ORDBase: A Common Architecture for Web Ontologies, Rules, and Data

Jing Mei, Zuoquan Lin

Department of Information Science
Peking University, Beijing 100871, China
{mayyam, lz}@is.pku.edu.cn

Abstract. The ORDBase architecture is introduced as a common platform for Web ontologies, rules, and data. The ontology base (here using OWL) is well-suited to represent structured knowledge. The rule base (here using RuleML) is the key to inferencing, whose predefined rules serve for translating the OWL semantics, also user-defined rules are considered. The database (here using SQL) provides for fast retrieval following the RDF data model of (subject predicate object) triples. Knowledge can be entered, queried, and derived with the ORDBase suite of tools provided as an open toolbox in Java.
Keywords: Semantic Web, ontologies, rules, databases, reasoning.

1 Introduction

While powerful expressively, Prolog and traditional (FOL) reasoning systems are hardly amenable to non-expert users. Support for non-standard reasoning services, in particular for queries and inferences over RDF knowledge bases, is not much better in this regard.

Ontology editors, such as Protege, have provided a development environment for OWL editing, and this "in-one-view" tool combines a sophisticated graphical layout with modern UI features for browsing and editing [10]. By now, building an OWL ontology is a relatively plain task. On the other hand, it is more straightforward to query a database than using an XML-based syntax, and main memory-based systems seem less attractive than databases, especially for scalable data [5].

The ORDBase – {Ontology, Rule, Data}Base – architecture is proposed to bridge the two above approaches as follows. Its ontology component generates a well-formed OWL[11] file, which is combined with a RuleML[2] rule component consisting of facts, predefined rules and user-defined rules. The information from the initial OWL file is completely transformed into triple facts, being of the form statement = (subject predicate object). Moreover, the characteristics of each OWL constructor is translated into the corresponding rules in advance; while user-defined rules can be embedded in OWL as a SWRL[8] file, or directly asserted in the targeted rule base. After running such an ontology-rule base in a rule engine, all resulting statements are exported into the third ORDBase component – a database, whose data model is still that of (subject predicate object) triples, so retrieval can use standard SQL.

The rest of the paper is organized as follows. The ORDBase architecture is presented in section 2, and section 3 illuminates a rule base to implement OWL semantics, consisting of five groups: transformed, user-defined, preprocessed, basic-part, and core-part. In section 4, we describe our implementation, and related work is discussed in section 5. Finally, section 6 gives our conclusions.

2 ORDBase Architecture

As shown in Figure-1, the author builds an ontology base in OWL, optionally with ontology editors. From this OWL file, facts are extracted into a RuleML rule base, whose predefined rules serve for translating OWL (RDF-compatible) Model-Theoretic Semantics[1]. SWRL rules will be included in the "transformed" part, while the "user-defined" part is for those directly in RuleML. The rest three parts, i.e., "preprocessed", "basic-part" and "core-part", are exactly predefined, and next section will introduce them in detail. Via (XSLT) translators for RuleML, a rule engine is run, resulting in most of desired statements. The results can be stored, using a tool for exporting, with retrieval in a database.

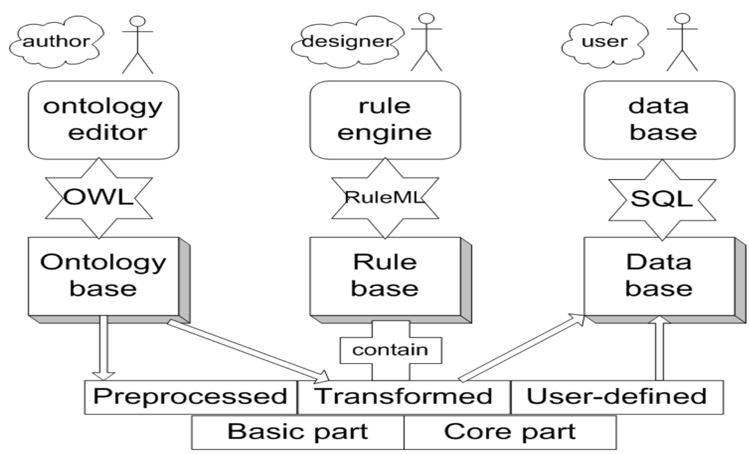

Fig. 1. ORDBase Architecture

Generally, the simple structures are more comprehensive than complex ones. A unique data model of statement = (subject predicate object) is adopted, not only as a template of rules, but also as a table of the database. On top of that, it is accessible for key reasoning tasks [12], including: concept consistency (? rdf:type foo:c), concept subsumption (foo:c1 rdfs:subClassOf foo:c2), instance

[1] in this paper, we call it as OWL semantics [11] for short

checking (foo:i rdf:type foo:c), instance retrieval (? rdf:type foo:c), classification (? rdfs:subClassOf ?). However, for KB consistency, error messages will be thrown if there is a contradiction; and it fails to entailment (i.e., from a KB to entail all axioms in another KB).

