September 2001

Exchanging Semantics with RDF

Wolfram Conen
XONAR GmbH, conen@gmx.de

Reinhold Klapsing
University of Essen, reinhold.klapsing@uni-essen.de

In: Buhl, Hans Ulrich, u.a. (Hg.) 2001. *Information Age Economy*; 5. Internationale Tagung Wirtschaftsinformatik 2001. Heidelberg: Physica-Verlag

ISBN: 3-7908-1427-X

Exchanging Semantics with RDF

Wolfram Conen
XONAR GmbH

Reinhold Klapsing
University of Essen

Abstract: E-Commerce applications require the exchange of data in an interoperable manner. XML enables syntactic interoperability but more sophisticated E-Commerce applications require semantic interoperability as well. RDF is an application of XML intended to exchange semantics between Web applications. RDF schemata can be used to describe the concepts and constraints of a specific application domain at a semantic level. However there is no formal mechanism to state sophisticated semantics beyond the static set of concepts and constraints provided by RDF. This paper presents an approach extending RDF in a standard manner and utilizing a host formalism for formally defining semantics of new RDF schemata. An elaborated example applies the approach to security management. A tool, the RDF Schema Explorer, is available on-line, allowing to process, validate and query a first-order-logic interpretation of(extended) RDF Schemata.

Keywords: Semantic Interoperability, Semantic Extensibility, RDF, Semantic Web

1 Exchange of Semantics in a Business Context

Open E-commerce applications require a flow of data along the value-adding chain of actors (e.g. between suppliers, distributors, retailers and end users). Metadata (data about data) is essential for e-commerce, as it provides standard data items to allow parties to communicate about their products, terms, conditions and organizations. The eXtensible Markup Language [XML00] is applied to many E-Commerce/E-Business applications. The main benefit of XML is that it provides a widely accepted and standardized exchange format. XML is designed to enable the exchange of data even across the boundaries of heterogenous systems. The flexibility of XML allows (business) communities to define data formats suitable for a certain domain area. XML focuses on *syntactic interoperability* not on *semantic interoperability*. Semantic interoperability is a key issue for new emerging E-Bussiness applications which require that machines are able to understand the intended meaning of the exchanged data. The least recent vision of the World Wide Web Consortium (W3C), the Semantic Web [SemWeb], fosters this aspect.

The Resource Description Framework (RDF) [RDF99; RDFS00] may develop into one of the foundations of the Semantic Web by enabling semantic interoperability. RDF is designed for data sharing and processing by automated tools as well as by people. For the Web to scale, independently designed programs must be able to exchange and process (meta-) data. Ideally, the programs should be able to process the meaning of (meta-) data indepent from specific application areas. Semantic interoperability can be achieved only if different users (agents, tools, etc.) interpret an RDF data description in the same way. Important aspects of the RDF model are, however, expressed in prose which may lead to misunderstandings. To avoid this, capturing the intended semantics of RDF in a formal manner is unremissible. In [CoKl00], the concepts and constraints of the RDF model have been expressed in first order logic (FOL), a well-studied expression mechanism with a commonly agreed-upon interpretation. This is utilized in the *RDF Schema Explorer*, a Prolog-based tool developed on top of Jan Wielemaker's RDF parser [SWI]. A Web-based version of the RDF Schema Explorer is accessible on-line [SE].

