RDFS(FA) and RDF MT: Two Semantics for RDFS

Jeff Z. Pan and Ian Horrocks

Department of Computer Science,
University of Manchester, UK M13 9PL
{pan,horrocks}@cs.man.ac.uk

Abstract. RDF Schema (RDFS) has a non-standard metamodeling architecture, which makes some elements in the model have dual roles in the RDFS specification. As a result, this can be confusing and difficult to understand and, more importantly, the specification of its semantics requires a non-standard model theory. This leads to semantic problems when trying to layer conventional first order languages, like DAML+OIL, on top of RDFS. In this paper we will first demonstrate how this problem with RDFS can be solved in a sub-language of RDFS - RDFS(FA), which introduces a Fixed layer metamodeling Architecture to RDFS, based on a (relatively) standard model-theoretic semantics. Logical layer Semantic Web languages such as DAML+OIL and OWL can, therefore, be built on top of both the syntax and semantics of RDFS(FA). We will also compare this approach with the existing RDF Model Theory and discuss the advantages and disadvantages of the two approaches.

1 Introduction

The Semantic Web [1] is a vision of the next generation Web, in which the current rendering markup, which specifies how to display Web resources for human consumption, will be enhanced with semantic markups (often called annotations), which will specify the meaning of Web resources so as to make them more accessible to automatic processes. Ontologies [16] will play an important role in the Semantic Web as a source of precisely defined important terms and properties in the domain, which can then be used in annotations, for communication.

There is a functional architecture [1,12] of semantic Web languages. On the bottom, XML (eXtensible Markup Language) [2] is used as syntax. On top of XML, RDF (Resource Description Framework) [10] is a simple metadata language, which provides a simple and general model of semantic assertions of the Web. E.g., RDF can be used to add annotations to Web resources. On top of RDF, RDF Schema (RDFS) [3] is a schema language (as well as a very simple Web ontology language), which provides facilities to define terms used in annotations. More powerful (logical layer) ontology languages, e.g. OIL [6,7], DAML-ONT [9], DAML+OIL [18] and OWL [5], are expected to stand on top of RDFS and supply a richer set of modelling primitives. Unfortunately, the relationships between adjacent layers are not specified, especially that between RDFS and more powerful ontology languages, e.g. DAML+OIL and OWL.

Initially RDF and RDFS had no formal model theory, nor any formal meaning at all. This made them unlikely foundations for the Semantic Web. As earlier works [11,4]

pointed out, RDFS has a non-standard and non-fixed layer metamodeling architecture, which makes some elements in the model have dual roles in the RDFS specification. In other words, multiple modelling primitives seems to be implicitly represented by a single RDFS primitive (see Section 2 for more details). Therefore, it makes the RDFS specification itself kind of confusing and difficult to understand for the modelers. One of the consequences is that when DAML+OIL is layering on top of RDFS, it uses the syntax of RDFS only, but defines its own semantics [17] for the ontological primitives of RDFS.

In order to clear up any confusion, [12] proposed a sub-language of RDFS - RDFS(FA)[1], which provides a Fixed layer metamodeling Architecture for RDFS. The implicitly represented modelling primitives in RDFS are explicitly stratified into different strata (layers) of RDFS(FA). In this way RDFS(FA) has clear semantics and there are no dual roles in RDFS(FA). Subsequently RDF Model Theory (RDF MT) [8] gave an official semantics for RDF and RDFS, justifying the dual roles by treating both classes and properties as objects in the universe. So RDF MT is another approach to clear up the kinds of confusion that can arise in RDFS.

In the remainder of this paper, we will first illustrate the kinds of confusion that can arise in RDFS (Section 2). We will then present the design philosophy and stratification of RDFS(FA), which were not covered by [12], and describe how RDFS(FA) clears up any possible confusion of RDFS (Section 3). For the purpose of comparison, we will also explain how RDF MT formalises RDFS (Section 4) and then compare the advantages and disadvantages of these two approaches (Section 5). Finally we will discuss what conclusions we can draw from the above comparison in Section 6.

2 RDFS Architecture

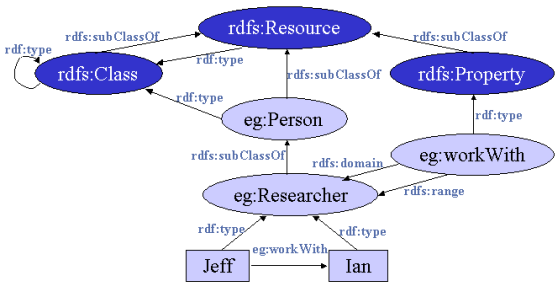

Fig. 1. An Example of Dual Roles in RDFS

The Resource Description Framework (RDF) [10] and its schema extension, RDF Schema Specification (RDFS) [3] form the lowest two layers of the Semantic Web. RDF is a foundation for processing metadata, which provides interoperability between applications that exchange machine-readable information on the Semantic Web. RDFS provides a standard mechanism for declaring classes and (global) properties as well as

[1] `http://DL-Web.man.ac.uk/rdfsfa/`

defining relationships between classes and properties, using RDF syntax. As a schema layer language, RDFS is responsible to define a basic metamodeling architecture for Web ontology languages.

RDFS, however, has a non-standard and non-fixed layer metamodeling architecture, which makes some elements in the model appear to have multiple roles - multiple modelling primitives seem to be implicitly represented by a single RDFS primitive.