3 A RuleML Rule Base

First, running a rule engine, the transformed part is imported, including initial facts and SWRL rules. Following, the preprocessed part is added, for authors who build ontologies manually. Next, the basic part can be ignored, if users do not care about constructors like (rdfs:Class rdf:type rdfs:Class). Besides, the core part must be considered, for reasoning about the initial OWL file. Finally, the user-defined part is also regarded as a first-class citizen in the targeted rule-based language.

3.1 Transformed

The transformation from OWL to RuleML is done by means of XSLT. Starting from the root, recurrent processes – ABox-class and ABox-property – are invoked via a set of named templates. The output file entirely consists of ruleml:Fact. Since, the semantics of the underlying ontology language has been already specified in a RuleML rule base, specific keyword matching is unnecessary in this XSL transformation. As follows, the ruleml:Rel of "statement" followed by its three arguments: (subject predicate object), is similar to an RDF statement.

```
   <xsl:template name="ABox-class" >
<Fact>
 <Atom>
  <Rel>statement</Rel>
  <Ind><xsl:call-template name="get-ID"/></Ind>
  <Ind>rdf:type</Ind>
  <Ind><xsl:value-of select="name(.)"/></Ind>
 </Atom>
</Fact>
     <xsl:for-each select="*">
      <xsl:call-template name="ABox-property"/>
     </xsl:for-each>
   </xsl:template>
```

Instead of <Rel>predicate</Rel><Ind>subject</Ind><Ind>object</Ind>, it provides <Rel>statement</Rel>, whose second argument is the predicate symbol, since the variable is not permitted as a ruleml:Rel. That is, statement/3 serves for reducing second-order syntax to first-order syntax, making it possible to express semantic conditions for OWL.

For those rules embedded in OWL, i.e., SWRL rules, another XSLT stylesheet helps to implement its transformation from RDF syntax to RuleML syntax, resulting in an enlargement of this part.

3.2 Basic Part

The metadata of OWL semantics mostly concern about the universe and syntactic categories, and RDF, RDFS, OWL axiomatic triples will directly be written as ruleml:Fact (i.e., an implication with empty body).

<body>	<head>
s p o	s | p | o rdf:type rdfs:Resource
s p o	p rdf:type rdf:Property
s rdf:type o	o rdf:type rdfs:Class
s rdf:type rdfs:Class	s rdfs:subClassOf s
	owl:Class rdfs:subClassOf rdfs:Class
	owl:Thing rdf:type owl:Class

3.3 Preprocessed

With the help of ontology editors, some semantic conditions have been implemented. The class (resp. property) hierarchy tree is the case, which presents explicitly the transitive relationship of rdfs:subClassOf (resp. rdfs:subPropertyOf).

<body>	<head>
u rdfs:subClassOf v , v rdfs:subClassOf w	u rdfs:subClassOf w
u rdfs:subPropertyOf v , v rdfs:subPropertyOf w	u rdfs:subPropertyOf w

Besides, object properties are permitted to be functional, inverse functional, symmetric or transitive; while datatype properties merely to functional. For those OWL built-in properties, such as owl:inverseOf, the domain and range have been scoped in a certain OWL built-in class. However, owl:unionOf and owl:intersectionOf actually relate to a series of owl:Class, rather than rdf:List. Consequently, we skip the verbose syntax of rdf:List, and group them here.

<body>	<head>
s rdf:type owl:Class	s rdfs:subClassOf owl:Thing
s rdf:type owl:ObjectProperty	s rdfs:domain owl:Thing , s rdfs:range owl:Thing

<Fact>
owl:FunctionalProperty rdfs:subClassOf owl:ObjectProperty | owl:DatatypeProperty
owl:InverseFunctionalProperty rdfs:subClassOf owl:ObjectProperty
owl:equivalentClass rdfs:domain(range) owl:Class
owl:inverseOf rdfs:domain(range) owl:ObjectProperty
owl:complementOf rdfs:domain(range) owl:Class
owl:unionOf rdfs:domain(range) owl:Class

3.4 Core Part

The important characteristics of RDF, RDFS and OWL vocabularies are illuminated in this group, from which we can draw key inferences.

