Enabling knowledge representation on the Web by extending RDF Schema

Jeen Broekstra,[†] Michel Klein,[‡]
Stefan Decker,[⋆] Dieter Fensel,[‡] Frank van Harmelen,[‡]
and Ian Horrocks[+]

[†]AIdministrator Nederland B.V.
jeen.broekstra@aidministrator.nl

[‡]Vrije Universiteit Amsterdam, Holland
{michel.klein|dieter|frankh}@cs.vu.nl

[⋆]Department of Computer Science, Stanford University, Stanford, USA
stefan@db.stanford.edu

[+]Department of Computer Science
University of Manchester, UK
horrocks@cs.man.ac.uk

10th November 2000

Abstract

Recently, there has been a wide interest in using ontologies on the Web. As a basis for this, RDF Schema (RDFS) provides means to define vocabulary, structure and constraints for expressing metadata about Web resources. However, formal semantics are not provided, and the expressivity of it is not enough for full-fledged ontological modeling and reasoning. In this paper, we will show how RDFS can be extended in such a way that a full knowledge representation (KR) 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 RDFS. An important advantage of our approach is a maximal backward compatability with RDFS: any meta-data in OIL format can still be partially interpreted by any RDFS-only-processor. The OIL extension of RDFS has been carefully engineered so that such a partial interpretation of OIL meta-data is still correct under the intended semantics of RDFS: simply ignoring the OIL specific portions of an OIL document yields a correct RDF(S) document whose intended RDFS semantics is precisely a subset of the semantics of the full OIL statements. In this way, our approach ensures maximal sharing of meta-data on the Web: even partial interpretation of meta-data by less semantically aware processors will yield a correct partial interpretation of the meta-data. We conclude that our method of extending is equally applicable to other KR formalisms.

1 Introduction

Currently, computers are changing from single isolated devices into entry points into a worldwide network of information exchange and business transactions (cf. [4]). 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. Recently, the notion of ontology has become widespread in fields such as intelligent information integration, cooperative information systems, information retrieval, electronic commerce, and knowledge management. The reason ontologies are becoming so popular is in large part due to what they promise: a shared and common understanding of some domain that can be communicated between people and application systems. Because ontologies aim at consensual domain knowledge, their development is often a cooperative process involving different people, possibly at different locations.

Many definitions of ontologies have been given in the last decade, but one that, in our opinion, best characterizes the essence of an ontology is based on the related definitions by [8]: An ontology is a formal, explicit specification of a shared conceptualisation. A *conceptualisation* 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 the fact that the ontology should be machine understandable, i.e. the machine should be able to interpret the semantics of the information provided. *Shared* reflects the notion that an ontology captures consensual knowledge, that is, it is not restricted to some individual, but accepted by a group.

Using ontologies, semantic annotations on Web resources will allow structural and semantic definitions of documents, providing completely new possibilities: intelligent search instead of keyword matching, query answering instead of information retrieval, document exchange between departments via ontology mappings, and definition of views on documents.

RDF Schema [3] 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* [2].

At the lowest level of the Semantic Web a generic mechanism for expressing machine readable semantics of data is required. The Resource Description Framework (RDF) [12] 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 [3]). 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 [6, 10] as an extension of RDF Schema.

OIL (Ontology Inference Layer), a major spin-off from the IST project On-To-Knowledge[1][7], 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, 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. The result is an RDF Schema definition of OIL primitives, which allows one to express any OIL ontology in RDF syntax. This enables the added benefits of OIL, such as reasoning support and formal semantics, to be used on the Web, while retaining maximal backward compatability with 'pure' RDF. Finally, we provide some conclusions and recommendations in section 6.

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) [12] 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.

[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/`

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. 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 publisher (the predicate) the W3C (the object). Since both the subject and the object of a statement can be resources, 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) is created by staff member no. 85740 (the object). In the next statement, this same resource (staff member 85740) plays the role of subject to state this his email address is lassila@w3.org. Finally, RDF statements are also resources, so that statements can be applied recursively to statements, allowing their nesting.

