Reasoning on the Web with Open and Closed Predicates

Gerd Wagner[1], Adrian Giurca[1], Ion-Mircea Diaconescu[1], Grigoris Antoniou[2], Anastasia Analyti[2] and Carlos Viegas Damasio[3]

[1] Brandenburg University of Technology, Germany
{G.Wagner, Giurca, M.Diaconescu}@tu-cottbus.de,
[2] Institute of Computer Science, FORTH-ICS, Greece
{antoniou, analyti}@ics.forth.gr
[3] Universidade Nova de Lisboa, Portugal
cd@di.fct.unl.pt

Abstract. SQL, Prolog, RDF and OWL are among the most prominent and most widely used computational logic languages. However, SQL, Prolog and RDF do not allow the representation of negative information, only OWL does so. RDF does not even include any negation concept. While SQL and Prolog only support reasoning with closed predicates based on negation-as-failure, OWL supports reasoning with open predicates based on classical negation, only. However, in many practical application contexts, one rather needs support for reasoning with both open and closed predicates. To support this claim, we show that the well-known Web vocabulary *FOAF* includes all three kinds of predicates i.e. *closed*, *open* and *partial* predicates. Therefore, reasoning with FOAF data, as a typical example of reasoning on the Web, requires a formalism that supports the distinction between open and closed predicates. We argue that *ERDF*, an extension of RDF, offers a solution to deal with this problem.

1 Introduction

1.1 Open and Closed Predicates

Many information management scenarios deal with predicates with their complete extension recorded (e.g. in a database). Such *closed* predicates use the computational mechanism of *negation-as-failure (NAF)* in order to infer negative conclusions based on the explicit absence (or non-inferability) of an information item. In other words, not for *open* but only for *closed* predicates, NAF is similar with standard negation.

The issue of reasoning with closed predicates and NAF has been researched in the field of Artificial Intelligence back in the 1980's, as a form of NAF has been implemented at that time both in the database language SQL and in the logic programming language Prolog. The resulting theories and formalisms, including the famous "Closed-World Assumption", have considered NAF to be the negation concept of choice in computational logic systems, and have downplayed the

significance of "open-world" reasoning with classical negation. 20 years later, however, a computational logic concept of classical negation has been chosen and implemented in a prominent computational logic formalism, viz the Web ontology language OWL [10]. While SQL and Prolog have a nonmonotonic computational logic semantics and support only closed predicates, OWL is based on a computational fragment of classical logic and therefore supports only open predicates. However, in many practical application contexts, one rather needs support for reasoning with both open and closed predicates.

1.2 Total and Partial Predicates

In fact, in addition to the distinction between *open* and *closed* predicates, it is useful to make another distinction between *total* and *partial* predicates. All these distinctions are related to the semantics of negative information and negation. The distinction between total and partial predicates is supported by *partial logic* (see [9]), which comes in different versions (with either 3 or 4 truth values) and can be viewed as a refinement of classical logic allowing both truth value gaps and truth value clashes. The law of the excluded middle only holds for total, but not for partial predicates. Both closed and open predicates are total. Consequently we obtain three kinds of predicates, as described in the following table:

	NAF=NEG	LEM (Law of Excluded Middle)
closed	yes	yes
open	no	yes
partial	no	no

The symbolic equation *NAF=NEG* denotes the condition that negation-as-failure and standard negation collapse, i.e. that both connectives are logically equivalent.

1.3 Three Kinds of Predicates in FOAF

A well-known example of a Web vocabulary is FOAF, the *Friends of a Friend* vocabulary [6], which is essentially expressed in RDFS (with a few additional constructs borrowed from OWL), and which has the purpose to create a Web of machine-readable information describing people, the links between them and the things they create and do. As examples of closed, open and partial predicates included in FOAF we consider the properties `foaf:member`, `foaf:knows` and `foaf:topic_interest`. Of course, one could simply stipulate that these predicates have a standard classical logic semantics. But we argue that their intended meaning in natural language implies that they are better treated as closed, open, respectively partial predicates according to partial logic.

