, and a set of inference rules that translate OWL into Prolog. It enables reasoning over OWL ontologies by rules via a translation of OWL subsets into simple Prolog predicates which a JIProlog engine[4] can understand and process.
SweetProlog consists of five principal functions:

1. **Translation of the OWL and OWLRuleML ontologies into RDF triples:** SweetProlog reads an OWL ontology and OWLRuleML rules, and extracts RDF triples[5] out of the ontologies, where each RDF triple consists of a subject, a predicate, and a object.
2. **Translation of the OWL assertions into Prolog:** The extracted RDF triples that represent OWL concepts and instances are translated into Prolog predicates via the mapping defined in Section 4.
3. **Translation of the OWLRuleML rules into CLP:** The RDF triples that represent the rules are translated into Courteous Logic Programs rules, more precisely into IBM CommonRules V3.0 SCLPfile format [6].
4. **Transformation of CLP Rules into Prolog:** The rules are transformed by a Courteous Compiler into Prolog rules.
5. **Interrogation of the output logic programs:** Finally, the translated predicates are then fed into the working memory of a JIProlog engine and it is able to infer new knowledge.

[4] http://www.ugosweb.com/jiprolog/index.shtml
[5] http://www-db.stanford.edu/ melnik/rdf/api.html
[6] http://alphaworks.ibm.com/tech/commonrules.

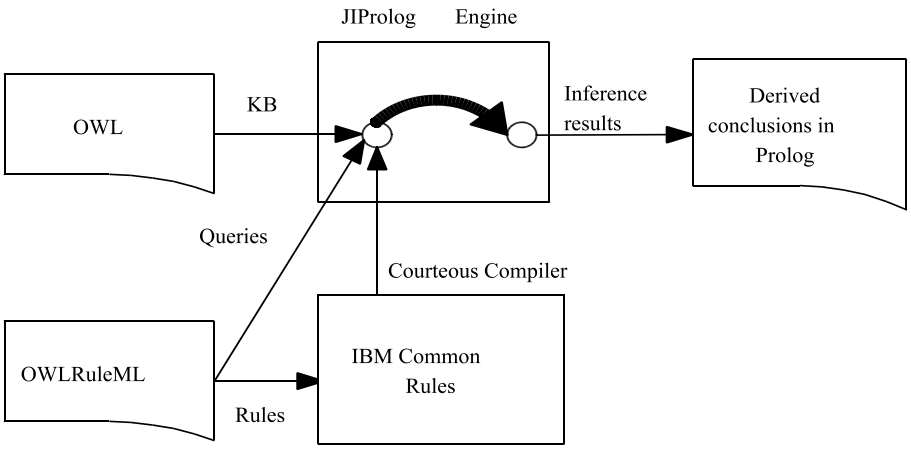

Fig. 2. Architecture of SweetProlog

Example

We consider a wine ontology, which is constituted of 91 concepts and 14 properties and is designed to find suitable wines for particular dishes. For brevity and ease of human-readability, we define our example rules in a Prolog-like syntax:

- pastaWithSpicyRedSouce(X) :- hasColour (X, Red), hasFlavor(X,Moderate), hasSugar(X,Dry), hasBody(X, Medium).
- pastaWithCreamSouce(X) :- hasColour(X,White), hasFlavor(X,Strong), hasSugar(X, Dry), hasBody(X, Full).

SweetProlog is able to derive the following new facts:

pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#SaucelitoCanyonZinfarel1998).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#WhitehallLaneCabernetFranc).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#MariettaPetiteSyrah).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#MariettaCabernetSauvignon).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#PageMillVineryCabernetSauvignon).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#SaucelitoCanyonZinfarel).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#MariettaZinfadel).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#MountadamPinotNoir). pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#GaryFarrellMerlot).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#ChiantiClassico).
pastaWithRedSouce(file:OWLToProlog/Esempi/WineOntology.owl#SaucelitoCanyonZinfarel1998).
pastaWithCreamSouce(file:OWLToProlog/Esempi/WineOntology.owl#MountadamChardonnay).

6 Conclusions

We have presented an approach for adding a rule layer on top of the ontology layer of the Semantic Web. This approach develops Grosof and Horrocks' idea

of specifying RuleML rules *on top of* OWL ontologies, suggesting a mapping of a Description Logic subset (OWL) into Logic Programs (Prolog). This intersection of DL with LP (Description Logic Programs) covers RDF Schema and a significant fragment of OWL.

The prototype realized, called SweetProlog, provides semantic and inferential interoperation between ontologies and rules. An immediate result of this project is that it possible to reason over the instances of an ontology within rule definitions that use vocabulary from these ontologies using a Prolog engine, thus through backward chaining. This is an initial step to layer rules on top of ontological languages, as proposed by the vision of the Semantic Web.

References

[Ant02] G. Antoniou. A nonmonotonic rule system using ontologies. In *Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2001. http://www.sciam.com/2001/0501issue/0501berners.

[BTW01] H. Boley, S. Tabet, and G. Wagner. Design rationale of RuleML: A markup language for semantic web rules. In *International Semantic Web Working Symposium (SWWS)*, 2001.

[CM87] W.F. Clocksin and C.S. Mellish. Programing in Prolog. Springer-Verlag, Berlin, 1987.

[DSHHHMPS03] M. Dean, G. Schreiber, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL Web Ontology Language Reference. http://www.w3.org/TR/2003/WD-owl-ref-20030331/.

[GGF03] Benjamin N. Grosof, Mahesh D. Gandhe, and Timothy W. Finin. SweetJess: Inferencing in situated courteous ruleml via translation to and from jess rules. Working paper, May 2, 2003. http://ebusiness.mit.edu/bgrosof/.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[Gro97] B. Grosof. Courteous logic programs: Prioritized conflict handling for rules. Technical report, IBM Research Report RC 20836, Dec. 30 1997, revised from May 8 1997.

[Gro99] B. Grosof. A courteous compiler from generalized courteous logic programs to ordinary logic programs (Preliminary Report). Technical report, IBM T.J. Watson Research Center, http://www.research.ibm.com/people/g/grosof/papers.html,July 1999.

[Gro03] Benjamin N. Grosof. SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *Proceedings of the twelfth international conference on World Wide Web*, Budapest, Hungary, 2003.