Encoding DL-Atoms in RuleML

Samy Sá[1] **and João Alcântara**[1]

[1]Departamento de Computação, Universidade Federal do Ceará, P.O.Box 12166, Fortaleza, CE, Brazil 60455-760

{samy,jnando}@lia.ufc.br

Abstract. *We consider Description Logic Programs (DLPs) as consisting of a description logic knowledge base L and a generalized extended logic program P, which may contain queries to L through the DL-Atoms. Aiming to provide general syntax for representing rules, RuleML is a markup language and candidate for a standard exchange rule language in the Web. Currently under development from version 0.91 to version 1.0, its syntax still lacks the ability to express DL-Atoms and, consequently, DL-Programs. In order to overcome this gap, we propose an extension for RuleML expressive enough to represent all sorts of DL-Atoms.*

Resumo. *Consideramos Description Logic Programs (DLPs) como estes que consistem em uma base de conhecimento L em Lógica Descritiva e um programa em lógica estendido generalizado P, o qual pode conter consultas a L através de DL-Átomos. Objetivando prover sintaxe geral para a representação de regras, RuleML é uma linguagem candidata a padrão para troca de regras na Web. Atualmente em desenvolvimento da versão 0.91 para 1.0, sua sintaxe ainda não tem a habilidade de expressar DL-Átomos e, consequentemente, DL-Programs. Para superar esse obstáculo, nós propomos uma extensão para RuleML expressiva o suficiente para representar todos os tipos de DL-Átomos.*

1. Introduction

The *Semantic Web* [Berners-Lee and Fischetti 1999] is an extension of the *World Wide Web* technology that aims to establish a pattern for expressing the intended meanings of documents in the web along with their content. The Semantic Web consists of several layers enabling to build ontologies with increasingly expressive power. XML is the first layer and all others are built on top of it, meaning that documents in all superior layers are valid XML documents. The current top layer of the stack being developed comprises OWL and the representation of rules in order to allow reasoning over ontologies [Antoniou and van Harmelen 2008]. Now that OWL reaches a satisfactory level of maturity, the urge to efficiently represent rules in the most variate formats has taken over the Semantic Web development efforts and the W3C (World Wide Web Consortium) has assigned a working group to design a Rule Interchange Format (RIF). As part of this effort, the XML based *Rule Markup Language* (RuleML) [Dema and Hirtle 2006] is a candidate for becoming the standard language for exchange and representation of rules in both syntax and semantics. The main goal of RuleML is to allow the translation from any rule systems to others, in a single general framework. Currently, RuleML is being developed from version 0.91 into version 1.0. Even though there are improvements in the language, it still lacks the ability to express some kinds of rules.

Various notions relating *Description Logics* (DL) [Baader et al. 2003] and *Logic Programming* (LP) [Lloyd 1987] were proposed, including a translation from ontologies in the Description Logic language $\mathcal{ALCIQ}$ to logic programs [Alsaç and Baral 2002] and an algorithm to reduce a $\mathcal{SHIQ}$ knowledge base to a disjunctive datalog program [Hustadt et al. 2007]. Other approaches involve proposing Description Logic Programs as the intersection of horn logic programs and description logics [Grosof et al. 2003] and an extension of answer set programming that supports inverted predicates and infinite domains to integrate description logics and answer set programming [Heymans and Vermeir 2003]. Eiter et al., however, proposed a different concept of *Description Logic Programming* (*DLP*) in [Eiter et al. 2004, Eiter et al. 2008] to represent and reason over ontologies. Their approach enables a great deal of ontology integration on reasoning, as a logic program can access and consider the results of queries over Description Logic ontologies (DL ontologies) represented in expressive languages such as OWL DL or OWL Lite [Eiter et al. 2008]. Also, the results of queries might be based on results from both the logic program and the ontology through special operators. Next, we show a DLP rule taken from [Eiter et al. 2008] and explain its meaning.

Example 1.1 (Sample DLP Rule)

The following rule from a DLP P contains a query to a Description Logic ontology (DL ontology) L as its only condition. The query is described in the DL-Atom $DL[keyword \cup +kw; inArea](P,\ A)$, in which $keyword$ and $inArea$ are terms from the ontology L and kw is a predicate from the program P.

$$r : paperArea(P,\ A) \leftarrow DL[keyword \cup +kw; inArea](P,\ A);$$

The above rule queries L to find to what area A the paper P belongs. In this case, the query returns the results of the role $inArea$ from L as a predicate $inArea(P, A)$ based on the consideration $keyword \cup +kw$. This consideration means that the results of DLP predicate kw augments ($\cup+$) the set of results of the DL ontology role $keyword$, so the result of the query is based on data from both the program P and ontology L.

