Metamodeling Architecture of Web Ontology Languages

Jeff Z. Pan and Ian Horrocks

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

Abstract. Recent research has shown that RDF Schema, as a schema layer Semantic Web language, has a non-standard metamodeling architecture. As a result, it is difficult to understand and lacks clear semantics. This paper proposes a fixed layer metamodeling architecture for RDF Schema (RDFS(FA)) and demonstrates how the problems of RDF Schema can be solved under RDFS(FA). Based on this metamodeling architecture, a clear model-theoretic semantics of RDFS(FA) is given. Interestingly, RDFS(FA) also benefits DAML+OIL by offering a firm semantic basis and by solving the "layer mistake" problem.

1 Introduction

The Semantic Web, with its vision stated by Berners-lee [1], aims at developing *languages* for expressing information in a machine understandable form. The recent explosion of interest in the World Wide Web has also fuelled interest in ontologies. It has been predicted (Broekstra et al. [3]) that ontologies will play a pivotal role in the Semantic Web since ontologies can provide shared domain models, which are understandable to both human being and machines.

Ontology (Uschold and Gruninger [20]) is, in general, a representation of a shared conceptualization of a specific domain. It provides a shared and common understanding of a *domain* that can be communicated between people and heterogeneous and distributed application systems. An ontology necessarily entails or embodies some sort of world view with respect to a given domain. The world view is usually conceived as a hierarchical description of important concepts (is-a hierarchy), a set of crucial properties, and their inter-relationships.

Berners-lee [1] outlined the architecture of Semantic Web. We would like to call it a *functional architecture* because the expressive primitives are incrementally introduced from languages in the lowest layer (i.e. matadata layer) to those in the higher layer (e.g. logical layer), so that the languages in each layer can satisfy the requirements of different kinds (or levels) of applications:

1. In the *metadata layer*, a simple and general model of semantic assertions of the Web is introduced. The simple model contains just the concepts of *resource* and *property*, which are used to express the meta information and will be needed by languages in the upper layers. The Resource Description Framework (RDF) (Lassila and R.Swick [13]) is believed to be the general model in metadata layer.

2. In the *schema layer*, simple Web *ontology* languages are introduced, which will define a hierarchical description of concepts (is-a hierarchy) and properties. These languages use the general model in metadata layer to define the basic metamodeling architecture of Web ontology languages. RDF Schema (RDFS) (Brickley and Guha [2]) is a candidate schema layer language.

3. In the *logical layer*, more powerful Web *ontology* languages are introduced. These languages are based on the basic metamodeling architecture defined in schema layer, and defines a much richer set of modelling primitives that can e.g. be mapped to very expressive Description Logics (Horrocks et al. [11], Horrocks [10]) to supply reasoning services for the Semantic Web. OIL (Horocks et al. [9]) and DAML+OIL (van Harmelen et al. [22]) are well known logical layer languages.

This paper will focus on the *metamodeling architecture* other than the functional architecture. We should point out that "metamodeling" and the "metadata layer" in the functional architecture are not the same. Metadata means data about data. *Metamodeling* concerns the definition of the modelling primitives (vocabulary) used in a modelling language. Many software engineering modelling languages, including UML, are based on metamodels. Among the Semantic Web languages, the schema layer languages are responsible to build the metamodeling architecture.

In this paper, we argue that RDFS, as a schema layer language, has a non-standard and non-fixed layer metamodeling architecture, which makes some elements in the model have dual roles in the RDFS specification. Therefore, it makes the RDFS specification itself quite difficult to understand by the modellers. The even worse thing is that since the logical layer languages (e.g. OIL, DAML+OIL) are all based on the metamodeling architecture defined by schema layer languages (RDFS), these languages therefore have the similar problems, e.g. the "layer mistake" discussed in Section 2.3.

We propose a fixed layer metamodeling architecture for RDFS (we call it *RDFS(FA)*) which is similar to the metamodeling architecture of UML. We analyse the problems of the non-fixed metamodeling architecture of RDFS and demonstrate how these problems can be solved under RDFS(FA). Furthermore, We give a clear semantics to RDFS(FA).

The rest of the article is organized as follows. In Section 2 we explain the data model of RDF, RDFS and DAML+OIL, the languages belonging to the matadata level, schema level and logical level of the Semantic Web Architecture respectively. We will focus on the metamodeling architecture of RDFS and locate what the problems are and where they come from. In Section 3 we discuss the advantages and disadvantages of fixed and non-fixed layer metamodeling architecture and then briefly explain the metamodeling architecture of UML. In Section 4 we propose RDFS(FA), and give a clear semantics to RDFS(FA). We also demonstrate how the "layer mistake" problem with DAML+OIL is solved in RDFS(FA). Section 5 briefly discuss the advantages of RDFS(FA) and our attitudes on how to make use of UML in the Web ontology languages.

2 Current Data Models of Semantic Web Languages

2.1 RDF Data Model

As a Semantic Web language in the *metadata layer* of the functional architecture, RDF is a foundation for processing metadata. It provides interoperability between applications that exchange machine-understandable information on the Web. The foundation of RDF is a model for representing named properties and property values. The RDF data model provides an abstract, conceptual framework for defining and using metadata. The basic data model consists of three object types:

Resources: All things being described by RDF expressions are called *resources*. A resource may be an entire Web page, a part of a Web page, a whole collection of pages (Web site); 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*.

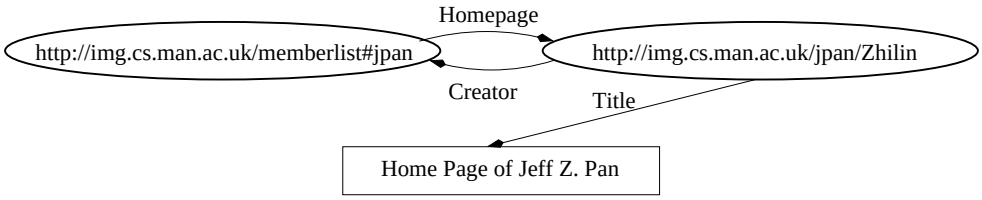

Figure 1: An Example of RDF in a Directed Labeled Graph

In a nutshell, the RDF data model is an object-property-value mechanism. The metadata information is introduced by a set of statements in RDF. There are several ways to express RDF statements. First, we can use the binary predicate form Property(object,value), e.g. Title('`http://img.cs.man.ac.uk/jpan/Zhilin`', "Home Page of Jeff Z. Pan"). Secondly, we can diagram an RDF statement pictorially using directed labeled graphs: '[object]-Property->[value]' (see Figure 1). Thirdly, RDF uses an Extensible Markup Language (XML) encoding as its interchange syntax:

```
<rdf:Description rdf:ID="http://img.cs.man.ac.uk/jpan/Zhilin">
  <Title>Home Page of Jeff Z. Pan</Title>
</rdf:Description>
```

The RDF data model is so called "property-centric". We can use the "about" attribute to add more properties to the existing resource. Generally speaking, with the object-property-value mechanism, RDF can be used to express:

- *attributes* of resources: in this case, the 'value' is a literal (e.g the "Title" property above);
- *relationships* between any two resources: in this case, the 'value' is a resource, and the involved properties represent different roles of the two resources with this relationship; in the following example, there exists a "creator-homepage" relationship between "http://img.cs.man.ac.uk/jpan/Zhilin" and "http://img.cs.man.ac.uk/memberlist#jpan" (see also Figure 1):

  ```
  <rdf:Description rdf:ID="http://img.cs.man.ac.uk/memberlist#jpan">
    <Homepage rdf:resource="http://img.cs.man.ac.uk/jpan/Zhilin"/>
  </rdf:Description>
  <rdf:Description about="http://img.cs.man.ac.uk/jpan/Zhilin">
    <Creator rdf:resource="http://img.cs.man.ac.uk/memberlist#jpan"/>
  </rdf:Description>
  ```

- *weak type* of resources: the 'type' is *weak* because RDF itself has no standard way to define a Class, so the type here is regarded only as a special attribute; for example,

  ```
  <rdf:Description about="http://img.cs.man.ac.uk/memberlist#jpan">
    <rdf:type rdf:resource="#Person"/>
  </rdf:Description>
  ```

- *statement about statement*: RDF can be used for making statements about other RDF statements, which are referred to as *higher-order statements*. This feature of RDF has yet to be clearly defined and is beyond the scope of this paper.

