# Utilizing Host Formalisms to Extend RDF Semantics

Wolfram Conen[+] and Reinhold Klapsing[++]

[+]XONAR GmbH,
Wodanstr. 7
D-42555 Velbert, Germany,
Conen@gmx.de

[++]Information Systems and Software Techniques,
University of Essen, Universitätsstraße 9,
D-45141 Essen, Germany,
Reinhold.Klapsing@uni-essen.de

**Abstract.** RDF may be considered as an application of XML intended to interoperably exchange semantics between Web applications. In its current form, this objective may be hard to reach. Even if the semantical gems hidden in the RDF/RDFS specification are precisely captured, as, for example, in the axiomatic formalizations currently available, the useabilty of RDF's concepts and constraints is limited: RDF offers a data model but does not specify the processing of RDF-encoded data. RDFS describes some basic (ontological) concepts and constraints but does not specify the processing of RDFS-encoded ontological information. The expressiveness of the constraints is rather limited and no clear means of providing semantics for new concepts and constraints are specified. This paper presents one possible approach to overcome this weaknesses. The definition and interpretation of semantics and the processing of the RDF-encoded information will be delegated to a host formalism (first order logic). An elaborated example specifies an extended set-algebraic range constraint and applies the extended vocabulary to a security management task. The definition of semantics is made explicit in the RDF Schemata. The new constraints and concepts are added to the concepts and constraints of an underlying axiomatic interpretation of RDF(S). A Prolog-based implementation of the approach, the RDF Schema Explorer, which is available on-line, is presented. The tool allows to process, validate, query and extend a FOL interpretation of (extended) RDF Schemata.[1]

**Keywords:** Semantic Web, RDF, Semantic Extensibility, Host Formalism, Prolog

[1]This paper draws from an earlier paper that we will present at the German Wirtschaftsinformatik conference.

## 1 Exchanging Semantics on the Web

Semantic annotation of data becomes increasingly important, as increasingly complex interactions, involving a multitude of actors, call for a shared and common understanding of the exchanged information. Semantic annotation may enable intelligent search instead of keyword matching, query answering instead of information retrieval [7][2], knowledge base definition instead of data format exchanges etc. The Semantic Web Activity of the World Wide Web Consortium (W3C) emphasizes the importance of semantics for the further development of the Web. The Resource Description Framework (RDF) [9, 2] may develop into one of the foundations of the Semantic Web by enabling semantic interoperability. RDF intends to provide a standard for describing the semantics of information via metadata descriptions (compare [7]).

For the Semantic Web to scale, independent and heterogenous actors (users, agent, tools) must be able to exchange and process (meta-)data based on a common semantic interpretation. One may question if RDF provides the means to achieve this. We want to emphasize two issues here: (1) most aspects of the RDF Schema specification are expressed informally, and (2) the concepts and constraints of the RDF Schema specification do not provide sufficient expressiveness and lack a clear extension mechanism.

The first issue has been addressed before[3] – in [4] we chose first order logic (FOL) to express the main concepts and constraints defined in the RDF specifications. The main benefit of using FOL is that it is a well-studied expression mechanism with a commonly agreed-upon interpretation. This has been utilized in the *RDF Schema Explorer*, a Prolog-based tool we developed that integrates Jan Wielemaker's RDF parser [11] and the axioms given in [4]. A Web-based version of the RDF Schema Explorer is accessible online [10]. It allows to query and validate RDF descriptions not only on the statement level but also with respect to the facts and rules that capture the semantic concepts and constraints of RDF.

The second issue has been discussed in the context of modeling ontologies in RDF(S), see [5]. Staab et al. state about RDF(S) that "the lack of capabilities for describing the semantics of concepts and relations beyond those provided by inheritance mechanisms makes it a rather weak language for even the most austere knowledge-based systems". They propose an approach that extends the semantics of vocabularies expressed in RDF(S) via axioms which are considered as objects that are describable in RDF(S).