As it can be taken from example 1.1, programs and DL ontologies can be combined in very complex ways. In fact, DLP rules that include DL-Atoms are very peculiar, as they present queries inside some atoms. This feature is not common and rule exchange languages (RuleML v1.0 included) are not capable of expressing such queries. We point out that this kind of integration is compliant to the the goals of the Semantic Web and, as such, should be considered. In this paper we propose a modular extension capable of comprising DLP rules while respecting XML compatibility and show how to encode the characteristic atoms of these rules. We defend that our extension to RuleML 0.91 should be considered for adoption in the development of RuleML 1.0. This extension makes possible to represent DLP rules and therefore integrate description logic ontologies and logic programs as rules from one another are exchanged in the Semantic Web.

A related work is [Damásio et al. 2006], in which the authors propose a general extension to RuleML 0.9 for representing uncertainty rules from several frameworks such as fuzzy and probabilistic logic programming. Their proposal is aimed at being modular, to make for natural, easy to read translations while also trying to keep changes to the language of

RuleML 0.9 to a minimum. In this paper we adopt the same goals to propose an extension in which it is also possible to represent DL-Atoms.

The rest of the paper is organized as follows. Section 2 briefly introduces Logic Programs, Description Logics and the definitions of programs, rules, atoms and operators used to relate ontologies and programs in *DLP*. Section 3 provides an analysis of available and missing features of RuleML needed to express DL-Atoms and presents our suggested extension. Section 4 shows how to encode DL-Rules in RuleML. We summarize the steps towards proposing such extension, criticize results and comment on future work in section 5.

2. Preliminaries

2.1. Extended Logic Programs

An Extended Logic Programs (ELP) [Gelfond and Lifschitz 1991] comprises a finite sets of inferential rules of the form $r_i : A \leftarrow B_1, \ldots, B_k, notB_{k+1}, \ldots, notB_m, m \geq k \geq 0$, where A, B_i, $m \geq i \geq 0$, are classical literals. We refer to the literal A as the *Head* of the rule and the conjunction of the literals $B_1, \ldots, B_k, notB_{k+1}, \ldots, notB_m$ as the *Body* of the rule and write $Head(r)$ and $Body(r)$ to refer to those. The semantics of such programs are given by its Answer Sets (AS) [Gelfond and Lifschitz 1991].

2.2. Description Logics

Description logics (DL) [Baader et al. 2003] are a family of formal knowledge representation languages based on concepts and roles possibly applied to variables or terms representing particular entities in a domain. A *concept* is a unary predicate that qualifies a set of entities and a *role* is a binary predicate defining a relationship between entities. A Description Logic knowledge base, sometimes referred to as an ontology, consists of two parts traditionally referred to as the *TBox* (Terminological Box) and *ABox* (Assertional Box). The first consists of assertions about concepts and roles in the sense of defining their meaning towards each other (e.g concept inclusion). The second presents specific information about particular entities belonging to concepts or pairs of entities belonging to roles.

2.3. Description Logic Programs

The literature comprises several approaches to relate DL and logic programming such as [Alsaç and Baral 2002, Hustadt et al. 2007, Grosof et al. 2003, Heymans and Vermeir 2003, Eiter et al. 2004, Eiter et al. 2008]. Except by *Description Logic Programs* (DLP) as in Eiter et al. [Eiter et al. 2004, Eiter et al. 2008], the proposed syntax from all others can be easily expressed in RuleML. For instance, in [Heymans and Vermeir 2003] it is proposed the inclusion of a *predicate inversion* operator to disjunctive logic programs syntax, resulting in the extended language *Conceptual Logic Programs* (CLP). This extension allows for direct translation from DL formulas using the *Role Inversion* construct. In CLP we write $P^-(X_1, X_2)$ to address the inverse of $P(X_1, X_2)$ (instead of using $P'(X_2, X_1) \leftrightarrow P(X_1, X_2)$), P being a binary predicate, or $P^-(X_1)$ as the inverse of $P(X_1)$, $P^-(X_1) = P(X_1)$, P being a unary predicate. There are a few ways we can represent such extension to syntax, such as by suggesting a new tag to involve atoms and account for its inversion, a new attribute to the tag <Rel>