All this leads to the underlying datamodel being a labelled hyper-graph, with each statement being a predicate-labelled link between object and subject. The graph is a hyper-graph since each node can itself again contain an entire graph.

2.2 Introduction to RDF Schema

The modeling primitives offered by RDF are very basic[2]. Therefore, the RDF Schema specification [3] defines further modeling primitives in RDF. That is, RDF Schema extends (or: enriches) RDF by giving an externally specified semantics to specific resources. e.g., to rdfs:subclassOf, to rdfs:Class etc. It is only because of this external semantics that RDF Schema is useful. Moreover, this semantics cannot be captured in RDF - if it could then there would be no need for RDFS. OIL stands in a similar relationship to RDFS - by defining a semantics for specific resources we further extend (or: enrich) RDF Schema. This allows OIL to capture meaning that cannot be captured in RDFS, and this is where the added value is. Furthermore, we will be careful to create this extension to RDF Schema in such a way that a partial interpretation without the additional OIL semantics will still yield a valid RDF Schema interpretation.

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

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 1 pictures the subclass-of hierarchy of RDFS and figure 2 pictures 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 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. [14].

2.2.2 The modeling primitives of RDF Schema

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

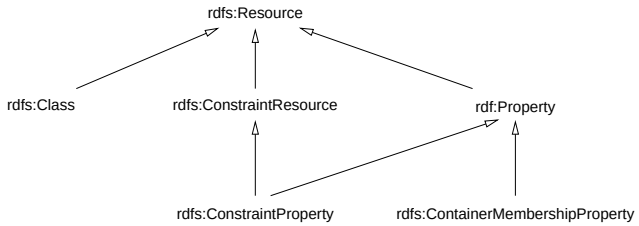

Figure 1: The subclass-of hierarchy of modeling primitives in RDFS.

- **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 rdfs:Resource. The class rdf:Property is the class of all properties used to characterize instances of rdfs:Resource, i.e., each slot /

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

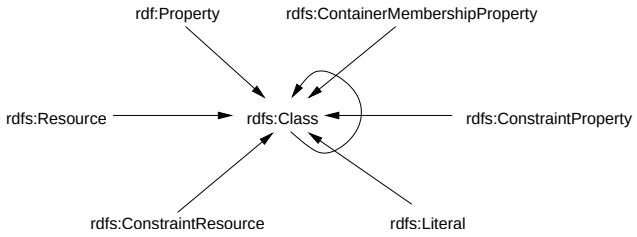

Figure 2: The instance-of relationships of modeling primitives in RDFS.

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 [10]. A small example ontology in OIL is provided below.

[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.

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://www.ontoknowledge.org/oil/oil-rdfs.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
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 mammmal
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

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 [1], a DL reasoner developed at the University of Manchester, can be (and has been) used to this end [17].

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, slot and axiom 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.

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. As another example, (**equal** "xyz") defines the data-type consisting of the string "xyz".

A **slot-constraint** (or property restriction) 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 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.

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

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

- **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 oriented towards XML and RDF. [10] defines a DTD and a XML schema definition for OIL. [11] 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. 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. 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#"
         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.

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 3).We conform to the informal RDF guideline to start property names with a lower-case letter, and class names with a capital.