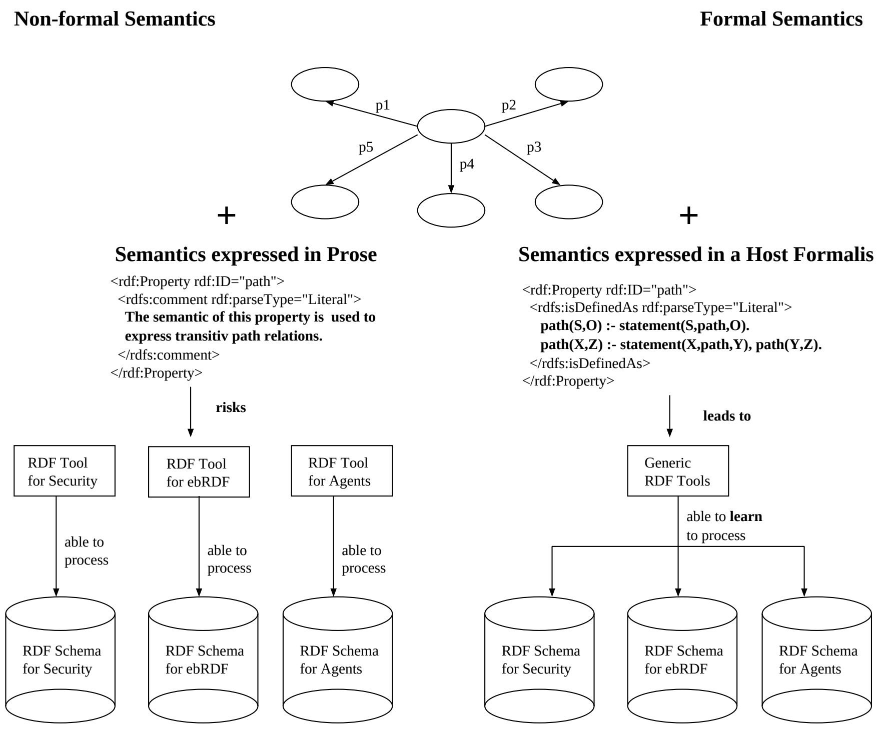
Our work, to be discussed below, can be seen as a combination of the work cited above[4]. We also provide means to explicitly specify the (axiomatic) semantic of properties from within RDF, compare Figure 1. This capability is implemented and available in the RDF Schema Explorer [10].

Furthermore, the definition of extended vocabularies is based on the axioms that capture the core RDF(s) concepts and constraints. These axioms are also available accessibly and explicitly. This tight integration of the RDFS concepts / constraints with the extended seman-

[2]Fensel provides an instructive overview of rationales for (ontology-driven) semantics in different networking contexts.

[3]Though, unfortunately, it is not yet on the issue list of the current RDF working group to provide some more formal (axiomatic) semantics for RDFS, so this effort documents only one possible, not-standardized attempt to capture the meaning of RDF Schema

[4]While we implemented the RDF Schema Explorer without knowledge of the approach of Staab et al., we nevertheless very much agree with their rationales for making axioms available "as objects that are describable in RDF(S)". We would like to recommend their paper as a complementary source of well-chosen arguments for extending RDFS with explicitly available axioms



Non-formal Semantics
Formal Semantics
p1
p2
p5
p3
p4
+
+
Semantics expressed in Prose
Semantics expressed in a Host Formalis
<rdf:Property rdf:ID="path">
<rdfs:comment rdf:parseType="Literal">
The semantic of this property is used to express transitiv path relations.
</rdfs:comment>
</rdf:Property>
<rdf:Property rdf:ID="path">
<rdfs:isDefinedAs rdf:parseType="Literal">
path(S,O) :- statement(S,path,O).
path(X,Z) :- statement(X,path,Y), path(Y,Z).
</rdfs:isDefinedAs>
</rdf:Property>
risks
leads to
RDF Tool for Security
RDF Tool for ebRDF
RDF Tool for Agents
Generic RDF Tools
able to process
able to learn to process
RDF Schema for Security
RDF Schema for ebRDF
RDF Schema for Agents

Figure 1: Defining more sophisticated semantics with a host formalism. In the left part of the figure, semantics are informally described within rdfs:comments. This may lead to the development of a plethora of interpretation-specific RDF tools. This is contrasted with the approach to make (axiomatic) meaning explicitly available, thus making it generally accessible for precise and interoperable interpretation (within the limits of the chosen host formalism as far as it extends RDFS).

tics, as well as the availability of a prolog-based implementation maybe considered as the main difference to the work of Staab et al.

Below, we will demonstrate this integration by means of an application that especially emphasizes the use of an extended range constraint in an access-control context. The remainder of this paper is structured as follows. In Section 2 the extension mechanisms is presented. We describe how the *RDF Schema Explorer* operates and which basic predicates are provided to query an RDF description. In Subsection 2.1 the extension mechanism, used to formally define more sophisticated semantics in RDF schemata, is explained. An example, taken from an access control context, is presented in Subsection 2.3 to demonstrate the extension mechanism and the related RDF syntax. We include a brief discussion of one of the core concepts of RDFS, the range constraint. In Section 3 the paper is concluded with a brief discussion of the presented approach.