to indicate inversion or just by using a simple translation that requires no change to RuleML, such as done in [Grosof et al. 2003]. Since the role inversion construct can be successfully represented in RuleML 0.91, we are left with only the approach due to Eiter et al. in [Eiter et al. 2008] and [Eiter et al. 2004] for consideration. In their work, characteristic atoms are introduced to the body of rules in order to integrate Description Logic ontologies (DL ontologies) and extended logic programs. These atoms require special attention, as they can not be expressed in RuleML 1.0. From this point and on, we mean by DLP the notion introduced in [Eiter et al. 2004, Eiter et al. 2008]. In this section we introduce the elements composing DLPs. Most of the definitions below are taken from [Eiter et al. 2008], but the name of considerations is a suggestion of our own.

Definition 2.1 (DL-Queries) *A* DL-Query *is a question to a DL knowledge base L in one of the forms:*

1. *a concept inclusion axiom F or its negation $\neg F$.*
2. *$C(t)$ or $\neg C(t)$, where C is a concept and t is a term.*
3. *$R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role and t_1, t_2 are both terms.*

Definition 2.2 (Considerations) *A* Consideration *is a sequence $S_i op_i p_i$, where S_i is either a concept or role from L, $op_i \in \{\cup+, \cup-, \cap-\}$ and p_i is, respectively, a unary or binary predicate from P relative to S_i.*

The operators op_i are able to augment or retract facts about S_i (from L) or its negation through p_i (from P) as (1.) $op_i = \cup+$ increases the set of positive results of S_i by adding instances of p_i while (2.) $op_i = \cup-$ increases the set of results of $\neg S_i$ by adding instances of p_i and (3.) $op_i = \cap-$ restricts possible results of S_i to the ones of p_i.

For instance, suppose a query $Q : inArea$ to L depends on instances of $keyword$ in the TBox of L. In case the consideration $keyword \cup +kw$ is applied to Q, the instances of predicate kw in P are taken as if they were translated to instances of $keyword$ in L before Q is processed and discarded right after, therefore possibly changing the output of Q.

Definition 2.3 (DL-Atoms) *Special atoms allowed in the syntax of DLPs that specify queries to L.* DL-Atoms *have the form*

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](t),\ m \geq 0,$$

where $Q(t)$ is a DL-Query and each sequence $S_i op_i p_i$ is a consideration to the query $Q(t)$.

A query inside DL-Atoms ask about the occurrence or evidence of either a description logic axiom or its negation in L. Whenever a query $Q(t)$ is to be calculated, it takes into account inferences driven from P according to considerations to the query inside the DL-Atom [1].

Definition 2.4 (DL-Rules) *A DL-Rule is an extended logic program rule possibly presenting DL-Atoms in its body. A DL-Rule has the form $r : L_0 \leftarrow L_1, \ldots, L_k, notL_{k+1}, \ldots, notL_m, m \geq k \geq 1$, where L_0 is a classical literal and L_i, $m \geq i \geq 1$, are either classical literals or DL-Atoms. We refer to L_0 as the* head *of r and the conjunction $L_1, \ldots, L_k, notL_{k+1}, \ldots, notL_m$ as the* body *of r.*

[1] For a more detailed explanation, we invite the reader to address to [Eiter et al. 2004, Eiter et al. 2008].

Definition 2.5 (Description Logic Programs) *A* Description Logic Program *(DLP) is a pair KB = (L, P), where L is a Description Logic knowledge base and P is a logic program possibly with occurrence of DL-Rules. DLPs can present queries to L in the body of rules of P and integrate the knowledge base and logic program allowing for rules to work on top of ontologies and ontologies to be built on top of rules.*

The following example is a partial transcript from [Eiter et al. 2004, Eiter et al. 2008].