Figure 1 shows an example of dual roles of some RDFS elements in a directed labelled graph. The top there ellipses represent three RDFS built-in modelling primitives rdfs:Class, rdfs:Resource and rdfs:Property. The rest is a very simple ontology. There are two classes in this ontology, where eg:Researcher is an rdfs:subClassOf eg:Person. eg:workWith is a property, whose rdfs:domain and rdfs: range are both eg:researcher. There are two instances of eg:Researcher, they are objects Ian and Jeff.

In this example, there seem to be more than one role for rdf:type and rdfs:subClassOf. For instance, rdf:type is used between objects and ontology classes (i.e. Jeff and eg:Researcher) and between ontology classes and built-in classes (i.e. eg:Person and rdfs: Class) etc. Similarly, rdfs:subClassOf is used between two ontology classes (i.e. eg:Researcher and eg:Person) and between two built-in classes (i.e. rdfs:Class and rdfs:Resource) etc.

Furthermore, there is a strange situation for rdfs:Class and rdfs:Resource as discussed in [12]. On the one hand, rdfs:Resource is an instance of rdfs:Class. On the other hand, rdfs:Class is a sub-class of rdfs:Resource. Thus rdfs:Resource is an instance of its sub-class. It is kind of confusing, isn't it?

While RDF is mainly used as standard *syntax*, RDFS is expected to be the lowest layer to provide *semantics* for the Semantic Web. However, the existence of dual roles in RDFS makes it difficult to give clear semantics to RDFS. E.g. it is unclear whether rdfs:Resource should be interpreted as an instance or a super-class of rdfs:Class. This might partially explain why Brickley and Guha [3] didn't define the semantics for RDFS. Up to now, there are at least two ways to clear up any confusion and give a clear semantics to the schema language: RDFS(FA) and RDF MT. We will present them individually in the following two sections.

3 RDFS(FA)

In [12] we proposed a sub-language of RDFS - RDFS(FA), which provides a Fixed layer metamodeling Architecture for RDFS. RDFS(FA) eliminates dual roles by defining the modelling primitives *explicitly*, instead of implicitly.

We call the solution *stratification*. The universe of discourse is divided up into different strata (layers). Built-in modelling primitives of RDFS are stratified into different strata of RDFS(FA), so that certain modelling primitives belong to a certain stratum (layer). Different prefixes, e.g. o-, l- or m-, are used to label which stratum modelling primitives belong to. The semantics of modelling primitives depend on the stratum they belong to. All these strata form the metamodeling architecture of RDFS(FA). Theoretically there can be infinite number of strata, while in practice, four strata are usually enough [12].

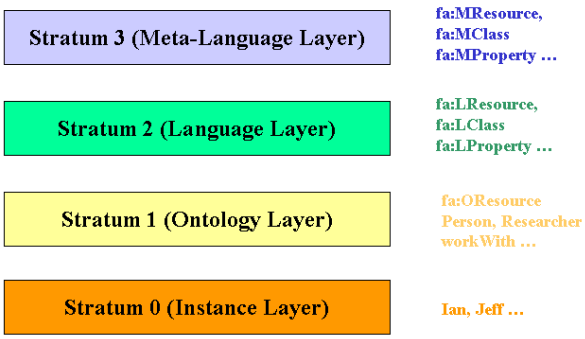

Fig. 2. Metamodeling Architecture (Four Strata) of RDFS(FA)

Figure 2 shows the metamodeling architecture (four strata) of RDFS(FA). Here are stratum 0,1,2 and 3. Some people like to call them layers, then they are called the Instance Layer, the Ontology Layer, the Language Layer and the Meta-Language Layer respectively.

Elements in the Instance Layer are objects, e.g. Ian and Jeff. Elements in the Ontology Layer are ontology classes, e.g. Person and Researcher, and ontology properties, e.g. workWith. Elements in the Language Layer are used to define and describe elements in the Ontology Layer, e.g. fa:LClass and fa:LProperty, and elements in the Meta-Language Layer are used to define and describe elements in the Language Layer.

As seen in Figure 2, rdfs:Resource is stratified into three layers, i.e. fa:OResource in the Ontology Layer, fa:LResource in the Language Layer and fa:MResource in the Meta-Language Layer. The same thing happens to rdfs:Class and rdfs:Property. They are stratified into the Language Layer and the Meta-Language Layer.

3.1 No Dual Roles in RDFS(FA)

There are no dual roles in RDFS(FA). Let's visit the same example again, but this time in RDFS(FA) (see Figure 3).

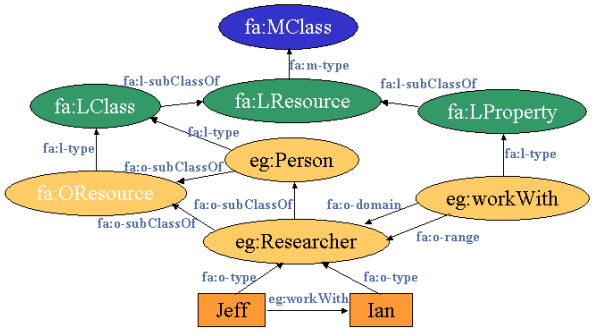

Fig. 3. No Dual Roles in RDFS(FA)

As we mentioned earlier, rdfs:Resource and rdfs:Class are stratified into different layers in RDFS(FA), such that fa:OResource is an instance of fa:LClass, and fa:LClass is a sub-class of fa:LResource, while fa:LResource is an instance of fa:MClass.

