Bootstrapping Semantic Web Languages using a UML Meta-Modeling Approach

Sherif Yacoub
Information Infrastructure Laboratory
HP Laboratories Palo Alto
HPL-2002-200
July 15th , 2002*

E-mail: sherif_yacoub@hp.com

RDF, UML, Semantic Web, meta modeling

RDF Schema (RDFS) and RDF are evolving as the de facto languages for the Semantic Web. Whereas RDFS can be used to define concepts in a domain ontology, RDF can be used to define instances of that ontology. In an attempt to make these modeling languages evolvable and extensible, the current languages lack a clear semantic support. While RDF and RDFS are easy to use, they are hard to understand and difficult to validate and extend as a result of their ambiguous semantics. In this paper, we use UML modeling and meta modeling concepts to resolve some of semantic ambiguities and provide support for validation and extension. We describe a UML meta modeling approach that defines five modeling layers and identify how RDFS and RDF as Semantic Web languages fit into the architecture. The approach also defines the meta languages required to create RDFS and RDF and to support extensions to both languages and their constructs. The meta modeling approach is based on concepts inspired by the UML meta modeling architecture. Moreover, we use UML to bootstrap the modeling hierarchy and we use the UML extension mechanisms to describe the models developed at each layer including the RDFS and RDF language constructs.

* Internal Accession Date Only
Approved for External Publication

Bootstrapping Semantic Web Languages using a UML Meta-Modeling Approach

Sherif Yacoub

Hewlett-Packard Labs, MS 1126,
1501 Page Mill Rd.,
Palo Alto, CA 94304, USA
sherif_yacoub@hp.com

Abstract. RDF Schema (RDFS) and RDF are evolving as the de facto languages for the Semantic Web. Whereas RDFS can be used to define concepts in a domain ontology, RDF can be used to define instances of that ontology. In an attempt to make these modeling languages evolvable and extensible, the current languages lack a clear semantic support. While RDF and RDFS are easy to use, they are hard to understand and difficult to validate and extend as a result of their ambiguous semantics. In this paper, we use UML modeling and meta modeling concepts to resolve some of semantic ambiguities and provide support for validation and extension. We describe a UML meta modeling approach that defines five modeling layers and identify how RDFS and RDF as Semantic Web languages fit into the architecture. The approach also defines the meta languages required to create RDFS and RDF and to support extensions to both languages and their constructs. The meta modeling approach is based on concepts inspired by the UML meta modeling architecture. Moreover, we use UML to bootstrap the modeling hierarchy and we use the UML extension mechanisms to describe the models developed at each layer including the RDFS and RDF language constructs.

Keywords: RDF, RDF Schema, Semantic Web, UML, and Meta Modeling.

1 Introduction

The vision of the Semantic Web is to create a web environment in which machines understand and interpret information on the web in a way similar to what web users do. To make this a reality, information has to be presented in a form that can be processed by machines. The Semantic Web approach defines and creates languages for expressing information in a machine readable and processable form [7]. These languages should provide consistent and stable means to create information and reason about this information. To provide such capability, Semantic Web uses a layered data model approach to present data on the web. Each layer provides some semantic information using a data representation language. As a result a four-layer architecture has been developed for data presentation and interpretation [7,5]:

a) The **data** layer, which presents the data itself such as the web pages or user data.
b) The **metadata** layer, which describes information about the data such as the creator of the web page and its date of creation. A simple model has been proposed that uses resources and properties to describe the metadata. For this purpose, the Resource Description Framework (RDF) Syntax and Model [2] is created as the general language for metadata description.
c) The **schema** layer. It was clear that the metadata layer is not sufficient because there are no common ways to describe metadata in general. To solve this problem, we need to define metadata for a particular context (or domain) and describe that domain using another language. For this purpose, the RDF Schema [1] has been created.
d) The **ontology** layer. At this layer, more comprehensive ontology languages are used to describe and reason about the meaning of the resources defined by the schema layer. The Ontology Information Language (OIL) [3,6] and the DARAP Agent Markup Language (DAML) [4] and their combination (DAML-OIL) [14] are examples of such languages.