Example 2.1 (Reviewer Selection) *Suppose we want to assign reviewers to papers, based on certain information about the papers and the available persons, using a description logic knowledge base L_S which contains knowledge about scientific publications. Consider then the DL-Program $KB_S = (L_S, P_S)$, where P_S contains in particular the following DL-Rules:*
(1) $paper(p_1)$*;* $kw(p_1,\ Semantic\_Web)$*;*
(2) $paper(p_2)$*;* $kw(p_2,\ Bioinformatics)$*;* $kw(p_2, Answer\_Set\_Programming)$*;*
(3) $kw(P,\ K_2) \leftarrow kw(P,\ K_1),\ DL[hasMember](S,\ K_1), DL[hasMember](S,\ K_2)$*;*
(4) $paperArea(P,\ A) \leftarrow DL[keyword \cup +kw; inArea](P,\ A)$*;*
(5) $cand(X,\ P) \leftarrow paperArea(P,\ A),\ DL[Referee](X),\ DL[expert](X,\ A)$.

Intuitively, (1) and (2) specify the keyword information of the papers p_1, p_2 which should be assigned to reviewers. Rule (3) augments the set of keywords from P_S for a paper P with similar ones obtained through queries to L_S. The DL-Atoms in (3) query L_S for keywords other than K_1 that are members of the same keyword cluster S. Rule (4) queries L_S to assert a paper P to an area A while considering all results of $keyword$ (from L_S) and kw (from P_S) of P. Finally, rule (5) singles out review candidates based on the information from experts among the reviewers according to L_S. Note that, in view of rules (3) to (5), information flows in both directions between the knowledge encoded in L_S and P_S.

3. Extension of RuleML

RuleML [Tabet et al. 2001] is a normalized modular markup language designed to express and exchange rules in the Web. RuleML resorts to a set of tags and attributes with preconceived nesting[2] admissibility and semantics. About to enter version 1.0 [Dema and Hirtle 2006, Boley et al 2010], RuleML is fully XML compatible and capable of expressing rules in various working frameworks ranging from Datalog (function-free horn logic) [Silberschatz et al. 2001] to HILOG [Chen et al. 1993]. Specialized markup languages are also available for specific rule terminology[3] such as FOL RuleML [Boley et al 2004] for first order logic. Nonetheless, RuleML up to version 0.91 is unable to express the special case of rules including sub-queries to external knowledge bases. In this section we analyze the main issues for representing DL-Atoms and DL-Rules in RuleML 0.91 and propose a minimal and modular extension. It means that (1) changes to RuleML are kept to a minimum while attempting to maximize expressiveness; (2) any valid RuleML document will still be a valid document in our extended version and that future work should also be easily compatible to ours. Our extension also provides a straight-forward translation of DLP-Rules.

[2]For any two particular tags, the tags nesting specifies whether or not one can be used in the scope of the other.

[3]Specialized rule markup languages might present a different nesting or set of tags/attributes than that of RuleML.

3.1. Special needs of DL-Atoms

We now summarize the features of DL-Atoms and which elements are missing for encoding those features in RuleML 0.91.

- DL-Atoms should somehow be identified as a different kind of atom;
- The adequate structure for representing queries is the performative <Query> [Dema and Hirtle 2006] but performatives can not occur nested in RuleML;
- Considerations relate concepts/roles in L to predicates in P and omit variables. Also, there are no performatives or tags suitable to encompass the semantics of a consideration.
- There are no tags for the operators $\cup+$, $\cup-$, $\cap-$.

Some of those issues are solved by employing tags already in RuleML. Relations as concepts, roles and predicates can be comprised under the tag <Rel>. Variables and constants can be expressed respectively by <Var> and <Ind>. Also, in order to associate relations, it is enough to wrap them as the only content inside a tag <Implies>. In such a case, the predicate (condition) comes first and the concept/role (conclusion) follows, just as the first extends the second and not the opposite. In addition, we need to:

- Allow sub-queries in the body of rules.
- Represent the special operators $\cup+$, $\cup-$, $\cap-$ used in considerations to DL-Queries.
- Propose a set of tags and nesting capable of relating concepts/roles to predicates in considerations.

3.2. Changes to syntax

In DLPs there may be atoms able to express taking action in which rules can query other rules. The general construct for expressing action in RuleML are the performatives [Dema and Hirtle 2006]. In plain RuleML there are three performatives: <Assert>, <Query> and <Retract> and they are meant to add information, express queries and eliminate information, respectively. In FOL RuleML 0.9 [Boley et al 2004], however, we have the performative <Consider> meant to add information only locally, i.e., such information is not part of the theory where inference is driven, but extra. Thus <Consider> seems to be the most adequate amongst performatives to express considerations to queries within DL-Atoms.

