ELSEVIER

Computer Networks 39 (2002) 609–634

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Enabling knowledge representation on the Web by extending RDF Schema

Jeen Broekstra [a], Michel Klein [b,*], Stefan Decker [c], Dieter Fensel [b], Frank van Harmelen [b], Ian Horrocks [d]

[a] *Aidministrator Nederland bv, Prinses Julianaplein 14-b, 3817 CS Amersfoort, Netherlands*
[b] *Division of Mathmatics and Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, Netherlands*
[c] *Department of Computer Science, Stanford University, Stanford, CA 94305-9040, USA*
[d] *Department of Computer Science, University of Manchester, Manchester, UK M13 9PL*

Abstract

Recently, a widespread interest has emerged in using ontologies on the Web. Resource Description Framework Schema (RDFS) is a basic tool that enables users to define vocabulary, structure and constraints for expressing meta data about Web resources. However, it includes no provisions for formal semantics, and its expressivity is not sufficient for full-fledged ontological modeling and reasoning. In this paper, we will show how RDFS can be extended to include a more expressive knowledge representation language. That, in turn, would enrich it with the required additional expressivity and the semantics of that language. We do this by describing the ontology language Ontology Inference Layer (OIL) as an extension of RDFS. An important advantage to our approach is that it ensures maximal sharing of meta data on the Web: even partial interpretation of an OIL ontology by less semantically aware processors will yield a correct partial interpretation of the meta data. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Semantic Web; Knowledge representation; Ontologies; RDF; OIL; DAML

1. Introduction

Currently, computers are changing from single isolated devices to entry points into a worldwide network of information exchange and business transactions (cf. [1]). Support in data, information, and knowledge exchange is becoming the key issue in current computer technology. *Ontologies* will play a major role in supporting information exchange processes in various areas.

Many definitions of ontologies have been formulated over the past decade. In our opinion, however, the one that best captures the essence of an ontology is based on the related definitions by [2]: an ontology is a formal, explicit specification of a shared conceptualization. A *conceptualization* refers to an abstract model of some phenomenon in the world which identifies the relevant concepts of that phenomenon. *Explicit* means that the type of concepts used and the constraints on their use are explicitly defined. *Formal* refers to

* Corresponding author.
E-mail address: michel.klein@cs.vu.nl (M. Klein).

the fact that the ontology should be machine processible, i.e., the machine should be able to interpret the information provided unambiguously. *Shared* reflects the idea that an ontology captures consensual knowledge, that is, it is not restricted to some individual, but accepted by a group.

Resource Description Framework Schema (or RDFS for short) [3] provides a means to define vocabulary, structure and constraints for expressing meta data about Web resources. However, formal semantics for the primitives defined in RDFS are not provided, and the expressivity of these primitives is not sufficient for full-fledged ontological modeling and reasoning. These tasks require an additional layer on top of RDFS. Tim Berners-Lee calls this layered architecture the *Semantic Web* [4].

The lowest level of the Semantic Web requires a generic mechanism for expressing machine readable semantics of data. The RDF [5] is this foundation for processing meta data, providing a simple data model and a standardized syntax for meta data. Basically, it provides the language for writing down factual statements. The next layer is the schema layer (provided by RDFS). 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 Ontology Inference Layer (OIL) [6,7] as an extension of RDFS.

OIL, a major spin-off of 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 Description Logics (DLs), epistemological rich modeling primitives as provided by the Frame community, and a standard proposal for syntactical exchange notations as provided by the Web community.

In this paper, we will show how RDFS can be extended to contain a more expressive knowledge representation language, which would enrich it with the required additional expressivity and the semantics of that language. We will do this by describing the ontology language OIL as an extension of RDFS. Enabling the use of more expressive formal languages on the Web offers the basis for inferences, and thus for automated services, such as information filtering and query answering. An important advantage peculiar to our approach is that our extension method ensures maximal sharing of meta data on the Web: even partial interpretation of an OIL ontology by less semantically aware processors will yield a correct partial interpretation of the meta data.

This paper is structured as follows. Section 2 presents a short introduction to RDF and RDFS. Section 3 contains a very brief introduction into OIL. Section 4 illustrates in detail how RDFS can be extended, using OIL as an example of a knowledge representation language. The result is an RDFS definition of OIL primitives, which makes it possible to express any OIL ontology in RDF syntax. In Section 5 we discuss how our approach makes it possible to tap into the additional advantages of OIL on the Web, such as reasoning support and formal semantics, while retaining maximal compatibility with 'pure' RDF(S). Finally, Section 6 presents our conclusions.

2. RDF and RDFS

This section presents the main features of RDF and RDFS and a critical review of some of their design decisions.

2.1. Introduction to RDF

The Semantic Web requires machine-processible semantics in information. RDF is a foundation for processing meta data; it provides interoperability between applications that exchange machine-processible

[1] On-To-Knowledge: Content-driven Knowledge-Management Tools through Evolving Ontologies (IST-1999-10132). http://www.ontoknowledge.org/ [8].

information on the Web. Basically, RDF defines a data model for describing machine-processible semantics in 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*: An RDF statement consists of a specific resource, together with a named property and the value of that property for the resource in question.

These three individual parts of a statement are called the *subject*, *predicate*, and the *object*, respectively. 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. For example:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org">
    <Publisher>World Wide Web Consortium</Publisher>
  </rdf:Description>
</rdf:RDF>
```

states that http://www.w3.org (the subject) has as its publisher (the predicate) the W3C (the object). Since both the subject and the object of a statement can be re-sources, these statements can be linked in a chain.

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <Creator rdf:resource="http://www.w3.org/staffId/85740"/>
  </rdf:Description>
  <rdf:Description about="http://www.w3.org/staffId/85740">
    <Email>lassila@w3.org</v:Email>
  </rdf:Description>
</rdf:RDF>
```

states that http://www.w3.org/Home/Lassila (the subject) was created by staff member 85740 (the object). In the next statement, this same resource (staff member 85740) acts as the subject, who states that his email address is lassila@w3.org. Finally, RDF statements are also resources. Thus, statements can be applied recursively to statements, allowing their nesting.

As a result of the above, the underlying data model is a labeled hyper-graph, and each statement acts as a predicate-labeled link between object and subject. The graph is a hyper-graph since each node, in itself, can again contain an entire graph.

2.2. Introduction to RDFS

The modeling primitives offered by RDF are very basic. [2] Therefore, the RDFS specification defines further modeling primitives in RDF. In other words, RDFS extends (or enriches) RDF by assigning an

[2] Actually they correspond to binary predicates of ground terms, where the predicates can also be used as terms.

externally specified semantics to specific resources, e.g., to rdfs:subclassOf, to rdfs:Class etc. It is only because of these external semantics that RDFS is useful. Moreover, these semantics cannot be captured in RDF. (If that were possible, there would be no need for RDFS.) OIL bears a similar relationship to RDFS: by defining semantics for specific resources we can further extend (or enrich) RDFS. This allows OIL to capture meaning that cannot be captured in RDFS; and this is where its added value lies. Furthermore, we will be careful to create this extension to RDFS in such a way that a partial interpretation without the additional OIL semantics will still yield a valid RDFS interpretation.