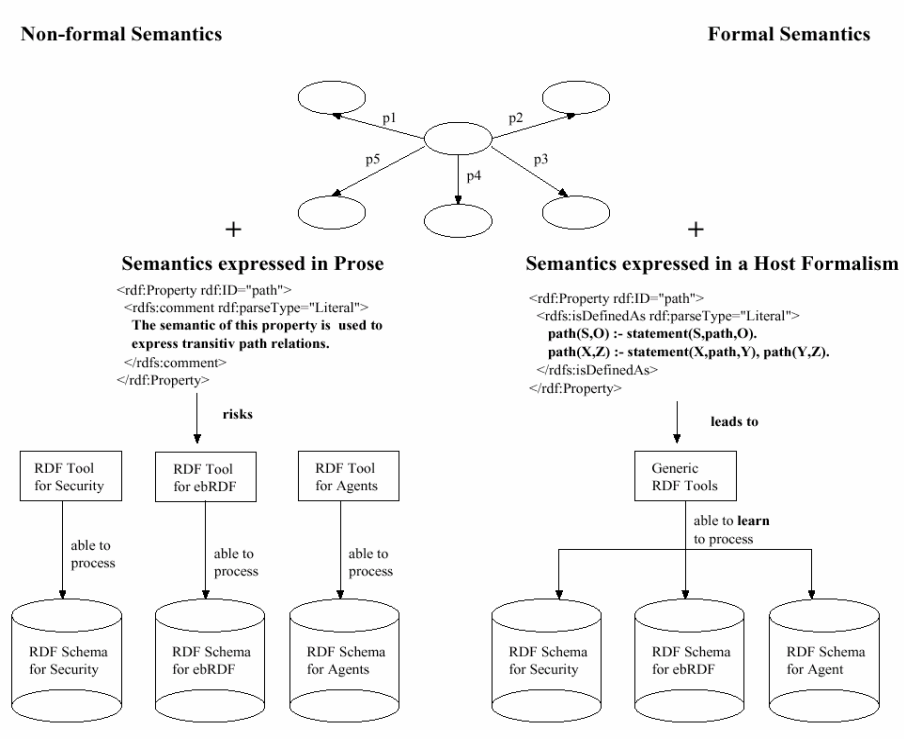

Figure 1: Defining more sophisticated semantics with a host formalism.

It allows to query RDF descriptions not only on the statement level but with respect to the facts and rules that capture the semantic concepts and constraints of

RDF. For this purpose, a number of pre-defined logic predicates is available. This allows to validate RDF descriptions against the RDF rule set. The concepts and constraints defined by the RDF specifications can be used to create new application-specific schemata which are also processible by generic RDF tools. However, problems arise if an application domain area requires semantics which are not expressible with the basic, static concepts and constraints provided by RDF (being mainly subclassing, typing and domain/range constraints). RDF lacks a formal approach for defining more sophisticated semantics beyond simple labeling (that is defining attribute/value pairs). As a consequence, the intended semantics of the properties provided by new RDF schemata can not be assessed generically. We think that this poses the risk for the emerging Semantic Web to develop into a babylonic jumble of semantics, as it entails the development of "specific purpose" RDF tools with built-in interpretation of application-specific RDF schemata/descriptions. This hard-coding of semantics of RDF properties hampers the further evolution and universal re-use of schemata. In addition, such tools will only be able to construe other RDF schemata/descriptions at the labeling niveau. A generic mechanism is needed to extend the expressibility of RDF schemata in a formal and interoperable manner (compare Fig. 1). Our mechanism achieves this by delegating the interpretation of RDF descriptions to a *host formalism*. RDF schemata are tied to the host formalism by two elementary devices: (1) an additional RDF property `isDefindAs` that allows to formulate semantic definitions with the host formalism, and (2) an elementary mapping of the RDF primitive *Triple* (i.e, `[s,p,o]`) to a corresponding primitive of the host formalism (the instantiation `statement(s,p,o)` of the predicate `statement` in our case).

To summarize: This paper presents a generic approach which equips RDF with a mechanism to formally and interoperably express sophisticated semantics in RDF schemata. We present a tool, the RDF Schema Explorer, that allows to parse, validate, query and extend RDF Schemata. The concepts and benefits of the approach are demonstrated along the presentation of an RDF schema that allows to capture security-relevant access constraints in a rule-based access control contexts. The vocabulary of the schema with its prolog-based semantics can be used to augment other RDF schemata with access control features. We view this application as prototypically demonstrating the benefits of precisely and interoperably specified semantics in collaboration-driven application domains.

The remainder of this paper is structured as follows. In Section 2 the extension mechanisms is presented. First, 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 2.2 to demonstrate the extension mechanism and the related RDF syntax. 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 [SE] 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 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. The tool works as follows. First, some RDF-File will be fed into the SWI-Prolog-based RDF parser. This file will be parsed and a relation will be created that contains the triples (relation `statement(S,P,O)`). 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, this simple triple database could already be queried. The tool offers a query field allowing to ask the Prolog engine things like `statement(S,rdf:type,O)` or `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 [CoKl00] is provided. Thus the knowledge level predicates that are briefly explained in Table 1 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.