## 2 Specifying Extensible Semantics in RDF

Below, the RDF Schema Explorer [10] is presented that allows to query RDF models not only on a statement level but also with respect to the facts and rules that capture the semantic concepts and constraints of RDFS. For this purpose, a number of pre-defined predicates is available. This also allows to validate the models against this RDFS rule set. In addition, it is possible to define the semantics of newly introduced predicates from within RDF and to query/check/validate these extended models.

| Predicate | Purpose |
|---|---|
| `statement(S,P,O)` | Contains the basic facts of the knowledge base. |
| `res(R)` | Gives the resources. |
| `lit(O)` | Gives the literals. |
| `reifies(R,S,P,O)` | R reifies the (not necessarily present) triple `[S,P,O]`. |
| `reifyingStatement(R)` | R fulfills `reifies/4` for some S,P,O. |
| `reifies_fact(R)` | R fulfills `reifies/4` for some S,P,O and the triple `[S,P,O]` is indeed in the knowledge base. |
| `subClassOf(C,D)` | Transitive predicate that captures the relation that is expressed with the `rdfs:subClassOf` property. |
| `instanceOf(R,C)` | Transitive predicate that captures the relation that is expressed with the `rdf:type/rdfs:subClassOf` properties. |
| `subClass_cycle_violation(C)` | This is true if the knowledge base allows to infer `subClassOf(C,C)`. |
| `subPropertyOf(X,Y)` | A transitive predicate that captures the relation that is expressed with the `subPropertyOf` predicate. |
| `subProperty_cycle_violation(P)` | This is true if the knowledge base allows to infer `subPropertyOf(P,P)`. |
| `domain_constrained_property(P)` | At least one statement that specifies a domain constraint is present for property P. |
| `domain(X,P)` | X is an instance of one of the classes that are in the domain of P. |
| `domain_violation(S,P,O)` | This is true if a statement `[S,P,O]` is in the knowledge base, and P is domain-constrained and S is not in the domain of P. |
| `is_range(C,P)` | C is (one of) the range restriction(s) for P. |
| `range_cardinality_violation(P)` | There are (at least) two different range restrictions for P. |
| `has_range(P)` | P is range-constrained. |
| `range(X,P)` | X is an instance of (one of) the class(es) to which the range of P is constrained to. |
| `range_violation(S,P,O)` | P is range-constrained, the statement `[S,P,O]` is in the knowledge base and O is not in the range of P. |
| `violation(T,S,P,O)` | A convenience predicate that collects the above violations. T will show the type of the violation and S,P,O will be the violating triple - with the exception of `range_cardinality`, where S will be the violating predicate and O will be one of the ranges S should be constrained to. In this case, all ranges that are given will be shown as different instances of `violation`. |

Table 1: A collection of the predicates that axiomatize the RDF Schema constraints.

The tool works as follows. First, some RDF-File will be fed into the SWI-Prolog-based RDF parser[5]. This file will be parsed and a relation will be created that contains the triples, e.g.[S,P,O], in a relation `statement(S,P,O)`).[6]

The slightly modified parser tries to *normalize* the URIs–no matter, if a resource is given in subject, predicate, or object position, the parser tries to transform it into the format `namespace:resource_name`. This makes querying much easier. Furthermore, some form of normalization is necessary to be able to discover that `xxx:yyy` and `URI_of_xxx#yyy` are (or better: “represent”) indeed the same resource.

Now, one could already query this simple triple database. The tool offers a query field allowing to ask the Prolog engine things like `statement(S,rdf:type,O)` or

[5]Credits go to Jan Wielemaker. Some minor modifications have been made related to namespaces.

[6]Note that we do not assume *per se* that every triple encodes an instance of a binary relation. As has been discussed in [5], a triple plus a reification and a simple negated truth predicate may easily be used to imply intentions that render the mapping to binary relations faulty – e.g. triple [S,P,O], plus Reification R representing [S,P,O], plus triple [R hasTruthValue FALSE] may express that it is known that [S,P,O] is not true.

`setof(O,statement(S,P,O),Z)`. While it is certainly useful to know a little bit about Prolog, it is not necessary, because the tool offers a choice of predefined queries from a pre-selection list.

However, this would not be completely satisfying. As one will normaly use concepts/constructs from RDFS, the fact and rule base that has been outlined in the paper “A logical interpretation of RDF” ([4]) is provided. The effect is that the knowledge level predicates that are briefly explained in Table 2 can be used to check and query a model with respect to the RDF schema constraints.