As far as rdf:type and rdfs:subClassOf in RDFS(FA), rdf:type is stratified into fa:o-type, fa:l-type and fa:m-type[2] where

- fa:o-type is used between objects and ontology classes, e.g. Jeff and eg:Researcher;
- fa:l-type is used between elements in the Ontology Layer and elements in the Language Layer, such as eg:Person and fa:LClass, as well as eg:workWith and fa:LProperty;
- fa:m-type is used between elements in the Language Layer and elements in the Meta-Language Layer, e.g. fa:LResource and fa:MClass.

Similarly, rdfs:subClassOf is stratified into fa:o-subClassOf, fa:l-subClassOf and fa:m-subClassOf:

- fa:o-subClassOf is used between two ontology classes, such as eg:Researcher and eg:Person;
- fa:l-subClassOf is used between two classes in the Language Layer, e.g. fa:LClass and fa:LResource;
- fa:m-subClassOf is used between two classes in the Meta-Language Layer.

3.2 Design Philosophy

We discuss the design philosophy of RDFS(FA) in this section. The principle is to build the fixed layer metamodeling architecture on the basis of semantics. There are two groups of fundamental modelling primitives in RDFS(FA), which are classes primitives and property primitives.

What is the semantics of a class primitives in RDFS(FA)? A Class primitive is interpreted as a set of objects or a set of sets. E.g. in Figure 3, eg:Researcher is a class in the Ontology Layer, since it is mapped to a set of objects (e.g. Ian and Jeff). For the same reason, eg:person is also a class. In the Language Layer, fa:LClass is a class since it is mapped to a set of sets (such as eg:Person and eg:Research). fa:Property is also class primitive, since it is mapped to a set of sets (such as eg:workWth).

What is the semantics of a property primitives in RDFS(FA)? A property primitive is interpreted as a set of binary relationships (or pairs) between two instance of class primitive(s) in the same stratum. E.g. in Figure 3, eg:workWith is a property primitive, since it is mapped to a set of binary relationships between two instances of eg:Researcher in the same stratum (the Ontology Layer). fa:o-subClassOf is also a property primitive, since it is mapped to a set of binary relationships between two instances of fa:LClass in the same stratum (the Language Layer).

Once a property primitive is defined in a certain stratum, it can be used in the adjacent lower stratum. E.g. in Figure 3, once eg:workWith is defined in stratum 1 (the Ontology Layer), it can be used in stratum 0 (the Instance Layer), e.g. Jeff eg:workWith Ian. Once fa:o-subClassOf is defined in stratum 2 (the Language Layer), it can be used in stratum 1 (the Ontology Layer), such as eg:Researcher fa:o-subClassOf eg:Person.

[2] In order to make it more readable, we change the syntax a bit and use fa:o-type, fa:l-type and fa:m-type, instead of fa:otype, fa:ltype and fa:mtype in [12].

The only exceptions are the type properties, because they are just the instance-of relationships, and always cross adjacent strata (layers). Please note that the type properties are very special, because they are just the *links* between classes and properties (see more details on the discussion of the type properties in Section 5).

3.3 Formal Description

Based on the design philosophy, we will give a formal description of the stratification of RDFS(FA) in this section. To clarify the presentation, we will not consider datatypes and values in this section; they will be discussed in Section 3.4.

Let V be a vocabulary, which is a set of urirefs. V is divided into disjoint sets $V_0, V_1, V_2, \ldots$, the vocabularies used in strata 0,1,2 ... respectively.

Let $\mathbf{R}_i$, $\mathbf{C}_i$, $\mathbf{P}_i$ be the modelling primitives which are interpreted as the sets of all elements, all classes and all properties respectively in stratum i. According to the design philosophy, since their *instances* are in stratum i, $\mathbf{R}_i$, $\mathbf{C}_i$, $\mathbf{P}_i$ are classes in stratum $i+1$. For example, $\mathbf{R}_1$, $\mathbf{C}_1$ and $\mathbf{P}_1$ are fa:LResource, fa:LClass and fa:LProperty respectively; fa:LResource is mapped to the *set* of all resources in stratum 1 (the Ontology Layer), fa:LClass is mapped to the *set* of all ontology classes (such as eg:Person) in stratum 1, and fa:LProperty is mapped to the *set* of all ontology properties (such as eg:workWith) in stratum 1; since their instances are in stratum 1, fa:LResource, fa:LClass and fa:LProperty exist in stratum 2.

Let $\mathbf{D}_i$ be the domain (a set) in stratum i and $\mathbf{IE}$ be an interpretation function.

