HOOPO: A Hybrid Object-Oriented Integration of Production Rules and OWL Ontologies

Georgios Meditskos and Nick Bassiliades[1]

Abstract. We describe a framework for the development of production rule programs on top of OWL ontologies, following a hybrid Object-Oriented (OO) approach. The hybrid nature is realized by separating ontologies and rules, interfacing an external DL reasoner and a production rule engine. The OO nature is realized by mapping OWL ontologies into the OO model, in such a way, so to preserve the extensional ontology semantics when the OO ontology constructs are matched in the production rule conditions.

1 INTRODUCTION

There are two main approaches towards the combination of rules and ontologies [1][7]:

- *Hybrid approach*: Rule and ontology predicates are strictly separated and the ontology predicates can be used as constraints in rules. Thus, existing reasoners may be used.
- *Homogeneous approach*: Rule and ontology predicates are treated homogeneously, as a new single logic language. Thus, a new reasoner is needed, able to handle the new language.

We present HOOPO, a hybrid framework that enables the definition of production rules over OWL ontologies, following an OO approach. More specifically, we enable the development of OO rule-based applications with ontology-based information, using an OO schema that stems from a vocabulary defined in ontologies. We follow the idea that rules may not be used to derive ontological knowledge and any knowledge about ontology information is provided by a DL component. This is achieved by allowing the OO ontology constructs to be matched only in OO production rule conditions, serving as restrictions for the development of *derived OO KBs*, that is user-defined classes, attributes and objects, disjoint from the OO ontology KB. Thus, we target at the monotonic combination of rules and ontologies.

HOOPO interfaces a production rule engine with an external DL reasoner, defining an OO mapping procedure of the ontological knowledge into the OO model of the rule engine. There are three motivations behind this OO mapping procedure. Firstly, we enable the development of rule-based ontology-based applications based on the well-known and established OO programming principles. Secondly, the generated OO ontology KB encapsulates the extensional (individual) ontology semantics that are needed during rule execution, and thus, there is no need for a runtime interaction between the rule engine and the reasoner, increasing rule execution performance. This is feasible, since we consider that the ontology information is not altered by the rule programs. Finally, the lack of any runtime interaction accounts for the utilization of the reasoner and the rule engine without modifications. In that way, the DL component reasons only once on the ontology and the information is used to generate the OO KB of the rule engine.

[1] Department of Informatics, Aristotle University of Thessaloniki, Greece, email: {gmeditsk, nbassili}@csd.auth.gr

2 OBJECT-ORIENTED MAPPING OF OWL

Let C be the set of *named classes*, R the set of *properties*, I the set of *individuals* of a DL reasoner's KB after the reasoning procedure over an ontology, and let C_o be the set of *classes*, R_o the set of *attributes* and I_o the set of *objects* of the OO model. Furthermore, $A \sqsubseteq B$ and $m : A$ is the DL syntax for class subsumption and class membership, and $EXT(A)$ is the *class extension* of A, that is the set of individuals that belong to the class A.

2.1 Class mapping

OWL classes are mapped into OO classes. The class transformation procedure implements the OWL axiom, stating that *owl:Thing subsumes every class and all individuals belong to the class extension of owl:Thing*.

The OO model is unable to represent directly the semantics of equivalent classes that impose mutual subclass relationships among them, in order to have the same class extension. For that reason, we introduce the notion of the *delegator* class.

C1. *For every set of equivalent classes D, we arbitrary choose a class* $A \in D$ *as the delegator class, such that* $\forall B \in D$, $dlg(B) = A$. *For each concept M with no equivalent classes,* $dlg(M) = M$.

Each class without any superclass becomes direct subclass of the OO $owl{:}Thing_o$ class.

C2. *Let a concept A for which* $\nexists N$ *such that* $A \sqsubseteq N$. *We define* $M_o \preccurlyeq owl{:}Thing_o$, *where* $M_o \leftarrow dlg(A_o)$.