It is straightforward about rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range, as well as the features of transitivity and so on.

<body>	<head>
s rdfs:subClassOf o , x rdf:type s	x rdf:type o
s rdfs:subPropertyOf o , x s y	x o y
s rdfs:domain o , x s y	x rdf:type o
s rdfs:range o , x s y	y rdf:type o
s rdf:type owl:FunctionalProperty , x s y1 , x s y2	y1 owl:sameAs y2
s rdf:type owl:InverseFunctionalProperty , x1 s y , x2 s y	x1 owl:sameAs x2
s rdf:type owl:SymmetricProperty , x s y	y s x
s rdf:type owl:TransitiveProperty , x s y , y s z	x s z

For OWL does not employ the Unique Name Assumption(UNA), the two ones y_1, y_2 related to owl:FuntionalProperty will refer to the same individual. Furthermore, as shown below, owl:sameAs is explained with ruleml:Equal, while ruleml:Neg(the classical negation) works in owl:disjointWith and owl:differentFrom.

<body>	<head>
s owl:equivalentClass o	s rdfs:subClassOf o , o rdfs:subClassOf s
s owl:disjointWith o , x rdf:type s	<Neg>x rdf:type o</Neg>
s owl:sameAs o	<Equal> s o </Equal>
s owl:differentFrom o	<Neg><Equal> s o </Equal></Neg>

However, to our knowledge, some rule engines only provide the string or arithmetic equations, rather than the equivalence predicate. As for a practical solution, (s owl:sameAs o) means that, unless the strings of s and o are identical, all positions where s appears should be substituted by o, i.e., (s a b) implies (o a b), (a s b) implies (a o b), (a b s) implies (a b o), and vice versa. About (s owl:differentFrom o), it is naturally satisfied if the strings are not the same, resulting in a constraint instead of an assertion in practice.

Facing to the expressions of OWL boolean combinations and restrictions, we would deal with the Open World Assumption(OWA) cautiously. As known, OWL adopts OWA, while the opposite approach, namely Closed World Assumption(CWA), is more useful in logic programming, in particular the negation-as-failure(Naf). Generally, OWL is written to be universally applicable under any circumstances, such as in a Web application; on the contrary, a rule base contains just pieces of information scattered around the Web, and at any moment we are working with the information we have found. Therefore, it is dangerous to close the current ontology for drawing inferences.

Suppose, a web page (in OWL or RDF) that lists some books the author has, for example, and the designer of the rule base assumes (wrongly) that this list is complete, and so infers (wrongly) that because "Gone With the Wind" is not in it, that the author has no this book, then it is the designer who made mistakes, neither OWL nor the author. To handle the dilemma, we point out some optional rules, which would take effects if the designer really has already known the information source being complete.

owl:unionOf

From (s owl:unionOf o), it easily infers that (o rdfs:subClassOf s). However, with (x rdf:type s) but no (x rdf:type o), it is not trivial to decide x belonging to which subclass o. And such mandatory assignation should not be done by the designer in an open world, hence we provide the caution messages to reminder the author, cf. the right column. Furthermore, $\exists w.C(w) \leftarrow A$ equals to $0 \leftarrow A \wedge \forall w.\neg C(w)$, where the explicit quantifier "ruleml:Forall"[2] in the body is converted to an implication $D \leftarrow B$, also to $\neg B \vee D$. On the other hand, being an optional one, the left directly employs the explicit quantifier "ruleml:Exists" in the head of a rule, beyond the form of general rules.

```
<Implies>
 <head>
  <Exists>
   <Var> y </Var>
   <And>
    s owl:unionOf y
    x rdf:type    y
   </And>
  </Exists>
 </head>
 <body>
  <And>
   s owl:unionOf o
   x rdf:type    s
  </And>
 </body>
</Implies>
```

```
<Implies>
 <head>
  caution messages
 </head>
 <body>
  <And>
   s owl:unionOf o
   x rdf:type    s
   <Or>
    <Neg>
     s owl:unionOf y
    </Neg>
    <Neg>
     x rdf:type y
    </Neg>
   </Or>
  </And>
 </body>
</Implies>
```

owl:intersectionOf

Similarly, from (s owl:intersectionOf o), it infers that (s rdfs:subClassOf o). With "ruleml:Forall" in the body, the below left column is optional, for it is possible that another currently unknown o' occurs, s.t. (s owl:intersectionOf o') without (x rdf:type o'), resulting in the previous result (x rdf:type s) invalid. And the right provides another way, if no support for an implication in the body.

```
<Implies>
 <head>
   x rdf:type s
 </head>
 <body>
  <Forall>
    <Var> y </Var>
    <Implies>
     <head>
      x rdf:type y
     </head>
     <body>
      s owl:intersectionOf y
     </body>
```

```
<Implies>
 <head>
  x rdf:type s
 </head>
 <body>
  <And>
   s owl:intersectionOf o
   x rdf:type           o
   <Or>
    <Neg>
     s owl:intersectionOf y
    </Neg>
    x rdf:type y
   </Or>
```