Despite the similarity in their names, RDFS fulfills a different role than does XML Schema. XML Schema, like DTDs, prescribes the order and combination of tags in an XML document. In contrast, RDFS only provides information about the interpretation of the statements in an RDF data model, but does not constrain the syntactical appearance of an RDF description. Therefore, the definition of OIL in RDFS presented in this document will not include 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 RDFS

Fig. 1 presents the subclass-of hierarchy of RDFS and Fig. 2 presents the instance-of relationships of RDFS primitives according to [3]. 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 RDFS specification and as specific instances of RDF properties. This dual role makes it possible to view, say, rdfs:subClassOf as an RDF property just like other predefined or newly introduced RDF properties. However, it does introduce a self referentiality into the RDFS definition, which makes it rather unique as compared to conventional model and meta modeling approaches, and makes the RDFS specification very difficult to read and to formalize, cf. [9].

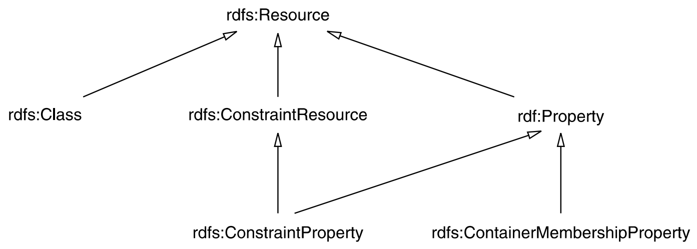

Fig. 1. The subclass-of hierarchy of modeling primitives in RDFS.

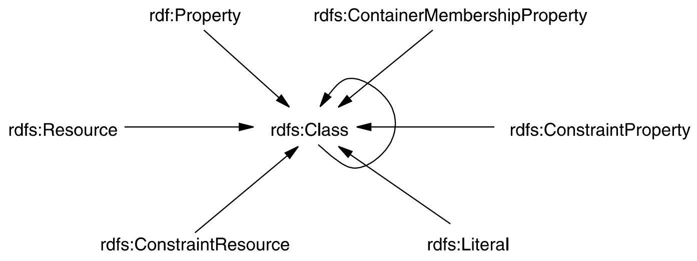

Fig. 2. The instance-of relations of modeling primitives in RDFS.

2.2.2. The modeling primitives of RDFS

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

- *Core classes* are rdfs:Resource, rdf:Property, and rdfs:Class. Everything that is described by RDF expressions is viewed to be an instance of the class 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 [3] relation models the subsumption hierarchy between classes and is supposed to be transitive. Again, a class may be a 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 a 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 the 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

This section offers a very brief description of the OIL language; more details can be found in [7]. A small example of an ontology in OIL is presented in Fig. 3.

This language has been designed such that

(1) it provides most of the modeling primitives commonly used in frame-based and DL oriented Ontologies;
(2) it features 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 [10], a DL reasoner developed at the University of Manchester, can be, and has been, used to this end [11].

This core language is expected to be extended in the future with sets of additional primitives. It should be noted however, 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 segment. For the container, we adopt the components defined by Dublin Core Metadata Element Set, Version 1.1. [4]

[3] 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.

[4] 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 and plants"
   description.release "2.0"
   publisher "I. Horrocks"
   type "ontology"
   format "pdf"
   identifier "http://.../oil-rdfs.pdf"
   source "http://www.africa.com/"
   language "en-uk"
ontology-definitions
   slot-def eats
      inverse is-eaten-by
   slot-def has-part
      inverse is-part-of
      properties transitive
   slot-def weight
      range (min 0)
      properties functional
   slot-def colour
      range string
      properties functional
   class-def animal
   class-def plant
   disjoint animal plant
   class-def tree
      subclass-of plant
   class-def branch
      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
      slot-constraint eats
         value-type (plant or
         (slot-constraint is-part-of
            has-value plant))
   disjoint carnivore herbivore
   class-def mammal
      subclass-of animal
   class-def elephant
      subclass-of herbivore mammal
      slot-constraint eats
         value-type plant
      slot-constraint colour
         has-filler "grey"
   class-def defined african-elephant
      subclass-of elephant
      slot-constraint comes-from
         has-filler Africa
   class-def defined indian-elephant
      subclass-of elephant
      slot-constraint comes-from
         has-filler India
   disjoint-covered elephant by
         african-elephant indian-elephant
   —— instance information ——
   instance-of Africa continent
   instance-of Asia continent
   related is-part-of India Asia
```

Fig. 3. An example OIL ontology, modeling the animal kingdom.

The ontology-definition segment consists of an optional import statement, an optional rule base and class, slot and axiom definitions.

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

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.

In some situations it is possible to use a *concrete-type-expression* instead of a class expression. A concrete-type-expression defines a range over some data type. Two data types that are currently supported in OIL are *integer* and *string*. Ranges can be defined using the expressions (**min X**), (**max X**), (**greater-than X**), (**less-than X**), (**equal X**) and (**range X Y**). For example, (**min** 21) defines the data type consisting of all the integers greater than or equal to 21. Another example is (**equal** "xyz"), which defines the data-type consisting of the string "xyz".

A slot constraint (**slot-constraint**) is a list of one or more constraints (restrictions) applied to a slot (property). 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, at the most, be related to 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.

An *axiom* asserts some additional facts about the classes in the ontology, for example that the classes carnivore and herbivore are disjoint (that is, have no instances in common). Valid axioms are

- **disjoint** (**class-expr**)**+** All of the class expressions in the list are pairwise disjoint.
- **covered** (**class-expr**) **by** (**class-expr**)**+** Every instance of the first class expression is also an instance of at least one of the class expressions in the list.
- **disjoint-covered** (**class-expr**) **by** (**class-expr**)**+** Every instance of the first class expression is also an instance of exactly one of the class expressions in the list.
- **equivalent** (**class-expr**)**+** All of the class expressions in the list are equivalent (i.e., they have the same instances).

The syntax of OIL is geared towards XML and RDF. Ref. [7] defines a DTD and a XML schema definition for OIL. Ref. [12] derives an XML Schema for writing down instances of an OIL ontology. In this paper, we will derive the RDFS syntax for OIL.

4. OIL as an extension of RDFS

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. This section presents a careful analysis of the relation between RDFS and OIL by defining OIL in RDFS, using existing vocabulary wherever possible. The reason for this is twofold. First, by reusing RDFS primitives we are effectively imposing formal semantics on them, specifically the formal semantics of OIL. Secondly, because we only extend RDFS with new primitives where necessary, RDFS becomes a full sub-language of OIL, thus providing backward compatibility from OIL to RDFS.

The complete schema can also be found at `http://www.ontoknowledge.org/oil/rdf-schema/`. The RDFS serialization of the example from the previous section is available at `http://www.ontoknowledge.org/oil/a-animals.rdfs`.

4.1. The ontology container and import mechanism

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. Thus, the OIL specification uses RDF and RDFS, and an actual ontology in OIL has namespace definitions which make both the RDF and RDFS definitions as well as the OIL specification itself available.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
  <rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:oil="http://www.ontoknowledge.org/
         oil/rdf-schema/2000/11/10-oil-standard"
  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>
```

It is important to note that namespace definitions are not import statements and are, therefore, not transitive. An actual ontology also has to define the namespaces for RDF and RDFS via xmlns:rdf and xmlns:rdfs, otherwise, all elements of OIL that directly correspond to RDF and RDFS elements would be unavailable.