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 (so, reifies may model a belief or a reification of a fact.).
`subClassOf(C,D)`	Transitive predicate that captures the relation that is expressed with the `subClassOf` property.
`instanceOf(R,C)`	Transitive predicate that captures the relation

	that is expressed with the `type/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 capture the relation that is expressed with the `subPropertyOf` property.
`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 constraints 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.

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

An example is `show_statements(S,P,O)` where a value for any of the variables S, P, or O can 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 explore the effects of RDF schema concepts and constraints, one will soon discover that the semantics implied by RDFS are rather general. 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.

The Extension Mechanism

The mechanism to be described allows to provide the semantics of 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 (the basic rule set can also be defined 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.

```
<?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#">

<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>
```

In the current version, only Prolog code may be provided (to be read by SWI-Prolog in the sequence that is implied by the XML serialization--unfortunately, in standard SLD-resolution-based Prolog, sequence does matter. This matches, naturally with XML--and not quite so naturally with RDF, which does not use sequence information with the notable exception of Seq-type containers). In future versions, other languages may be allowed as well (and an additional statement will then denote which language is used in a document. A similar approach is proposed in [SEMD00]). Note the use of `statement` above, which is meaningful because all predicates that are defined in the basic rule set are usable.

Sharing Security Schemata - An Example

Below, an example of extending the semantics of RDF schema constructs is described. In particular, a more expressive version of the range constraint is presented. In RDFS, the applicability and expressiveness of the range constraint is

rather limited. To see this, first a brief review of the intended semantics of the range constraint in the current version of RDFS is given.

- At most one range constraint is allowed.
- Only two distinguished sets of entities, namely *Resources* and *Literals* exist.
- The semantics of subclassing can be captured with the rule
- instanceOf(x,B) :- instanceOf(x,A), subClassOf(A,B)

With an open-world assumption, not much could be deduced from a range constraint[1], because knowing that the range of a property `p` is constrained to the set `X` $\subset$ `Resources` and 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 reender earlier decisions invalid (because adding a type information to a resource may show that two classes are in fact not distinct but overlapping etc.) --considering a world as complete is dangerous with respect to inter-temporal validity. In addition,only a richer set of constraints (including union, difference and disjunction) would allow to specify all constraints that seem reasonable if the

[1] We do not infer types from range *constraints*. Rationales: Two possible interpretations of the *range constraint* have been discussed (see 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 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).

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 ∪ C2	(y in C1 OR y in C2)
Exp := C1 ∩ C2	(y in C1 AND y in C2)
Exp := C1 \ C2	(y in C1 AND y NOT in C2)
Exp := C2 \ C1	(y in C2 AND y NOT in C1)
Exp := (C1 \ C2) ∪ (C2 \ 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. 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 constraint using a constructed class is:

```
[ 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 `Exp` "`C2\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. To us, 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 re-

spect 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 ∩ C2)\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, rdfs:range, A ]
```

The semantics, building upon the basic rules given in [CoKl00] could than be:

```
/* C is a constructed class */
constructed_class(C) :- instanceOf(C,'ConstructedClass').

/* instanceOfSet collects all instances of
   a constructed class */
instanceOfSet(X,A) :- constructed_class(A),
   reifies(A,S,P,O), in(X,S,P,O).
/* ... and all instances of „Standard“ classes */
instanceOfSet(X,A) :- instanceOf(X,A).

/* Range is extended to include ranges over
   constructed classes  (the rule is added to
   standard interpretation of range) */
range(X,P) :- is_range(C,P), instanceOfSet(X,C).

/* Difference */
in(X,S,P,O) :-
 P = difference, instanceOfSet(X,S), not(instanceOfSet(X,O)).

/* Union */
in(X,S,P,O) :- P = union, instanceOfSet(X,S).
in(X,S,P,O) :- P = union, instanceOfSet(X,O).

/* Intersection */
in(X,S,P,O) :-
 P = intersection, instanceOfSet(X,S), instanceOfSet(X,O).
```

We will now demonstrate how this new (meta) schema constructs can be defined in an RDF-conform manner by applying the above introduced extensions mechanism to the domain of role-based access control. In the example below (the source is accessible at [SE]), 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. Figure 2 depicts the specific situation.