In addition, we have defined a number of additional *convenience predicates*. Most of them can be chosen from the pre-selection menu on the query form. An example is `show_statements(S,P,O)` where a value for any of the variables S,P, or O cab be substituted in and a list of the triples containing the substituted value at the corresponding position will be generated.

While this all makes it rather easy to play with the effects of RDF schema concepts and constraints, one will soon discover that the semantics implied by RDFS are pretty general (not to say “weak”). We therefore allow to introduce *semantics on top of the basic facts and rules* which makes it possible to specify more precisely what a modeler intends with her predicates. This can be done in two ways:

1. Either, some Prolog rules may be directly keyed into the query field, for example

   ```
   assert(trans_rel(S,O):- statement(S,path,O)).
   assert(trans_rel(S,O):- statement(S,path,Z), trans_rel(Z,O)).
   ```

   which defines the predicate `trans_rel` to represent a transitive property `path`. This would allow to inquire if two resource are transitively related, or

2. the RDF-level mechanism that we provide to define the semantics of predicates within RDF documents is used. This mechanism will be discussed in some detail in the following subsections.

## *2.1 The Extension Mechanism*

The mechanism to be described allows to provide the semantics for properties within RDF schema declarations. A special predicate `rdfs:isDefinedAs` is available to extend the basic rule set with additional semantics for newly defined properties (it is also possible to define the basic rule set this way). The interpretation of the schemata will rely on a suitably chosen host formalism. For the current implementation, the Prolog-flavor of first-order logic has been selected.

The example below, defining the transitive property `path`, can be fed directly into the RDF Schema Explorer.[7]

```
<?xml version="1.0"?>
<RDF
 xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

[7]Note, however, that to make it exiting, some resource that are related via the path property would be required.

```
 xmlns:rdfs="http://.../TR/2000/CR-rdf-schema-20000327#">

 <rdf:Property rdf:ID="path">
   <rdfs:isDefinedAs rdf:parseType="Literal">
      path(S,O) :- statement(S,path,O).
      path(X,Z) :- statement(X,path,Y), path(Y,Z).
   </rdfs:isDefinedAs>
 </rdf:Property>
</RDF>
```

Note the use of `statement` above, which is meaningful because all predicates that are defined in the basic rule set are accessible.

In the current version of the RDF Schema Explorer, only *Prolog code* may be provided (to be read by SWI-Prolog in the sequence that is implied by the XML serialization[8]). In future versions, other languages (such as implementations of Description Logics [1]) may be allowed as well.

### *2.2 An Advocacy for a set-algebraic* `range`*-constraint*

In RDFS, the applicability and expressiveness of the range constraint is rather limited. To see this, first a brief review of (our version of) the intended semantics of the range constraint in the current version of RDFS is given. In [4] the range constraint has been captured as