We start from stratum 0. Every individual name $x \in V_0$ is mapped to an object in the domain $\mathbf{D}_0$:

$$IE(x) \in \mathbf{D}_0,$$

the set of all the elements in stratum 0 (the interpretation of $\mathbf{R}_0$) is $\mathbf{D}_0$:

$$IE(\mathbf{R}_0) = \mathbf{D}_0.$$

In stratum $i+1$ (where $i = 0, 1, 2, \ldots$), the set of all elements is equal to the domain of stratum i+1:

$$IE(\mathbf{R}_{i+1}) = \mathbf{D}_{i+1},$$

the domain $\mathbf{D}_{i+1}$ is equal to the union of the set of all classes and the set of all properties in stratum $i+1$:

$$IE(\mathbf{R}_{i+1}) = IE(\mathbf{C}_{i+1}) \cup IE(\mathbf{P}_{i+1})$$

Each class primitive $c_{i+1} \in V_{i+1}$ is interpreted as a set of elements in stratum i:

$$IE(c_{i+1}) \subseteq IE(\mathbf{R}_i),$$

and each property primitive $p_{i+1} \in V_{i+1}$ is interpreted as a set of pairs of elements in stratum i:

$$IE(p_{i+1}) \subseteq IE(\mathbf{R}_i) \times IE(\mathbf{R}_i).$$

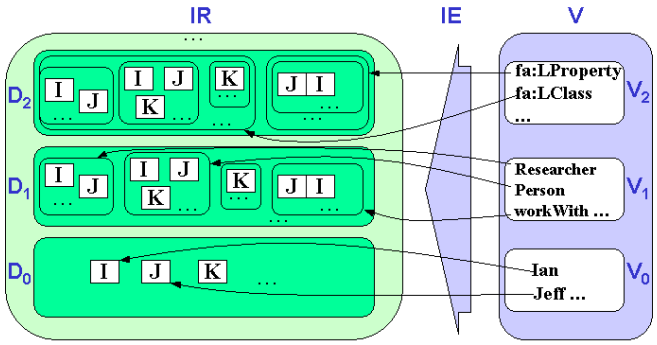

Fig. 4. Interpretation of RDFS(FA)

The $type_{i+1}$ property is interpreted as a set of pairs, where the first element is in stratum i, and the second element is a class in stratum $i+1$:

$$IE(type_{i+1}) \subseteq IE(\mathbf{R}_i) \times IE(\mathbf{C}_{i+1}).$$

Since $IE(c_{i+1}) \subseteq IE(\mathbf{R}_i)$, we have $IE(c_{i+1}) \in 2^{IE(\mathbf{R}_i)}$, i.e. $IE(c_{i+1}) \in 2^{\mathbf{D}_i}$. According to the definition of $\mathbf{C}_{i+1}$, we have

$$IE(\mathbf{C}_{i+1}) = 2^{\mathbf{D}_i}.$$

Similarly, since $IE(p_{i+1}) \subseteq IE(\mathbf{R}_i) \times IE(\mathbf{R}_i)$, we have $IE(p_{i+1}) \in 2^{IE(\mathbf{R}_i) \times IE(\mathbf{R}_i)}$, i.e. $IE(p_{i+1}) \in 2^{\mathbf{D}_i \times \mathbf{D}_i}$. According to the definition of $\mathbf{P}_{i+1}$, we have

$$IE(\mathbf{P}_{i+1}) = 2^{\mathbf{D}_i \times \mathbf{D}_i}.$$

Since $IE(\mathbf{R}_{i+1}) = \mathbf{D}_{i+1} = IE(\mathbf{C}_{i+1}) \cup IE(\mathbf{P}_{i+1})$,we have

$$\mathbf{D}_{i+1} = 2^{\mathbf{D}_i} \cup 2^{\mathbf{D}_i \times \mathbf{D}_i}.$$

The pair $\langle IR, IE \rangle$ is an interpretation for RDFS(FA), where

$$IR = \mathbf{D}_0 \cup \mathbf{D}_1 \cup \mathbf{D}_2 \cup \cdots.$$

Figure 4 illustrates the interpretation of RDFS(FA). Vocabularies in stratum 0 (the Instance Layer), e.g. Ian and Jeff, are interpreted as objects (i.e., elements of D_0). Vocabularies for ontology classes (in V_1), such as eg:Researcher and eg:Person, are interpreted as sets of objects. Vocabularies for ontology properties (in V_1), such as eg:workWith, are interpreted as sets of pairs of objects. In stratum 2 (the Language Layer), fa:LClass is interpreted as a set of sets of objects (a set of ontology classes), and fa:LProperty is interpreted as a set of sets of pairs of objects (a set of ontology properties).

3.4 RDFS(FA) and DAML+OIL

DAML+OIL is consistent with the stratification of RDFS(FA). The DAML+OIL language itself exists in stratum 2 (the Language Layer). Its semantics [17] covers stratum

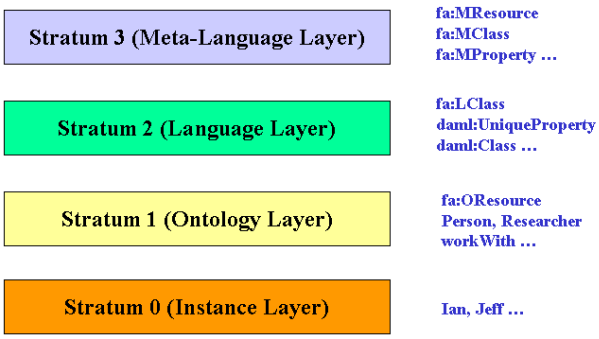

Fig. 5. DAML+OIL and the Stratification of RDFS(FA)

0 and 1. In Figure 5, DAML+OIL constructs, e.g. daml:Class and daml:UniqueProperty are in stratum 2 (the Language Layer).