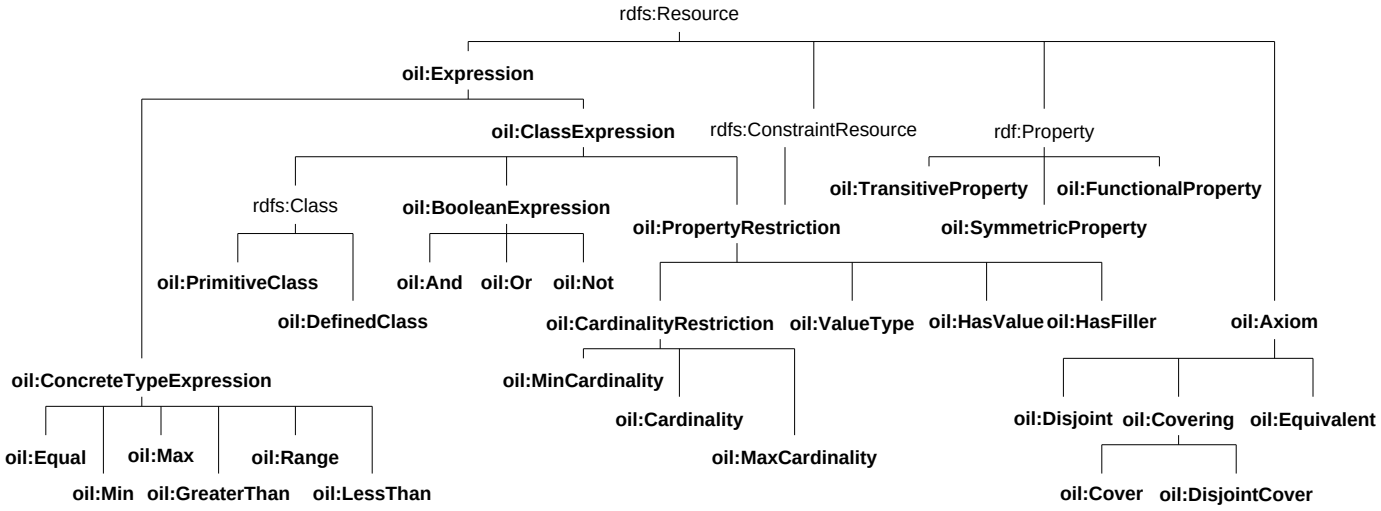

Figure 3: The OIL extensions to RDFS in the subsumption hierarchy.

To illustrate the use of these extensions, we will walk through them by means of some 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))
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 straightforward manner, using the rdfs:Class construct and the rdf:ID property to assign a name:

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

From this definition it is not yet clear that this class is a defined class. We chose to introduce two extra classes in the OIL namespace, named PrimitiveClass and DefinedClass. In a particular class definition, we can use one of these two ways to express that a class is a defined class:

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

or:

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

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

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 OIL in general, if it is not explicitly stated that a class is defined, the class is assumed to be primitive.

4.2.2 Class Subsumption

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

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

However, if one want to define a class as a subclass of a class *expression*, one should use the oil:subClassOf property.

4.2.3 Slot Constraints

We still need to serialize the slot constraint on the class ”carnivore”. In RDFS, there is 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 introduce the oil:hasPropertyRestriction 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:PropertyRestriction 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 the OIL namespace as subclasses of oil:PropertyRestriction:

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

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

In our example ontology, the first part of the slot constraint would be serialized using the primitives introduced above as follows:

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

4.2.4 Class 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 *class expression*. A class expression is an expression that evaluates to a class definition. Such an expression can be a simple class name, or it can be a boolean expression of classes and/or slot constraints. In the example, we have a boolean ’or’ expression that evaluates to the class of all things that are plants or that are parts of plant.

We introduce oil:ClassExpression as a placeholder class[6]. A second placeholder class, oil:BooleanExpression, is introduced as a subclass of ClassExpression to hold the operators ‘and’, ‘or’ and ‘not’ as subclasses.

[6] 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.

Also, since a single class is a essentially a simple kind of class-expression, rdfs:Class itself should be a subclass of oil:ClassExpression.

The ‘and’, ‘or’ and ‘not’ operators are connected to operands using the oil:hasOperand property. This property again 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.

In our example, we need to serialize a boolean or. The RDF Schema definition of the operator looks like this:

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

and 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 apply what we defined above to the example:

```
<rdfs:Class rdf:ID="herbivore">
  <rdf:type rdf:resource="http://www.ontoknowledge.org/oil/rdfs-schema/#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 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. In this statement, a list of superclasses is given. In the RDFS syntax, we model these as seperate 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 it is serialized in the same manner as we showed 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 serialize this like we do with the other slot constraints: we 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>
```

IN RDF(S), there is unfortunately no direct way 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. The best we can do is constrain it to rdfs:Literal:

```
<rdf:Property ID="stringFiller">
  <rdfs:domain rdf:resource="#HasFiller"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

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.

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>
```

Observe that it is allowed to have more than one property restriction within the hasPropertyRestriction element.