OIL's **ontology-container** provides meta data describing an OIL ontology. Because the structure and RDF format of the Dublin Core element set is used, it is sufficient to import the namespace of the Dublin Core element set. It is important to note that an OIL ontology's provision of a container definition is an informal guideline in its RDFS syntax, because it is impossible to enforce this in the schema definition.

Aside 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 to be included in the ontology. We make use of the XML namespace mechanism to incorporate this mechanism in our RDFS specification. Again, in contrast to the import statement in OIL, inclusion via the namespace definition is not transitive.

4.2. Class and attribute definitions

In OIL, a class definition links a class with a name, documentation, a type, its superclasses, and the attributes defined for it. In RDFS, classes are simply declared by assigning them a name (with the ID attribute). We will demonstrate how to write OIL class definitions in RDF, making maximum use of existing RDFS constructs, but extending RDFS with additional constructs where necessary (see Table 1 and Fig. 4). We have followed the informal RDF guideline of starting property names with a lower-case letter, and class names with a capital.

To illustrate the use of these extensions, we will examine them systematically with various example OIL class definitions that need to be represented in RDFS syntax:

```
class-def defined herbivore
  subclass-of animal
    slot-constraint eats
      value-type (plant or
      (slot-constraint is-part-of has-value plant))
```

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 BooleanExpression)	class
or	`oil:Or` (subclass of BooleanExpression)	class
not	`oil:Not` (subclass of BooleanExpression)	class
slot-constraint	`oil:PropertyRestriction` (placeholder only)	class
	`oil:hasPropertyRestriction` (rdf:type of rdfs:ConstraintProperty)	property
	`oil:CardinalityRestriction` (placeholder only) (subclass of oil:PropertyRestriction)	class
has-value	`oil:HasValue` (subclass of oil:PropertyRestriction)	class
has-filler	`oil:HasFiller` (subclass of oil:PropertyRestriction)	class
value-type	`oil:ValueType` (subclass of oil:PropertyRestriction)	class
max-cardinality	`oil:MaxCardinality` (subclass of oil:CardinalityRestriction)	class
min-cardinality	`oil:MinCardinality` (subclass of oil:CardinalityRestriction)	class
cardinality	`oil:Cardinality` (subclass of oil:CardinalityRestriction)	class

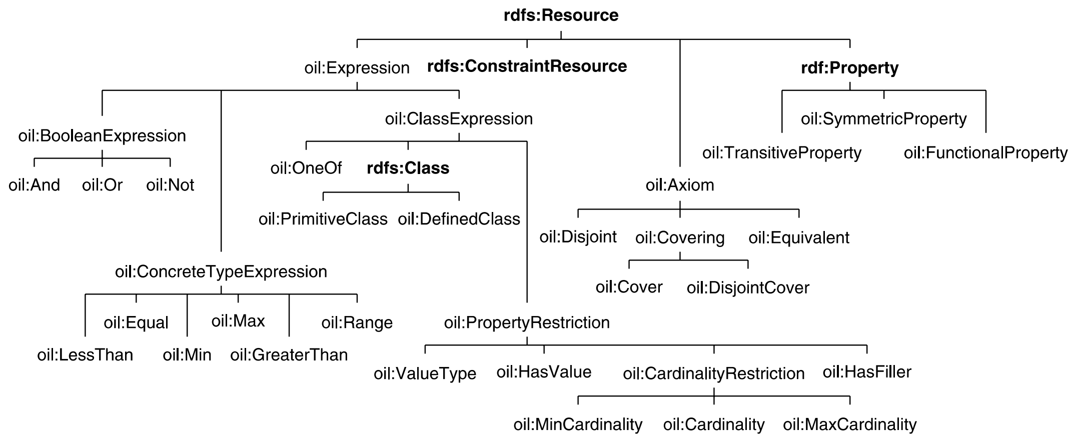

Fig. 4. The OIL extensions to RDFS in the subsumption hierarchy.

class-def elephant
  subclass-of herbivore mammal
  slot-constraint eats
    value-type plant
  slot-constraint colour
  has-filler "grey"

The first defines a class "herbivore", a subclass of animal, whose instances eat plants or parts of plants. The second defines a class "elephant", which is a subclass of both herbivore and mammal.

4.2.1. Defined classes and primitive classes

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

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

This definition does not yet clearly identify this class as a defined class. We chose to introduce two extra classes in the OIL namespace: PrimitiveClass and DefinedClass. In a particular class definition, we can use one of these two options to identify a class as a defined class:

```
<rdfs:Class rdf:ID="herbivore" >
  <rdf:type rdf:resource="http://www.ontoknowledge.org/oil/
      rdf-schema/2000/11/10-oil-standard#DefinedClass"/>
</rdfs:Class>
```

or:

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

We will use the first method of serialization throughout this article, but it is important to note that both models are exactly the same.

This method 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, we can consider the class "herbivore" to be an instance of DefinedClass. Generally speaking, if a class in OIL is not explicitly identified as a defined class, it is assumed to be primitive.

4.2.2. Class subsumption

Next, we need to translate the subclass-of statement to RDFS. This also can be done in a straightforward manner, simply by re-using existing RDFS expressiveness:

```
<rdfs:Class rdf:ID="herbivore">
  <rdf:type rdf:resource="http://www.ontoknowledge.org/oil/
        rdf-schema/2000/11/10-oil-standard#DefinedClass"/>
  <rdfs:subClassOf rdf:resource="#animal" />
</rdfs:Class>
```

However, in order to define a class as a subclass of a class expression, we would need to use the oil:subClassOf property.

4.2.3. Slot constraints

We still need to serialize the slot constraint on the class "herbivore". RDFS provides no mechanism for restricting the attributes of a class on a local level. This is 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 will introduce the oil:hasPropertyRestriction property, which is an rdf:type of rdfs:ConstraintProperty (analogous to rdfs:domain and rdfs:range). In doing so, we will be using RDFS's full potential capacity for extensibility. We will also introduce oil:PropertyRestriction as a placeholder class [5] for specific classes of slot constraints, such as has-value, value-type, cardinality, etc. These are all modeled in the OIL namespace as subclasses of oil:PropertyRestriction:

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

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

They are also similar for the other slot constraints. For the three cardinality constraints, an extra property "number" will be introduced, which will serve to assign a concrete value to the cardinality constraints.

