Adding formal semantics to the Web

building on top of RDF Schema

Jeen Broekstra[1], **Michel Klein**[2], **Stefan Decker**[3], **Dieter Fensel**[2], **Ian Horrocks**[4]

[1] AIdministrator Nederland b.v., Amersfoort, The Netherlands, e-mail: jeen.broekstra@aidministrator.nl
[2] Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, e-mail: {mcaklein,dieter}@cs.vu.nl
[3] Department of Computer Science, Stanford University, Stanford, USA, e-mail: stefan@db.stanford.edu
[4] Department of Computer Science, University of Manchester, UK, e-mail: horrocks@cs.man.ac.uk

4th September 2000

Abstract RDF Schema provides means to define vocabulary, structure and constraints for expressing metadata about Web resources. However, formal semantics for the primitives defined in RDF Schema are not provided, and the expressivity of these primitives is not enough for full-fledged ontological modeling and reasoning. To perform these tasks, an additional layer on top of RDF Schema is needed. In this paper, we will show how RDF Schema can be extended in such a way that a full knowledge representation language can be expressed in it, thus enriching it with the required additional expressivity and the semantics of this language. We do this by describing the ontology language OIL as an extension of RDF Schema. First, we give a short introduction to both RDF Schema and OIL. We then proceed to define a Schema to express OIL ontologies in RDF, where the aim is to use existing RDF terminology where possible, and extending RDF(S) where necessary. The result is an RDF Schema definition of OIL primitives, which allows one to express any OIL ontology in RDF syntax, thus enabling the added benefits of OIL, such as reasoning support and formal semantics, to be used on the Web. We conclude that our method of extending is equally applicable to other knowledge representation formalisms.

1 Introduction

RDF Schema provides means to define vocabulary, structure and constraints for expressing metadata about Web resources. However, formal semantics for the primitives defined in RDF Schema are not provided, and the expressivity of these primitives is not enough for full-fledged ontological modeling and reasoning. To perform these tasks, an additional layer on top of RDF Schema is needed. Tim Berners-Lee calls this layered architecture the *Semantic Web* [Berners-Lee, 1998].

At the lowest level of the Semantic Web (see figure 1) a generic mechanism for expressing machine readable semantics of data is required. The Resource Description Framework (RDF) [Lassila and Swick, 1999] is this foundation for processing metadata, providing a simple data model and a standardized syntax for metadata. Basically, it provides the language for writing down factual statements. The next layer is the schema layer (provided by the RDF Schema specification [Brickley and Guha, 2000]). We will show how a formal knowledge representation language can be used as the third, logical, layer. We will illustrate this by defining the ontology language OIL [Fensel et al., 2000, Horrocks et al., 2000] as an extension of RDF Schema.

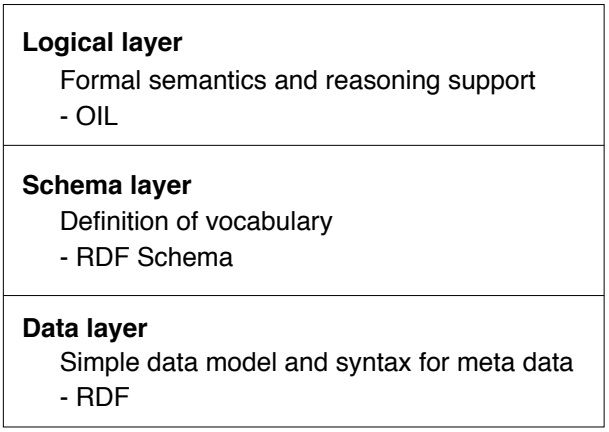

Figure 1. The three-layered architecture of the Semantic Web

OIL (Ontology Inference Layer), a major spin-off from the IST project On-To-Knowledge[1], is a Web-based representation and inference layer for ontologies, which unifies three important aspects provided by different communities: formal semantics and efficient reasoning support as provided by De-

[1] *On-To-Knowledge: Content-driven Knowledge-Management Tools through Evolving Ontologies* (IST-1999-10132). Project partners are the Vrije Universiteit Amsterdam (VU); the Institute AIFB, University of Karlsruhe, Germany; AIdministrator, the Netherlands; British Telecom Laboratories, UK; Swiss Life, Switzerland; CognIT, Norway; and Enersearch, Sweden. http://www.ontoknowledge.org/

scription Logics, epistemological rich modeling primitives as provided by the Frame community, and a standard proposal for syntactical exchange notations as provided by the Web community.

The content of the paper is organized as follows. In section 2 we provide a short introduction to RDF and RDF Schema. Section 3 provides a very brief introduction into OIL. Section 4 illustrates in detail how RDF Schema can be extended, using OIL as an example knowledge representation language. Finally, we provide some conclusions and recommendations in section 5.

2 RDF and RDF Schema

In this section we will discuss the main features of RDF and RDF Schema (or RDFS for short) and we will critically review some of their design decisions.