[2] ruleml:Forall and ruleml:Exists are in FOL RuleML[1], hence, our rule base is actually a use case of FOL RuleML

```
  </Forall>                     </And>
 </body>                       </body>
</Implies>                    </Implies>
```

owl:allValuesFrom

On the premise of OWA, it is impossible to find out "all" issues, because there are much more unknown things. For the optional left, we attempt to check all existing elements with its type, and assert the proper ones; while the right column presents an obvious definition corresponding to OWL semantics.

```
<Implies>                          <Implies>
 <head>                             <head>
   x rdf:type s                      y rdf:type o
 </head>                            </head>
 <body>                             <body>
  <And>                              <And>
   s owl:allValuesFrom o              s owl:allValuesFrom o
   s owl:onProperty    p              s owl:onProperty    p
   <Forall>                           x      rdf:type     s
    <Var> y </Var>                    x         p         y
    <Implies>                        </And>
     <head>y rdf:type o</head>      </body>
     <body>x    p    y</body>     </Implies>
    </Implies>
   </Forall>
  </And>
 </body>
</Implies>
```

owl:someValuesFrom

For the uncertainty, the effect of owl:someValuesFrom is similar as owl:unionOf, and we declare it optional in the right column. However, if an individual y matches successfully, then x will be typed of the corresponding class s.

```
<Implies>                     <Implies>
 <head>                        <head>
   x rdf:type s                 <Exists>
 </head>                         <Var> y </Var>
 <body>                          <And>
  <And>                           x     p    y
   s owl:someValuesFrom o         y rdf:type o
   s owl:onProperty    p         </And>
   x         p         y        </Exists>
   y      rdf:type     o       </head>
  </And>                       <body>
 </body>                        <And>
</Implies>                       s owl:someValuesFrom o
                                 s owl:onProperty     p
                                 x rdf:type           s
                                </And>
```

```
 </body>
</Implies>
```

owl:cardinality

The cardinality restrictions rely on "counting" distinct individuals, and some rule engines provide such operations as Jess function "count-query-results". For RuleML, we employ "Cterm" (complex term) and "Ctor" (constructor) to express the functions, as shown below.

```
<Implies>
 <head>
   x rdf:type s
 </head>
 <body>
  <And>
   s owl:cardinality o
   s owl:onProperty  p
   x rdf:type owl:Thing
   <Equal>
    <Ind> o </Ind>
    <Cterm>
     <Ctor>count</Ctor>
     <statement>
        x   p   y
     </statement>
    </Cterm>
   </Equal>
  </And>
 </body>
</Implies>
```

```
<Implies>
 <head>
  <Equal>
   <Ind> o </Ind>
   <Cterm>
    <Ctor>count</Ctor>
    <statement>
       x   p   y
    </statement>
   </Cterm>
  </Equal>
 </head>
 <body>
  <And>
   s owl:cardinality o
   s owl:onProperty  p
   x rdf:type        s
  </And>
 </body>
</Implies>
```

3.5 User-defined

This part is special for the rules defined by a user personally, on the assumption that the user is also familiar with the corresponding targeted rule-based language.

4 Implementation

Being an early stage, the above RuleML rule base has been published on OWL-Trans[3], where OWL2Jess is similar but with a Jess rule base, while SWRL2Jess is used to handle SWRL rules. With the mentioned XSLT, an OWL file produces a lot of triples. Combining the desired components, a customized rule base is generated. Running it, implicit information is made explicit. Finally, by a simple lexical translator, each triple of (subject predicate object) is exported to the database as one record.

[3] http://www.inf.fu-berlin.de/inst/ag-nbi/research/owltrans/

Due to the XSLT function of "generate-id", the blank node in RDF syntax also has a machine code as its ID. For those OWL constructors without URI references, the generated IDs serve as OWL class names. Suppose an OWL restriction ∃hasChild.Person, for example, it provides no URI like foo:Parent, but actually returns a machine code of N40003E, s.t., the instance retrieval is dealt with (? rdf:type N40003E).