2.2 RDF Schema Data Model

As we have seen, on the one hand, RDF data model is enough for defining and using metadata. On the other hand, the modelling primitives offered by RDF are very basic. Although you can define "Class" and "subClassOf" as resources in RDF (no one can stop you doing that), RDF provides no standard mechanisms for declaring classes and (global) properties, nor does it provide any mechanisms for

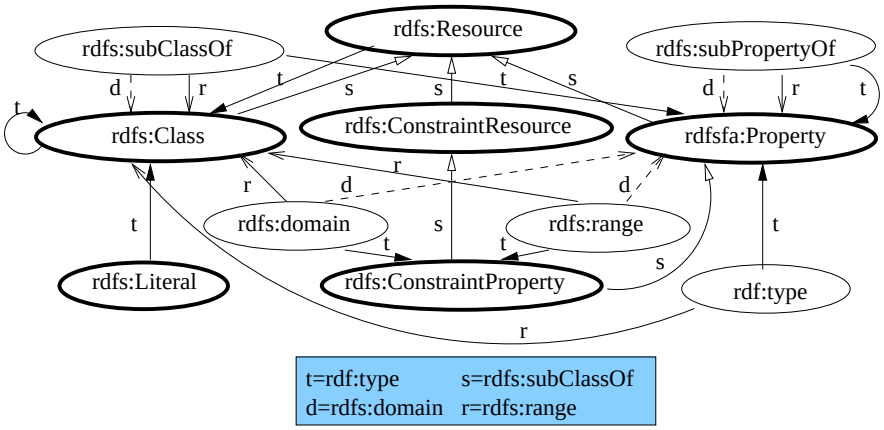

Figure 2: Directed Labeled Graph of RDF Schema

defining the relationships between properties or between classes. That is the role of RDFS–a Semantic Web language in the schema layer.

RDFS is expressed by using RDF data model. It extends RDF by giving an externally specified semantics to specific resources. In RDFS, rdfs:Class is used to define concepts, i.e. every class must be an instance of rdfs:Class. Resources that are described by RDF expressions are viewed to be instances of the class rdfs:Resource. The class rdf:Property is the class of all properties used to characterize instances rdfs:Resource. The rdfs:ConstraintResource defines the class of all constraints. The rdfs:ConstraintProperty is a subset of rdfs:ConstraintResource and rdf:Property, all of its instances are properties used to specify constraints, e.g. rdfs:domain and rdfs:range. For example, the following RDFS expressions

```
<rdfs:Class rdf:ID="Animal">
  <rdfs:comment>This class of animals is illustrative of a number of
  ontological idioms.</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</rdfs:Class>
<rdf:Description rdf:ID="John">
  <rdf:type rdf:resource="#Person"/>
  <rdfs:comment>John is a person.</rdfs:comment>
</rdf:Description>
<rdf:Description rdf:ID="Mary">
  <rdf:type rdf:resource="#Person"/>
  <rdfs:comment>Mary is a person.</rdfs:comment>
</rdf:Description>
```

define the classes “Animal” and “Person”, with the latter being the subclass of the former, and two individuals “John” and “Mary”, which are instances of the class “Person”. Individual “John” can also be defined in this way,

```
<Person rdf:ID="John">
  <rdfs:comment>John is a person.</rdfs:comment>
</Person>
```

which is an implicit way to define rdf:type property. Note that here “Person” is a class.

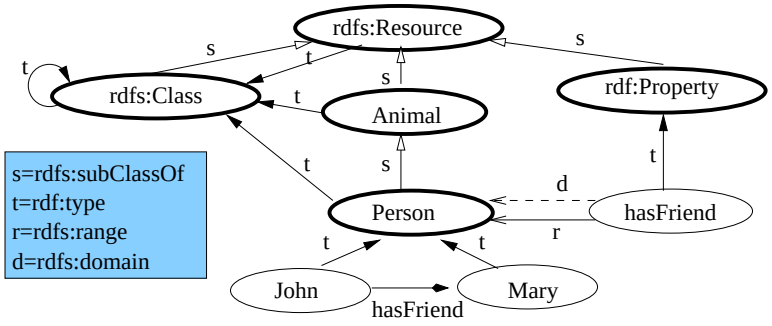

Figure 3: A “Person–hasFriend” Example of RDF Schema

Figure 2 pictures the subclass-of and instance-of hierarchy of RDFS: rdfs:Resource, rdfs:Class, rdf:Property, rdfs:ConstraintResource and rdfs:ConstraintProperty are all instances of rdfs:Class, while rdfs:Class, rdf:Property and rdfs:ConstraintResource are subclass of rdfs:Resource. It is confusing that rdfs:Class is a sub-class of rdfs:Resource, while rdfs:Resource itself is an instance of rdfs:Class at the same time. It is also strange that rdfs:Class is an instance of itself.

In RDFS, all properties are instances of rdf:Property. The rdf:type property models instance-of relationships between resources and classes. The rdfs:subClassOf property models the subsumption hierarchy between classes, and is transitive. The rdfs:subPropertyOf property models the subsumption hierarchy between properties, and is also transitive. The rdfs:domain and rdfs:range properties are used to restrict domain and range properties. For example, the following RDFS expressions

```
<rdf:Property rdf:ID="hasFriend">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
<rdf:Description about="#John">
  <hasFriend rdf:resource="#Mary"/>
</rdf:Description>
```

define a property “hasFriend” between two “Person”s and ⟨‘John’, ‘Mary’⟩ is an instance of “hasFriend” (see Figure 3).

In RDFS, properties are regarded as sets of binary relationships between instances of classes, e.g. a property “hasFriend” is a set of binary tuples between two instances of the class “Person”. One exception is the rdf:type, since it is just the *instance-of* relationship. In this sense, rdf:type is regarded as a special predefined property.

Figure 2 also shows the range and domain constraints in RDFS–rdfs:domain and rdfs:range can be used to specify the two classes that a certain property can associate with. So the rdfs:domain of rdfs:domain and rdfs:range is the class rdf:Property, the rdfs:range of rdfs:domain and rdfs:range is the class rdfs:Class. The rdfs:domain and rdfs:range of rdfs:subClassOf is rdfs:Class. The rdfs:domain and rdfs:range of rdfs:subPropertyOf is rdf:Property. The rdfs:range of rdf:type is the class rdfs:Class. The rdf:type property is regarded as a set of binary links between *instances* and classes (as mentioned above), while the value of the rdfs:domain property should be a *class*, therefore rdf:type does not have the rdfs:domain property (cf. Brickley and Guha [2]).