To connect a ValueType slot constraint with its actual values, such as the property to which it refers and the class to which it restricts that property, we will introduce a pair of helper properties. These helper properties have no direct counterpart in terms of OIL primitives, but do serve to connect two classes. We will define a property oil:onProperty to connect a property restriction with the subject property, and a property oil:toClass to connect the property restriction to its class restriction.

In our example ontology, we would serialize the first part of the slot constraint using the primitives introduced above. This would proceed as follows:

```
<rdfs:Class rdf:ID="herbivore">
  <rdf:type rdf:resource="http://www.ontoknowledge.org/oil/
       rdf-schema/2000/11/10-oil-standard#DefinedClass"/>
  <rdfs:subClassOf rdf:resource="#animal"/>
  <oil:hasPropertyRestriction>
    <oil:ValueType>
      <oil:onProperty rdf:resource="#eats"/>
      <oil:toClass> </oil:toClass>
    </oil:ValueType>
  </oil:hasPropertyRestriction>
</rdfs:Class>
```

To restrict the value type of a property to a string or an integer, we could use the toConcreteType property:

```
...
  <oil:ValueType>
    <oil:onProperty
    rdf:resource="#age"/>
    <oil:toConcreteType
    rdf:resource="http://www.ontoknowledge.org/oil/
         rdf-schema/2000/11/10-oil-standard#Integer"/>
  </oil:ValueType>
...
```

4.2.4. *Boolean expressions*

The slot constraint has not been completely translated yet: the toClass element is not yet filled. Here we come across a feature of OIL that is not available in RDFS: the *boolean expression*. A boolean expression is an expression that evaluates to either a class definition or a concrete type. In the case of a class definition, such an expression is a boolean combination of classes and/or slot constraints. In the case of a concrete type definition, the expression can be a simple string or integer value, or a more complex expression (see Section 4.2.6). In the example, we have a boolean 'or' expression that evaluates to the class of all things that are plants or parts of plant.

We will introduce oil:Expression as a common placeholder, along with oil:ConcreteTypeExpression and oil:ClassExpression as specialization placeholders. However, oil:BooleanExpression will be introduced as a sibling of these two, since we want to be able to construct boolean expressions with either kind of expression. The specific boolean operators, 'and', 'or' and 'not', are introduced as subclasses. We should also note that since a single class is essentially a simple kind of class expression, rdfs:Class itself should be a subclass of oil:ClassExpression (see Fig. 4).

The 'and', 'or' and 'not' operators are connected to operands using the oil:hasOperand property. Again, this property has no direct equivalent in OIL primitive terms. Rather, it serves as a helper in connecting two class expressions, as the only way to relate two classes in the RDF data model is by means of a Property.

In our example, we need to serialize a boolean 'or'. The RDFS definition of the operator reads as follows:

```
<rdfs:Class rdf:ID="Or">
  <rdfs:subClassOf rdf:resource="#BooleanExpression"/>
</rdfs:Class>
```

The helper property is defined as follows:

```
<rdf:Property rdf:ID="hasOperand">
  <rdfs:domain rdf:resource="#BooleanExpression"/>
  <rdfs:range rdf:resource="#ClassExpression"/>
</rdf:Property>
```

The fact that hasOperand is only to be used on boolean class expressions is expressed using the rdfs:domain construction. This type of modeling stems directly from the RDF property-centric approach.

Now, we will apply what we defined above to the example:

```
<rdfs: Class rdf: ID="herbivore">
  <rdf:type rdf:resource="http://www.ontoknowledge.org/oil/
       rdf-schema/2000/11/10-oil-standard#DefinedClass"/>
  <rdfs:subClassOf rdf:resource="#animal"/>
  <oil:hasPropertyRestriction>
    <oil:ValueType>
      <oil:onProperty rdf:resource="#eats"/>
      <oil:toClass>
        <oil:Or>
          <oil:hasOperand rdf:resource="#plant"/>
          <oil:hasOperand>
            <HasValue>
              <oil:onProperty
                rdf:resource="#is-part-of"/>
              <oil:toClass rdf:resource="#plant"/>
            </HasValue>
          </oil:hasOperand>
        </oil:Or>
      </oil:toClass>
    </oil:ValueType>
  </oil:hasPropertyRestriction>
</rdfs:Class>
```

Observe that the HasValue property restriction is not related to the class by a hasPropertyRestriction property, but by a hasOperand property. This stems from the fact that the property restriction plays the role of a boolean operand here.

4.2.5. Lists of statements

Now, we will illustrate some more features by translating the second class definition, "elephant".

The first bit is trivial:

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

Next, we need to translate the OIL subsumption statement to RDFS. This statement contains a list of superclasses. In the RDFS syntax, we will model these as separate subClassOf statements:

```
<rdfs:Class rdf:ID="elephant">
  <rdfs:subClassOf rdf:resource="#mammal"/>
  <rdfs:subClassOf rdf:resource="#herbivore"/>
</rdfs:Class>
```

Next, we have two slot constraints. The first of these is a value-type restriction, and is serialized in the same manner demonstrated in the "herbivore" example:

```
<rdfs:Class rdf:ID="elephant">
  <rdfs:subClassOf rdf:resource="#mammal"/>
  <rdfs:subClassOf rdf:resource="#herbivore"/>
  <oil:hasPropertyRestriction>
    <oil:ValueType>
      <oil:onProperty rdf:resource="#eats"/>
      <oil:toClass rdf:resource="#plant"/>
    </oil:ValueType>
  </oil:hasPropertyRestriction>
</rdfs:Class>
```

4.2.6. Slot constraints to concrete types

The second slot constraint is a restriction to a particular concrete type. In OIL, a shortcut syntax for such restrictions has been introduced in the form of a "has filler" primitive. We will serialize this as we did with the other slot constraints: we will introduce a class oil:HasFiller and helper properties, oil:stringFiller and oil:integerFiller, to connect to the value:

```
<oil:HasFiller>
  <oil:onProperty rdf:resource="#colour"/>
  <oil:stringFiller>grey</oil:stringFiller>
</oil:HasFiller>
```

Unfortunately, there is no direct way in RDFS to constrain the value of a property to a particular datatype. Therefore, the range value of oil:stringFiller can not be constrained to contain only strings. We will create two subclasses of rdfs:Literal, named oil:String and oil:Integer only for the sake of clarity.

```
<rdfs:Class rdf:ID="String">
  <rdfs:comment> The subset of Literals that are strings.
  </rdfs:comment>
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdfs:Class>
```

The range of the filler properties can now be set to the appropriate class, although it is still possible to use any type of Literal. The semantics of rdfs:Literal are only that anything of this type is atomic, i.e., it will not be processed further by an RDF processor. The fact that in this case it should be a string value can only be made an informal guideline.

```
<rdf:Property ID="stringFiller">
  <rdfs:domain rdf:resource="#HasFiller"/>
  <rdfs:range rdf:resource="#String"/>
</rdf:Property>
```

Using all this, we get the following complete translation of the class "elephant":

```
<rdfs:Class rdf:ID="elephant">
  <rdfs:subClassOf rdf:resource="#mammal"/>
  <rdfs:subClassOf rdf:resource="#herbivore"/>
  <oil:hasPropertyRestriction>
    <oil:ValueType>
      <oil:onProperty rdf:resource="#eats"/>
      <oil:toClass rdf:resource="#plant"/>
    </oil:ValueType>
    <oil:HasFiller>
      <oil:onProperty rdf:resource="#colour"/>
      <oil:stringFiller>grey</oil:stringFiller>
    </oil:HasFiller>
  </oil:hasPropertyRestriction>
</rdfs:Class>
```

Note that more than one property restriction is allowed within the hasPropertyRestriction element.