Since DAML+OIL supports datatypes, the domain in stratum 0 is divided into two disjoint sets, the "object domain" $\mathbf{OD}$ and "datatype domain" $\mathbf{DD}$[3] such that $\mathbf{D_0} = \mathbf{OD} \cup \mathbf{DD}$.

In stratum 0, every individual name $x \in V_0$ is interpreted as an object in the domain $\mathbf{OD}$:

$$IE(x) \in \mathbf{OD},$$

every literal $t \in V_0$ is interpreted as a data value in the domain $\mathbf{DD}$

$$IE(t) \in \mathbf{DD},$$

and the set of all the elements in stratum 0 (the interpretation of $\mathbf{R}_0$) is $\mathbf{D_0}$

$$IE(\mathbf{R}_0) = \mathbf{D}_0 = \mathbf{OD} \cup \mathbf{DD}.$$

In stratum 1, each class primitive $c_1 \in V_1$ is interpreted as a set of objects:

$$IE(c_1) \subseteq \mathbf{OD},$$

each datatype name $d_1 \in V_1$ is interpreted as a set of data values:

$$IE(d_1) \subseteq \mathbf{DD},$$

each object property $p_1^o \in V_1$ is interpreted as a set of pairs of objects:

$$IE(p_1^o) \subseteq \mathbf{OD} \times \mathbf{OD},$$

each datatype property $p_1^d \in V_1$ is interpreted as a set of pairs, where the first element is an object and the second element is a data value:

$$IE(p_1^d) \subseteq \mathbf{OD} \times \mathbf{DD}.$$

[3] Note that datatype domain (a set) is in stratum 1, since it is a set of data values that are in stratum o.

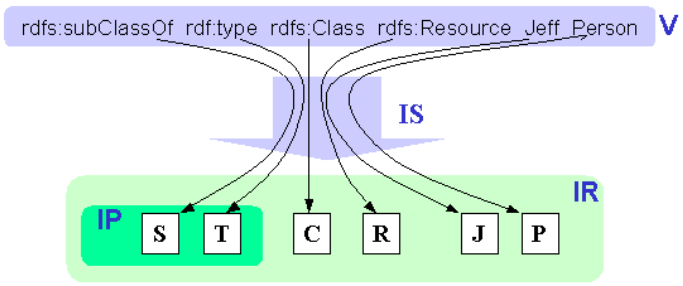

Fig. 6. Resources in RDF MT

In general, DAML+OIL, and other first order logical layer Semantic Web languages, can be built on top of both the syntax and semantics of RDFS(FA). Furthermore, the stratification of RDFS(FA) can benefit such logical layer Semantic Web languages by offering possibilities of extending them in stratum 3 (the Meta-Language Layer). It can also help avoiding "layer mistakes" [12] in DAML+OIL.

4 RDF Model Theory

Another way to clear up the kinds of confusion of RDFS is RDF Model Theory (RDF MT) [12], which gives a precise semantic theory for RDF and RDFS. It is a W3C working draft when this paper is being written.

An interpretation in the RDF model theory is a triple $\langle IR, IEXT, IS\rangle$, where **IR** is the domain (of resources); IS is a function that maps URI references to elements of **IR**. and $IEXT$ is an extension function from **IR** to $IR \times IR$. In RDF MT, meaning is given to properties by first mapping the property URI references to an object of the domain of discourse via IS. The domain object is then mapped into a set of pairs via $IEXT$.

In RDF MT, all resources (including all classes and properties) are objects (see Figure 6). IS maps the URI references of resources to objects in the domain **IR**, e.g. IS maps rdfs:subClassOf to object S, or IS(rdfs:subClassOf) , rdf:type to object T, or IS(rdf:type), rdfs:Class to object C, or IS(rdfs:Class) etc. IP is a special sub-set of **IR**. It is a set of all property objects.

Property objects are special in the sense that they can have non-empty extensions. Extension function $IEXT$ maps property objects to their extensions. E.g. in Figure 7, $IEXT$ maps S to $IEXT$(S), which is a set of pairs $\{\langle$P,R$\rangle$, $\langle$C,R$\rangle\}$. $IEXT$ maps T to $IEXT$(T), which is a set of pairs $\{\langle$P,C$\rangle$, $\langle$R,C$\rangle\}$. Class primitives are not fundamental primitives in RDF MT. Class extension $ICEXT$ is defined through the extension of IS(rdf:type):

$$ICEXT(x) = \{y \mid \langle y, x\rangle \text{ is in } IEXT(IS(\text{rdf:type}))\}$$

In Figure 7, $IEXT$(T)=$\{\langle$P,C$\rangle$, $\langle$R,C$\rangle\}$, so P and R are in ICEXT(C).

4.1 No Confusion in RDF MT

RDF MT justify dual roles in RDFS by treating classes and properties as objects. In other words, class primitives in RDF MT are interpreted as objects that can have non-

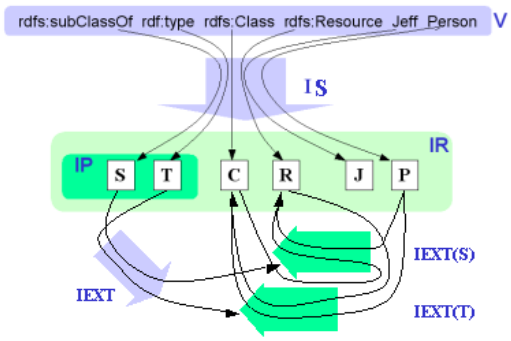

Fig. 7. Interpretation of RDF MT

empty class extensions; property primitives in RDF MT are interpreted as objects that can have non-empty extensions. Even though it is a bit strange to some people, there is no confusion in RDF MT.