The Semantic Web community recognizes that ontologies play an important role in this data model architecture. The ontology languages build on top of the two main and fundamental data modeling languages, the RDF and the RDF Schema (RDFS).

However, the RDF and RDFS specifications are difficult to read and formalize as acknowledged in [5,15] for a variety of reasons, which we summarize as follows:

- *Mingling RDFS and RDF constructs.* RDFS elements are defined in terms of RDF elements and vice versa. For example `rdf:Property` is a subclass of `rdfs:Resource`. This makes the two languages dependent on one another and makes it hard to describe one language on its own. This is one reason the specification is hard to read.
- *Self-Instantiation and Referencing.* For instance, in RDFS the `rdfs:Class` is an instance of itself which makes it difficult for object oriented developer to accept that a construct is a class and an object at the same time!
- *Ambiguous Relationships.* In RDFS, an element can be an instance of another element, which is a subclass of the first. For example, `rdfs:Class` is a subclass of `rdfs:Resource` but `rdfs:Resource` is an instance of `rdfs:Class`.
- *Layer Mistake for Classes.* Using the current specification, the user can define his own class, which will be treated at the same level as a class describing the language itself. For example, if the user creates a resource `Mammal` as an instance of `rdfs:Class` and a subclass of `rdfs:Resource`, we end up with the `Mammal` and the `rdfs:Class` at the same modeling level while the first is a user defined class and the later is an RDFS defined class.
- *Layer Mistake for Properties.* Similar to the layer mistake with classes, the RDFS has a layer mistake in using properties. For example, the `rdfs:domain` and `rdfs:range` properties are used in defining constraints in the RDFS language itself and at the same time could be used by a user to define constraints on his domain ontology.

To solve some of these problems, a proposal is developed by Pan *et.al* [5] to distinguish *metadata* models from *meta modeling* by creating a fixed architecture for

modeling languages in the Semantic Web. Though their proposal is inspired by the Unified Modeling Language (UML) meta modeling architecture[13], Pan *et.al.* build their architecture using Directed Label Graphs (DLG) and not using the UML itself. In addition, the top layer in their architecture is not bootstrapped from any other language; this leaves it to varieties of interpretations. Pan *et.al.* only address a subset of the RDFS constructs. They do not address RDF constructs or the full RDFS syntax.

In this paper, we develop a meta modeling approach for the Semantic Web that is based on UML concepts. We use: a) the UML meta modeling concepts to structure the Semantic Web meta modeling layers, b) the UML models and extension mechanisms to develop the languages at each layer of the meta model, and c) the UML modeling elements to bootstrap the top layer of the meta model. We describe all elements of the RDFS and the RDF languages and identify their relevance to other meta modeling layers. Hence, UML modeling concepts, extension mechanism, and meta modeling proves to provide a solid foundation for analyzing Semantic Web languages.

Section 2 describes the layers of the meta model. In sections 3 through 7, we use UML to describe each of those layers. In Section 8, we discuss the usefulness and applications of the meta modeling approach. Finally, we summarize related work and ideas for future research in sections 9 and 10.

2 The Meta Modeling Approach

Figure 1 illustrates the various layers for the meta modeling approach.

Layer 0: The Instance Layer. At this level, we express instances and objects of concepts from a specific domain. These are the concrete objects or documents that we want to specify using RDF. This layer could also be called the User Objects layer. As an example, at this level we can create statements such as: the *homepage* (property) of "*http://hpl.hp.com/staff#John*" (subject) is "*http://hpl.hp.com/~john*" (object) or "*Mary*"(subject) has a *friend* (property) called "*John* " (Object).

Layer 1: The Ontology Layer. At this level, we abstract the concepts and properties in a particular domain and represent them using a domain modeling language. The developed concepts and relationships are represented in what is called an ontology. An ontology is developed for each particular domain. It represents the common terms and language used to describe instances (Layer 0) in that domain. For example, in an ontology for species we may define the concepts `Mammal` and `Human`. An instance of a `Human` (layer 1) could be `John` (layer 0) and `Mary` (layer 0). Similarly we capture property concepts such as `homepage`. Whereas layer 1 resembles a class model in UML, layer 0 resembles an object model.