When a `foaf:Group` is defined, we may assume that such a definition is not made in an uncontrolled distributed manner, but rather in a controlled way where one specific person (or agent) has the authority to define the group, typically in the context of an organization that empowers the agent to do so.

In this case, it is natural to consider the definition of the group membership to be a complete specification, and, consequently, to consider the `foaf:member` property to be a closed predicate. For the following example,

```
<foaf:Group rdf:ID="http://tu-cottbus.de/lit/erdf-team">
  <foaf:name>BTU Cottbus ERDF Team</foaf:name>
  <foaf:member rdf:resource="#Gerd"/>
  <foaf:member rdf:resource="#Adrian"/>
  <foaf:member rdf:resource="#Mircea"/>
</foaf:Group>
<foaf:Person rdf:ID="Gerd">
<foaf:Person rdf:ID="Adrian">
<foaf:Person rdf:ID="Mircea">
```

this would mean that we can draw the (negative) conclusion that

> Grigoris is not a member of the BTU Cottbus ERDF Team

based on the absence of a fact statement that "Grigoris is a member of the BTU Cottbus ERDF Team".

In the case of the property `foaf:knows`, however, we could argue that the standard RDF and OWL treatment of classes and properties as open predicates is adequate, since one does normally not make a complete set of statements about all persons one knows in a FOAF file. Consequently, the absence of a fact statement that "Grigoris knows Gerd" does not justify to draw the negative conclusion that "Grigoris does not know Gerd".

Both `foaf:member` and `foaf:knows` can be considered as *total* predicates that are subject to the *law of the excluded middle*, implying that the following disjunctive statements hold:

> Either Grigoris is a member of the BTU Cottbus ERDF Team or Grigoris is not a member of the BTU Cottbus ERDF Team.
> Either Grigoris knows Gerd or Grigoris does not know Gerd.

In the case of the property `foaf:topic_interest`, the situation is different. First, notice that while in the previous cases of `foaf:member` and `foaf:knows` there is no need of representing negative fact statements, we would like to be able to express both topics in which we are interested and topics in which we are definitely not interested (and would therefore prefer not to receive any news messages related to them). For instance, we may want to express the negative triple "Gerd is definitely not interested in the topic *motor sports*". Therefore, we should declare `foaf:topic_interest` to be a *partial* property, which means (1) that we want to be able to represent negative fact statements along with positive fact statements involving this predicate and (2) that the *law of the excluded middle* does not hold for it: it is not the case that for any topic x,

> Gerd is interested in the topic x or Gerd is (definitely) not interested in the topic x.

There may be topics, for which it is undetermined whether Gerd is interested in them or not.

1.4 Extended RDF

Since RDF(S) (see [7, 5, 3]) does not allow to represent negative information and does not support any negation concept, we need to extend it for turning it into a suitable reasoning formalism for FOAF and similar Web vocabularies.

In [12], it was argued that a database, as a knowledge representation system, needs two kinds of negation, namely *weak negation* for expressing *negation-as-failure* (or *non-truth*), and *strong negation* for expressing *explicit negative information* or *falsity*, to be able to deal with partial information. In [13], this point was also made for the Semantic Web as a framework for knowledge representation in general, and in [1, 2] for the Semantic Web language RDF with a proposal how to extend RDF for accommodating the two negations of partial logic as well as derivation rules. The extended language, called *Extended RDF*, or in short *ERDF*, has a model-theoretic semantics that is based on partial logic [9].

1.5 Plan of the Paper

While the theoretical foundation of ERDF has been presented in [1, 2], the novel contributions of this paper are

1. an exposition and discussion of the RDF-style syntax of ERDF, and
2. a presentation of a case study that shows how a practical Web vocabulary (FOAF) would benefit from the extended logical features offered by ERDF (the support of two kinds of negation and three kinds of predicates).
3. a discussion about our current implementation of ERDF tool set, including an inference engine, and our future plans for improvements.

2 The ERDF Abstract Syntax

This section describes the abstract syntax of ERDF in terms of a MOF/UML metamodel that is aligned with the RDF metamodel of OMG's *Ontology Definition Metamodel (ODM)* [8].