As we have seen, RDFS use some primitive modelling primitives to define other modelling primitives (e.g. rdf:type, rdfs:domain, rdfs:range, rdf:type and rdfs:subClassOf). At the same time, these primitives can be used to define ontologies as well, which makes it rather unique when compared to conventional model and metamodeling approaches, and makes the RDFS specification very difficult to read and to formalize (Nejdl et al. [16], Broekstra et al. [3]). For example, in Figure 3, it is confusing that although rdfs:Class is the rdf:type of “Animal”, both “Animal” and rdfs:Class are rdfs:subClassOf

rdfs:Resource, where rdfs:Class is a modelling primitive and “Animal” is an user-defined ontology class.

2.3 DAML+OIL Data Model

DAML+OIL is an expressive Web ontology language in the logical layer. It builds on earlier W3C standards such as RDF and RDFS, and extends these languages with much richer modelling primitives. DAML+OIL inherits many aspects from OIL, and provides modelling primitives commonly found in frame-based languages. It has a clean and well defined semantics based on description logics.

A complete description of the data model of DAML+OIL is beyond the scope of this paper. However, we will illustrate how DAML+OIL extends RDFS by introducing some new subclasses of rdfs:Class and rdf:Property. One of the most important classes that DAML+OIL introduces is daml:Datatype. DAML+OIL divides the universe into two disjoint parts, the object domain and the datatype domain. The object domain consist of objects that are members of classes described in DAML+OIL. The datatype domain consists of the values that belong to XML Schema datatypes. Both daml:Class (object class) and daml:Datatype are rdfs:subClassOf rdfs:Class. Accordingly, properties in DAML+OIL should be either object properties, which relate objects to objects and are instances of daml:ObjectProperty; or datatype property, which relate objects to datatype values and are instances of daml:Datatype- Property. Both daml:ObjectProperty and daml:DatatypeProperty are rdfs:subClassOf rdf:Pro- perty. For example, we can define a datatype property called “birthday”:

```
<daml:DatatypeProperty rdf:ID="birthday">
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProp-
  erty"/>
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</daml:DatatypeProperty>
```

Besides being an instance of daml:datatypeProperty, the “birthday” property is also an instance of daml:UniqueProperty, which means that “birthday” can only have one (unique) value for each instance of the “Animal” class. In fact, daml:UniqueProperty is so useful that some people even want to use it to refine DAML+OIL predefined properties, e.g. daml:maxCardinality:

```
<rdf:Property rdf:about="#maxCardinality">
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProp-
  erty"/>
</rdf:Property>
```

This statement seems obviously right, however, it is wrong because the semantics of daml:Uniq- ue-Property requires that only the ontology properties can be regarded as its instances (cf. van Harmelen et al. [21]). This is the so called “layer mistake”. The reason that people can easily make the above “layer mistake” lies in the fact that the schema layer language RDFS doesn’t *distinguish* the modelling information in the ontology level and that in the language level. Another example is what we had mentioned before in Figure 3, it is not appropriate that both rdfs:Class and “Animal” are rdfs:subClassOf rdfs:Resource.

It is the existence of the dual roles of some RDFS modelling elements, e.g. rdfs:subClassOf, that makes RDFS have unclear semantics. This partially explains why Brickley and Guha [2] didn’t define the semantics of RDFS. We should stress that DAML+OIL is built on top of the *syntax* of RDFS, but not the *semantics* of RDFS. On the contrary, RDFS relies on DAML+OIL to give semantics to its modelling primitives. In other words, DAML+OIL not only defines the semantics of its newly introduced modelling primitives, e.g. daml:UniqueProperty, daml:maxCardinality etc., but also the modelling primitives of RDFS, e.g. rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range etc (van Harmelen et al. [see 21]). This breaks the dependency between logical layer languages and schema

layer languages and indicates that RDFS is not yet a fully qualified schema layer Semantic Web language.

3 Fixed or Non-fixed Metamodeling Architecture?

3.1 The Advantages and Disadvantages of Non-fixed Metamodeling Architecture

The dual roles of some RDFS modelling elements indicate that something might be wrong with the metamodeling architecture of RDFS. The RDFS has a non-fixed metamodeling architecture, which means that it can have possibly infinite layers of classes. The advantage is that it makes itself compact. However, it has at least the following three disadvantages or problems:

1. The class rdfs:Class is an instance of itself. Usually, a class is regarded as a set, and an instance of the class is a member of the set. A Class of classes can be interpreted as a set of sets, which means its members are sets. In RDFS, all classes (including rdfs:Class) are instances of rdfs:Class, which is suspicious by close to Russell paradox. The paradox arises when considering the set of all sets that are not members of themselves. Such a set appears to be a member of itself if and only if it is not a member of itself, hence the paradox.

2. The class rdfs:Resource is a superclass and instance of rdfs:Class at the same time, which means that the super set (rdfs:Resource) is a member of the subset (rdfs:Class).

3. The properties rdfs:subClassOf, rdf:type, rdfs:range and rdfs:domain are used to define both the other RDFS modelling primitives and the ontology, which makes their semantics unclear and makes it very difficult to formalize RDFS. E.g. it is not clear that the semantic of rdfs:subClassOf is a set of binary relationships between two sets of objects or a set of binary relationships between two sets of sets of objects, or else.

As a result, RDFS has no clear semantics, it even rely on DAML+OIL to give itself semantics, which makes RDFS a not so satisfactory schema layer semantic Web language.

3.2 The Advantages and Disadvantages of Fixed Metamodeling Architecture

We can demonstrate the advantages of fixed metamodeling architecture by showing how the problems of RDF Schema mentioned in Section 3.1 are solved under the fixed metamodeling architecture.

The reason that problem 1 exists is that RDFS uses a single primitive rdfs:Class to implicitly represent possibly infinite layers of classes. *But do we really need infinite layers of classes?* In practice, rdfs:Class usually acts as a modelling primitive in the ontology language and is used to define ontology classes (e.g. “Person”). One reasonable solution is to explicitly specify a certain number of layers of *class* primitives, with one being an instance of another, and the *class* primitives in the top layer having no type at all, which means that it is not an instance of anything. It isn’t because it can’t have a type, but because it doesn’t have to have a type, from the pragmatic point of view. This is the main difference between the fixed and non-fixed metamodeling architecture.

But how many class primitives do we really need? Problem 2 indicates that we need at least *two* class primitives in different metamodeling layers–one as the type of rdfs:Resource, the other as a subclass of rdfs:Resource. In fact, in the four-layer metamodeling architecture of UML, there exist two class primitives in different metamodeling layers, which are *Class* in metamodel layer and *MetaClass* in meta-metamodel layer (see Section 3.3). In practice, it has not been found useful to have more than two class primitives in the metamodeling architecture (technology Inc. [19, pg. 298]). Therefore, it is reasonable to explicitly define two class primitives in different metamodeling layers of RDF Schema,

one is MClass in Metalanguage Layer and the other is LClass in Ontology Language Layer[1] (see Section 4.1). This makes RDFS have a similar metamodeling architecture to that of the well known UML, so that it is easy for the modellers to understand.

Problem 3 is mainly about predefined properties. It can be solved by specifying which level of class we intend to refer to when we use these predefined properties. (see Section 4.1).