4.2.7. Conclusion

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. This is to enable RDF agents that are not OIL-aware to understand as much of the specification as possible, while retaining the option of translating 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. In addition, the "subslot-of", "domain", and "range"

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
	`oil:domain`	property
range	`rdfs:range`	property
	`oil:range`	property
inverse	`oil:inverseRelationOf`	property
transitive	`oil:TransitiveProperty`	class
functional	`oil:FunctionalProperty`	class
symmetric	`oil:SymmetricProperty`	class

properties have almost direct equivalents in RDFS. Table 2 presents an overview of the OIL constructs and the corresponding RDFS constructs.

There are a few subtle differences between domain and range restrictions in OIL and their equivalents in RDFS. OIL allows multiple domain and range restrictions on a single slot. The interpretation of such a set of restrictions is the intersection of the classes in the individual statements (conjunctive semantics). In RDFS, multiple domain statements are allowed, but their interpretation is the *union* of the classes in the statements (disjunctive semantics). This limits the reasoning capabilities of RDFS drastically. [6]

Despite these semantics for domain, a Property can have at most one range restriction in RDFS. However, the current consensus within the RDF community is that the semantics of domain and range should change in the next release of RDFS. We anticipate such a change, and will interpret both multiple domain and multiple range restrictions with conjunctive semantics.

Another difference with RDFS is that OIL not only allows classes as range and domain of properties, but also class *expressions*, and, in the case of range, concrete type expressions. It is not possible to reuse rdfs:range and rdfs:domain for these sophisticated expressions, because of the conjunctive semantics of multiple range statements: we cannot extend the range of rdfs:range or rdfs:domain, we can only restrict it. In our RDFS serialization of OIL, we will, therefore, introduce two new ConstraintProperties oil:domain and oil:range. They have the same domain as their RDFS equivalent (i.e., rdf:Property), but have a broader range. For domain, class expressions are valid fillers; for range, both class expressions and concrete type expressions may be used:

```
<rdfs:ConstraintProperty rdf:ID="domain">
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/
         22-rdf-syntax-ns#Property"/>
  <rdfs:range rdf:resource="#ClassExpression"/>
</rdfs:ConstraintProperty>
<rdfs:ConstraintProperty rdf:ID="range">
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/
         22-rdf-syntax-ns#Property"/>
  <rdfs:range rdf:resource="#Expression"/>
</rdfs:ConstraintProperty>
```

[6] For example, it is never possible to derive class membership from a domain statement when union semantics are used.

When translating a slot definition, rdfs:domain and rdfs:range should be used for simple (one class) domain and range restrictions. For example:

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

will be translated into:

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

For more complicated statements, the oil:range or oil:domain properties should be used:

slot-def *age*
  domain (elephant **or** lion)
  range (**range** 0–70)

is in the RDFS representation:

```
<rdf:Property rdf:ID="age">
  <oil:domain>
    <oil:Or>
      <oil:hasOperand rdf:resource="#elephant"/>
      <oil:hasOperand rdf:resource="#lion"/>
    </oil:Or>
  </oil:domain>
  <oil:range>
    <oil:Range>
      <oil:integerValue>0</oil:integerValue>
      <oil:integerValue>70</oil:integerValue>
    </oil:Range>
  </oil:range>
</rdf:Property>
```

To specify that the range of a property is string or integer, we will use our definitions of oil:String and oil:Integer as subclasses of rdfs:Literal. For example, in stating that the range of age is integer, we could say:

```
<rdf:Property ID="age">
  <rdfs:range rdf:resource="http://www.ontoknowledge.org/
        oil/rdf-schema/2000/11/10-oil-standard#Integer">
</rdf:Property>
```

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

In light of this, we will add a property “inverseRelationOf” with “rdf:Property” as the domain and range. We also add the classes “TransitiveProperty”, “FunctionalProperty” and “SymmetricProperty” 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

reads as follows in RDFS:

```
<rdf:Property rdf:ID="has-part">
  <rdf:type rdf:resource= "http://www.ontoknowledge.org/oil/rdf-schema/
    2000/11/10-oil-standard#TransitiveProperty"/>
  <oil:inverseRelationOf rdf:resource="#is-part-of"/>
</rdf:Property>
```

In the abbreviated syntax, it reads:

```
<oil:TransitiveProperty rdf:ID="has-part">
  <oil:inverseRelationOf rdf:resource="#is-part-of"/>
</oil:TransitiveProperty>
```

This method of translating the qualities of properties features the same nice object–meta distinction (between the OIL language 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 Transitive-Property. A property can be made an instance of more than one class, and thus assigned 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 [13]. This approach makes the same distinction between language-level constructs and schema-level constructs.

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 turn, be defined as subproperties of these qualities (e.g., transitiveProperty). However, this would mix-up the use of properties at the OIL specification level as well as at the actual ontology level.

A third approach would be to model the qualities again as subproperties of rdf:Property, but to define properties in the actual ontology as instances (rdf:type) of such qualities. This approach preserves the object–meta level distinction. 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.

In our opinion, the first solution is preferable, because of the clean distinction it makes between the meta and object level.

4.4. Axioms

Axioms in OIL are factual statements about the classes in the ontology. They correspond to n-ary relations between class expressions, where n is 2 or greater.

RDF features only binary relations (properties). Therefore, we cannot simply map OIL axioms to RDF properties. Instead, we chose to model axioms as classes, with helper properties connecting them to the class expressions involved in the relation. Since axioms can be considered objects, this is a very natural approach towards modeling them in RDF (see also [13,14]). Note also that binary relations (properties) are modeled as objects in RDFS as well (i.e., any property is an instance of the class rdf:Property). We simply introduce a new primitive *alongside* rdf:Property for relations with higher arity (see Fig. 4).

We introduce a placeholder class oil:Axiom, and model specific types of axioms as subclasses:

```
<rdfs:Class ID="Disjoint">
  <rdfs:subClassOf rdf:resource="#Axiom"/>
</rdfs:Class>
```

We do the same for Equivalent.

We also introduce a property to connect the axiom object with the class expressions it relates to each other: oil:hasObject is a property connecting an axiom with an object class expression. We will illustrate this below by serializing the axiom that herbivores, omnivores and carnivores are (pairwise) disjoint:

```
<oil:Disjoint>
  <oil:hasObject rdf:resource="#herbivore"/>
  <oil:hasObject rdf:resource="#carnivore"/>
  <oil:hasObject rdf:resource="#omnivore"/>
</oil:Disjoint>
```

Since in a disjointness axiom (or an equivalence axiom) the relation between class expressions is bidirectional, we can connect all class expressions to the axiom object using the same type of property.