2.1 The ERDF-Vocabulary

ERDF adds the following classes to the RDFS vocabulary: `erdf:PartialClass`, `erdf:PartialProperty`, `erdf:TotalClass`, `erdf:TotalProperty`, `erdf:OpenClass`, `erdf:OpenProperty`, `erdf:ClosedClass` and `erdf:ClosedProperty`. These classes specialize `erdf:Class` and `erdf:Property` as depicted in Figure 1.

ERDF allows to designate properties and classes that are completely represented in a knowledge base – they are called *closed*. The classification if a predicate is closed or not is up to the owner of the knowledge base: the owner must know for which predicates there is complete information and for which there is not.

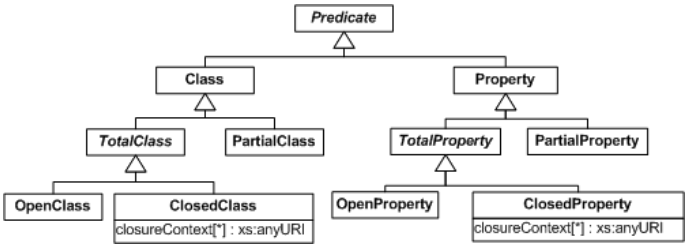

Fig. 1. The ERDF vocabulary as an extension of the RDFS vocabulary

2.2 ERDF Descriptions and Atoms

ERDF descriptions, as depicted in the metamodel diagram in Figure 2, extend RDF descriptions by

1. adding to RDF property-value slots an optional attribute `negationMode` that allows to specify three kinds of negation (`Naf` for *negation-as-failure*, `Sneg` for *strong negation* and `NafSneg` for *negation-of-failure over strong negation*). An optional value, `None` is also possible and it is the default value (i.e. when the attribute *negationMode* is missing);
2. allowing not only data literals, URI references and blank node identifiers as *subject* and *object* arguments (called `subjectExpr` and `valueExpr` in Figure 2), but also variables.

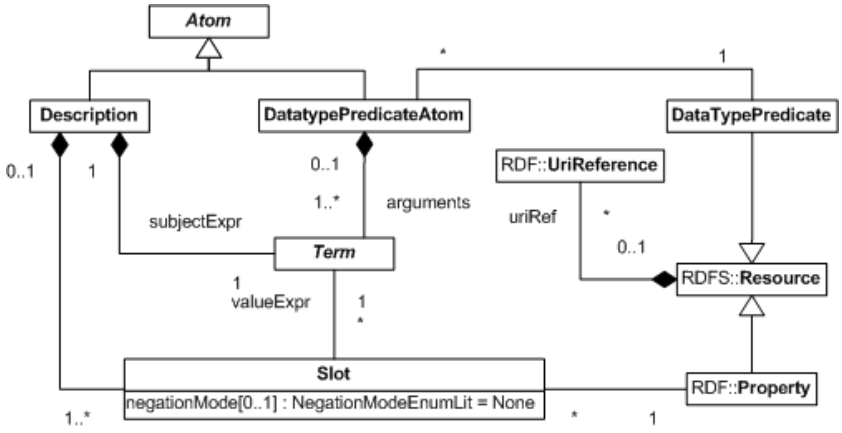

Fig. 2. ERDF Descriptions

An ERDF description consists of the following components:

– One *subject expression*, denoted by the `subjectExpr` property in the metamodel diagram, being an *ERDF term*, that is a `URIReference`, a `Variable`,

an `ExistentialVariable` (blank node identifier) or `rdfs:Literal` (see Figure 3 for the definition of ERDF term).
– A non-empty set of *slots* being property-value pairs consisting of a URI reference denoting a property and an ERDF term as the value expression.