From the discussion above, we believe that although the schema layer language won't be as compact as it is, there will be several advantages if it has a fixed metamodeling architecture:

1. We don't have to worry about Russell's Paradox.
2. It has clear formalized semantics.
3. DAML+OIL and other logical layer Semantic Web languages can be built on top of both the syntax and semantics of the RDFS with fixed metamodeling architecture.
4. It is similar to the metamodeling architecture of UML, easy to understand and use.

3.3 UML Metamodeling Architecture

The Unified Modelling Language (OMG [17]) is a general-purpose visual modelling language that is designed to specify, visualise, construct and document the artifacts of a software system. It is a standard object-oriented design language that has gained virtually global acceptance. UML has a four-layer metamodeling architecture.

1) The *Meta-metamodel Layer* forms the foundation for the metamodeling architecture. The primary responsibility of this layer is to define the language for specifying a metamodel. A meta-metamodel can define multiple metamodels, and there can be multiple meta-metamodels associated with each metamodel. Examples of meta-objects in the metamodeling layer are: MetaClass, MetaAttribute.

2) A *Metamodel* is an instance of a Meta-metamodel. The primary responsibility of the Metamodel layer is to define a language for specifying models. Examples of meta-objects in the metamodeling layer are: Class, Attribute.

3) A *Model* is an instance of a Metamodel. The primary responsibility of the Model Layer is to define a language that describes an information domain. Examples in Model layer are class "Person" and property "hasFriend".

4) *User Objects* are an instance of a Model. The primary responsibility of the User Objects Layer is to describe a specific information domain. Examples in User Objects Layer are "John", "Mary" and $\langle$'John', 'Mary'$\rangle$.

The four-layer metamodel architecture is a proven methodology for defining the structure of complex models that need to be reliably stored, shared, manipulated and exchanged (Kobryn [12]). In the next section, we will use the metamodeling methods of UML to build a fixed layer metamodeling architecture for RDFS.

4 Web Ontology Language Data Model with Fixed Metamodeling Architecture

We will now illustrate what the data model of an RDF-based Web ontology language will look like under the fixed metamodeling architecture.

[1] In this sense, there are three kinds of classes: meta classes in the Metalanguage Layer, language classes in the Language Layer and ontology classes, which are instance of LClass, in Ontology Layer.

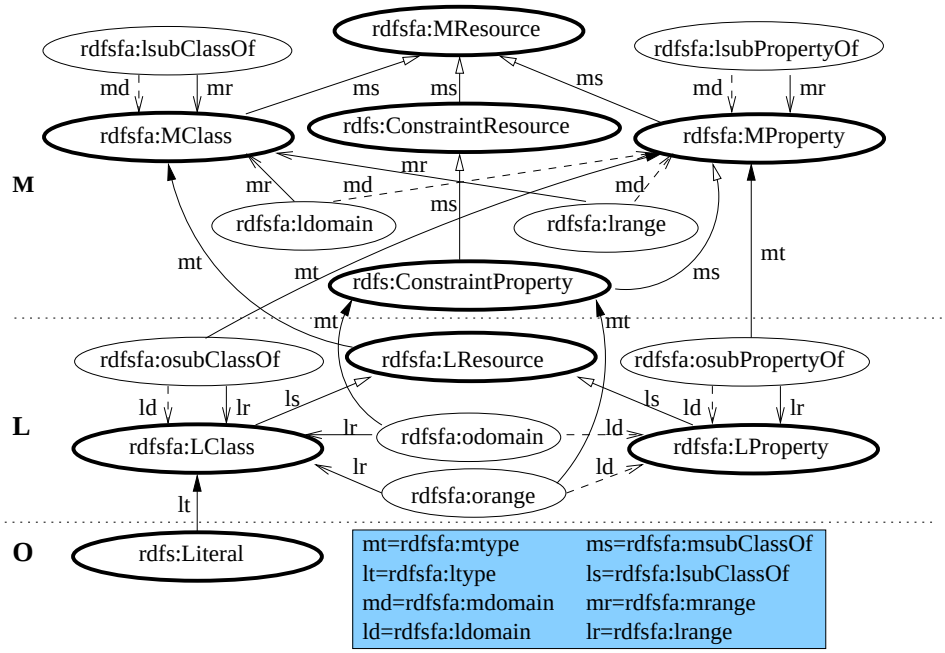

Figure 4: Directed Labeled Graph of RDFS(FA)

4.1 RDF Schema Data Model with Fixed Metamodeling Architecture

Firstly, we will map the original RDFS into RDFS under the Fixed metamodeling Architecture (or RDFS(FA) for short). One principle during this mapping is that we try to minimise the changes we make to RDFS.

As we discussed in Section 3.2, we believe it is reasonable to define a four-layer metamodeling architecture for RDFS(FA). These four metamodeling layers are:

1. The *Metalanguage Layer* (M Layer, corresponding to the Meta-metamodel Layer in UML) forms the foundation for the metamodeling architecture. Its primary responsibility is to define the language layer. All the modelling primitives in this layer have no types (see Section 3.2). Examples of modelling primitives in this layer are rdfsfa:MClass and rdfsfa:MProperty.

2. The *Language Layer* (L Layer, corresponding to the Metamodel Layer in UML), or *Ontology Language Layer*, is an instance of the Metalanguage Layer. Its primary responsibility is to define a language for specifying ontologies. Examples of modelling primitives in this layer are rdfsfa:LClass, rdfsfa:LProperty. Both of them are instances of rdfsfa:MClass.

3. The *Ontology Layer* (O Layer, corresponding to the Model Layer in UML) is an instance of Language Layer. Its primary responsibility is to define a language that describes a specific domain, i.e. an ontology. Examples of modelling primitives in this layer are “Person” and “Car”, which are instances of rdfsfa:LClass, and “hasFriend”, which is an instance of rdfsfa:LProperty.

4. The *Instance Layer* (I Layer, corresponding to the User Objects Layer in UML) is an instance of Ontology Layer. Its primary responsibility is to describe a specific domain, in terms of the ontology defined in the Ontology Layer. Examples in this layer are “Mary”, “John” and hasFriend$\langle$ ‘John’,‘Mary’$\rangle$.

RDFS(FA) is illustrated in Figure 4. We map the modelling primitives of RDFS to the primitives in corresponding metamodeling layers of RDFS(FA), so that *no* modelling primitives will have dual

roles in the metamodeling architecture of RDFS(FA).