In previous versions, RuleML had an attribute @kind exclusive to tags <Implies>. This attribute was used to identify the kind of implication by values ”fo” for first order and ”lp” for logic programming, but it was renamed in RuleML 0.91 to @material. Other work such as [Damásio et al. 2006] proposes applying @kind back to elements and connectives to express the language in which such elements should be interpreted. We defend the usefulness of @kind as such, as its use in <Atom> with a new possible value ”dl” is enough for identifying DL-Atoms. Also, note that the nesting of queries inside tags <Atom> identified as DL-Atoms is adequate to express the queries inside DL-Atoms. Considerations should be expressed inside the same element <Atom> as its related query, and we use the performative <Consider> to do so. Relations from both P and L are related inside each consideration through the use of <Implies>. Finally, instead of proposing new tags for the operators $\cup+$, $\cup-$ and $\cap-$, we show how they can

be translated into RuleML as specific sequences of nested tags inside DL-Atoms. We have then extended RuleML by:

- allowing the performatives <Query> and <Consider> to occur inside elements <Atom>;
- adding @kind to elements <Atom> and a new possible value "dl".

We now show the mentioned elements and performatives content models from RuleML 0.91 in the same style as in [Damásio et al. 2006, Dema and Hirtle 2006]. The exact same changes would be applied to the current proposal of RuleML version 1.0. Below, our proposed changes are typed in bold to emphasize which elements are being added as part of our proposed extension.

Query	**Consider**
attribute: @closure	**attribute: @closure**
content: (oid?, (formula)*)	**content: (oid?, (formula)*)**
within Atom …	**within Atom …**
content: (oid?, (formula))	**content: (oid?, (formula))**

Figure 1. Content Model for Performatives

We change the content models for performatives in two ways. First, performative <Query> can now occur inside <Atom> elements. The second change introduces <Consider> in standard RuleML with the same possible content as <Query> and also able to occur inside <Atom> elements. Optional parameters receive a question mark (?). When a parameter can occur zero or multiple times we use a star mark (*).

Atom
attribute: @closure, **@kind**
content: (oid?, **((formula)* | ((Consider)*, Query))**

@kind
[optional] (default:fo | lp | **dl**)

Figure 2. Content Model for Elements

Content models for elements are changed in three ways. Elements <Atom> can now show performatives <Query> and <Consider> and have a new attribute @kind. We also enable the possible value "dl" to @kind. Having introduced these changes, we can now encode *DLP* rules in RuleML as expounded in the sequel.

4. Encoding DLP Rules

In this section we show how to use the extended RuleML syntax proposed in section 3 to encode DLPs. As extended logic programs can be easily expressed in RuleML 1.0, only DL-Atoms are left for encoding. Indeed, each kind of consideration in a DL-Atom requires a specific codification in RuleML. For an easier comprehension of the reader, we opt to show our translation through examples of DLP rules.

4.1. DL-Atoms with no Considerations to the Query

The first kind of DL-Atom to be considered consists of just a query to L, with no considerations. RuleML provides a performative <Query>, but does not allow for its use inside tags <body>. We simply extend the syntax of RuleML so we can have this behavior of tags, and identify the atom as being of kind DL through attribute @kind inside <Atom>. The value "dl" for @kind is a new possible value, added to our extension.

Example 4.1 *Consider the DL-Atom $DL[inArea](P,\ A)$, that specify a query to find in L what papers (P) qualifies in which areas (A). This single DL-Atom can be represented in our extension as*

```
<Atom kind = "dl">
     <Query>
          <Atom>
             <Rel>inArea</Rel>
             <Var>P</Var>
             <Var>A</Var>
          </Atom>
     </Query>
</Atom>
```

In this example, @kind has the value "dl" in <Atom> and the performative <Query> inside it. Both changes are necessary to translate DL-Atoms to RuleML and back. Content of <Query> is still as usual.

4.2. DL-Atoms with Considerations to the Query

In order to represent more complex DL-Rules, we need to deal with considerations and, in particular, to translate the operators $\cup+$, $\cup-$, $\cap-$. In order to introduce considerations inside DL-Atoms, we use the performative <Consider> from FOL RuleML (0.9) [Boley et al 2004]. For simplicity, our examples will only have one consideration at a time and its query content omitted, as they can still be encoded as shown in Subsection 4.1 in a DL-Atom with considerations. For the general case, multiple occurrences of <Consider> take place. We now present a brief account for each kind of consideration found in DLPs.