Let's revisit the same example in RDF MT. rdfs:Class and rdfs:Resource are mapped to objects C and R in the domain of resource by IS, therefore rdfs:Class is rdfs:subClassOf rdfs:Resource means the pair of R and C is in the extension of the rdfs:subClassOf object S

$$\langle \text{C,R} \rangle \in IEXT(\text{S})$$

while rdfs:Resource is instance of rdfs:Class means the pair of R and C is in the extension of the rdf:type object T

$$\langle \text{R,C} \rangle \in IEXT(\text{T}).$$

According to the definition of $ICEXT$, we have

$$\text{R} \in ICEXT(\text{C}).$$

In this way, the situation between rdfs:Class and rdfs:Resource is considerably reasonable.

5 Comparing the Two Approaches

In Section 3 and 4, we described two known approaches to clear up any confusion of RDFS. In this section, we will first compare these two approaches, and then discuss their advantages and disadvantages.

5.1 Main Differences

On how to clear up confusion of RDFS, RDFS(FA) stratifies dual roles into different strata and define modelling primitives explicitly; while RDF MT justifies dual roles by treating classes and properties as objects, suggesting having dual roles is a feature of the language, instead of a problem.

Differences in Syntax RDFS(FA) adds restriction of stratification on the syntax of RDFS, in order to avoid dual roles. Since elements of RDFS(FA) exists in different strata of the domain of universe, (prefixes of) symbols of elements should indicates this fact. E.g. fa:OResource is in stratum 1 (the Ontology Layer), fa:LProperty is in stratum 2 (the Language Layer), and fa:MClass is in stratum 3 (the Meta-Language Layer). [4]

Furthermore, valid RDFS(FA) statements should be consistent with the design philosophy (see Section 3.2) of RDFS(FA). If one defines a class in stratum $i+1$, then the instances of that class should be in stratum i, e.g.

```
<fa:LClass rdf:about=''#Person''>
</fa:LClass>
```

Since fa:LClass is in stratum 2 (the Language Layer), the Class "Person" should be in stratum 1 (the Ontology Layer), because "Person" is an instance of fa:LClass.

If one defines a property in stratum $i+1$, then the classes as the domain and range of the property should be in stratum $i+1$ as well.

```
<fa:LProperty rdf:about=''#hasFriend''>
  <fa:o-domain rdf:resource=''#Person''/>
  <fa:o-range rdf:resource=''#Person''/>
</fa:LProperty>
```

Since fa:LProperty is in stratum 2 (the Language Layer), the property "hasFriend" should be in stratum 1 (the Ontology Layer). Thus the class "Person", as the fa:o-domain and fa:o-range of "hasFriend", should be in stratum 1 (the Ontology Layer).

Differences in Semantics Besides differences in syntax, there are also differences in semantics between RDFS(FA) and RDF MT.[5]

First of all, the domain in RDFS(FA) is smaller than the domain of RDF MT, in other words, RDFS is more expressive than RDFS(FA). The reason for being less expressive is that the stratification of RDFS(FA) disallows most cross-stratum binary relationships (except the type properties). However, it could be argued that these kinds of relationship are too confusing for most users.

Secondly, there are different *fundamental* primitives in RDFS(FA) and RDF MT. In RDFS(FA), both class and property primitives are fundamental primitives, i.e., both are directly interpreted by the interpretation function IE. As seen in Section 3.3, RDFS(FA) class primitives in stratum $i+1$ are interpreted as sets of elements in stratum i, while property primitives in stratum $i+1$ are interpreted as sets of pairs of elements in stratum i. The type properties are *special* properties: their interpretations are just the *instance-of* relationships.

In RDF MT, although both class and property primitives are objects, *only* property primitives are fundamental primitives, i.e., only property primitives can be given non empty extensions by $IEXT$. The class extension $ICEXT$ is simply derived from the

[4] Note that properties, except the type properties, are always used one stratum lower than the one where they are defined, e.g. fa:o-subClassOf is defined in stratum 2 (the Language Layer) and used in stratum 1 (the Ontology Layer).

[5] Readers are advised to refer to Figure 4 and 7 for better understanding of these differences.

$IEXT$ extension of the rdf:type object. Note, however that although the rdf:type property is used to define membership of classes, in all other respects it is treated in the same way as any other property.

Thirdly, RDFS(FA) and RDF MT interpret property (and class) primitives in different ways. In RDFS(FA) (as well as conventional FOL), there is a mapping directly from property (class) symbols to a set of pairs of elements (elements) in the domain. While in RDF MT, on the other hand, meaning is given to property and class symbols by first mapping them (via IS) to objects in the domain. A property object is then mapped (via $IEXT$) to a set of pairs of objects in the domain.

Based on the differences between RDFS(FA) and RDF MT, we will discuss their advantages and disadvantages in next section.

5.2 Advantages of RDF MT

Since there is no restriction of stratification, RDF MT (RDFS) is more expressive than RDFS(FA). This advantage of RDF MT is believed to be consistent with the following philosophy: anyone can say anything about anything. In RDF MT, this means

- properties can be defined between any two resources;
- any resource can be an instance of any resource (including itself).