First, we map rdfs:Class and its instance primitives in RDFS to the metamodeling architecture of RDFS(FA) as follows:

1. rdfs:Class is mapped to rdfsfa:MClass in Metalanguage Layer and rdfsfa:LClass in Language Layer, so that rdfsfa:LClass is an instance of rdfsfa:MClass.

```
<rdf:Description rdf:ID="MClass">
  <rdfs:comment>The concept of class in the Metalanguage Layer.
  </rdfs:comment>
  <rdfsfa:msubClassOf rdf:resource="#MResource"/>
</rdf:Description>
<rdfsfa:MClass rdf:ID="LClass">
  <rdfs:comment>The concept of class in the Language Layer.</rdfs:comment>
  <rdfsfa:lsubClassOf rdf:resource="#LResource"/>
</rdfsfa:MClass>
```

2. rdfs:Resource is mapped to rdfsfa:MResource in the Metalanguage Layer and rdfsfa:LResource in Language Layer, so that rdfsfa:MResource is the super class of all the modelling primitives in the Metalanguage Layer, while rdfsfa:LResource is an instance of rdfsfa:MClass and the superclass of rdfsfa:LClass.

```
<rdf:Description rdf:ID="MResource">
  <rdfs:comment>The most general resource in the Metalanguage Layer.
  </rdfs:comment>
</rdf:Description>
<rdfsfa:MClass rdf:ID="LResource">
  <rdfs:comment>The most general resource in the Language Layer.
  </rdfs:comment>
</rdfsfa:MClass>
```

3. The rdfs:Property is mapped to rdfsfa:MProperty in the Metalanguage Layer and rdfsfa:LProperty in the Language Layer.

```
<rdf:Description rdf:ID="MProperty">
  <rdfs:comment>The concept of property in the Metalanguage Layer.
  </rdfs:comment>
  <rdfsfa:msubClassOf rdf:resource="#MResource"/>
</rdf:Description>
<rdfsfa:MClass rdf:ID="LProperty">
  <rdfs:comment>The concept of property in the Language Layer.
  </rdfs:comment>
  <rdfsfa:lsubClassOf rdf:resource="#LResource"/>
</rdfsfa:MClass>
```

4. The rdfs:ConstraintResource is in the Metalanguage Layer, where it is rdfsfa:msubClassOf rdfsfa: MResource.

```
<rdf:Description rdf:ID="ConstraintResource">
  <rdfsfa:msubClassOf rdf:resource="#MResource"/>
</rdf:Description>
```

5. The rdfs:ConstraintProperty is in the Metalanguage Layer, where it is rdfsfa:msubClassOf rdfsfa: MProperty and rdfs:ConstraintResource.

```
<rdf:Description rdf:ID="ConstraintProperty">
  <rdfsfa:msubClassOf rdf:resource="#MProperty"/>
  <rdfsfa:msubClassOf rdf:resource="#ConstraintResource"/>
</rdf:Description>
```

As shown in Figure 4, modelling primitives are divided into three groups in the Metalanguage Layer, Language Layer and Ontology Layer. rdfsfa:LClass is not an instance of itself, but an instance of rdfsfa:MClass. rdfsfa:LResource is an instance of rdfsfa:MClass and a super class of rdfsfa:LClass. In general, there are three kinds of "classes" in the metamodeling architecture of RDFS(FA)[2]: *meta classes* in the Metalanguage Layer (e.g. rdfsfa:MClass, rdfsfa:MProperty), *language classes* in the Language Layer (instances of rdfsfa:MClass, e.g. rdfsfa:LClass, rdfsfa:LProperty) and *ontology class* in the Ontology Layer (instance of rdfsfa:LClass, e.g. "Person", "Car").

In order to solve problem 3 mentioned in Section 3.1, we need to be able to specify which kind of class (out of the three kinds of "classes" mentioned above) we want to refer to. In RDFS(FA), we add the layer prefix (e.g. m- for Metalanguage Layer, l- for Language Layer etc.) on the properties when we use the predefined property primitives. Based on the above principle, we can map the property primitives in RDFS to the metamodeling architecture of RDFS(FA) as follows:

1. rdfs:domain is a set of binary relationships between instances of rdf:Property and rdfs:Class. As classes and properties occur in three different layers of RDFS(FA), rdfs:domain is mapped to three different properties in RDFS(FA): rdfsfa:odomain, rdfsfa:ldomain and rdfsfa:mdomain. As shown in Figure 4, the rdfsfa:ldomain is defined in the Metalanguage Layer and used in the Language Layer, while rdfsfa:odomain is defined in the Language Layer and used in the Ontology Layer (see Figure 5).

   ```
   <rdfs:ConstraintProperty rdf:ID="odomain">
     <rdfs:comment>This is how we specify that all instances of a particular
     ontology property describes instances of a particular ontology class.
     </rdfs:comment>
   </rdfs:ConstraintProperty>
   <rdf:Description rdf:ID="ldomain">
     <rdfs:comment>This is how we specify that all instances of a particular
     language property describes instances of a particular language class.
     </rdfs:comment>
   </rdf:Description>
   <rdf:Description rdf:ID="mdomain">
     <rdfs:comment>This is how we specify that all instances of a particular
     meta property describes instances of a particular meta class.
     </rdfs:comment>
   </rdf:Description>
   ```

2. Similarly, rdfs:range is mapped to rdfsfa:orange, rdfsfa:lrange and rdfsfa:mrange.

   ```
   <rdfs:ConstraintProperty rdf:ID="orange">
     <rdfs:comment>This is how we specify that all instances of a particular
     ontology property have values that are instances of a particular ontolo-
     gy class.</rdfs:comment>
   </rdfs:ConstraintProperty>
   <rdf:Description rdf:ID="lrange">
     <rdfs:comment>This is how we specify the values of an instance of a
     particular language property have values that are instances of a
     particular language class. </rdfs:comment>
   ```

[2] Accordingly, there are three kinds of "properties" as well.

```
</rdf:Description>
<rdfsfa:MProperty rdf:ID="mrange">
  <rdfs:comment>This is how we specify the values of an instance of a
  particular meta property should be instances of a particular meta class.
  </rdfs:comment>
</rdf:Description>
```

3. rdf:type is a set of binary relationship between resource and rdfs:Class. As RDFS(FA) has meta classes, language classes and ontology classes, rdf:type is mapped to rdfsfa:otype, rdfsfa:ltype and rdfsfa:mtype. E.g. in Figure 4, rdfsfa:MClass is the rdfsfa:mtype of rdfsfa:LResource and rdfsfa:LClass.

```
<rdfsfa:MProperty rdf:ID="otype">
  <rdfs:comment>Indicates membership of an instance of rdfsfa:LClass
  </rdfs:comment>
  <rdfsfa:lrange rdf:resource="#LClass"/>
</rdfsfa:MProperty>
<rdf:Description rdf:ID="ltype">
  <rdfs:comment>Indicates membership of rdfsfa:LClass or rdfsfa:LProperty
  </rdfs:comment>
  <rdfsfa:mrange rdf:resource="#MClass"/>
</rdf:Description>
<rdf:Description rdf:ID="mtype">
  <rdfs:comment>Indicates membership of rdfsfa:MClass or rdfsfa:MProperty.
  </rdfs:comment>
</rdf:Description>
```

4. rdfs:subClassOf is a set of binary relationship between two instances of rdfs:Class, so rdfs:subClassOf is mapped to rdfsfa:osubClassOf and rdfsfa:lsubClassOf . E.g. in Figure 4, rdfsfa:LClass is an rdfsfa:lsubClassOf rdfsfa:LResource and rdfsfa:MClass is an rdfsfa:msubClassOf rdfs:MResource.

```
<rdfsfa:MProperty rdf:ID="osubClassOf">
  <rdfs:comment>Binary relationship between two ontology classes.
  </rdfs:comment>
  <rdfsfa:ldomain rdf:resource="#LClass"/>
  <rdfsfa:lrange rdf:resource="#LClass"/>
</rdfsfa:MProperty>
<rdf:Description rdf:ID="lsubClassOf">
  <rdfs:comment>Binary relationship between two language classes.
  </rdfs:comment>
  <rdfsfa:mdomain rdf:resource="#MClass"/>
  <rdfsfa:mrange rdf:resource="#MClass"/>
</rdf:Description>
<rdf:Description rdf:ID="msubClassOf">
  <rdfs:comment>Binary relationship between two meta classes.
  </rdfs:comment>
</rdf:Description>
```