4.2.1. Considerations to the Query with operator $\cup+$

Whenever the operator $\cup+$ is used in a consideration $S_i op_i p_i$, the instances of p_i in P add to the instances of S_i in L for processing the query. As such, the consideration works in similar way to a normal logic program rule with S_i in the head and p_i in the body. Its result, however, is to be reconsidered each time the query is needed during execution of the DLP. We encode the consideration $S_i \cup +p_i$ using the same structure of general logic programs rules inside an occurrence of the performative <Consider>

Example 4.2 *Let us recall the DL-Atom $DL[keyword \cup+ kw;\ inArea](P,\ A)$ from rule (4) of Example 2.1. The query is the same of Example 4.1, but it has to take into account the extension of keyword (from L_S) by the results of kw (from P_S). By the application of the operator $\cup+$, kw extends the set of positive occurrences of keyword as $kw(p, k)$ allows*

to consider $keyword(p, k)$. *We write the DL-Atom* $DL[keyword \cup + \ kw;\ inArea](P,\ A)$ *in RuleML as:*

```
<Atom kind = "dl">
     <Consider>
          <Implies closure = "universal">
             <Atom>
                 <Rel>kw</Rel>
             </Atom>
             <Atom>
                 <Rel>keyword</Rel>
             </Atom>
          </Implies>
     </Consider>
     <Query> ... </Query>
</Atom>
```

Our considerations are represented through the performative <Consider> with a connective <Implies> as its only content. We call attention to the lack of variables in each occurrence of <Atom> above, as considerations in DL-Atoms do not enumerate the variables for any of the predicates. Both predicates in each consideration are supposed to have the same number and order of arguments.

4.2.2. Considerations to the Query with operator $\cup -$

The operator $\cup -$ in $S_i \cup -p_i$ means that any instance of p_i in P adds to the instances of $\neg S_i$ in L for processing the query. As in Subsection 4.2.1, we use the performative <Consider>, but now the conclusion inside <Implies> is negated.

Example 4.3 *The DL-Atom* $DL[isMale \cup - \ isFemale;\ gender](A,\ S)$ *express a query about the gender of animals who accounts that females are not males. The instances of the predicate* isFemale *in* P *extend the set of negative instances of the concept* isMale *in* L. *It is represented in RuleML as follows:*

```
<Atom kind = "dl">
     <Consider>
          <Implies closure = "universal">
             <Atom>
                 <Rel>isFemale</Rel>
             </Atom>
             <Neg>
                 <Atom>
                    <Rel>isMale</Rel>
                 </Atom>
              </Neg>
          </Implies>
     </Consider>
     <Query> ... </Query>
</Atom>
```

The only difference between two considerations (i) $S_i \cup + p_i$ and (ii) $S_i \cup - p_i$ is that results of p_i add to the set of *positive* instances of S_i in (i) and to the *negative* instances of S_i in (ii). Their representation differs by the negation of the second atom (head/conclusion) through the use of the connective <Neg>.

4.2.3. Considerations to the Query with operator $\cap -$

Considerations of the kind $S_i \cap -p_i$ are such that the failure in proving instances of p_i in P adds to the instances of $\neg S_i$ in L for processing the query, restricting the results of S_i to those of p_i. In this sense, the negation as failure is applied to the condition inside <Implies> and conclusion is negated as in 4.3.

Example 4.4 *Consider the DL-Atom* $DL[Referee \cap - poss\_Referee;\ Referee](X)$ *meant to restrict the concept* Referee *(from* L*) up to the values to which* poss_Referee *(from* P*) is true. It can be represented in RuleML as follows:*

```
<Atom kind = "dl">
     <Consider>
          <Implies closure = "universal">
             <Naf>
                 <Atom>
                    <Rel>poss_Referee</Rel>
                 </Atom>
             </Naf>
             <Neg>
                 <Atom>
                     <Rel>Referee</Rel>
                 </Atom>
             </Neg>
          </Implies>
     </Consider>
     <Query> ... </Query>
</Atom>
```

Just as the operator $\cap -$ is defined, we use negation as failure of *poss_Referee* as premise in order to extend the set of negative instances of *Referee*.