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

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, 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 are a few subtle differences between domain and range restrictions in OIL and their equivalents in RDFS. First, the specification of OIL is very clear on multiple domain and range restriction: these are allowed, and the semantic is the *intersection* of 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. Despite these semantics for domain, a Property can have at most one range restriction in RDFS. However, according to discussions on the `rdf-interest` mailinglist the semantics of domain and range will very likely change in the next release of RDFS. We already anticipated on such a change, and interpret both multiple domain and multiple range restrictions with conjunctive semantics.

Secondly, in contrast to RDFS, OIL not only allows classes as range and domain of properties, but also class-*expressions*, and – for 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. If we would add an new range statement on rdfs:range, we would thereby restrict the range to the intersection of the existing and new range statement.

Therefore, we introduced 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>
```

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>
```

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 ”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

is in RDFS:

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

or, in the abbreviated syntax:

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

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 TransitiveProperty. 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 [15]. 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
	`oil:domain`	property
range	`rdfs:range`	property
	`oil:range`	property
inverse	`oil:inverseRelationOf`	property
transitive	`oil:TransitiveProperty`	class
transitive	`oil:FunctionalProperty`	class
symmetric	`oil:SymmetricProperty`	class

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 only knows 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 [16, 15]). Observe 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 figure 3).

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>
```

and likewise 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. For example, to serialize 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 the 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 (like cover or disjoint-cover), the relation between class expressions is not bidirectional: one class expression plays the role of covering, several other class expressions play the role of being part of that covering.

For modeling covering axioms, we introduce a seperate 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>
```

Furthermore, two additional properties are introduced: oil:hasSubject, to connect a covering axiom with its subject, and oil:isCoveredBy, which is a subproperty of oil:hasObject, to connect a covering axiom with the classes that cover the subject.

For example, we serialize 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>
```

5 Backward compatability with RDF Schema

In this section we will discuss the backward compatability that we have achieved between the semantic extension (OIL), and the underlying language (RDF Schema).

As for any ontology language, we can distinguish three levels: [1] the ontology language, the language in which to state for example class-definitions, subclass-relations, attribute-definitions etc. In our case, RDF Schema and OIL. [2] the ontological classes, for example the classes ”giraffe” or ”herbivore”, their subclass relationships, and their properties (such as eats). These are of course expressed in the language of level [1] [3] the instances of the ontology, such as individual giraffes or lions that belong to classes defined at level [2].

If we look at the existing W3C RDF/RDF Schema recommendation, these levels have the following form:

1. the ontology language is of course RDF Schema
2. specific classes, their properties and relations are therefore written in RDF Schema, eg:

   ```
   <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* RDF Schema), eg:

   ```
   <rdf:Description about="http://www.cs.vu.nl/~frankh">
     <rdf:type resource="#herbivore"/>
   </rdf:Description>
   ```

If we consider a semantic extension of RDF Schema such as OIL, the situation is as follows:

1. the ontology language is OIL, but it is important to realise that OIL includes significant parts of RDF Schema as a sublanguage
2. as a result, some class expressions (written in OIL) are actually also legal RDF Schema. For example, besides being legal RDF Schema, the class definition of ”herbivore” in item 2 above is also a legal example of an OIL class-definition. Of course, since OIL is an *extension* of RDF Schema, not all OIL definitions are interpretable as RDF Schema definitions. For example, in

```
<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>
```

the semantics of the hasPropertyRestriction statement will not be interpretable by an RDF Schema processor. 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 RDF Schema processor.

3. OIL instances are written as RDF! This is an important consequence of the fact that level [2] is organised as an extension of RDF Schema.