However, in a covering axiom (such as cover or disjoint-cover), the relation between class expressions is not bidirectional: one class expression may serve as the covering, while several others function as part of that covering.

For modeling covering axioms, we will introduce a separate placeholder class, oil:Covering, which is a subclass of oil:Axiom. The specific types of coverings available are modeled as subclasses of oil:Covering again:

```
<rdfs:Class ID="Cover">
  <rdfs:subClassOf rdf:resource="#Covering"/>
</rdfs:Class>
<rdfs:Class ID="DisjointCover">
  <rdfs:subClassOf rdf:resource="#Covering"/>
</rdfs:Class>
```

We will also introduce two additional properties: oil:hasSubject, to connect a covering axiom with its subject, and oil:isCoveredBy (a subproperty of oil:hasObject) to connect a covering axiom with the classes that cover the subject.

We will illustrate this below by serializing the axiom that the class animal is covered by carnivore, herbivore, omnivore, and mammal (i.e., every instance of animal is also an instance of at least one of the other classes).

```
<oil:Cover>
  <oil:hasSubject rdf:resource="#animal"/>
  <oil:isCoveredBy rdf:resource="#carnivore"/>
  <oil:isCoveredBy rdf:resource="#herbivore"/>
  <oil:isCoveredBy rdf:resource="#omnivore"/>
  <oil:isCoveredBy rdf:resource="#mammal"/>
</oil:Cover>
```

4.5. Restrictions to valid expressions

In the previous sections, we demonstrated how the knowledge representation constructs in OIL can be defined as an extension to RDFS. With these constructs, every OIL ontology can be fully expressed in an RDFS representation. However, it was not possible to define the extension in such a way that all schemas that follow it are also valid OIL ontologies. In other words, there are some restrictions to valid ontologies that are not expressible in the RDFS extension. [7]

First, there is a problem with data types. It cannot be enforced that instances of oil:String are really strings or that instances of oil:Integer are really integers. Consequently, it is syntactically possible to state:

```
<rdf:Property rdf:ID="weight">
  <rdf:range>
    <oil:Min>
      <oil:integerValue>nonsense</oil:integerValue>
    </oil:Min>
  </rdf:range>
</rdf:Property>
```

This is due to the fact that the RDFS specification has (intentionally) not specified any primitive data types. According to the specification, the work on data typing in XML itself should be the foundation for such a capability.

Second, the RDFS specification of OIL does not prevent the intertwining of boolean expressions of classes with boolean expressions of concrete data types. Although a statement like (dog **and** (**min** 0)) is not allowed in OIL, it is syntactically possible to state:

```
<oil:And>
  <oil:hasOperand rdf:resource="#Dog">
  <oil:hasOperand>
    <oil:Min>
      <oil:integerValue>0</oil:integerValue>
    </oil:Min>
  </oil:hasOperand>
</oil:And>
```

To prevent this kind of mixing, we could have introduced separate boolean operators for class expressions and concrete type expressions. In our opinion, however, this would have made the schema too convoluted.

Finally, another kind of problem is that the schema cannot prevent the unnecessary use of the OIL variants of standard RDFS constructs, such as oil:subClassOf, oil:range and oil:domain. Although this

[7] By "valid" we mean: not allowed by the BNF grammar of OIL. From the logical point of view, there is nothing wrong with a statement such as (dog **and** (**min** 0)); it just happens to be equivalent to the empty class.

unnecessary use does not affect the semantics of the ontology, it limits the compatibility of ontologies with plain RDFS.

5. Compatibility with RDFS

In this section, we will discuss the extent of the compatibility that we have achieved between the semantic extension (OIL), and the underlying language (RDFS).

We can distinguish three levels in all ontology languages. The first of these is the ontology language itself, such as OIL. This is the language in which to state class definitions, subclass-relations, attribute-definitions, etc. The second level consists of the ontological classes (e.g., “giraffe” or “herbivore”), their subclass relations, and their properties (e.g., “eats”). Naturally, these are expressed in the language of the first level. The third level contains the instances of the ontology, such as individual giraffes or lions that belong to classes defined at the second level.

A look at the existing W3C RDF/RDFS recommendation would reveal the following about these levels:

(1) The ontology language is, of course, RDFS;
(2) Specific classes, their properties and relations are, therefore, written in RDFS, e.g.:
```
<rdfs:Class rdf:ID="herbivore">
  <rdfs:subClassOf rdf:resource="#animal">
  </rdfs:Class>
<rdf:Property rdf:ID="eats"/>
```
(3) Instances are written in RDF (note: *not* RDFS), e.g.:
```
<rdf:Description about="http://www.cs.vu.nl/~frankh">
  <rdf:type rdf:resource="#herbivore"/>
</rdf:Description>
```

If we were to examine a semantic extension of RDFS such as OIL, we would find the following:

(1) The ontology language is OIL, but it is important to realize that OIL includes RDFS as a sublanguage.
(2) As a result, class expressions written in OIL are actually also legal RDFS. For example, besides being a meaningful OIL definition, the class definition of “herbivore” in item 2 above is also a legal example of an RDFS definition. Of course, since OIL is an *extension* of RDFS, not all parts of an OIL definition are *meaningful* RDFS. This is illustrated below.

```
<rdfs:Class rdf:ID="herbivore">
  <rdfs:subClassOf rdf:resource="#animal"/>
  <oil:hasPropertyRestriction>
    <oil:ValueType>
      <oil:onProperty rdf:resource="#eats"/>
      <oil:toClass>
    <oil:Or>
      <oil:hasOperand rdf:resource="#plant"/>
      <oil:hasOperand>
        <oil:HasValue>
          <oil:onProperty
            rdf:resource="#is-part-of"/>
          <oil:toClass rdf:resource="#plant"/>
          </oil:HasValue>
        </oil:hasOperand>
      </oil:Or>
        </oil:toClass>
      </oil:ValueType>
    </oil:hasPropertyRestriction>
  </rdfs:Class>
```

Note that the semantics of the hasPropertyRestriction statement would be impossible for an RDFS processor to interpret. The entire state is legal RDF syntax, so it can be parsed, but the intended semantics of the property restriction itself can only be understood by an OIL-aware application. Notice that the first subClassOf statement is still fully interpretable even by an OIL-unaware RDFS processor.

(3) OIL instances are written as RDF! This is an important consequence of the fact that the second level is organized as an extension of RDFS.