5. Similarly, rdfs:subPropertyOf is a set of binary relationships between instances of rdf:Property, so it is mapped to rdfsfa:osubPropertyOf, rdfsfa:lsubPropertyOf and rdfsfa:msubPropertyOf .

```
<rdfsfa:MProperty rdf:ID="osubPropertyOf">
  <rdfs:comment>Binary relationship between two ontology properties.
```

```
    </rdfs:comment>
    <rdfsfa:ldomain rdf:resource="#LProperty"/>
    <rdfsfa:lrange rdf:resource="#LProperty"/>
  </rdfsfa:MProperty>
  <rdf:Description rdf:ID="lsubPropertyOf">
    <rdfs:comment>Binary relationship between two language properties.
    </rdfs:comment>
    <rdfsfa:mdomain rdf:resource="#MProperty"/>
    <rdfsfa:mrange rdf:resource="#MProperty"/>
  </rdf:Description>
  <rdf:Description rdf:ID="msubPropertyOf">
    <rdfs:comment>Binary relationship between two meta properties.
    </rdfs:comment>
  </rdf:Description>
```

6. The rdfs:comment, rdfs:label, rdfs:seeAlso and rdfs:isDefinedBy are treated as documentation in RDFS, and are not related to the semantics of RDFS(FA), so we are not going to discuss them in this paper.

Below is an RDFS(FA) version of the "Person–hasFriend" example. As with other Web ontology languages, these statements describe resources in the Ontology Layer and the Instance Layer.

```
<rdfsfa:LClass rdf:ID="Animal">
  <rdfs:comment>This class of animals is illustrative of a number of
  ontological idioms.</rdfs:comment>
</rdfsfa:LClass>
<rdfsfa:LClass rdf:ID="Person">
  <rdfs:osubClassOf rdf:resource="#Animal"/>
</rdfsfa:LClass>
<rdf:Property rdf:ID="hasFriend">
  <rdfsfa:odomain rdf:resource="#Person"/>
  <rdfsfa:orange rdf:resource="#Person"/>
</rdf:Property>
<rdf:Description rdf:ID="John">
  <rdfsfa:otype rdf:resource="#Person"/>
  <rdfs:comment>John is a person.</rdfs:comment>
</rdf:Description>
<rdf:Description rdf:ID="Mary">
  <rdfsfa:otype rdf:resource="#Person"/>
  <rdfs:comment>Mary is a person.</rdfs:comment>
</rdf:Description>
<rdf:Description about="#John">
  <hasFriend rdf:resource="#Mary"/>
</rdf:Description>
```

In the Ontology Layer, "Animal" and "Person" are ontology classes, so they are instances of rdfsfa:LClass. The ontology class "Person" is the rdfsfa:odomain and rdfsfa:orange of the property "hasFriend", so both the values of and resource described by instances of "hasFriend" are instances of "Person". In the Instance Layer, the rdfsfa:otype of individuals such as "John" and "Mary" is the ontology class "Person". Figure 5 is a directed labeled graph of the above RDFS(FA) statements. Here the rdfsfa:mtype of rdfsfa:LClass is the meta class rdfsfa:MClass, the rdfsfa:ltype of "Person" is the language class rdfsfa:LClass and the rdfsfa:otype of "John" is the ontology class "Person". The language class rdf:Property is rdfsfa:lsubClassOf the language class rdfs:Resource while the ontology

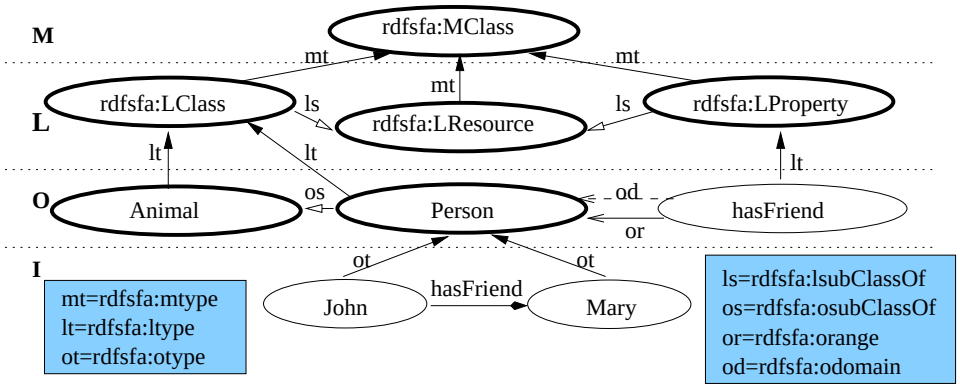

Figure 5: A "Person–hasFriend" example in RDFS(FA)

class "Person" is rdfsfa:osubClassOf the ontology class "Animal". This example clearly shows that the modelling primitives in RDFS(FA) no longer have dual roles. Thus a clear semantics can be given to them.

Note that, as in RDFS (see Section2.2), we can define rdfsfa:otype, rdfsfa:ltype and rdfsfa:mtype properties within RDFS(FA) in an implicit way as well. E.g., individual "John" can also be defined as

```
<Person rdf:ID="John">
  <rdfs:comment>John is a person.</rdfs:comment>
</Person>
```

Here "Person" is an *ontology* class, so the above expressions use an implicit way to define rdfsfa:otype property.

4.2 Data Model Semantics of RDFS(FA)

In this section, we use a Tarski style ([18]) model theoretic semantics to interpret the data model of RDFS(FA). Classes and properties are taken to refer to sets of objects in the domain of interests and sets of binary relationships (or tuples) between these objects.

In RDFS(FA), the meaning of individuals, pairs of individuals, ontology classes and properties is given by an interpretation $\mathcal{I}$, which is a pair$(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain (a set) and $\cdot^{\mathcal{I}}$ is an interpretation function, which maps every individual name x to an object in the domain $\Delta^{\mathcal{I}}$:

$$x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$$

every pair of individual names x, y to a pair of objects in $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$:

$$\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

every ontology class name OC to a subset of $\Delta^{\mathcal{I}}$:

$$OC^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$

every ontology property name OP to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$:

$$OP^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}.$$

In the Language Layer, the interpretation function $\cdot^{\mathcal{I}}$ maps rdfsfa:LClass (LC) to $2^{\Delta^{\mathcal{I}}}$:

$$LC^{\mathcal{I}} = 2^{\Delta^{\mathcal{I}}}$$

rdfsfa:LProperty (LP) to $2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$:

$$LP^{\mathcal{I}} = 2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$$

rdfsfa:LResource (LR) to $LC^{\mathcal{I}} \cup LP^{\mathcal{I}}$:

$$LR^{\mathcal{I}} = LC^{\mathcal{I}} \cup LP^{\mathcal{I}}$$

so that the interpretation of every possible ontology class ($\text{OC}^{\mathcal{I}}$) is an element of the interpretation of rdfsfa:LClass ($\text{LC}^{\mathcal{I}}$), the interpretation of every possible ontology property ($\text{OP}^{\mathcal{I}}$) is an element of the interpretation of rdf:Property ($\text{LP}^{\mathcal{I}}$). Note that $\text{LR}^{\mathcal{I}}$ is interpreted as the union of $\text{LC}^{\mathcal{I}}$ and $\text{LP}^{\mathcal{I}}$, and not as $2^{\Delta^{\mathcal{I}} \cup (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})}$, so instances of rdfsfa:LResource must be either ontology classes (sets of objects), or ontology properties (sets of tuples), and can't be interpreted as a "mixture" of sets of objects and tuples.