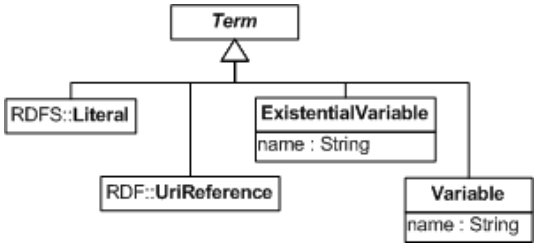

Fig. 3. ERDF Terms

Obviously, descriptions with just one slot correspond to the usual concept of an atomic statement (or triple), while descriptions with multiple slots correspond to conjunctions of such statements. However, as can be seen in Figure 2, all descriptions are considered as ***ERDF atoms***, which in addition subsume *datatype predicate* atoms (datatype predicates are often also called 'built-ins').

ERDF fact statements are variable-free ERDF descriptions such that no slot has a negation mode other than `None` or `Sneg`. That is, only strong negation may occur in fact statements (in the case of negative information).

ERDF descriptions with variables correspond to conjunctive query formulas that can be used as rule conditions.

2.3 ERDF Rules

The abstract syntax of ERDF rules is defined in the metamodel diagram in Figure 4. ERDF rules are derivation rules of the form $D \leftarrow A_1, \ldots, A_n.$ where D is an ERDF description with only `None` or `Sneg` as slot negation modes and $A_1, \ldots, A_n$ are *ERDF atoms*, that is, descriptions or datatype predicate atoms.

3 A Concrete Syntax for ERDF

Our approach is to follow the RDF/XML syntax as much as possible and derive an RDF-style syntax for ERDF atomic formulas, "triple patterns" from the abstract syntax metamodel presented above.

3.1 Expressing a Vocabulary in ERDF

Using the ERDF predicate categories defined in section 2.1, we can refine the FOAF vocabulary definition of `foaf:member`, `foaf:knows` and `foaf:topic_interest` as follows:

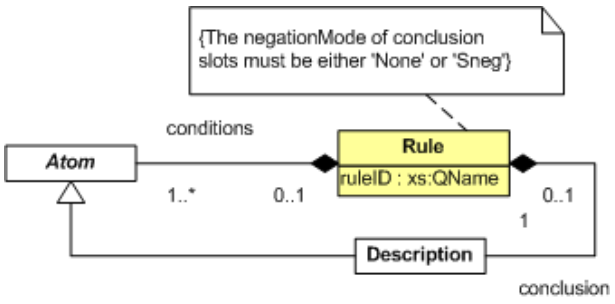

Fig. 4. ERDF Rule

```
<erdf:OpenProperty rdf:about="http://xmlns.com/foaf/0.1/knows">
 <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
 <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
</erdf:OpenProperty>

<erdf:PartialProperty rdf:about="http://xmlns.com/foaf/0.1/topic_interest">
 <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
 <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</erdf:PartialProperty>

<erdf:ClosedProperty rdf:about="http://xmlns.com/foaf/0.1/member">
 <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Group"/>
 <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Agent"/>
</erdf:ClosedProperty>
```

We identify `erdf:OpenProperty` with `rdf:Property` and `erdf:OpenClass` with `rdfs:Class`. Thus, by default, all RDF predicates are considered to be open. One may argue that is no need of `erdf:OpenProperty` and `erdf:OpenClass` constructs, but for names uniformity and expressivity these constructs are defined as part of the ERDF vocabulary.

3.2 Expressing ERDF Terms

ERDF terms are URI references, blank node identifiers, variables or data literals. They are expressed in two ways, depending on their occurrence as *subject expressions* or as *value expressions*.

Terms as subject expressions are values of the `erdf:about` attribute, which may be URI references, blank node identifiers or variables (using the SPARQL syntax for blank node identifiers and variables).

Terms as value expressions are expressed either with the help of one of the attributes `rdf:resource`, `rdf:nodeID` or `erdf:variable`, or as the text content of the property-value slot element in the case of a data literal.

3.3 Descriptions and Datatype Predicate Atoms

ERDF descriptions are encoded by means of the `erdf:Description` element. Each description contains a non-empty list of (possibly negated) property-value slots.