Only delegator classes are involved in OO subclass relations.

C3. *Each* $M \sqsubseteq N$ *relation is mapped into the subclass relation* $A_o \preccurlyeq B_o$, *where* $A_o \leftarrow dlg(M_o)$ *and* $B_o \leftarrow dlg(N_o)$.

Class intersection and union are mapped into multiple OO subclass relations.

C4. *Let the concept A be the intersection of a set D of concepts. We define* $A_o \preccurlyeq M_o$, $\forall M_o \in D_o$.

C5. *Let a concept A be the union of a set D of concepts. We define* $M_o \preccurlyeq A_o$, $\forall M_o \in D_o$.

In class equivalence, the delegator becomes subclass of all its equivalent classes.

C6. *Let the set D of equivalent classes and* $A = dlg(N)$, $\forall N \in D$. *We define* $A_o \preccurlyeq M_o$, $\forall M_o \in D_o - \{A_o\}$.

2.2 Property mapping

Properties are mapped into class attributes. Let a property P with a domain set D.

P1a. *If* $D = \emptyset$, *we define* P_o *as an attribute of the owl:Thing*$_o$ *class in order to be inherited by all classes.*

P1b. *If* $D = \{M\}$, *then if* $\exists K$ *such that* $M \equiv K$ *and* E *is the equivalent class set, then we define* P_o *as an attribute of all* $N_o \in E_o - \{dlg(M)^{\Delta}\}$. *If* $\nexists K$ *such that* $M \equiv K$, *the property P is mapped directly as an attribute* P_o *in the* M_o *class.*

P1c. *If* $|D| \geq 2$ *then we create a class* T_o, *such that* $T_o \preccurlyeq N_o$, $\forall N_o \in D_o$, *and* P_o *is defined as an attribute of* T_o.

We follow the same approach for ranges. In the case of OWL datatype properties, we map range restrictions to actual datatypes, for example *xsd:int* restrictions into *Integer* types.

2.3 Individual mapping

Individuals are mapped into objects. Let the set D of concepts and an individual m, where $\forall N \in D, m : D$.

I1a. *If* $D = \{K\}$, *then* $m_o \mapsto A_o$, *where* $m_o \leftarrow m^G$ *and* $A_o \leftarrow dlg(K)^{\Delta}$.

I1b. *If* $|D| \geq 2$, *we create (or reuse, if exists from P1c) the class* $T_o \in C_o$, *such that* $T_o \preccurlyeq dlg(N_o)$, $\forall N_o \in D_o$, *where* $D_o = \{M_o \mid M_o \leftarrow M^{\Delta}, \forall M \in D\}$ *and we define* $m_o \mapsto T_o$, *where* $m_o \leftarrow m^G$.

Furthermore, individual property values are mapped into object attribute values.

I2. *Each* $\langle m, y\rangle : P$ *axiom is mapped by inserting the value y in the attribute* P_o *of the* m_o *object, where* $P_o \leftarrow P^S$ *and* $m_o \leftarrow m^G$. *If y is an individual (P is an object property) then* $y_o \in m_o.P$, *where* $y_o \leftarrow y^G$, *else (P is a datatype property)* $y \in m_o.P$.

3 EXAMPLES

Intersection: Consider the OWL ontology (DL syntax): $Father \equiv Male \sqcap \exists hasChild.Child$, $m : Male$, $n : Child$, $\langle m, n\rangle : hasChild$. The *m* instance will be classified in the *Father* concept by the reasoner, since it satisfies the existential restriction. Then, $Father \preccurlyeq Male$ (*C4*), $Child \preccurlyeq owl{:}Thing$ (*C2*), $hasChild \in Att(owl{:}Thing)$ (*P1a*), $m \mapsto Father$ and $n \mapsto Child$ (*I1a*) and $n \in m.hasChild$ (*I2*). Thus, $m \in Obj(Male)$ and $m \in Obj(Father)$, since $Father \preccurlyeq Male$. Notice, that only named concepts are mapped into classes.