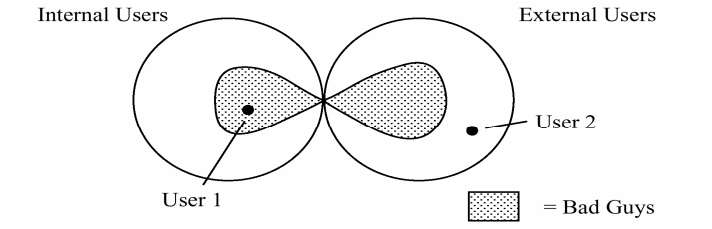

Figure 2: Access shall only be granted to users in the white section of the above venn diagramm, i.e., *bad guys* like user 1 should not get access.

Three new predicates are introduced, namely *union*, *difference* and *intersection*. 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.

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://..schema..#Class"/>
 </rdfs:Class>
```

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

<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="http://..schema...#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 All_Users} = Internal_Users ∪ External_Users}` and `Trusted_Users = All_Users \ 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="http://...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="http://...rdf-syntax-ns#Statement"/>
</rdfsets:ConstructedClass>
```

Access will be granted according to a closed security policy, that is, all accesses are to be allowed. 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. The rules given above allow to derive that `user_1` is *not* a member of the constructedclass `Trusted_Users` and, thus, the range constraint on `AccessAllowedFor` is violated.

To summarize: together with a Prolog engine, the above mechanism provides 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*, i.e. one can leave the simplistic triple structure behind and start to precisely capture what the intentions behind the model are. Furthermore, application domain specific vocabularies can be developed that build upon the formalized RDF/RDFS constraints. These vocabularies can be re-used in schema definitions in other domains as well. The RDF Schema Explorer will support this with dynamic loading and interpretation of schema defintions (via HTTP).

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. This allows to apply the RDF concepts to describe/relate the semantic definitions as well. For ex-

ample, new properties expressing containment, semantic dependencies, abstraction etc. can be defined and used, which may ease to maintain and re-use the (meta-) schemata. The approach is paradigm-independent, as it allows to select different host formalisms for specific purposes.

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. A closely related approach has been presented by Staab et al. [SEMD00]. Here, a similar mechanism to embed axioms explicitly in RDF Schema documents, is proposed (while we implemented the RDF Schema Explorer without knowledge of this approach, we nevertheless very much agree with their rationales for making axioms available "as objects that are describable in RDF(S)"). Its usefulness is demonstrated for the domain of modeling ontologies. The main difference between the approaches is that our approach is tighly coupled to an (explicitly available FOL) interpretation of the RDF schema concepts and constraints. This interpretation is used for validation in the host formalism itself, making it possible to *jointly* validate (extended) schemata against RDFS *core* constraints and *extension* constraints from within one instrument, without the necessity to delegate the validation of RDFS conformance to an external validator. Also, as the example above demonstrates for the range constraint, explicit extensions of RDFS constraints may require access to the explicit interpretation of the constraints.

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.

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, is identified as a key requirement for the success of the underlying collaborations.

References

[XML00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). Recommendation, W3C, October 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[RDFS00] 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.

[CoKl00] 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/.

[RDF99] 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.

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

[SEMD00] Steffen Staab, Michael Erdmann, Alexander Mädche, and Stefan Decker. An extensible approach for Modeling Ontologies in RDF(S). In Proceedings of ECDL-2000. Workshop "Semantic Web: Models, Architectures and Management, September 2000. http://www.aifb.uni-karlsruhe.de/ sst/Research/Publications/onto-rdfs.pdf.

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

[SemWeb] W3C. Semantic Web Activity. http://www.w3.org/2001/sw/, February 2001.