```
is_range(X,P) :- statement(P,rdfs:range,X).
has_range(P)  :- is_range(_,P).
range(X,P)    :- is_range(C,P), instanceOf(X,C).
range_violation(S,P,O) :- statement(S,P,O), has_range(P), not(range(O,P)).
```

In RDFS, the following further restrictions apply.

1. At most one range constraint is allowed.
2. Only two distinguished sets of entities, namely *Resources* and *Literals* exist.
3. The semantics of subclassing can be captured with the rule

   ```
   instanceOf(I,C) :- statement(I,rdf:type,C).
   instanceOf(I,D) :- statement(I,rdf:type,C), subClassOf(C,D).
   ```

With an open-world assumption, not much could be deduced from a range constraint[9], because knowing that the range of a property $p$ is constrained to the set $X \subset Resources$ and

[8]Unfortunately, in standard SLD-resolution-based Prolog, sequence does matter. This matches, however, naturally with XML (and not quite so naturally with RDF, which does not use sequence information with the notable exception of Seq-type containers). If one would parse the XML serialization, compute triples from it, scramble the triple sequence and subsequently start to assert the property definitions, this might lead to a behaviour that was not intended – however, it would conform to the notion of RDF as being set-oriented.

[9]We do not infer types from range *constraints*. Rationales: Two possible interpretations of the *range constraint* have been discussed (RDF-IG, Rdf-logic), (a) the *constraint* and the (b) *axiom* interpretation. Roughly, (a) says that a property $p$ may (only) be applied to instances of classes that are in the range of $p$ while (b) states that, from using a resource $r$ as a value of a range-constrained property $p$, it can be infered that $r$ has the type of the range of $p$. Formally, both interpretation can be formulated as $instanceOf(O, C) \leftarrow statement(S, P, O), range(P, C)$, with the difference that, with the constraint interpretation, we have to ask if this is a (logical) *consequence* of the known statements (facts) and rules (axioms) while, with the axiom interpretation, this will be treated as one of the rules/axioms that allows us to infer type information (and no

knowing that a resource $r$ is an element of a set $Y \subset Resources$ does not allow to conclude that attaching a value $r$ to $p$ would violate the range constraint. This would only be reasonable if it would be known that $X$ and $Y$ are disjoint. However, this information is only available for *Literals* and *Resources* and is not expressible in RDF for the relation between two (or more) arbitrary subsets of *Resources*. Assuming that the world is closed and complete, one could argue that two subclasses $X, Y$ of a class $R$ are disjoint if no entity is known that is an instance of both classes. Nevertheless, two problems remain: schemata are mostly used to guide the design/evolution of models, ie. not all instances will be known at schema design time – and introducing further information may render earlier decisions inconsistent (because adding a type information to a resource may show that two classes are in fact not distinct but overlapping etc.) – SO, considering a world as complete is dangerous with respect to inter-temporal validity. In addition, only a richer set of constraints (including set-union, set-difference and set-disjunction) would allow to specify all constraints that seem reasonable if the range of a property should be restricted. To see this consider the following: The are two classes, $C1$ and $C2$, and a property $p$. With “reasonable” we mean the following range constraints: for `[x,p,y]`, `range(p,Exp)` may constrain $y$ to be an element of $Exp$ defined as

| | |
|---|---|
| $Exp := C1 \cup C2$ | ($y$ in $C1$ OR $y$ in $C2$) |
| $Exp := C1 \cap C2$ | ($y$ in $C1$ AND $y$ in $C2$) |
| $Exp := C1 \backslash C2$ | ($y$ in $C1$ AND $y$ NOT in $C2$) |
| $Exp := C2 \backslash C1$ | ($y$ in $C2$ AND $y$ NOT in $C1$) |
| $Exp := (C1 \backslash C2) \cup (C2 \backslash C1)$ | ($y$ in $C1$ XOR $y$ in $C2$) |
| $Exp := \,!(C1)$ | ($y$ not in $C1$) |

An often suggested extension of RDFS is to allow multiple range constraints and to interpret these constraints as binding the allowed range to the disjunction of the classes. However, this would restrict the interpretation of multiple range constraints to one (limited) case of the cases given above[10]. Below, we will suggest a solution that not only conforms to RDF but also offers a flexible and general way to specify range constraints. The required interpretation can be encoded on schema level, making it possible to specify and enforce different *types* of range constraints in different application domains.

Below, only one range constraint will be allowed. This is sufficient if classes (or class expressions) can be constructed from other classes (or class expressions). In this case, each range constraint will point to exactly one class and the *construction* of the class directly expresses the constraint. Above, the *Exp* term represents the constructed class and the right hand side gives the construction expression. An example for applying a range constraints using a constructed class is:

validation will be possible). We adopt the practice of the examples in (Sec. 3.1, Sec. 7.1 of [2]), where types are assigned to resources with the *rdf:type*/*rdfs:subClassOf* properties, and the range-constraint is used to “state that a ... property only ´makes sense’ when it has a value which is an instance of the class ... ”, allowing for **validation**. This conforms to interpretation (a) above. Please note that now, no types of resources will nor should be infered, instead it is possible to check (with the range constraint) if properties are applied to resources of the correct type (with rdf:type, rdfs:subClassOf or subproperties of these properties as the available devices to provide typing information).

[10]A solution could be to introduce specific range constraints / range constraint types for all of the above cases. This is, however, problematic, because it does not scale very good to “mixed” range dependencies with $3, 4, \ldots, n$ classes.

```
[C1,rdf:type,rdfs:Class]
[C2,rdf:type,rdfs:Class]
[A,rdf:type,ConstructedClass]
[A,isConstructedFrom,"C2 \ C1"]
[p, rdfs:range, A]
```

With `[X, rdf:type, C1]`, $X$ would violate the intended range constraint if it would be chosen as a value for $p$.

If it is assumed that the object "`C2` $\setminus$ `C1`" is modeled as a literal, the above solution can be formulated as well-formed RDF easily. However, to interpret it, an application-level check of the class construction semantics would be required. This is not really nice, because range constraints seem to be too important to leave their semantics to "proprietary" vocabularies and interpretations, but this might be a matter of taste. With respect to the intended interoperability based on RDF schemata, making the semantics of the constructs expressible within RDF seems to offer a more interoperable solution. In fact, the property `isConstructedFrom` denotes a multi-ary relation between classes. This can be transformed (generally) into a sequence of (3-ary) "atomic" set-algebraic operations (expressed below as nested tuples), as in the following example that expresses $A = (C1 \cap C2) \setminus C3$.