In the Metalanguage Layer, interpretation function $\cdot^{\mathcal{I}}$ maps rdfsfa:MClass (MC) to $2^{LR^{\mathcal{I}}}$:

$$MC^{\mathcal{I}} = 2^{LR^{\mathcal{I}}}$$

rdfsfa:MProperty (MP) to $2^{LC^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LC^{\mathcal{I}} \times LP^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LP^{\mathcal{I}}}$:

$$MP^{\mathcal{I}} = 2^{LC^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LC^{\mathcal{I}} \times LP^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LP^{\mathcal{I}}}$$

rdfsfa:MResource (MR) to $MC^{\mathcal{I}} \cup MP^{\mathcal{I}}$:

$$MR^{\mathcal{I}} = MC^{\mathcal{I}} \cup MP^{\mathcal{I}}$$

rdfs:ConstraintResource (CR) to subset of $MR^{\mathcal{I}}$:

$$CR^{\mathcal{I}} \subseteq MR^{\mathcal{I}}$$

Predefined Property	Interpretation	Semantic Constraint
osubClassOf (OSC)	$OSC^{\mathcal{I}} \subseteq LC^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \rangle \in OSC^{\mathcal{I}}$ iff. $C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
lsubClassOf (LSC)	$LSC^{\mathcal{I}} \subseteq MC^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \rangle \in LSC^{\mathcal{I}}$ iff. $C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
msubClassOf (MSC)	$MSC^{\mathcal{I}} \subseteq 2^{MR^{\mathcal{I}}} \times 2^{MR^{\mathcal{I}}}$	$\langle C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \rangle \in MSC^{\mathcal{I}}$ iff. $C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
osubPropertyOf (OSP)	$OSP^{\mathcal{I}} \subseteq LP^{\mathcal{I}} \times LP^{\mathcal{I}}$	$\langle P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \rangle \in OSP^{\mathcal{I}}$ iff. $P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \in LP^{\mathcal{I}}$ and $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
lsubPropertyOf (LSP)	$LSP^{\mathcal{I}} \subseteq MP^{\mathcal{I}} \times MP^{\mathcal{I}}$	$\langle P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \rangle \in LSP^{\mathcal{I}}$ iff. $P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \in MP^{\mathcal{I}}$ and $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
msubPropertyOf (MSP)	$MSP^{\mathcal{I}} \subseteq \Phi \times \Phi$	$\langle P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \rangle \in LSP^{\mathcal{I}}$ iff. $P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \in \Phi$ and $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
odomain (OD)	$OD^{\mathcal{I}} \subseteq LP^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in OD^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in LP^{\mathcal{I}}$, $C^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
ldomain (LD)	$LD^{\mathcal{I}} \subseteq MP^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in LD^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in MP^{\mathcal{I}}$, $C^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
mdomain (MD)	$MD^{\mathcal{I}} \subseteq \Phi \times 2^{MR^{\mathcal{I}}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in MD^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in \Phi$, $C^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
orange (ORG)	$ORG^{\mathcal{I}} \subseteq LP^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in ORG^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in LP^{\mathcal{I}}$, $C^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
lrange (LRG)	$LRG^{\mathcal{I}} \subseteq MP^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in LRG^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in MP^{\mathcal{I}}$, $C^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
mrange (MRG)	$MRG^{\mathcal{I}} \subseteq \Phi \times 2^{MR^{\mathcal{I}}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in MRG^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in \Phi$, $C^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
otype (OT)	$OT^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle x^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in OT^{\mathcal{I}}$ iff. $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}, C^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $x^{\mathcal{I}} \in C^{\mathcal{I}}$
ltype (LT)	$LT^{\mathcal{I}} \subseteq LR^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle R^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in LT^{\mathcal{I}}$ iff. $R^{\mathcal{I}} \in LR^{\mathcal{I}}, C^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $x^{\mathcal{I}} \in C^{\mathcal{I}}$
mtype (MT)	$MT^{\mathcal{I}} \subseteq MR^{\mathcal{I}} \times 2^{MR^{\mathcal{I}}}$	$\langle R^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in MT^{\mathcal{I}}$ iff. $R^{\mathcal{I}} \in MR^{\mathcal{I}}, C^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $x^{\mathcal{I}} \in C^{\mathcal{I}}$

Figure 6: Semantics of Predefined Properties in RDFS(FA)

rdfs:ConstraintProperty (CP) to subset of both $CR^{\mathcal{I}}$ and $MP^{\mathcal{I}}$:

$$CP^{\mathcal{I}} \subseteq CR^{\mathcal{I}} \cap MP^{\mathcal{I}}$$

so that the interpretations of rdfsfa:LClass ($LC^{\mathcal{I}}$), rdfsfa:LProperty ($LP^{\mathcal{I}}$) and rdfsfa:LResource ($LR^{\mathcal{I}}$) are all elements of the interpretation of rdfsfa:MClass ($MC^{\mathcal{I}}$), and all the possible pairs of subsets of $LC^{\mathcal{I}}$ and subsets of $LP^{\mathcal{I}}$ are elements of $MP^{\mathcal{I}}$.

Unlike rdfs:Class in RDFS, classes in RDFS(FA) have clear semantics. Clean semantics can also be given to the predefined properties of RDFS(FA) as shown in Figure 6, where

$$\Phi = 2^{MC^{\mathcal{I}} \times MC^{\mathcal{I}}} \cup 2^{MC^{\mathcal{I}} \times MP^{\mathcal{I}}} \cup 2^{MP^{\mathcal{I}} \times MC^{\mathcal{I}}} \cup 2^{MP^{\mathcal{I}} \times MP^{\mathcal{I}}}$$

As mentioned above, in order to specify which kind of class we want to refer to when we use the predefined properties, we add the layer prefixes to these properties. Subclass-of and subproperty-of are the subset relationship between the classes or properties within the same layer. Domain and range are foundation modelling primitives of RDFS(FA) properties, which can be used to specify two classes that a certain property can describe/use in descriptions in a certain layer. Type is a special cross-layer property, which is used to link instances to classes.

4.3 DAML+OIL Data Model with Fixed Metamodeling Architecture

With a fixed metamodeling architecture, RDFS(FA) has its own semantics and makes itself a fully qualified schema layer Semantic Web language. Thus, DAML+OIL (or any other logical layer Semantic Web language) can be built on both its syntax and semantics.

From the point of view of metamodeling architecture, the modelling primitives that DAML+ OIL introduces are mainly located in the Language Layer (a complete description of the DAML+OIL data model with fixed metamodeling architecture will be given in a forthcoming paper). daml:Class is rdfsfa:lsubClassOf rdfsfa:LClass and daml:ObjectProperty is rdfsfa:lsubClassOf rdfsfa:LProperty; both daml:Datatype and daml:DatatypeProperty are rdfsfa:lsubClassOf rdfsfa:LResource. The above four are disjoint with each other. The "birthday" property lies in the Ontology Layer and can be defined in the following way:

```
<daml:DatatypeProperty rdf:ID="birthday">
  <rdfsfa:ltype rdf:resource="http://www.daml.org/2001/03/daml+oil#Unique-
  Property"/>
  <damlfa:odatadomain rdf:resource="#Animal"/>
  <damlfa:odatarange rdf:resource="http://www.w3.org/2000/10/XMLSchema#da-
  te"/>
</daml:DatatypeProperty>
```

where damlfa:odatadomain is a set of binary relationships between instances of daml:DatatypeProperty and daml:Class, and damlfa:odatarange is a set of binary relationships between instances of daml:DatatypeProperty and daml:Datatype.