Example 1. Gerd knows Adrian, has some topic interest, but is not interested in the topic 'motor sports'

```
<erdf:Description erdf:about="#Gerd">
  <foaf:knows rdf:resource="#Adrian"/>
  <foaf:topic_interest rdf:nodeID="x"/>
  <foaf:topic_interest erdf:negationMode="Sneg"
                       rdf:resource="urn:topics:motor_sports"/>
</erdf:Description>
```

`erdf:Description`, as an extension of `rdf:Description` element, allows negated slots and two other possible values for triples subject: variables and literals (as values of `erdf:about` attribute). For expressing RDF triples it is possible to use any of `rdf:Description` or `erdf:Description` elements.

Datatype predicate atoms are n-ary logical atoms. The value of `erdf:arguments` property represent an ordered list of arguments. The `erdf:predicate` XML attribute encodes the URI reference to the predicate.

Example 2. Using built-ins

```
<erdf:DatatypePredicateAtom erdf:predicate="swrlb:add">
 <erdf:arguments>
  <erdf:Variable>?sum</erdf:Variable>
  <rdfs:Literal rdf:datatype="xs:int">40</rdfs:Literal>
  <rdfs:Literal rdf:datatype="xs:int">20</rdfs:Literal>
 </erdf:arguments>
</erdf:DatatypePredicateAtom>
```

3.4 Rules and Rulesets

Two syntaxes for ERDF rules are proposed: (1) a more concise non-XML syntax based on SPARQL triple patterns, and (2) an XML-based syntax, which is useful for rule transformations and interchange.

To express ERDF rules in XML, constructs from R2ML[14] rule markup language are used. Later, it may be an option to use the W3C rule interchange format.

Inspired by Jena Rules[4], the non-XML syntax for ERDF rules is based on SPARQL triple patterns: universal quantified variables prefixed by the '?' symbol, literals, typed literals, URI's or QNames to denote full URI's. Five types of atoms can be used:

[4] Jena Rules Syntax - `http://jena.sourceforge.net/inference/#rules`

– *built-ins*, available in a predefined set [5] and offering the possibility of defining new ones (e.g. `sum(?a,?b,?c)` bound `c` to the value of sum from `a` and `b`).
– *positive triples*, formally expressed as `(subject predicate object)`, e.g. `(ex:John foaf:knows ex:Tom)`;
– *strong negated triples*, denoted by adding the '-' symbol in front of the second node, namely `predicate`, e.g. `(?x -foaf:topic_interest ?t)`;
– *weak-negated triples*, expressed as a built-in, namely `naf`. It's arguments, are the triple's nodes, i.e. : `naf(?x foaf:knows ex:Tom)`.
– *negation-as-failure over strong negation*, the '-' symbol is added in front of the second argument, namely the `predicate`, when the `naf` built-in is used, e.g. `naf(?x -foaf:topic_interest ?t)`.

4 The ERDF Application Programming Interface

The ERDF Application Programming Interface was implemented as an extension of Jena Rules. An extended rule syntax was defined for allowing the two ERDF negation connectives. The rule language is backward compatible, therefore the Jena rules are also supported.

Adding support for reasoning in top of ERDF facts has required modifications in the structure of the Jena API. In Figure 5 are reflected some important changes of the informational model. These improvements were made for allowing representation of negated triples and for dealing with these triple types. The ERDF reasoner was defined as an extension of the Jena backward engine.