Union: Consider the OWL ontology: $Human \equiv Man \sqcup Woman$, $m : Man$, $n : Woman$. Then, $Man \preccurlyeq Human$ and $Woman \preccurlyeq Human$ (*C5*), $m \mapsto Man$ and $n \mapsto Woman$ (*I1a*). Thus, $Obj(Human) = Obj(Man) \cup Obj(Woman) = \{m, n\}$.

Equivalence: Consider the OWL ontology: $Student \equiv Pupil$, $m : Student$, $n : Pupil$. Assuming that $dlg(Student) = dlg(Pupil) = Student$, then $Student \preccurlyeq Pupil$ (*C6*), $m \mapsto Student$ and $n \mapsto Student$ (*I1a*). Thus, $Obj(Student) = Obj(Pupil) = \{m, n\}$.

4 RELATED WORK

The OO transformation procedure of HOOPO is inspired by [6]. In this work, we target at the *hybrid* paradigm where OWL semantics are handled by a DL reasoner, without involving entailments.

The most closely related approaches to HOOPO are the [3][5] [9] and [2], where the ontology axioms are not altered and the ontology predicates are used as constraints in rule bodies. HOOPO differs from the above approaches on the fact that we approach the integration from an OO perspective and the DL constraints are determined directly by the OO KB, using both ontology class and property constraints in rule bodies, without runtime interaction between the DL and rule components. There are also approaches that target at the use of ontology predicates in rule heads, altering ontology axioms. Some examples are [8][10][4].

5 CONCLUSIONS

In this work we investigated the possibility of representing OWL extensional semantics following object-oriented principles in order to enable OO production rules to operate over OWL ontologies. The results show that it is possible to preserve the extensional ontology semantics of the transformed ontology. We plan to use HOOPO in the domain of semantic Web service discovery and composition.

ACKNOWLEDGEMENTS

This work was partially supported by a PENED program (EPAN M.8.3.1, No. 03ΕΔ73), jointly funded by the European Union and the Greek Government (General Secretariat of Research and Technology/GSRT).

REFERENCES

[1] G. Antoniou, C.V. Damasio, B. Grosof, I. Horrocks, M. Kifer, J. Maluszynski, P.F. Patel-Schneider, Combining Rules and Ontologies. A Survey, *Reasoning on the Web with Rules and Semantics*, REWERSE Deliverables, 2005.

[2] W. Drabent, J. Henriksson, J. Maluszynski, HD-rules: A Hybrid System Interfacing Prolog with DL-reasoners, *2nd International Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services*, 2007

[3] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, AL-log: Integrating datalog and description logics, *J. of Intelligent Information Systems*, vol 10(3), pp. 227-252, 1998.

[4] S. Heymans, L. Predoiu, C. Feier, J. Bruijn, D. Van Nieuwenborgh, G-Hybrid Knowledge Bases, *Applications of Logic Programming in the Semantic Web and Semantic Web Services* (ALPSWS), 2006

[5] A.Y. Levy, M. Rousset, Combining Horn rules and description logics in CARIN, *Artificial Intelligence*, vol 104 (1-2), 1998.

[6] G. Meditskos, N. Bassiliades, A Rule-based Object-Oriented OWL Reasoner, *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 3, pp. 397-410, Mar., 2008.

[7] J. Mei, Z. Lin, H. Boley, ALC^u: An Integration of Description Logic and General Rules, *Web Reasoning and Rule Systems*, 2007

[8] B. Motik, U. Sattler, R. Studer, 'Query answering for OWL-DL with rules', *J. of Web Semantics*, 3 (1), pp. 41-60, 2005.

[9] R. Rosati, Towards expressive KR systems integrating Datalog and description logics: Preliminary report. *In Proc. of DL'99*, 1999.

[10] R. Rosati, DL+log: Tight Integration of Description Logics and Disjunctive Datalog, *In Proceedings of the 10th KR*, 2006