The above shows that we have now achieved two important compatibility results: first, OIL is *backwardly compatible* with RDFS, i.e., every RDFS specification is also a valid OIL ontology declaration. Secondly, we have achieved *partial forward compatibility*. This means that even if an ontology is written in the richer modeling language (OIL), a processor for the simpler ontology language (RDFS) can still:

(a) fully interpret all the instance information of the ontology, and
(b) partially interpret the class-structure of the ontology. This can be achieved by simply ignoring any statement not from the RDF or RDFS namespaces. (In our example, these came from the oil namespace). In the above definition of "herbivore", for instance, an RDFS processor would interpret the statement simply as asserting that herbivores are a subclass of animals, and that they have some other property that it cannot interpret. This is a correct, albeit partial, interpretation of the definition.

Such partial interpretability of semantically rich meta data by semantically poor processing agents is a crucial step towards sharing meta data on the Semantic Web. We cannot realistically hope that all of the Semantic Web will be built on a single standard for semantically rich meta data. The above shows that multiple semantic modeling languages do not have to lead to meta data that are completely impossible for others to interpret. Instead, simpler processors can still pick up as much of the meta data from rich processors as they can " understand". They can safely ignore the rest in the knowledge that their partial interpretation is still correct with respect to the original intention of the meta data.

6. Problems with RDFS

The previous section explained how it is possible to define a more expressive knowledge representation language as an extension to RDFS, effectively implementing the "third layer of the Semantic Web". However, 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 we 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 that we think would be highly undesirable. This is because they are really semantic rather than syntactic constraints (i.e., they limit the kinds of models that can be represented), even if the reasoning required to detect constraint violation is very basic.

Secondly, in contrast to RDFS, OIL allows more than one range restriction on a property (as already discussed in Section 5). 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 perspective, allowing more than one range restriction would be 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. [9]). 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 relation consistently as an object–meta relation.

Moreover, 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. The meaning of the subPropertyOf relation with respect to the inheritance of the domain and range restrictions is also unclear.

7. Related work

Work on ontology representation languages dates back to the work on frame languages in the early days of AI. However, efforts towards designing ontology representation languages that are Web-enabled only date from recent years. The most prominent (or even: the only) efforts in this area have been SHOE [15,16], Ontobroker [17], OIL and DAML-ONT (Initial Release, http://www.daml.org/2000/10/daml-ont.html), and more recently, as a replacement for DAML-ONT, DAML+OIL (http://www.daml.org/2001/03/daml+oil-index.html).

Of these, only the last three have been defined on top of RDF and RDFS. Since DAML+OIL is essentially a merger between OIL and DAML-ONT, we will focus on comparing our own proposal (OIL) to DAML-ONT.

DAML-ONT shares a principle with our own proposal: an ontology language should maintain maximum backwards compatibility with existing Web standard languages, in particular RDFS. The difference between OIL and DAML-ONT lies in the degree to which the languages succeed in maximizing the ontological content that can be understood by an "RDFS agent". (These "agents" are applications that understand RDFS but do not recognize the language specific extensions, OIL or DAML-ONT). Unlike OIL, DAML-ONT is built on top of RDFS in a way that allows little, if any, ontology content to be understood by an RDFS agent. OIL, for example, uses the RDFS subClassOf property to state simple subclass relations between classes:

```
<rdfs:Class ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</rdfs:Class>
```

This part of OIL ontologies is, therefore, accessible to any RDFS agent. In contrast, DAML-ONT uses its own locally defined "subClassOf" property, for example:

```
<daml:Class ID="Male">
  <daml:subClassOf resource="#Animal"/>
</daml:Class>
```

The DAML-ONT subClassOf property is then defined to be "equivalentTo" rdfs:subClassOf. However, the definition of "daml:equivalentTo" itself relies cyclically on the definition of daml:sub-PropertyOf. Thus, even simple subclass relationships in a DAML ontology are inaccessible to an RDFS agent. The situation is even worse when it comes to more complex class definitions. For example, the definition of the class "TallMan", which is the intersection of the classes "Man" and "TallThing", is expressed in DAML-ONT as:

```
<daml:Class ID=" TallMan" >
  <daml:intersectionOf parseType=" daml:collection" >
    <daml:Class about=" '#TallThing" />
    <daml:Class about=" #Man" />
  </daml:intersectionOf>
</daml:Class>
```

This is completely opaque to an RDFS agent as it will not understand the semantics of "daml:intersectionOf". In OIL, the definition of TallMan would rely on the fact that intersection is implicit in the semantics of rdfs:subClassOf:

```
<rdfs:Class ID=" TallMan" >
  <rdf:type rdf:resource=" http://www.ontoknowledge.org/oil/
         rdf-schema/2000/11/10-oil-standard#DefinedClass" />
  <rdfs:subClassOf rdf:resource="#TallThing" />
  <rdfs:subClassOf rdf:resource="#Man" />
</rdfs:Class>
```

This, in turn, would make the subclass relations accessible to any RDFS agent. In conclusion, we would argue that:

- OIL and DAML-ONT are currently the only two Web-based ontology languages that are built on top of RDFS; and that
- of the two, OIL achieves a much larger degree of "backwards compatibility" with RDF.

As mentioned earlier, DAML+OIL is a proposal for an ontology language that merges the ideas incorporated in DAML-ONT with those in OIL. Specifically, many of the ideas, presented in this article, on how to represent a KR language in RDFS have been adopted by DAML+OIL. In effect, DAML+OIL is "backwardly compatible" with RDF to a much larger degree than the initial DAML-ONT language is.

8. Conclusion

In this article, we have demonstrated why a machine-accessible representation of the information available on the Web is both useful and necessary. We have also shown that RDFS is only a small step towards the expressiveness required for the Semantic Web. Finally, we have illustrated how RDFS can still be used to this end, by extending it with additional modeling primitives as defined by a more formal knowledge representation scheme, such as OIL.

An important advantage to our approach is that it maximizes the compatibility with RDFS. Not only is every RDFS document a valid OIL ontology declaration, every OIL ontology can also be partially interpreted by a semantically poorer processing agent. Needless to say, this partial interpretation is incomplete. All the same, it is correct under the intended semantics of the ontology. We firmly believe that our extension method is generally applicable across knowledge representation formalisms.

Acknowledgements

We would like to thank Sean Bechhofer, Monica Crubezy, Michael Erdmann, and Arjohn Kampman for their helpful comments and for reviewing early drafts of this paper.