2.1 Introduction to RDF

A prerequisite for the Semantic Web is machine-processable semantics of the information. The Resource Description Framework (RDF) [Lassila and Swick, 1999] is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. Basically, RDF defines a data model for describing machine processable semantics of data. The basic data model consists of three object types:

- **Resources**: A resource may be an entire Web page; a part of a Web page; a whole collection of pages; or an object that is not directly accessible via the Web; e.g. a printed book. Resources are always named by URIs.
- **Properties**: A property is a specific aspect, characteristic, attribute, or relation used to describe a resource.
- **Statements**: A specific resource together with a named property plus the value of that property for that resource is an RDF statement.

These three individual parts of a statement are called, respectively, the *subject*, the *predicate*, and the *object*. In a nutshell, RDF defines object-property-value-triples as basic modeling primitives and introduces a standard syntax for them. An RDF document will define properties in terms of the resources to which they apply. As RDF statements are also resources, statements can be recursively applied to statements allowing their nesting.

2.2 Introduction to RDF Schema

The modeling primitives offered by RDF are very basic[2]. Therefore, the RDF Schema specification [Brickley and Guha, 2000] defines further modeling primitives in RDF. Examples are class, subclass relationship, domain and range restrictions for property, and subproperty. With these extensions, RDF Schema comes closer to existing ontology languages.

Despite the similarity in their names, RDF Schema fulfills a different role than XML Schema does. XML Schema, and also DTDs, prescribes the order and combination of tags in an XML document. In contrast, RDF Schema only provides information about the interpretation of the statements given in an RDF data model, but it does not constrain the syntactical appearance of an RDF description. Therefore, the definition of OIL in RDFS that will be presented in this document will not provide constraints on the structure of an actual OIL ontology.

In this section we will briefly discuss the overall structure of RDFS and its main modeling primitives.

2.2.1 The data model of RDF Schema

Figure 2 pictures the subclass-of hierarchy of RDFS and figure 3 pictures the instance-of relationships of RDFS primitives according to [Brickley and Guha, 2000]. The 'rdf' prefix refers to the RDF name space (i.e., primitives with this prefix are already defined in RDF) and 'rdfs' refers to new primitives defined by RDFS. Note that RDFS uses a non-standard object-meta model: the properties rdfs:subClassOf, rdf:type, rdfs:domain and rdfs:range are used both as primitive constructs in the definition of the RDF schema specification and as specific instances of RDF properties. This dual role makes it possible to view e.g. rdfs:subClassOf as an RDF property just like other predefined or newly introduced RDF properties, but introduces a self referentiality into the RDF schema definition, which makes it rather unique when compared to conventional model and meta modeling approaches, and makes the RDF schema specification very difficult to read and to formalize, cf. [Nejdl et al., 2000].

2.2.2 The modeling primitives of RDF Schema

In this section, we will discuss the main classes, properties, and constraints in RDFS.

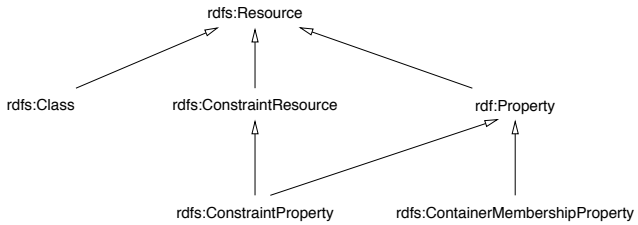

Figure 2. The subclass-of hierarchy of modeling primitives in RDFS.

- **Core classes** are rdfs:Resource, rdf:Property[3], and rdfs:Class. Everything that is described by RDF expressions is viewed to be an instance of the class

[2] Actually they correspond to binary predicates of ground terms, where, however, the predicates may be used as terms, as well.

[3] Note, that in this sense a property is an instance of a class.

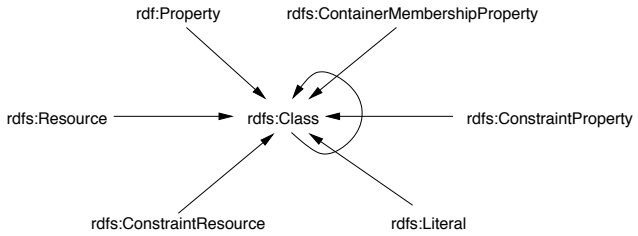

Figure 3. The instance-of relationships of modeling primitives in RDFS.

rdfs:Resource. The class rdf:Property is the class of all properties used to characterize instances of rdfs:Resource, i.e., each slot / relation is an instance of rdf:Property. Finally, rdfs:Class is used to define concepts in RDFS, i.e., each concept must be an instance of rdfs:Class.

- **Core properties** are rdf:type, rdfs:subClassOf, and rdfs:subPropertyOf. The rdf:type relation models instance-of relationships between resources and classes. A resource may be an instance of more than one class. The rdfs:subClassOf[4] relation models the subsumption hierarchy between classes and is supposed to be transitive. Again, a class may be subclass of several other classes, however, a class can neither be a subclass of its own nor a subclass of its own subclasses, i.e., the inheritance graph is cycle-free. The rdfs:subPropertyOf relation models the subsumption hierarchy between properties. If some property P_2 is a rdfs:subPropertyOf another property P_1 , and if a resource R has a P_2 property with a value V, this implies that the resource R also has a P_1 property with value V. Again, the inheritance graph is supposed to be cycle-free.
- **Core constraints** are rdfs:ConstraintResource, rdfs:ConstraintProperty, rdfs:range, and rdfs:domain. rdfs:ConstraintResource defines the class of all constraints. rdfs:ConstraintProperty is a subset of rdfs:ConstraintResource and rdf:Property covering all properties that are used to define constraints. At the moment, it has two instances: rdfs:range and rdfs:domain that are used to restrict range and domain of properties. It is not permitted to express two or more range constraints on a property. For domains this is not enforced and is interpreted as the union of the domains.