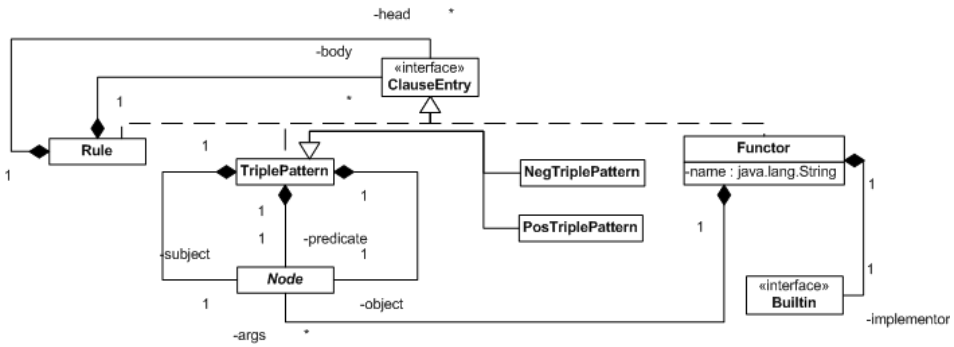

Fig. 5. ERDF Triples extension

The RDF(S)-based reasoner implemented by the Jena API uses an internal set of axioms and rules. For instance, the following axioms are used to express relations used by the RDF Schema:

```
-> (rdf:type      rdfs:range rdfs:Class).
-> (rdfs:Resource  rdf:type  rdfs:Class).
```

[5] Jena built-ins - `http://jena.sourceforge.net/inference/#RULEbuiltins`

An internal set of rules is used for computing the transitive closure in RDF(S). As an example, the following rule consider the `subClassOf` relationship:

```
[(?a rdfs:subClassOf ?b), (?b rdfs:subClassOf ?c) -> (?a rdfs:subClassOf ?c)]
```

ERDF defines `erdf:Class` as being the superclass of all its classes and `erdf:Property` the superclass of all properties. This was reflected by extending the Jena axioms set:

```
-> (rdf:type rdfs:range erdf:Class).
-> (rdfs:Resource rdf:type erdf:Class).
-> (erdf:TotalClass rdfs:subClassOf erdf:Class)
-> (erdf:PartialClass rdfs:subClassOf erdf:Class)
-> (erdf:ClosedClass rdfs:subClassOf erdf:TotalClass)
-> (erdf:OpenClass rdfs:subClassOf erdf:TotalClass)
-> (erdf:TotalProperty rdfs:subClassOf erdf:Property)
-> (erdf:PartialProperty rdfs:subClassOf erdf:Property)
-> (erdf:ClosedProperty rdfs:subClassOf erdf:TotalProperty)
-> (erdf:OpenProperty rdfs:subClassOf erdf:TotalProperty)
-> (erdf:OpenClass rdfs:subClassOf rdfs:Class)
-> (rdfs:Class rdfs:subClassOf erdf:OpenClass)
-> (erdf:OpenProperty rdfs:subClassOf rdf:Property)
-> (rdf:Property rdfs:subClassOf erdf:OpenProperty)
```

Since ERDF deals also with closed properties, the internal rules set was extended to support this feature:

```
[close1: (?s -?p ?o)
         <-
         (?p rdf:type erdf:ClosedProperty)
         (?p rdf:range ?r)(?p rdf:domain ?d)
         (?s rdf:type ?d)(?o rdf:type ?r) naf(?s ?p ?o)]
```

Some other information about the ERDF API might be accessed on the ERDF Web Page[6]. An AJAX based Web Application [7] is provided for testing ERDF rules. The application needs as input data: (1) a set of RDF/ERDF facts (using XML/RDF syntax), (2) a set of rules (using Jena Rules extended syntax), and (3) a set of queries (expressed by using Jena Rules extended syntax). The input data is processed by the ERDF API and query results are returned.

5 Case Study - Building FOAF-Based Working Groups

This section presents a scenario involving FOAF data and ERDF rules. As a short story, an organizing committee needs to create working groups with peoples from different communities taking part at some meeting. The assumption is that every member has his own FOAF file where their topic interests and contacts are provided. The FOAF files are available to the organizing committee.

[6] ERDF - `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=ERDF`
[7] ERDF Rules frontend - `http://oxygen.informatik.tu-cottbus.de/JenaRulesWeb`

The organizers task is to offer a solution (or more) for grouping participating members, having different topic interests areas, by their common interests. Therefore, two members are considered qualified for the same group if they have at least one common interest, but no contradictory interests. The second goal is to extend the community by grouping those members which does not know yet one each other. The meaning of "contradictory topic interest" between two members is that one topic interest of a member is negated in the FOAF file of the other member. The `foaf:knows` property express possible contacts between participants, and `foaf:topic_interest` property denotes interest topics of the meeting participants.