```
[ A1, intersection, [C1,C2] ]
[ A,  difference, [A1, C3] ]
```

In RDF, this is expressible using reification and a suitable interpretation of the reified statements:

```
[ A1, rdf:type, rdf:Statement ]
[ A1, rdf:subject, C1 ]
[ A1, rdf:predicate, rdfsets:intersection]
[ A1, rdf:object, C2 ]

[ A, rdf:type, rdf:Statement ]
[ A, rdf:subject, A1 ]
[ A, rdf:predicate, rdfsets:difference]
[ A, rdf:object, C3 ]
```

Suitably interpreted, this allows to express a set algebraic range constraint like:

```
[ p, rdfsets:range, A ]
```

### *2.3 Sharing Security Schemata – An Example*

In the following we demonstrate how such set constructs can be defined in an RDF-conform manner by applying the above introduced extensions mechanism to the domain of role-based access control. The semantics are build upon the basic RDF rules given in [4]. In the example below[11], the task is to decide if access to certain documents should be granted to certain users. The decision depends on the membership of users in certain groups[12]. Figure 2 depicts the specific situation.

[11]The RDF source of the following example is easily accessible as part of the RDF Schema Explorer on-line demonstration [10].

[12]Conceptually, membership in groups or role assignment can both be represented with set-algebraic class expression – and this is the mechanism used in this example.

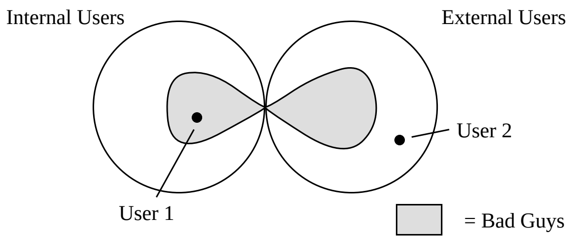


Figure 2: Access shall only be granted to users in the white section of the above venn diagrams, i.e., *bad guys* like user 1 should not get access.

Three new predicates are introduced, namely *union*, *difference* and *intersection*[13]. These predicates can be used to construct classes from other classes with the help of binary relations and reification, both being completely valid RDF constructs. This will be utilized to construct classes from set-algebraic expressions over other (constructed) classes.

The extension is based on the already introduced semantic primitive *isDefinedAs* (to ease the demonstration, we assume that the property is in the `rdfs` namespace). To make it possible to mix meta-schema, schema and instance expressions in the example below, we adopted the following convention: if a namespace `this#` is introduced, the namespace abbreviation will be omitted during the parsing process. This makes it possible to use the namespace within the document while still being able to normalize the resource names to make them easily useable for querying the model.[14].

First, a subclass of `rdfs:Class`, `ConstructedClass` is introduced. The rules described above are used to define the semantics of the newly introduced predicates. Additionally, the semantics of both the `type` and the `range` property are (monotonically) extended to be able to cope with constructed classes.

```
<?xml version="1.0"?>
<RDF
 xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://.../TR/2000/CR-rdf-schema-20000327#"
 xmlns:rdfsets="this#">

 <!-- Meta Schema definitions -->
 <rdfs:Class rdf:ID="ConstructedClass">
  <rdfs:subClassOf rdf:resource=
    "http://.../TR/2000/CR-rdf-schema-20000327#Class"/>
 </rdfs:Class>

 <Description
  about="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    constructed_class(C):-instanceOf(C,'ConstructedClass').
  </rdfs:isDefinedAs>
 </Description>
```

[13]A NOT will not be introduced because it allows to formulate unbounded class expressions, ie. expressions that depend on an (unknown) universal set. Set-difference contains implicit (bounded) NOT constraints and is sufficient for most purposes.

[14]The reader may adopt this practice with self-developed extension schemata to make it easy to feed schemata and instances as one document into the RDF Schema Explorer [10].