References

[1] D. Fensel, Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer, Berlin, 2000.
[2] T.R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (2).
[3] D. Brickley, R. Guha, 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, March 2000.
[4] T. Berners-Lee, Semantic web road map, Internal note, World Wide Web Consortium, see http://www.w3.org/DesignIssues/Semantic.html, September 1998.
[5] O. Lassila, R.R. Swick, Resource Description Framework (RDF): Modeland Syntax Specification, Recommendation, World Wide Web Consortium, see http://www.w3.org/TR/REC-rdf-syntax/, February 1999.
[6] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, M. Klein, OIL in a nutshell, in: R. Dieng, O. Corby (Eds.), Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000, Juan-les-Pins, France, Lecture Notes in Artificial Intelligence, Vol. 1937, Springer, Berlin, 2000, pp. 1–16.
[7] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, E. Motta, OIL: The Ontology Inference Layer, Tech. Rep. IR-479, Vrije Universiteit Amsterdam, Faculty of Sciences, see http://www.ontoknowledge.org/oil/, September 2000.
[8] D. Fensel, F. van Harmelen, M. Klein, H. Akkermans, J. Broekstra, C. Fluit, J. van der Meer, H.-P. Schnurr, R. Studer, J. Hughes, U. Krohn, J. Davies, R. Engels, B. Bremdal, F. Ygge, T. Lau, B. Novotny, U. Reimer, I. Horrocks, On-to-knowledge: Ontology-based tools for knowledge management, in: Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference, Madrid, Spain, 2000.
[9] W. Nejdl, M. Wolpers, C. Capella, The RDF schema revisited, in: Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik: Modellierung 2000 St. Goar, Foelbach Verlag, Koblenz, 2000.
[10] S. Bechhofer, I. Horrocks, P.F. Patel-Schneider, S. Tessaris, A proposal for adescription logic interface, in: Proceedings of DL'99, 1999, pp. 33–36.
[11] H. Stuckenschmidt, Using OIL for intelligent information integration, in: V. Benjamins, A. Gomez-Perez, N. Guarino (Eds.), Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI 2000, Berlin, Germany, 2000.
[12] M. Klein, D. Fensel, F. van Harmelen, I. Horrocks, The relation between ontologies and schema-languages: Translating OIL-specifications in XML-schema, in: V.R. Benjamins, A. Gomez-Perez, N. Guarino (Eds.), Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence, ECAI 2000, Berlin, Germany, 2000.
[13] S. Staab, M. Erdmann, A. Mädche, S. Decker, 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, 2000.
[14] S. Staab, A. Mädche, Axioms are objects, too - ontology engineering beyond the modeling of concepts and relations, in: V. Benjamins, A. Gomez-Perez, N. Guarino (Eds.), Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence, ECAI 2000, Berlin, Germany, 2000.
[15] S. Luke, L. Spector, D. Rager, Ontology-based knowledge discovery on the world-wide web, in: A. Franz, H. Kitano (Eds.), Working Notes of the Workshop on Internet-Based Information Systems at the 13th National Conference on Artificial Intelligence (AAAI96), AAAI Press, 1996, pp. 96–102.
[16] J. Heflin, J. Hendler, Dynamic ontologies on the web, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), AAAI/MIT Press, Menlo Park, CA, 2000, pp. 443–449.

[17] D. Fensel, S. Decker, M. Erdmann, R. Studer, Ontobroker: The very high idea, in: Proceedings of the 11th International Flairs Conference (FLAIRS-98), Sanibal Island, Florida, 1998.

Jeen Broekstra is a Ph.D. student at the Knowledge Representation & Reasoning group of the Vrije Universiteit Amsterdam, and a Senior Software Developer for Aidministrator Nederland b.v. He is one of the leading designers and implementors of Sesame, the first ever implementation of a storage and retrieval engine for RDFS, the first ontology language standardised by W3C. Sesame is currently attracting international attention. He is one of the designers of OIL'S RDF syntax, ensuring full compatibility between various W3C standards and candidate standards. He has also written influentially on the language design of both RDFS (in particular its meta model) and DAML+OIL (in particular its design to be partially processable by RDFS only agents). Contact him at Aidministrator Nederland b.v., Julianaplein 14b, 3817 CS Amersfoort, NL; jeen.broekstra@aidministrator.nl; http://www.cs.vu.nl/~jbroeks/.

Michel Klein is a Ph.D. student in the Business Informatics group at the Vrije Universiteit in Amsterdam, The Netherlands. He received his Master's degree from the same university in October 1996. After finishing his master, he worked for two and a half year as a software engineer at the Leiden University Medical Center on a system for the classification and presentation of multi-media medical information. His research interests include the use of ontologies for information integration, representational issues of ontologies, and support for the dynamic aspects of knowledge representation on the web. You may contact him at the Vrije Universiteit, division Math & CS, De Boelelaan 1081a, 1081 HV, Amsterdam, NL; Email: Michel.Klein@cs.vu.nl; Homepage: http://www.cs.vu.nl/~mcaklein/.

Stefan Decker is a Postdoctoral Fellow at the Computer Science Department at Stanford University and the cofounder of Ontoprise, a startup company focusing on ontology-based knowledge management. He consults frequently on RDF, XML, and interoperability issues.

Dieter Fensel is an Associated Professor at the Vrije Universiteit Amsterdam in the area of business informatics. After studying Mathematics, Sociology and Computer Science in Berlin, he joined in 1989 the Institute AIFB at the University of Karlsruhe. His major subject was knowledge engineering and his Ph.D. thesis in 1993 was about a formal specification language for knowledge-based systems. Currently, his focus is on the use of Ontologies to mediate access to heterogeneous knowledge sources and to apply them in knowledge management and electronic commerce. He has published around 150 papers as journal, book, conference, and workshop contributions. He organized around 100 scientific workshops and conferences and has edited several special issues of scientific journals. He is Associated Editor of Knowledge and Information Systems: An International Journal (KAIS), Springer-Verlag, and IEEE Intelligent Systems. He is involved in several national and internal research projects, for example, in the running 1ST projects IBROW and On-to-Knowledge, and Ontoweb. Dieter Fensel is the author of several books, for example, Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, and Problem-Solving Methods: Understanding, Development, Description, and Reuse.

Frank van Harmelen (1960) studied Mathematics and Computer Science in Amsterdam. In 1989, he was awarded a Ph.D. from the Department of AI in Edinburgh for his research on meta level reasoning. After his Ph.D. research, he moved back to Amsterdam where he was involved in the REFLECT project on the use of reflection in expert systems, and in the KADS project, where he contributed to the development of the (ML)2 language for formally specifying Knowledge-Based Systems. In 1995 he joined the AI research group at the Vrije Universiteit Amsterdam, where he is heading the Knowledge Representation and Reasoning group. He was one of the designers of OIL, which in its incarnation of DAML+OIL is the basis for the activities of the W3C working group on a Web Ontology language. He is author of a book on meta level inference, editor of a book on knowledge-based systems, and has published over 60 research papers.

Ian Horrocks is a Lecturer in Computer Science with the Information Management Group at the University of Manchester, UK. He graduated in Computer Science from Manchester in 1982, and after working in industry he returned to complete a Ph.D. in 1997. His research interests include knowledge representation, automated reasoning, optimising reasoning systems and ontological engineering, with particular emphasis on the application of these techniques to the World Wide Web. His work on the FaCT system, a highly optimised DL reasoner, has revolutionised the design of DL systems, redefining the notion of tractability for DLs and establishing a new standard for DL implementations. He is a member of the OIL language steering committee, of the Joint EU/US Committee on Agent Markup Languages, and is co-editor of the DAML+OIL language specification. Contact him at the Dept. of Computer Science, Univ. of Manchester, Oxford Road, Manchester, M13 9PL, UK; horrocks@cs.man.ac.uk; www.cs.man.ac.uk//~horrocks.