3 OIL

In this section we will give a very brief description of the OIL language; more details can be found in [Horrocks et al., 2000]. A small example ontology in OIL is provided in figure 4. This language has been designed so that:

1. it provides most of the modeling primitives commonly used in frame-based and Description Logic (DL) oriented Ontologies;
2. it has a simple, clean and well defined first-order semantics;
3. automated reasoning support, (e.g., class consistency and subsumption checking) can be provided. The FaCT system [Bechhofer et al., 1999], a DL reasoner developed at the University of Manchester, can be (and has been) used to this end [Stuckenschmidt, 2000].

It is envisaged that this core language will be extended in the future with sets of additional primitives, with the proviso that full reasoning support may not be available for ontologies using such primitives.

An ontology in OIL is represented via an *ontology container* and an *ontology definition* part. For the container, we adopt the components defined by Dublin Core Metadata Element Set, Version 1.1[5].

The ontology-definition part consist of an optional import statement, an optional rule-base and class and slot definitions.

A class definition (**class-def**) associates a class name with a class description. This class description in turn consists of the type of the definition (either primitive, which means that the stated conditions for class membership are necessary but not sufficient, or defined, which means that these conditions are both necessary and sufficient), a subclass-of statement and zero or more slot-constraints.

The value of a **subclass-of** statement is a (list of) class-expression(s). This can be either a class name, a slot-constraint, or a boolean combination of class expressions using the operators **AND**, **OR** and **NOT**, with the standard DL semantics.

A **slot-constraint** (a slot may also be called a role or an attribute) is a list of one or more constraints (restrictions) applied to a slot. Typical constraints are:

- **has-value (class-expr)** Every instance of the class defined by the slot constraint must be related, via the slot relation, to an instance of each class expression in the list.
- **value-type (class-expr)** *If* an instance of the class defined by the slot-constraint is related via the slot relation to some individual x, *then* x must be an instance of each class-expression in the list.
- **max-cardinality n (class-expr)** An instance of the class defined by the slot-constraint can be related to at most n distinct instances of the class-expression via the slot relation (also min-cardinality and, as a shortcut for both min and max, cardinality).

A slot definition (**slot-def**) associates a slot name with a slot definition. A slot definition specifies global constraints that apply to the slot relation. A slot-def can consist of a **subslot-of** statement, **domain** and **range** restrictions, and additional qualities of the slot, such as **inverse** slot, transitive, and symmetric.

[4] It is not really clear from the RDFS specification whether rdfs:subClassOf can be applied to rdf:Property. This seems possible because the latter is also an instance of rdfs:Class.

[5] See `http://purl.org/DC/`

ontology-container
title "African Animals"
creator "Ian Horrocks"
subject "animal, food, vegetarians"
description "A didactic example ontology describing African animals"
description.release "1.01"
publisher "I. Horrocks"
type "ontology"
format "pseudo-xml"
format "rdf"
identifier "http://www.ontoknowledge.org/oil/rdfs-oil.pdf"
source "http://www.africa.com/nature/animals.html"
language "en-uk"

ontology-definitions
slot-def *eats*
inverse *is-eaten-by*
slot-def *has-part*
inverse *is-part-of*
properties transitive
class-def animal
class-def plant
subclass-of NOT animal
class-def tree
subclass-of plant
class-def plant
slot-constraint *is-part-of*
has-value tree
class-def leaf
slot-constraint *is-part-of*
has-value branch
class-def defined carnivore
subclass-of animal
slot-constraint *eats*
value-type animal
class-def defined herbivore
subclass-of animal, **NOT** carnivore
slot-constraint *eats*
value-type
plant **OR**
slot-constraint *is-part-of*
has-value plant
class-def giraffe
subclass-of herbivore
slot-contraint *eats*
value-type leaf
class-def lion
subclass-of animal
slot-constraint *eats*
value-type herbivore
class-def tasty-plant
subclass-of plant
slot-constraint *is-eaten-by*
has-value herbivore,carnivore

Figure 4. An example OIL ontology.

The syntax of OIL is oriented towards XML and RDF. [Horrocks et al., 2000] defines a DTD and a XML schema definition for OIL. [Klein et al., 2000] derives an XML Schema for writing down instances of an OIL ontology. In this paper, we will derive the RDFS syntax of OIL.

4 OIL as an extension of RDF Schema

RDF provides basic modeling primitives: ordered triples of objects and links. RDFS enriches this basic model by providing a vocabulary for RDF, which is assumed to have a certain semantics. In this section we will provide a careful analysis of the relation between RDFS and OIL by defining OIL in RDFS, using existing vocabulary where possible and extending RDFS with OIL primitives where necessary.