```
<Property rdf:ID="union">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    in(X,S,P,O) :- P = union, instanceOfSet(X,S).
    in(X,S,P,O) :- P = union, instanceOfSet(X,O).
  </rdfs:isDefinedAs>
</Property>

 <Property rdf:ID="difference">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    in(X,S,P,O) :- P = difference,
       instanceOfSet(X,S), not(instanceOfSet(X,O)).
  </rdfs:isDefinedAs>
</Property>

<Property rdf:ID="intersection">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    in(X,S,P,O) :- P = intersection,
       instanceOfSet(X,S), instanceOfSet(X,O).
  </rdfs:isDefinedAs>
</Property>

<Description about=".../CR-rdf-schema-20000327#range">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    instanceOfSet(X,A) :- constructed_class(A),
       reifies(A,S,P,O), in(X,S,P,O).
    instanceOfSet(X,A) :- instanceOf(X,A).
    range(X,P) :- is_range(C,P), instanceOfSet(X,C).
  </rdfs:isDefinedAs>
</Description>
```

Now the schema definitions follow, expressing that `Internal_Users`, `External_Users`, and `Bad_Guys` are plain classes and that `All_Users` and `Trusted_Users` are constructed classes, with $\texttt{All\_Users} = \texttt{Internal\_Users} \cup \texttt{External\_Users}$ and $\texttt{Trusted\_Users} = \texttt{All\_Users} \setminus \texttt{Bad\_Guys}$.

```
<rdfs:Class rdf:ID="Internal_Users"/>
<rdfs:Class rdf:ID="External_Users"/>
<rdfs:Class rdf:ID="Bad_Guys"/>

<rdfsets:ConstructedClass rdf:ID="All_Users">
  <subject   rdf:resource="#Internal_Users"/>
  <predicate rdf:resource="#union"/>
  <object    rdf:resource="#External_Users"/>
  <type      rdf:resource=".../22-rdf-syntax-ns#Statement"/>
</rdfsets:ConstructedClass>

<rdfsets:ConstructedClass rdf:ID="Trusted_Users">
  <subject   rdf:resource="#All_Users"/>
  <predicate rdf:resource="#difference"/>
  <object    rdf:resource="#Bad_Guys"/>
  <type      rdf:resource=".../22-rdf-syntax-ns#Statement"/>
</rdfsets:ConstructedClass>
```

Access will be granted according to a closed security policy that is, all accesses have to be allowed explicitly. This will be expressed by attaching a property `AccessAllowedFor` to resources that is constrained to the range `Trusted_Users`.

```
 <Property rdf:ID="AccessAllowedFor">
   <rdfs:range rdf:resource="#Trusted_Users"/>
 </Property>
```

The following instance definitions will entail a range constraint violation.

```
 <Description rdf:ID="user_1">
   <type rdf:resource="#Internal_Users"/>
 </Description>

 <Description rdf:ID="user_1">
   <type rdf:resource="#Bad_Guys"/>
 </Description>

 <Description rdf:ID="user_2">
    <type resource="#External_Users"/>
 </Description>

 <!-- Objects to restrict access to: -->
 <rdfs:Class rdf:ID="Important_Documents"/>

 <rdfsets:Important_Documents rdf:ID="Weak_Secret_1">
   <rdfsets:AccessAllowedFor rdf:resource="#user_1"/>
   <rdfsets:AccessAllowedFor rdf:resource="#user_2"/>
 </rdfsets:Important_Documents>
</RDF>
```

Here, `user_1` is known as a bad guy, accordingly, he should not be granted access. In fact, the range constraint on `AccessAllowedFor` is violated. To see this, consider the extended rule set for the set-algebraic range constraint:

```
/* RDFS rule set */
is_range(X,P) :- statement(P,rdfs:range,X).
has_range(P)  :- is_range(_,P).
range(X,P)    :- is_range(C,P), instanceOf(X,C).

/* Extension */
range(X,P) :- is_range(C,P), instanceOfSet(X,C).