Note that, such transformational implementation fails to supporting OWL reasoning fully, since it lacks of a practical FOL RuleML engine. However, Mandarax[4], the first complete input-processing-output environment for RuleML, also provides its jdbc driver for query the knowledge bases as relational database. So, in our planning work, it is considered to exploit Mandarax as the engine for our RuleML rule base, with some work-arounds to avoid the undeveloped fragment.

5 Related Work

Being the key to Semantic Web, reasoning has been greatly investigated, where sorts of OWL reasoners are available, as well as DL(description logic) reasoners. A companion to the OWL implementations is listed in [3], including some well-known ones such as: (1) F-OWL, related to F-logic and XSB closely; (2) Pellet, special for the OWL DL sublanguage; (3) Hoolet, using a First Order Prover; (4) FaCT, a DL classifier (without ABox reasoning); (5) ConsVISor, a consistancy checker for OWL (Full,DL,Lite). Besides, being a DL reasoner, RACER has been used with Protege for developing ontologies in OWL format, and Cwm (named from "Closed World Machine") is also featured for implementing as much of the Semantic Web layer cake as possible.

However, it is still an challenge to cope with scalable ontologies (such as in bioinformatics) and real Web environment (there are currently over four billion web pages indexed by Google). To tackle this problem, database has been taken into account, and one representative is the Instance Store[7]. Different from our work, "iS" depends on TBox reasoners such as FaCT or RACER to classify the descriptions of instances firstly, and then ABoxes are dynamically added to and retrieved from a database. In our architecture, a RuleML rule base is proposed for implementing OWL semantics, and the ontology base is for "telling" while the database is for "asking".

Besides, approaches to extension as well to restriction are both deserved to investigate. A first order prover, such as Vampire[12], can potentially solve its OWL fragment, also its SWRL fragment. Instead, DLP(Description Logic Program) [4], a fragment of OWL, can be reasoned fully with standard (deductive) databases. As regards our ORDBase architecture, it shows an operational suite of tools for non-expert users, laying emphasis on its utilization.

[4] http://mandarax.sourceforge.net/

6 Conclusion

Our proposal of ORDBase intends to demonstrate an architecture for ontologies, rules and data in Sematic Web, with the provision of RDF and OWL reasoning services, from structuring to querying and inferencing.

Considering that, it is relatively simple to build ontologies with graphical editors and to query statements in relational databases. We present a RuleML rule base, which imports the well-formed OWL file from an ontology base and exports the resulting triples to a database. In particular, predefined RuleML rules serve for translating OWL semantics, to make implicit information explicit.

It has been published this RuleML rule base and another Jess one on OWL-Trans. In our planned future work, (XSLT) translators will transform the RuleML rule base into sorts of rule engines. Alternatively, a RuleML engine, such as Mandarax, is also expected to use directly. More important issues are some evaluations to illustrate the performance of the ORDBase architecture.

References

1. Harold Boley, Mike Dean, Benjamin Grosof, Michael Sintek, Bruce Spencer, Said Tabet, and Gerd Wagner. First order logic ruleml. *Available at http://www.daml.org/2004/11/fol/folruleml*, 2004.
2. Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium (SWWS'01)*, pages 381–401. Stanford University, July/August 2001.
3. Jeremy J. Carroll and Jos De Roo. Owl test results (semi-official semi-static view). *Available at http://www.w3.org/2003/08/owl-systems/test-results-out*, 2004.
4. Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logics. In *WWW 2003, Budapest, Hungary, May 2003*, 2003.
5. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. An evaluation of knowledge base systems for large owl datasets. In *Third International Semantic Web Conference, Hiroshima, Japan, LNCS 3298*, pages 274–288. Springer, 2004.
6. Patrick Hayes and Brian McBride. Rdf semantics. *http://www.w3.org/TR/rdf-mt/*.
7. Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The instance store: Description logic reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.
8. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. *Available at http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/*, 2004.
9. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
10. Thorsten Liebig, Holger Pfeifer, and Friedrich W. von Henke. Reasoning services for an owl authoring tool: An experience report . In *Description Logics*, 2004.
11. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. *http://www.w3.org/TR/owl-absyn/*.
12. Dmitry Tsarkov, Alexandre Riazanov, Sean Bechhofer, and Ian Horrocks. Using Vampire to reason with OWL. In *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*, pages 471–485. Springer, LNCS 3298, 2004.