On the other hand, it is clear that Language Layer primitives can't be used to define/modify other Language Layer primitives, e.g. Uniqueproperty can not be used to restrict the numbers of values of the maxCardinality as follows:

```
<rdfsfa:MProperty rdf:about="#maxCardinality">
  <rdfsfa:ltype rdf:resource="http://www.daml.org/2001/03/daml+oil#Unique-
  Property"/>
</rdfsfa:MProperty>
```

In RDFS(FA), Language Layer properties can only be defined using Metalanguage Layer primitives, for which DAML+OIL doesn't provide any semantics. This is not clear in RDFS, where modellers

might be tempted to think that they can modify DAML+OIL in the above manner, exploiting the semantics of DAML+OIL itself. To solve the above problem, one can define MUniqueProperty in the Metalanguage Layer and then set daml:maxCardinality as its instance.

In short, RDFS(FA) not only provides a firm semantic basis for DAML+OIL, it also eradicates the possibility of the "layer mistake" mentioned above.

5 Discussion

A fixed layer metamodeling architecture for RDFS is proposed in this paper. We demonstrate how to map the original RDFS to RDFS under the Fixed metamodeling Architecture (RDFS(FA)) and give a clear model-theoretic semantics to RDFS(FA). We believe that although RDFS(FA) won't be as compact as RDFS, there will be several advantages if RDFS has a fixed metamodeling architecture:

1. We don't have to worry about Russell's Paradox. (Other ways of thinking may include non well-founded sets.)
2. RDFS(FA) has a clear formalized semantics.
3. DAML+OIL and other logical layer Semantic Web languages can be built on top of both the syntax and semantics of RDFS(FA).
4. The metamodeling architecture of RDFS(FA) is similar to that of UML, so it is easier for people to understand and use.

Some other papers have also talked about UML and the Web ontology language. Chang [4] summarized the relationship between RDF-Schema and UML. Melnik [14] tried to make UML "RDF-compatible", which allows mixing and extending UML models and the language elements of UML itself on the Web in an open manner. Cranefield and Purvis [5] investigated the use of UML and OCL (Object Constraint Language) for the representation of information system ontologies. Cranefield [6] proposed UML as a Web ontology language. Cranefield [7] described technology that facilitates the application of object-oriented modelling, and UML in particular, to the Semantic Web. However, none of these works address the problem of the metamodeling architecture of RDFS itself.

It is well known that UML has a well-defined metamodeling architecture (Kobryn [12]). It refines the semantic constructs at each layer, provides an infrastructure for defining metamodel extensions, and aligns the UML metamodel with other standards based on a four-layer metamodeling architecture, such as the Case Data Interchange Format (EIA [8]), Meta Object Facility (MOF-Parners [15]) and XMI Facility for model interchange (XMI-Parners [23]).

However, We believe Semantic Web languages and UML have different motivation and application domain. Besides the metamodeling architecture, Semantic Web languages also have a *functional architecture*. Within this functional architecture, RDF is a good candidate for the metadata layer language, while UML is obviously not designed as a metadata language. The schema layer languages must support global properties (anyone can say anything about anything) rather than the local ones, while the considerations of UML mainly focus on the local properties. The modelling primitives of logical layer languages, e.g. OIL and DAML+OIL, are carefully selected so that they can be mapped onto very expressive description logics (DLs), so as to facilitate the provision of reasoning support; on the UML side, reasoning over OCL is still under research.

Therefore, we prefer to enhance Web ontology languages by using the methodologies in UML, rather than making UML a component in Web ontology languages. Accordingly, we have used the metamodeling methods of UML to build a fixed layer metamodeling architecture for RDFS in this paper. Further research will include a detailed study of the data model of DAML+OIL based on RDFS(FA) and the reasoning support provided by corresponding Description Logics.

References

[1] Tim Berners-lee. Semantic Web Road Map. W3C Design Issues. URL http://www.w3.org/DesignIssues/Semantic.html, Oct. 1998.

[2] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommentdation, URL http://www.w3.org/TR/rdf-schema, Mar. 2000.

[3] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. can Harmelen, and I. Horrocks. Enabling knowledge representation on the Web by extending RDF Schema, Nov. 2000.

[4] Walter W. Chang. A Discussion of the Relationship Between RDF-Schema and UML. W3C Note, URL http://www.w3.org/TR/NOTE-rdf-uml/, Aug. 1998.

[5] S. Cranefield and M. Purvis. UML as an ontology modelling language. In *IJCAI-99 Workshop on Intelligent Information Integration*, 1999.

[6] Stephen Cranefield. Networked Knowledge Representation and Exchange using UML and RDF. In *Journal of Digital Information*, volume 1 issue 8. Journal of Digital Information, Feb. 2001.

[7] Stephen Cranefield. UML and the Semantic Web, Feb. 2001. ISSN 1172-6024. Discussion Paper.

[8] EIA. CDIF Framework for Modeling and Extensibility, EIA/IS-107. Nov. 1993.

[9] I. Horocks, D.Fensel, J.Broestra, S.Decker, M.Erdmann, C.Goble, F.van Harmelen, M.Klein, S.Staab, R.Studer, and E.Motta. The Ontology Inference Layer OIL. Aug. 2000.

[10] I. Horrocks. Benchmark Analysis with FaCT. In *TABLEAUX-2000*, number 1847 in LNAI, pages 62–66. Springer-Verlag, 2000.

[11] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[12] Cris Kobryn. UML 2001: A Standardization Odyssey. In *Communications of the ACM*, Vol.42, No. 10, Oct. 1999.

[13] Ora Lassila and Ralph R.Swick. Resource Description Framework (RDF) Model and Syntax Specification. Feb. 1999.

[14] S. Melnik. Representing UML in RDF. URL http://www-db.stanford.edu/~melnik/rdf/uml/, 2000.

[15] MOF-Parners. Meta Object Facility Revision 1.1b1. Jan. 1997.

[16] W. Nejdl, M. Wolpers, and C. Capella. The RDF Schema Specification Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000*, Apr. 2000.

[17] OMG. OMG Unified Modeling Language Specification vertion 1.3. Jun. 1999.

[18] A. Tarski. *Logic, Semantics, Mathemetics: Papers from 1923 to 1938*. Oxford University Press, 1956.

[19] PLATINUM technology Inc. Object Analysis and Design Facility, Response to OMG/OA&D RFP-1,Version 1.0, Jan. 1997.

[20] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 1996.

[21] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. A Model-Theortis Semantics of DAML+OIL(March 2001). Mar. 2001.

[22] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference Description of the DAML+OIL(March 2001) Ontology Markuk Language. DAML+OIL Document, `URL http://www.daml.org/2000/12/reference.html`, Mar. 2001.

[23] XMI-Parners. XML Metadata Interchange (XMI) v. 1.0. Oct. 1998.