The Table 1 summarizes the uses of connectives <Neg> and <Naf> for each operator:

\	<Neg>	<Naf>
$\cup +$	No	No
$\cup -$	Yes	No
$\cap -$	Yes	Yes

Table 1. Usage of <Neg> and <Naf> for each operator in DLP

DL-Rules can be encoded as most extended logic program rules, the only difference being the possible DL-Atoms in the body. We presented how to translate these atoms to RuleML

in a way that both syntax and semantics of DL-Atoms are properly represented. We reinforce that multiple occurrences of the performative <Consider> are needed in case of a DL-Atom with multiple considerations to its query.

5. Conclusion

We have proposed an extension for RuleML capable of expressing DL-Atoms and allowing exchange of DL-Programs rules in the Web. This extension relies on the use of the attribute @kind and occurrence of performatives <Query> and <Consider> inside <Atom> elements and should be considered for inclusion in RuleML 1.0. We also show through examples how this simple extension enables expression of DL-Atoms and, consequently, DL-Programs. This extension does not introduce new terminology to RuleML, for it only suggests new uses and nesting for current tags and an attribute. It provides the necessary expressive power and a straightforward encoding capable of coping with both syntax and semantics of DL-Rules. We defend that allowing performatives to occur inside atoms in the body of rules is very useful, as it also makes possible to easily represent commands from rule based language such as *append(X)* from Prolog through the use of a performative <Assert>. A more profound study of performatives inside atoms is left for future work. Also, we intend to work on the development of RuleML editors to include our suggestions.

6. Acknowledgements

Research partially supported by CAPES (PROCAD).

References

Alsaç, G. and Baral, C. (2002). Reasoning in description logics using declarative logic programming. Technical report, Arizona State University, Arizona.

Antoniou, G. and van Harmelen, F. (2008). *A Semantic Web Primer*. MIT Press, Cambridge, MA, 2. edition.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Berners-Lee, T. and Fischetti, M. (1999). *Weaving the Web*. Harper.

Boley et al, H. (2004). The first-order logic web language v0.9. `http://ruleml.org/fol/`.

Boley et al, H. (2010). Ruleml v1.0 specification. `http://www.ruleml.org/1.0/`.

Chen, W., Kifer, M., and Warren, D. S. (1993). HILOG: A foundation for higher-order logic programming. *JOURNAL OF LOGIC PROGRAMMING*, 15(3):187–230.

Damásio, C. V., Pan, J. Z., Stoilos, G., and Straccia, U. (2006). An approach to representing uncertainty rules in ruleml. In Eiter, T., Franconi, E., Hodgson, R., and Stephens, S., editors, *RuleML*. IEEE Computer Society.

Dema, T. and Hirtle, D. (2006). Content models for ruleml 0.91. `http://ruleml.org/1.0/xsd/content_models_10.pdf`.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., and Tompits, H. (2008). Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13):1495–1539.

Eiter, T., Lukasiewicz, T., Schindlauer, R., and Tompits, H. (2004). Well-founded semantics for description logic programs in the semantic web. In Antoniou, G. and Boley, H., editors, *RuleML*, volume 3323 of *LNCS*, pages 81–97. Springer.

Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386.

Grosof, B., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: Combining logic programs with description logics. In *Proceedings of the World Wide Web Conference (WWW2003)*, Hungary.

Heymans, S. and Vermeir, D. (2003). Integrating description logics and answer set programming. In Bry, F., Henze, N., and Maluszynski, J., editors, *Proc. of International Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2003)*, volume 2901 of *LNCS*, pages 146–159. Springer.

Hustadt, U., Motik, B., and Sattler, U. (2007). Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reason.*, 39(3):351–384.

Lloyd, J. (1987). *Foundations of Logic Programming (2nd Extended Edition)*. Springer-Verlag.

Silberschatz, A., Korth, H., and Sudarshan, S. (2001). *Database System Concepts, 4th Edition.* McGraw-Hill.

Tabet, S., Boley, H., and Wagner, G. (2001). Design rationale of RuleML: A markup language for semantic web rules. In Cruz, I. F., Decker, S., Euzenat, J., and McGuinness, D. L., editors, *Proc. Semantic Web Working Symposium*, Stanford University, California.