For instance, the organizers have collected the following data from FOAF files of some meeting participants:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:erdf="http://www.informatik.tu-cottbus.de/IT/erdf#">

 <erdf:Description erdf:about="http://www.tu-cottbus.de/staff#Gerd">
   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
   <foaf:topic_interest rdf:resource="urn:topics:RDF"/>
   <foaf:topic_interest rdf:resource="urn:topics:AgentBasedSimulation"/>
   <foaf:topic_interest erdf:negationMode="Sneg"
                        rdf:resource="urn:topics:motor_sports"/>
 </erdf:Description>

 <rdf:Description rdf:about="http://www.ics.forth.gr/staff#Grigoris">
   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
   <foaf:knows rdf:resource="http://www.tu-cottbus.de/staff#Gerd"/>
   <foaf:topic_interest rdf:resource="urn:topics:RDF"/>
 </rdf:Description>

 <rdf:Description rdf:about="http://www.tu-cottbus.de/staff#Adrian">
   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
   <foaf:topic_interest rdf:resource="urn:topics:RDF"/>
   <foaf:topic_interest rdf:resource="urn:topics:motor_sports"/>
 </rdf:Description>

 <rdf:Description rdf:about="http://www.tu-cottbus.de/staff#Mircea">
   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
   <foaf:knows rdf:resource="http://www.tu-cottbus.de/staff#Adrian"/>
   <foaf:topic_interest rdf:resource="urn:topics:AgentBasedSimulation"/>
 </rdf:Description>
</rdf:RDF>
```

Notice that only the first description, since it includes negative triples, needs to be marked up as an ERDF description. For the other (positive) fact statements there is possible to use RDF (but also ERDF).

The following rule, expressed by using XML-based syntax, is defined to establishes if two persons (meeting participants) classify for the same group.

If persons X and Y do not know each other, they have at least one common topic interest and have no contradictory topic interest, then it is recommended that X and Y are members of the same group.

```
<r2ml:DerivationRule r2ml:ruleID="sameGroupAs">
 <r2ml:conditions>
  <erdf:Description erdf:about="?x">
   <rdf:type rdf:resource="foaf:Person"/>
   <foaf:topic_interest erdf:variable="?t"/>
   <foaf:knows erdf:negationMode="Naf" erdf:variable="?y"/>
   <conf:contradictoryInterest erdf:negationMode="Naf" erdf:variable="?y"/>
  </erdf:Description>
  <erdf:Description erdf:about="?y">
   <rdf:type rdf:resource="foaf:Person"/>
   <foaf:topic_interest erdf:variable="?t"/>
   <foaf:knows erdf:negationMode="Naf" erdf:variable="?x"/>
   <conf:contradictoryInterest erdf:negationMode="Naf" erdf:variable="?x"/>
  </erdf:Description>
 </r2ml:conditions>
 <r2ml:conclusion>
  <erdf:Description erdf:about="?x">
   <conf:sameGroupAs erdf:variable="?y"/>
  </erdf:Description>
 </r2ml:conclusion>
</r2ml:DerivationRule>
```

The non-XML syntax for ERDF rules can be used to express the same rule:

```
[sameGroup: (?x conf:sameGroupAs ?y)
            <-
            (?x rdf:type foaf:Person)(?y rdf:type foaf:Person)
            naf(?x foaf:knows ?y) naf(?y foaf:knows ?x)
            naf(?x conf:contradictoryInterest ?y)
            naf(?y conf:contradictoryInterest ?x)]
```

The following rule define how the values of the `conf:contradictoryInterest` predicate are computed:

```
[conInterest: (?p1 conf:contradictoryInterest ?p2)
              <-
              (?p1 rdf:type foaf:Person)(?p2 rdf:type foaf:Person)
              (?p1 foaf:topic_interest ?t)(?p2 -foaf:topic_interest ?t)]
```

Other rules/queries are then used to create `foaf:Groups`. Values computed for `conf:sameGroupAs` property might by considered for this purpose. Consider the following queries:

```
@prefix btu: http://www.tu-cottbus.de/staff#
[q1: <- (btu:Gerd conf:sameGroupAs btu:Mircea)]
[q2: <- (btu:Gerd conf:sameGroupAs btu:Adrian)]
[q3: <- (?p1 conf:sameGroupAs ?p2)]
```

The answer is "true" for *q1* (no contradictory interests and persons does not know each other) and "false" for *q2* ("motor sports" is a contradictory interest). The last query (*q3*) will return all possible combinations of two persons which might be in the same group.

This use case is available for online testing by using the AJAX frontend. The Figure 6 shows an results excerpt obtained by using the above facts, rules and queries as input data in the frontend.

Fig. 6. Query result using ERDF API

6 Related Work

Variables in triples have also been introduced in languages such as N3 [4] and *Jena Rules* [11]. A form of negation-as-failure has been implemented in Jena Rules by using a special built-in predicate. In N3, there is also a form of negation-as-failure, which allows one to test for what a formula does not say, with the help of `log:notIncludes`. But neither N3 nor Jena Rules has a systematic treatment of negative information and open and closed predicates.

7 Conclusion and Future work

The paper presents an abstract and an RDF-style concrete syntax for ERDF, allowing to represent negative fact statements and supports reasoning with open and closed predicates. We have argued that these issues are of practical significance by showing how they affect the popular FOAF vocabulary. Finally a prototype of the ERDF API is described and a practical use case is considered.

Future work includes further extensions of the language, constructs for handling uncertainty and reliability, and their implementation in the ERDF API.

References

1. Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damasio, and Gerd Wagner. Negation and Negative Information in the W3C Resource Description Framework. *Annals of Mathematics, Computing and Teleinformatics*, 1(2):25–34, 2004.
2. Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damasio, and Gerd Wagner. Stable Model Theory for Extended RDF Ontologies. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of the 4th International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science (LNCS)*, pages 21–36, Galway, Ireland, 6-10 November 2005. Springer-Verlag.
3. Grigoris Antoniou and Frank Van Harmelen. *A Semantic Web Primer*. MIT Press, 2004.
4. Tim Berners-Lee. N3 (Notation 3). http://www.w3.org/DesignIssues/Notation3.html, 1998.
5. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation February 2004. http://www.w3.org/TR/rdf-schema/.
6. Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.91. http://xmlns.com/foaf/spec/, November 2007.
7. Klyne G. and Caroll J.J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-concepts/.
8. Object Management Group. Ontology Definition Metamodel. http://www.omg.org/docs/ptc/07-09-09.pdf, November 2007.
9. Heinrich Herre, Jan O. M. Jaspars, and Gerd Wagner. Partial Logics with Two Kinds of Negation as a Foundation for Knowledge-Based Reasoning. In D.M. Gabbay and H. Wansing, editors, *What is Negation?* Kluwer Academic Publishers, 1999.
10. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language. Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, February 2004.
11. Dave Reynolds. Jena Rules experiences and implications for rule use cases. In *W3C Workshop on Rule Languages for Interoperability*, 2005.
12. Gerd Wagner. A database needs two kinds of negation. In B. Talheim and H.D. Gerhardt, editors, *3rd Symposium on Mathematical Fundamentals of Database and KnowledgeBase Systems*, volume 495 of *Lecture Notes in Computer Science (LNCS)*, pages 357–371. Springer-Verlag, 1991.
13. Gerd Wagner. Web rules need two kinds of negation. In F. Bry, N. Henze, and J. Maluszynski, editors, *Principles and Practice of Semantic Web Reasoning, Proceedings of the 1st International Workshop, PPSWR '03*, volume 2901 of *Lecture Notes in Computer Science (LNCS)*, pages 33–50. Springer-Verlag, 2003.
14. Gerd Wagner, Adrian Giurca, and Sergey Lukichev. A General Markup Framework for Integrity and Derivation Rules. In F. Bry, F. Fages, M. Marchiori, and H. Ohlbach, editors, *Dagstuhl Seminar Proceedings 05371*, Principles and Practices of Semantic Web Reasoning, 2005.