The above shows that we have now achieved an important compatability result: even if an ontology is written in the richer modelling language (OIL), a processor for the simpler ontology language (RDF Schema) 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 those from the oil namespace). For example, in the above definition of ”herbivore”, an RDF Schema processor will interpret this statement simply as stating that herbivores are a subclass of animals, and that they 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 is a crucial step towards the sharing of meta-data on the Semantic Web. We cannot realistically hope that all of the Semantic Web will be build on a single standard for semantically rich meta-data. The above shows that multiple semantic modelling languages do not have to lead to meta-data that are totally uninterpretable by others. Instead, simpler processors can still pick up as much of the meta-data from rich processors as they can ”understand”, and safely ignore the rest in the knowledge that their partial interpretation is still a correct with respect to the original intention of the meta-data.

6 Uncovered problems with RDF Schema

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”. 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 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. [14]). 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.

7 Related work

Work on ontology representation languages dates back to the work on frame-languages in the early days of AI. However, efforts of 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 [13, 9], Ontobroker [5], and more recently OIL and DAML-ONT [7]. Of these, only the last two have been defined on top of RDF(S). We can therefore focus the comparison of our own proposal (OIL) to DAML-ONT.

DAML-ONT shares with our own proposal the principle that an ontology language should maintain maximum backwards compatibility with existing web standard languages, and in particular RDF

[7] DAML-ONT Initial Release, `http://www.daml.org/2000/10/daml-ont.html`

Schema. The difference between OIL and DAML-ONT lies in the degree to which the languages succeed in maximising the ontological content that can be understood by an "RDF Schema agent" (ie. an application that understands RDF Schema but does not recognise 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. In OIL, for example, stating simple subclass relationships between classes is done using the RDFS subClassOf property:

```
<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, but the definition of "daml:equivalentTo" itself relies cyclicly on the definition of daml:sub-PropertyOf. Therefore 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" 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/rdfs-schema/#DefinedClass"/>
  <rdfs:subClassOf rdf:resource="#TallThing"/>
  <rdfs:subClassOf rdf:resource="#Man"/>
</rdfs:Class>
```

making the sub-class relationships accessible to any RDF Schema agent. In conclusion, we argue that:

- OIL and DAML-ONT are currently the only two web-based ontology languages that are built on top of RDF;
- of these, OIL achieves a much larger degree of "backward compatability" with RDF.

8 Conclusion

In this article, we have shown 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 required expressiveness for the Semantic Web.

Finally, we have shown how RDFS still can 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 of our approach is the maximization of the backward compatibility with RDFS. We firmly believe that our way of extending is generally applicable across knowledge representation formalisms.

Acknowledgements

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

References

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

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

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

[4] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, Berlin, 2000.

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

[6] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In R. Dieng and O. Corby, editors, *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, Juan-les-Pins, French Riviera, Oct. 2000. Springer-Verlag.

[7] D. Fensel, F. van Harmelen, H. Akkermans, M. Klein, J. Broekstra, C. Fluit, J. van der Meer, H. Schnurr, R. Studer, J. Davies, J. Hughes, U. Krohn, R. Engels, B. Bremdahl, F. Ygge,

U. Reimer, and I. Horrocks. OnToKnowledge: Ontology-based Tools for Knowledge Management. In *Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference*, Madrid, Spain, Oct. 2000.

[8] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 1993.

[9] J. Heflin and J. Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 443–449. AAAI/MIT Press, Menlo Park, CA, 2000.

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

[11] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. 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*, Aug. 2000.

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

[13] S. Luke, L. Spector, and D. Rager. Ontology-based knowledge discovery on the world-wide web. In A. Franz and H. Kitano, editors, *Working Notes of the Workshop on Internet-Based Information Systems at the 13th National Conference on Artificial Intelligence (AAAI96)*, pages 96–102. AAAI Press, 1996.

[14] W. Nejdl, M. Wolpers, and C. Capella. The RDF Schema Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000, St. Goar*. Foelbach Verlag, Koblenz, Apr. 2000.

[15] S. Staab, M. Erdmann, A. Mädche, and 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*, Sept.18-20 2000.

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

[17] H. Stuckenschmidt. 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*, Aug. 2000.