4.1 The ontology container, import mechanism and rulebase

The outer box of the OIL specification in RDFS is defined by the XML prologue and the namespace definitions xmlns:rdf and xmlns:rdfs, which refer to RDF and RDFS, respectively. Namespace definitions make externally defined RDF constructs available for local use. Therefore, the OIL specification imports RDF and RDFS, and an actual ontology in OIL has namespace definitions which import both the RDF and RDFS definitions as well as the OIL specification itself.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/
                                22-rdf-syntax-ns#"[6]
    xmlns:rdfs="http://www.w3.org/TR/1999/
                                PR-rdf-schema-19990303#"
    xmlns:oil="http://www.ontoknowledge.org/oil/rdfschema"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcq="http://purl.org/dc/qualifiers/1.1/"

    <!-- The ontology defined in OIL with RDFS syntax-->
</rdf:RDF>
```

As one can see, the namespace definitions are not transitive. An actual ontology even needs to *reimport* RDF and RDFS definitions via xmlns:rdf and xmlns:rdfs, otherwise, all elements of OIL that directly correspond to RDF and RDFS elements would not be available.

The **ontology-container** of OIL provides metadata describing an OIL ontology. Because the structure and RDF-format of the Dublin Core element set is used, it is enough to import the namespace of the Dublin Core element set. Note that the fact that an OIL ontology should provide a container definition is an *informal* guideline in its RDFS syntax, because it is not possible to enforce this in the schema definition.

Apart from the container, an OIL ontology consists of a set of definitions. The **import** definition is a simple list of references to other OIL modules that are to be included in this ontology. We make use of the XML namespace mechanism to incorporate this mechanism in our RDFS specification. Notice again that, in contrast to the import statement in OIL, inclusion via the namespace definition is not transitive.

[6] Due to space limitations, we had to chop several URIs in two.

There are no constraints on the format or content of the **rule-base** in OIL. It could simply consist of untyped rules (a text string), or structured rules according to an externally defined format. For the latter, OIL provides a mechanism for referring to an external definition[7]. In the RDFS specification of OIL we provide a class RuleBase to classify the part of the ontology which defines the rules:

```
<rdfs:Class rdf:ID="RuleBase">
    <rdfs:comment>
        A user-defined rulebase possibly described
        by an external RDF-Schema
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/
                1999/PR-rdf-schema-19990303#Resource"/>
</rdfs:Class>
```

In an actual ontology, a simple example of a rulebase might look like:

```
<rdf:Description xmlns:syllogism=
                "http://old.greece/syllogism/">
    <rdf:type rdf:resource="http://www.ontoknowledge.org/
                oil/rdfschema#RuleBase"/>
    <syllogism:premise>
        if it rains, you get wet
    </syllogism:premise>
    <syllogism:fact>
        it rains
    </syllogism:fact>
    <syllogism:conclusion>
        you get wet
    </syllogism:conclusion>
</rdf:Description>
```

With help of the XML namespace facility it is very easy to refer to an externally defined format of the rules and axioms. See [Staab and Mädche, 2000].

4.2 Class and attribute definitions

In OIL, a class definition links a class with a name, a documentation, a type, its superclasses, and the attributes defined for it. In RDFS, classes are simply declared by giving them a name (with the ID attribute). We will show how OIL class definitions can be written down in RDF, while trying to make use of existing RDFS constructs as much as possible, but where necessary extending RDFS with additional constructs (see table 1 and figure 5).We conform to the informal RDF guideline to start property names with a lower-case letter, and class names with a capital.

To illustrate the use of these extensions, we will walk through them by means of a running example of a class definition in OIL that needs to be serialized in RDFS. Consider the following class definition (see also figure 4):

class-def defined herbivore
  subclass-of animal, **NOT** carnivore
    slot-constraint *eats*
      value-type
        plant **OR**
        slot-constraint *is-part-of* **has-value** plant

This defines a class named ”herbivore”, which is a subclass of all animals that are not carnivores (i.e. herbivore is *disjoint* from the class carnivore), and whose instances only eat plants or parts of plants.

We start by translating the class definition header. This can be done in a straightforward manner, using the rdfs:Class construct and the rdf:ID property to assign a name:

```
<rdfs:Class rdf:ID="herbivore"> </rdfs:Class>
```

However, from this definition it is not yet clear that this class is a defined class. We chose to introduce two classes in the OIL namespace, named PrimitiveClass and DefinedClass. In a particular class definition, whenever a class is a defined class, we use the rdf:type property to express this:

```
<rdfs:Class rdf:ID="herbivore">
    <rdf:type rdf:resource="http://www.ontoknowledge.org/
                        oil/rdfs-schema/#DefinedClass"/>
</rdfs:Class>
```

Alternatively, one could serialize this as:

```
<oil:DefinedClass rdf:ID="herbivore"> </oil:DefinedClass>
```

We will use the first method of serialization throughout this article, because, although both are equal in their meaning, the first one is, in our opinion, clearer to a human reader.

This way of making an actual class an instance of either DefinedClass or PrimitiveClass introduces a nice object-meta distinction between the OIL RDFS schema and the actual ontology: using rdf:type you can consider the class ”herbivore” to be an *instance* of DefinedClass. In general, if no explicit rdf:type is given to a class definition, the class is assumed to be primitive.

Next, we have to translate the subclass-of statement to RDFS. Here, we come across two features of OIL that are not directly expressible in RDFS: first, OIL allows *class expressions* as an extension to simple class names, and second, the subclass-of statement in OIL allows cycles in the subsumption hierarchy, which the RDFS equivalent does not. We ignore this last difference in our procedure for now (we come back to this problem in section 5).

In OIL it is possible to say that a class is a subclass of some class-expression, which is a boolean expression of classes. Three boolean operators are allowed in OIL: AND, OR and NOT. Since in RDFS the value of an rdfs:subClassOf statement can only be an instance of rdfs:Class, we decided to serialize the three boolean operators as classes in RDFS. This makes sense from a modeling perspective as well, as the result of the application of a boolean operator is the definition of a (nameless) class.

We introduce oil:ClassExpression as a placeholder class[8], with the operators AND, OR and NOT defined as subclasses of oil:ClassExpression. Also, since a single class is a essentially a simple kind of class-expression, rdfs:Class itself should be a subclass of oil:ClassExpression. Exploiting the credo of RDFS that anyone can say anything they want about

[7] However, OIL does not provide a syntax for these external definitions. The definition could simply consist of a description of the format of the rule-base in plain text.

[8] A placeholder class in the OIL RDFS specification is only used in the to apply domain- and rangerestrictions to a group of classes, and will not be used in the actual OIL ontology.

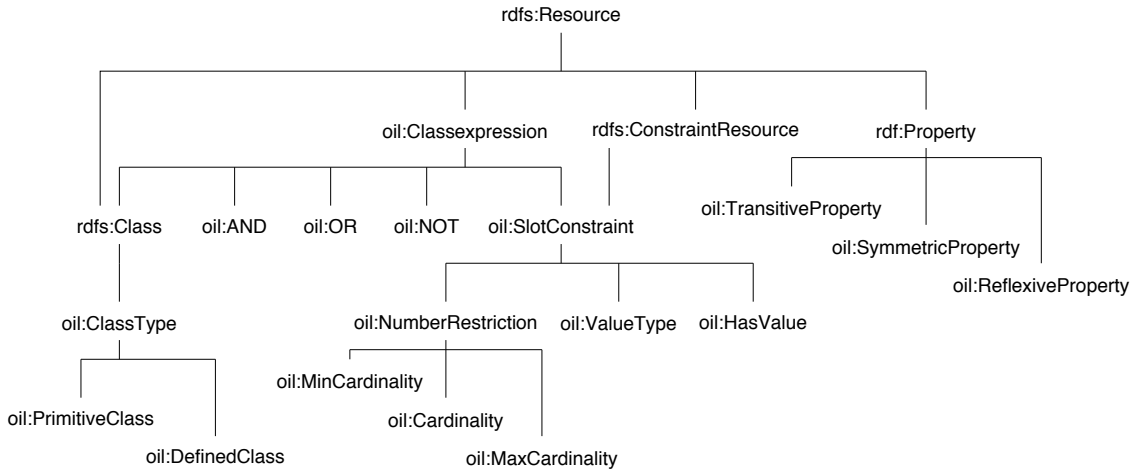

Figure 5. The OIL extensions to RDFS in the subsumption hierarchy.

existing resources, we add in the OIL-specification an extra subClassOf relation to the existing rdfs:Class, using the rdf:about construction:

```
<rdf:Description rdf:about="http://www.w3.org/2000/
                        01/rdf-schema#Class">
    <rdfs:subClassOf rdf:resource="#ClassExpression"/>
</rdf:Description>
```

Consequently, we also have to extend the range of rdfs:subClassOf with oil:ClassExpression.

```
<rdf:Description rdf:about="http://www.w3.org/2000/
                        01/rdf-schema#subClassOf">
    <rdfs:range rdf:resource="#ClassExpression"/>
</rdf:Description>
```

The AND, OR and NOT operators are connected to operands using the oil:hasOperand property. This property has no direct equivalent in OIL primitive terms, but is a helper to connect two class-expressions, because in the RDF data model one can only relate two classes by means of a Property. The oil:hasClass and oil:hasSlotConstraint properties have more or less the same meaning, but are used in different contexts, and thus have different domain and range restrictions. Because of this same function in more specific contexts, we would prefer to model oil:hasOperand and oil:hasSlotConstraint as rdfs:subPropertyOf oil:hasClass, because essentially oil:hasOperand is a specific case of oil:hasClass. However, the RDFS specification is unclear about the way domain and range restrictions are inherited. Because we are uncertain about this, we chose to keep them as separate properties.

In the case of our example, we need an RDFS-equivalent for NOT. The OIL RDF Schema definition of this operator looks like this:

```
<rdfs:Class rdf:ID="NOT">
    <rdfs:subClassOf rdf:resource="#ClassExpression"/>
</rdfs:Class>
```

and the helper property is defined as follows:

```
<rdf:Property rdf:ID="hasOperand">
    <rdfs:domain rdf:resource="#AND"/>
    <rdfs:domain rdf:resource="#OR"/>
    <rdfs:domain rdf:resource="#NOT"/>
    <rdfs:range rdf:resource="#ClassExpression"/>
</rdf:Property>
```

The fact that hasOperand is only to be used on specific class expressions (AND, OR and NOT) is expressed using the rdfs:domain construction. This type of modeling stems directly from the RDF property-centric approach.

Back to the example now. The subclass-of statement contains a comma seperated list. To serialize this in RDFS, we simply use one subClassOf statement for each item in this list:

```
<rdfs:Class rdf:ID="herbivore">
    <rdf:type rdf:resource="http://www.ontoknowledge.org/
                        oil/rdfs-schema/#DefinedClass"/>
    <rdfs:subClassOf rdf:resource="#animal"/>
    <rdfs:subClassOf>
        <oil:NOT>
            <oil:hasOperand rdf:resource="#carnivore"/>
        </oil:NOT>
    </rdfs:subClassOf>
</rdfs:Class>
```

As we can see, the first subClassOf statement is normal RDFS usage, where a named class is used as the value. In the second statement however, RDFS is extended using OIL primitives as explained above.

An alternative to this solution would be to serialize the list as one subClassOf statement, using the oil:AND construct to combine the items (after all, the comma seperated list is an implicit conjunction). However, there are some significant disadvantages to this approach:

- While none of the semantics of the original OIL expression is lost, its original modeling is. The difference between a comma seperated list within the subclass-of statement or an explicit conjunction is unretrievable. In other words: translating from OIL to RDF and back is no longer guaranteed to give an identical ontology from a modeling perspective (though semantic equivalence is of course still preserved).
- While using an RDFS-construct to model subsumption is a good idea, in this case one might get the wrong impression that an RDF agent would be able to deal with the information provided in the subClassOf statement. However, in the case of oil:AND as the outer class, it would

only be able to see that the defined class is a subClass of some other class, but no specifics of this other class are known because it does not know the semantics of oil:AND. By presenting it as a list, any RDFS agent can understand the subclass statements.

We still need to serialize the slot constraint. In RDFS, there is no mechanism for restricting the attributes of a class on a local level. This is again due to the property-centric nature of the RDF data model: properties are defined globally, with their domain description coupling them to the relevant classes.

To overcome this problem, we introduce the oil:hasSlotConstraint property, which is an rdf:type of rdfs:ConstraintProperty (analogous to rdfs:domain and rdfs:range). Here we take full advantage of the intended extensibility of RDFS. We also introduce oil:SlotConstraint as a placeholder class for specific classes of slot constraints, such as has-value, value-type, cardinality and so on. These are all modeled in RDFS as subclasses of oil:SlotConstraint:

```
<rdfs:Class rdf:ID="ValueType">
    <rdfs:subClassOf rdf:resource="#SlotConstraint"/>
</rdfs:Class>
```

and similar for the other slot constraints. For the three cardinality constraints, an extra property ”number” is introduced, which is used to assign a concrete value to the cardinality constraints.

We now define the oil:hasSlotConstraint property:

```
<rdf:Property rdf:ID="hasSlotConstraint">
    <rdf:type rdf:resource="http://www.w3.org/TR/1999/
          PR-rdf-schema-19990303#ConstraintProperty"/>
    <rdfs:domain rdf:resource="http://www.w3.org/2000/
          01/rdf-schema#Class/">
    <rdfs:range rdf:resource="#SlotConstraint"/>
</rdf:Property>
```

In our example, the slot-constraint would be serialized using the primitives introduced above, giving us the following complete translation:

```
<rdfs:Class rdf:ID="herbivore">
    <rdf:type rdf:resource="http://www.ontoknowledge.org/
              oil/rdfs-schema/#DefinedClass"/>
    <rdfs:subClassOf rdf:resource="#animal"/>
    <rdfs:subClassOf>
        <oil:NOT>
            <oil:hasOperand rdf:resource="#carnivore"/>
        </oil:NOT>
    </rdfs:subClassOf>
    <oil:hasSlotConstraint>
        <oil:ValueType>
            <oil:hasProperty rdf:resource="#eats"/>
            <oil:hasClass>
                <oil:OR>
                    <oil:hasOperand rdf:resource="#plant"/>
                    <oil:hasOperand>
                        <oil:HasValue>
                            <oil:hasProperty rdf:resource=
                              "#is-part-of"/>
                            <oil:hasClass rdf:resource=
                              "#plant"/>
                        </oil:HasValue>
                    </oil:hasOperand>
                </oil:OR>
            </oil:hasClass>
        </oil:ValueType>
    </oil:hasSlotConstraint>
</rdfs:Class>
```

In the second operand of the OR operator we see an example of a slot constraint used as a nameless class definition. It basically specifies the class of all things that have a property ”is-part-of” of which the value is ”plant”.

An alternative to the introduction of oil:hasSlotConstraint would be to serialize all slot constraints as part of the subClassOf statement. After all, a slot constraint is effectively the definition of a (nameless) class and can as such be used within any class expression. Also, this would eliminate the need for an extension of RDFS with the oil:hasSlotConstraint construct. However, there are some significant disadvantages to this approach, quite similar to the disadvantages stated earlier with respect to the serialization of a comma seperated list of class names: stuffing the entire class definition within a single subClassOf statement makes the RDF specification unclear, part of the modeling is lost, and an RDF agent is less likely to understand the subsumption hierarchy.

The serialization we propose gives us enough expressiveness to translate any possible OIL class definition to an RDF syntax. Use of RDF(S) specific constructs is maximized without sacrificing clarity of the specification, to enable RDF agents that are not OIL-aware to understand as much of the specification as possible, while retaining the possibility to translate back to OIL unambiguously.

In the next section, we will examine how to serialize global slot definitions.

4.3 Slot definitions

Both OIL and RDFS allow slots as first-class citizens of an ontology. Therefore, slot definitions in OIL map nicely onto property definitions in RDFS. Also the ”subslot-of”, ”domain”, and ”range” properties have almost direct equivalents in RDFS. In table 2, an overview of the OIL constructs and the corresponding RDFS constructs is given.

There is a subtile difference between range restrictions in OIL and their equivalent in RDFS: in the latter, only one restriction per Property is allowed. In contrast with RDFS, OIL allows more than one range restriction on a property. Although this can be circumvented by defining a dummy superclass of all classes in the range restriction, we see no reason for this restriction in RDFS. From a modeling point of view, allowing more than one range restriction is a much cleaner solution.

Translating a slot definition which comprises only constructs with more or less direct counterparts in RDFS is straightforward. For example:

slot-def *gnaws*
subslot-of *eats*
domain Rodent

would become:

```
<rdf:Property rdf:ID="gnaws">
    <rdfs:subPropertyOf rdf:resource="#eats"/>
    <rdfs:domain rdf:resource="#Rodent"/>
</rdf:Property>
```

Table 1. Class-definitions in OIL and the corresponding RDF(S) constructs

OIL primitive	RDFS syntax	type
class-def	`rdfs:Class`	class
subclass-of	`rdfs:subClassOf`	property
class-expression	*`oil:ClassExpression`* (placeholder only)	class
AND	`oil:AND` (subclass of ClassExpression)	class
OR	`oil:OR` (subclass of ClassExpression)	class
NOT	`oil:NOT` (subclass of ClassExpression)	class
slot-constraint	*`oil:SlotConstraint`* (placeholder only)	class
	`oil:hasSlotConstraint` (rdf:type of rdfs:ConstraintProperty)	property
	`oil:NumberRestriction` (placeholder only) (subclass of oil:SlotConstraint)	class
has-value	`oil:HasValue` (subclass of oil:SlotConstraint)	class
value-type	`oil:ValueType` (subclass of oil:SlotConstraint)	class
max-cardinality	`oil:MaxCardinality` (subclass of oil:NumberRestriction)	class
min-cardinality	`oil:MinCardinality` (subclass of oil:NumberRestriction)	class
cardinality	`oil:Cardinality` (subclass of oil:NumberRestriction)	class

However, global slot-definitions in OIL allow specification of more aspects of a slot than property definitions in RDFS do. Besides the domain and range restrictions, OIL slots can also have an ”inverse” attribute and qualities like ”transitive” and ”symmetric”.

We therefore added a property ”inverseRelationOf” with ”rdf:Property” as domain and range. We also added the classes ”TransitiveRelation” and ”SymmetricRelation” to reflect the different qualities of a slot. In the RDFS-serialization of OIL, the rdf:type property can be used to add a quality to a property. For example, the OIL definition of:

slot-def *has-part*
inverse *is-part-of*
properties transitive

is in RDFS:

```
<rdf:Property rdf:ID="has-part">
    <rdf:type rdf:resource="http://www.ontoknowledge.org/
                oil/rdf-schema/#TransitiveRelation"/>
    <oil:inverseRelationOf rdf:resource="#is-part-of"/>
</rdf:Property>
```

This way of translating the qualitities of properties features the same nice object-meta distinction between the OIL RDFS schema and the actual ontology as the translation of the ”type” of a class (see section 4.2). In an actual ontology, the property ”has-part” can be considered as an *instance* of a TransitiveRelation. Note that it is allowed to make a property an instance of more than one class, and thus giving it multiple qualities. Note that this way of representing qualities of properties in RDFS follows the proposed general approach of modeling axioms in RDFS, presented in [Staab et al., 2000]. In this approach, the same distinction between language-level constructs and schema-level constructs is made.

One alternative way of serializing the attributes of properties would be to define the qualities ”transitive” and ”symmetric” as subproperties of rdf:Property. Properties in the actual ontology (e.g. ”has-part”) would in their turn be defined as subProperties of these qualities (e.g. transitiveProperty). However, this would mixup the use of properties at the OIL-specification level and at the actual ontology level.

A third way would be to model the qualities as subproperties of rdf:Property again, but to define properties in the actual ontology as instances (rdf:type) of such qualities. In this aproach, the object-meta level distinction is preserved. However, we dislike the use of rdfs:subPropertyOf at the meta-level, because then rdfs:subPropertyOf has two meanings, at the meta-level and at the object-level.

We therefore prefer the first solution because of the clean distinction between the meta and object level.

Table 2. Slot-definitions in OIL and the corresponding RDF(S) constructs.

OIL primitive	RDFS syntax	type
slot-def	`rdf:Property`	class
subslot-of	`rdfs:subPropertyOf`	property
domain	`rdfs:domain`	property
range	`rdfs:range`	property
inverse	`oil:inverseRelationOf`	property
transitive	`oil:TransitiveRelation`	class
symmetric	`oil:SymmetricRelation`	class

5 Conclusion

In the previous section we have shown that it is possible to define a formal knowledge representation schema as an extension to RDFS, effectively implementing the "third layer of the Semantic Web". We did this by defining the ontology language OIL in RDFS, using existing primitives as much as possible while retaining a unambiguous mapping between the original OIL specification and its RDFS serialization. The resulting extension of RDFS allows the specification of domain ontologies that are already partially understandable by non-OIL-aware RDFS applications, while OIL-aware applications can fully benefit of the added features, such as formal semantics and reasoning support.

There are still a few unsolved problems with the specification of OIL into RDFS. First, we did not take into account a restriction on the rdfs:subClassOf statement, i.e. the restriction that no cycles are allowed in the subsumption hierarchy. We think that this restriction should be dropped: without cycles one cannot even represent equivalence between two classes — in our view this is an essential modeling primitive for any knowledge representation language. Moreover, these kinds of constraint significantly add to the complexity of parsing/validating RDF documents in a way which we think would be highly undesirable. This is because they are really semantic constraints rather than syntactic ones (they limit the kinds of models that can be represented), even if the reasoning required in order to detect constraint violation is of a very basic kind.

Second, in contrast with RDFS, OIL allows more than one range restriction on a property. Although this can be circumvented by defining a dummy superclass of all classes in the range restriction, we see no reason for this restriction in RDFS. From a modeling point of view, allowing more than one range restriction is a much cleaner solution.

During the process of extending RDFS, we encountered a couple of peculiarities in the RDFS definition itself. The most striking of these is the non-standard object-meta model, as already discussed in section 2.2.1. The main problem with this non-standard model is that some properties have a dual role in the RDFS specification, both at the schema level and instance level (cf. [Nejdl et al., 2000]). This makes it quite a challenge for modelers to understand the RDFS specification. We tried to make this distinction clear in our extensions by using the rdf:type relationship consistently as an object-meta relationship.

Furthermore, the semantics of several relationships are unclear. It is not obvious that the meaning of a list of domain (or range) restrictions is the union of the elements. Also, the meaning of the subPropertyOf relation with respect to the inheritance of the domain and range restrictions is unclear.

Despite these problems, we think that this procedure of extending RDFS is also applicable to other knowledge representation formalisms.

Acknowledgements. We would like to thank Monica Crubezy, Ying Ding, Michael Erdmann, Frank van Harmelen, Arjohn Kampman, and Borys Omelayenko for their helpful comments and for reviewing early drafts of this paper.

References

Bechhofer et al., 1999. Bechhofer, S., Horrocks, I., Patel-Schneider, P. F., and Tessaris, S. (1999). A proposal for a description logic interface. In *Proc. of DL'99*, pages 33–36.

Berners-Lee, 1998. Berners-Lee, T. (1998). Semantic web road map. Internal note, World Wide Web Consortium. See http://www.w3.org/DesignIssues/Semantic.html.

Brickley and Guha, 2000. Brickley, D. and Guha, R. (2000). Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium. See http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

Fensel et al., 2000. Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., and Klein, M. (2000). OIL in a nutshell. In *Proceedings of 12th International Conference on Knowledge Engineering and Knowledge Management, Juan-les-Pins, French Riviera*.

Horrocks et al., 2000. Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Staab, S., and Studer, R. (2000). OIL: The Ontology Inference Layer. Technical report, University of Manchester / Vrije Universiteit Amsterdam. See http://www.ontoknowledge.org/oil/.

Klein et al., 2000. Klein, M., Fensel, D., van Harmelen, F., and Horrocks, I. (2000). The relation between ontologies and schema-languages: Translating OIL-specifications in XML-Schema. In *Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI 2000, Berlin, Germany*.

Lassila and Swick, 1999. Lassila, O. and Swick, R. R. (1999). Resource Description Framework (RDF): Model and Syntax Specification. Recommendation, World Wide Web Consortium. See http://www.w3.org/TR/REC-rdf-syntax/.

Nejdl et al., 2000. Nejdl, W., Wolpers, M., and Capella, C. (2000). The RDF Schema Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000, St. Goar*. Foelbach Verlag, Koblenz.

Staab et al., 2000. Staab, S., Erdmann, M., Mädche, A., and Decker, S. (2000). An extensible approach for modeling ontologies in RDF(S). In *First Workshop on the Semantic Web at the Fourth European Conference on Digital Libraries, Lisbon, Portugal*.

Staab and Mädche, 2000. Staab, S. and Mädche, A. (2000). Axioms are objects, too - ontology engineering beyond the modeling of concepts and relations. Technical Report 399, Institut AIFB, Universität Karlsruhe.

Stuckenschmidt, 2000. Stuckenschmidt, H. (2000). Using OIL for Intelligent Information Integration. In *Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI 2000, Berlin, Germany*.