However, this *"unlimited"* expressive power can lead to problems, as we will see in the following section.

5.3 Disadvantages of RDF MT

The syntax rules in RDFS are very weak, and there are not many restrictions on writing RDFS triples. As a result, this can be confusing and difficult to understand and, more importantly, the specification of its semantics requires a non-standard model theory, i.e. RDF MT.

This leads to semantic problems when trying to layer[6] conventional FOL, like DAML+OIL and OWL. E.g., as DAML+OIL is more expressive than RDFS, a large and more complex set of semantic conditions[7] is required to capture the meaning and characteristic of its additional constructs. It is very difficult to get such semantic conditions correct, not to mention that one should also prove that they are right.

There are at least three known problems if we extend RDFS with more expressive FOL constructs, e.g. conjunctions and qualified number restrictions, and extend RDF MT to so called "RDF+ MT" to give meaning to this extended language. These known problems are: (i) too few entailments; (ii) contradiction classes; (iii) size of the universe.

[6] RDFS, in some sense, is a very limited language, and serves as the bottom semantic layer of Semantic Web languages. So it is both necessary and desirable to layer more expressive ontology languages on top of it.

[7] Since the constructs of RDFS are simple, the set of semantic conditions for RDFS is relatively small.

Too Few Entailments Entailment is the key idea which connects model-theoretic semantics to real-world applications. What is entailment? In RDF MT, entailment means "If A entails B, then any interpretation that makes A true also makes B true," so that an assertion of A already contains the same "meaning" as an assertion of B [12].

[13] first addressed the problem of too few entailments and gave the following example: if John is an instance of the class Student ∩Employee∩European, is John an instance of the class Employee ∩Student?

In RDFS(FA) and FOL, the answer is simply "yes", since Student∩Employee∩ European is a sub-class of Employee ∩Student, so every instance of the former class is also an instance of the later one.

However, in "RDF+ MT", since every concept is also an object, "John is an instance of the concept Student∩Employee ∩European" can't guarantee there exists an object for Employee ∩Student in all the interpretations that make "John is an instance of the concept Student∩ Employee∩European" true. So the answer in RDF+ MT is "no".

In this case, the "RDF+ MT" semantics seems to be broken, because the semantics of an ontology language should give meaning to any possible class expressions. In order to fix the problem, one can/should introduce comprehension axioms to add all possible missing objects into the domain, e.g. the Employee ∩Student in this example. But that is surely a very difficult task. Theoretically, it is yet to be proved that proper objects are all added into the universe, no more and no less. Practically, there will be infinite numbers of possible class expressions.[8] It is still unknown whether there exists a practical approach to solve the problem.

Contradiction Classes [13,14] also addressed the problem of contradiction classes. In RDFS, resources can be defined as instances of themselves, and rdf:type is treated as any other property. So, if the extended language supports qualified number restrictions, one can define a class eg:C as an instance of itself, and add a cardinality constraint "=0" on the rdf:type property (see Figure 8).

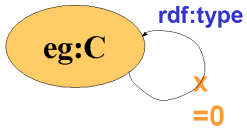

Fig. 8. Contradiction Classes

It is impossible for one to determine the membership of this class. If an object is an instance of this class, then it isn't, because instances should have no rdf:type property pointing to itself. But if it isn't then it is. This is a contradiction class.

One might argue that we can simply avoid defining such classes. However, with the comprehension axioms (see Section 5.3.1), we must add all possible class objects into the domain, and the above contradiction class is one of them. In this way, all the interpretations will have such contradiction classes, and thus have ill-defined class member-

[8] Think about all the possible conjunctions, disjunctions, exist restrictions, value restrictions and qualified number restrictions ...

ships. Again, the "RDF+ MT" semantics seems to be broken[9]. RDFS(FA) does't have this problem, because the type properties are not treated as ordinary properties.

Size of the Universe Like RDF MT, in "RDF+ MT" there is a pre-defined vocabulary (e.g. rdf:type, rdfs:Property etc), terms from which are mapped to elements of the universe. Thus these elements *must* exist in all the possible interpretations. A problem arises if we set constraints on the size of the universe. Even if we don't consider such pre-defined vocabulary, this problem still exists. Here is an example.

Let us consider the following question: is it possible to have an interpretation such that John is a member of Person, but not a member of Car, and there is only one object in the universe?

In RDFS(FA) and FOL, the answer is simply "yes". There is only one object in the universe, and John is interpreted as that object. The Person class thus has one instance, i.e. the interpretation of John. The Car class has no instances. So it is possible to have such an interpretation.

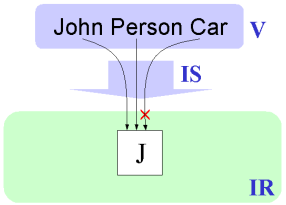

Fig. 9. Size of the Universe

But it is impossible to have only one object in the "RDF+ MT" universe in this example. Since classes are also objects, John, Person and Car should all be mapped to the only one object in the universe (see Figure 9). However, since the interpretation of John is a member of Person, but not a member of Car, Person and Car should be different. Thus there should at least two objects in the universe. In other words, the required interpretation is impossible in "RDF+ MT", and the answer to our question is "no".

This example shows that the interpretation of RDF MT has different features than the interpretation of standard FOL model theoretic semantics. This raises the question as to whether it is possible to layer FOL languages on top of both the syntax and semantics of RDFS.