/* Detecting the violation (from RDFS rule set) */
range_violation(S,P,O) :-  statement(S,P,O), has_range(P), not(range(O,P)).
```

The RDF descriptions above allow to derive that `user_1` is *not* a member of the constructed class `Trusted_Users` and thus, is not in the range of `AccessAllowedFor`.

We hope that this simple example may already demonstrate that the above mechanism, together with a Prolog engine, is a pretty powerful instrument to *define/extend semantics*, to *validate documents* against RDFS and user-provided constraints, and to *query a model on the knowledge level*. This may help to leave the simplistic triple structure behind and to capture the meaning of (extended) vocabularies more precisely. It allows to develop domain specific vocabularies build upon the formalized RDF/RDFS constraints. These vocabularies can be re-used in schema definitions for other domains as well. The RDF Schema Explorer will support this with dynamic loading and incremental interpretation of schema definitions (via HTTP).

## 3 Discussion

The approach outlined above allows to define RDF (meta-)schemata that precisely capture the semantic intentions if interpreted within a suitable host formalism. The approach represents the intended semantics of RDF schemata explicitly, making it possible to treat the definition as first-class resources within RDF[15]. The approach is paradigm-independent, as it allows to select different host formalisms for specific purposes [16]. The specific Prolog-based instantiation of the approach is expressive as it allows to utilize the available expressiveness of Prolog. Furthermore, production-quality implementations of Prolog are widely available. It may be asked why pure Prolog or any other KR Language (like KIF/SKIF) has not been chosen as an implementation language for the semantic web. We think that constraining people to a certain implementation language may not always be a good idea. There are always pros and cons for a certain implementation language. We propose to give an implementer the possibility to use a suitable implementation language for her application domain. Pure RDF/RDFS remains to be an exchange mechanism for (rudimentary) knowledge while an implementer should have the choice to integrate this basic knowledge (for example based on an axiomatization of RDFS) with more elaborate semantics defined on top of a suitable host formalism (with the consequence that this part of the knowledge may not be interpretable in different host formalisms).

To summarize: We presented a detailed example that demonstrates the use of the involved techniques in an access control context. The Prolog-based RDF Schema Explorer that we developed allows to validate and query such extended models. Both, the tool and a workable version of the example are accessible on-line. Besides being able to interpret (extended) RDF schemata, the tool is suitable to support the prototyping of domain-specific schemata, as the semantics of the defined properties can be changed on the fly and the consequences can be inspected utilizing the convenience predicates (such as `violation`, `show_classes`, etc.).

We expect that the interoperable definition of meta-schemata will develop into a necessity, once the formulation of complex semantic constraints in various emerging application domains such as cooperative security management, automated business contract negotiation etc. – all involving a number of autonomous partners and, thus, exhibiting a need for semantic interoperability – is identified as a key requirement for the success of the underlying collaborations.

## References

[1] Alexander Borgida. Description Logics in Data Management. *Knowledge and Data Engineering*, 7(5):671–682, 1995. http://citeseer.nj.nec.com/borgida95description.html.

[2] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate Recommendation, W3C, March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[15]This allows to apply the RDF concepts to describe/relate the semantic definitions as well. For example, new properties expressing containment, semantic dependencies, abstraction etc. can be defined and used, which may ease to maintain and re-use the (meta-) schemata.

[16]Both, making the semantics of the underlying (meta-)concepts explicit and being not bound to a specific world view / paradigm (such as, for example, ontology-based agent modeling), renders our approach different from such languages as OIL [3, 6] or DAML [8] that offer a set of non-manipulable primitives whose semantics are not expressed in the RDF-based languages themselves. This necessarily restricts the applicability of the languages to domains/applications that exhibit a “natural” and “close” fit with the concepts the languages offer.

[3] Jeen Broekstra, Michel Klein, Dieter Fensel, Stefan Decker, and Ian Horrocks. OIL: a case-study in extending RDF-Schema. Technical report, ontoknowledge.org, 2000. http://www.ontoknowledge.org/oil/oil-rdfs.pdf.

[4] Wolfram Conen and Reinhold Klapsing. A Logical Interpretation of RDF. *Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841*, 5(13), December 2000. http://www.ep.liu.se/ea/cis/2000/013/.

[5] Wolfram Conen, Reinhold Klapsing, and Eckhart Köppen. Rdf m&s revisited: From reification to nesting, from containers to lists, from dialect to pure xml. In *Proceedings of the Semantic Web Working Symposium (SWWS)*, Stanford, August 2001.

[6] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a Nutshell. Technical report, ontoknowledge.org, 2000. http://www.cs.vu.nl/ dieter/oil/oil.nutshell.pdf.

[7] Dieter Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, Heidelberg, 2001.

[8] Ian Horrocks, Frank van Harmelen, Tim Berners-Lee, Dan Brickley, Dan Connolly, Mike Dean, Stefan Decker, Dieter Fensel, Pat Hayes, Jeff Heflin, Jim Hendler, Ora Lassila, Deb McGuinness, Peter Patel-Schneider, and Lynn Andrea Stein. DAML+OIL Language. http://www.daml.org/2000/12/daml+oil-index.html, December 2000.

[9] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[10] Web-based RDF Schema Explorer. http://wonkituck.wi-inf.uni-essen.de/rdfs.html.

[11] SWI-Prolog. http://www.swi.psy.uva.nl/projects/SWI-Prolog/.