6 Discussion

As we have seen, RDFS has a non-standard and non-fixed layer metamodeling architecture, which makes some elements in the model have dual roles in the RDFS specification. This can be confusing and difficult to understand for modelers. One of the consequences is that when DAML+OIL is layering on top of RDFS, it uses the syntax of RDFS only, and defines its own semantics for the ontological primitives of RDFS.

[9] One might solve the problem by making the comprehension axioms more complex. It is yet to be proved that we keep the objects of all possible contradiction classes outside the universe.

Up to now, there are at least two approaches to clear up the kinds of confusion that can arise w.r.t. RDFS: RDFS(FA) and RDF MT. RDFS(FA), as a sub-language of RDFS, clears up any confusions via stratification, while RDF MT justifies dual roles by treating classes and properties as objects.

The advantage of RDF MT is that it is more expressive than RDFS(FA), because it doesn't have the restriction of stratification. The philosophy is that anyone can say anything about anything. Properties can be defined between any two resources, and a resource can be an instance of any resource (including itself). Some people, however, worry about this "unlimited" expressive power, in particular when layering more expressive languages on top of RDFS.

The advantage of RDFS(FA) is that FOLs, e.g. DAML+OIL, can be built on top of both the syntax and semantics of RDFS(FA). Furthermore, the stratification of RDFS(FA) can benefit such logical layer Semantic Web languages by offering possibilities of extending them in stratum 3 (the Meta-Language Layer).

The disadvantage of RDF MT is that there are at least three known problems if we extend RDFS with more expressive FOL constructs, and extend RDF MT to the so called "RDF+ MT" to give meaning to this extended language (see Section 5.3). Moreover, layering FOL on top of RDFS doesn't lead directly to any "computational pathway", i.e. it is not clear whether/how applications would be able to reason with languages layered on top of RDFS.

Generally speaking, on the one hand, RDF MT allows for a larger number of models of the universe, and can represent more heterogeneous states of affairs. On the other hand, RDFS(FA) allows more expressive ontology languages, e.g. DAML+OIL, to be layered on top of it, so that people can say more things about a smaller number of (more homogeneous) models of the universe.

It has yet to be proved that RDF MT can be extended to give a coherent meaning to more expressive ontology languages[10]. Moreover, it is not clear if the more heterogeneous models supported by RDF MT would be needed in many realistic applications. Given that the set of RDFS(FA) statements is a subset of the set of RDFS statements, one possible solution would be to support both semantics, with users able to choose if they are willing to constrain the kinds of model that can be represented in order to facilitate the extension of the language with more expressive modelling primitives. This solution could also provide a good guideline for more expressive logical ontology languages designed on top of RDFS.

7 Acknowledgments

We would like to thank Peter Patel-Schneider for helpful discussion on the stratification of RDFS(FA).

[10] Note that in OWL the RDFS-compatible model-theoretic semantics has the same consequence as the direct semantics on OWL ontologies, *only* when the separate vocabulary restriction is satisfied [15].

References

1. T. Berners-lee. Semantic Web Road Map. W3C Design Issues. URL http://www.w3.org/DesignIssues/Semantic.html, Oct. 1998.
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition) – W3C Recommendation 6 October 2000. Technical report, World Wide Web Consortium, 2000. Available at http://www.w3.org/TR/REC-xml.
3. D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommentdation, Mar. 2000.
4. J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge representation on the web by extending rdf schema. Hong Kong, May 2001.
5. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. S. eds. OWL Web Ontology Language 1.0 Reference. URL http://www.w3.org/TR/owl-ref/, Nov 2002.
6. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In R. Dieng, editor, *Proc. of the 12th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW'00)*, number 1937 in Lecture Notes in Artificial Intelligence, pages 1–16. Springer-Verlag, 2000.
7. D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
8. P. Hayes. RDF Model Theory. Apr 2002. W3C Working Draft, URL http://www.w3.org/TR/rdf-mt/.
9. J. Hendler and D. L. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):67–73, 2000.
10. O. Lassila and R. R.Swick. Resource Description Framework (RDF) Model and Syntax Specification – W3C Recommendation 22 February 1999. Technical report, World Wide Web Consortium, 1999.
11. W. Nejdl, M. Wolpers, and C. Capella. The RDF Schema Specification Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000*, Apr. 2000.
12. J. Z. Pan and I. Horrocks. Metamodeling Architecture of Web Ontology Languages. In *Proceeding of the Semantic Web Working Symposium (SWWS)*, July 2001. URL http://www.cs.man.ac.uk/~panz/Zhilin/download/Paper/Pan-Horrocks-rdfsfa-2001.pdf.
13. P. F. Patel-Schneider. Layering the Semantic Web: Problems and Directions. . In *2002 International Semantic Web Conference*, Jun 2002.
14. P. F. Patel-Schneider. Two Proposals for a Semantic Web Ontology Language. In *2002 International Description Logic Workshop*, Apr 2002.
15. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, Mar. 2003. W3C Working Draft.
16. M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 1996.
17. F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. A Model-Theortis Semantics of DAML+OIL(March 2001). Mar. 2001.
18. F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. Reference Description of the DAML+OIL(March 2001) Ontology Markuk Language. DAML+OIL Document. Available at http://www.daml.org/2000/12/reference.html, Mar. 2001.