Layer 2: The Modeling Layer. At the modeling layer, we define the languages that we will use to develop an ontology. RDFS and RDF represent two languages that are defined at this layer. Constructs from both languages are used to define an ontology (this is based on the two specifications as they currently stand). On creating an instance (at layer 0) the user not only uses the ontology created at layer 1 but also he uses the RDF language constructs defined at layer 2.

Layer 3: The Meta Modeling Layer. At this layer, we identify the concepts that tie together all the constructs and elements used in the modeling layer. It defines the meta language that we use to create modeling languages such as RDFS and RDF. This layer is useful if we want to create new languages at the modeling layer and tie them to the RDFS and RDF languages at a higher level. For example, the creation of other languages such as OIL [3,6] and DAML [4] should relate to each other and to the RDFS and RDF languages by using meta models defined at this layer.

Layer 4: The Meta-Meta Modeling Layer. This is a set of simple primitives that we use in creating the meta modeling layer. This layer was not recognized in [5]. In this layer, we find it useful to capture all the concepts used to create meta modeling constructs as discussed later.

Layer 5: The UML layer. Finally, we need a well-defined language with syntax and semantics to describe the elements at layer 4. For example, we should be able to look at any element in layer 4 and easily identify what it means because it is represented in a well-defined language. We choose the UML language to be the top layer of the meta model.

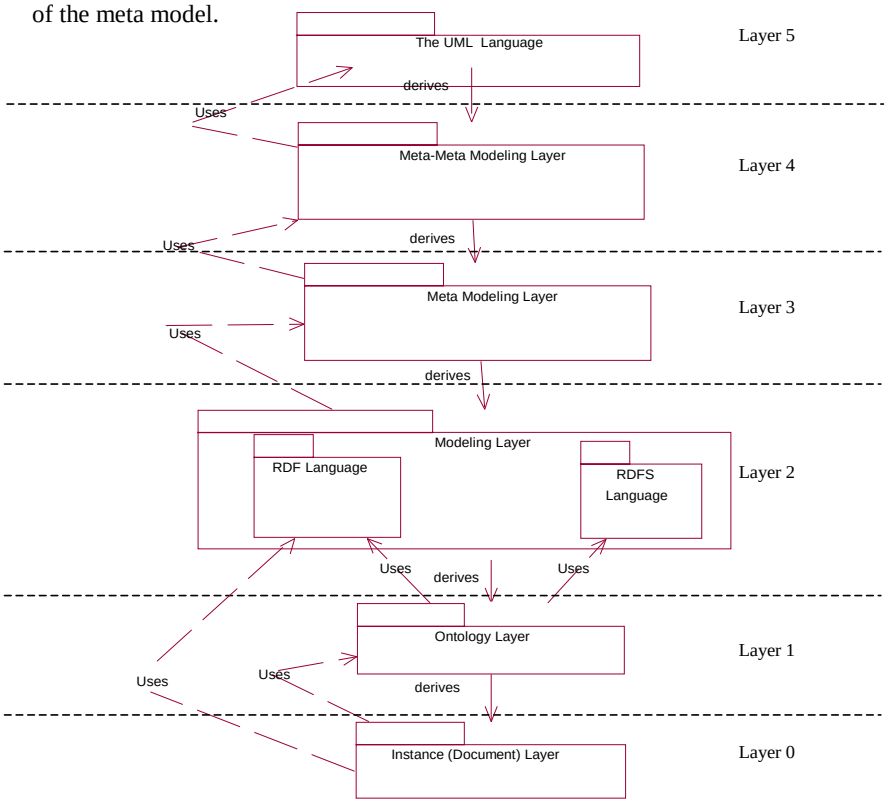

Fig. 1 Various Layers of the meta model

The rationale behind pursuing a meta modeling approach lies in:

1) *Separation of Concern*. Using the meta modeling approach, we can distinguish between constructs used at different levels.
2) *Resolution of the Layer Mistake*. The meta modeling approach provides a solution for the class and property layer problems discussed in section 2, by defining the responsiblies and the functions of elements at each layer.
3) *Extensibility*. With this approach, we are able to distinguish and define where extensions to a language should be made. If we want to develop other modeling languages that uses RDFS and RDF languages we are able to determine whether modifications and extensions are made at the modeling or meta-modeling levels.
4) *UML for Ontology Modeling*. We illustrate how UML could be used in describing the various layers. We also promote the research in using UML to model an ontology and instance of that ontology [9,10].

In the following sections we illustrate how we start from well-formed UML constructs to reach a low level instance of some domain ontology. In developing this meta-modeling approach, we started from low-level models such as RDF documents. We then abstracted those instances up to the modeling layers; i.e. what RDFS and RDF languages offer as building constructs. Then we abstracted the elements needed in those two languages and so on. At some level, we realize that we need to stop by using some predefined modeling constructs. We used the UML constructs for top level. While the development of the meta-modeling approach started from bottom layers and developing abstraction layers on top of it, we find it easier to explain using top down approach as illustrated in the sequel.

3 Layer 5: Bootstrapping using UML

3.1 UML Models

The top layer of the meta model starts with a predefined semantics using UML. UML offers some useful constructs that we will use in developing the lower level layers. We are specifically using the following UML constructs to bootstrap the meta modeling layers:

- `Class`. A Class is an abstraction of instances that represent the same concept and carry the same features. A class has attributes and methods. For simplicity we eliminate the placeholder for attributes and methods.
- `Inheritance` relationship. Inheritance is a relationship that shows that a child class is a subclass of the parent. It has all the properties of the parent in addition to some other child-specific features.
- `Association` relationship. An association defines a semantic relationship between two classes. The type of relationship is captured in the names and roles of the association. Each of the two classes plays a role in the association. Associations are directed relationships.

3.2 UML Presentation Notation for the Next Layer

UML defines specific presentation notation for each of its modeling elements. Figure 2 illustrates the modeling notations for the modeling elements `Class`, `Inheritance`, and `Association`. For more details about UML syntax and semantics we refer to the official UML web site [13].

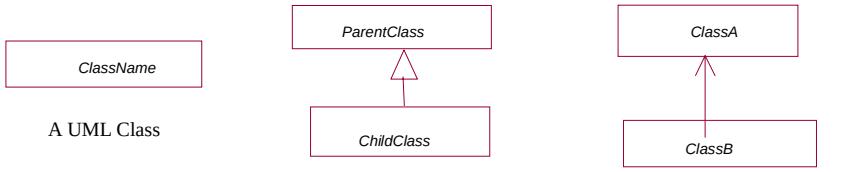

Fig. 2 UML constructs used to bootstrap the meta model

4 Layer 4: Meta-Meta Modeling Layer

At this level we define some constructs that we will use to build elements in the meta modeling layer (the layer below). To build this layer, we use the three elements that we selected from UML: Class, Inheritance, and Association.

4.1 UML Class Diagram for the Meta-Meta Modeling Layer

At this abstraction level, we define five elements:

- `MMClass`. The `MMClass` is the top-level element that is used to derive any modeling constructs in the layers below. An instance of the `MMClass` defines a construct in the meta-modeling layer; it does not define a relationship, however.
- `MMRelationship`. `MMRelationship` is an abstraction for all permissible relationships between the instances of the `MMClass`. It defines a relationship between two and only two instances of `MMClass` in the layer below. `MMRelationship` is an **abstract** class. Concrete implementations of the `MMRelationship` define the nature and semantics of the relationship. `MMRelationship` is a directed relationship between **two** `MMClass` instances, which means one instance will play the role of a source and another instance will play the role of the destination.
- `MMSubClassOf`. `MMSubClassOf` is a concrete implementation of the `MMRelationship` class. `MMSubClassOf` defines the relationship between two `MMClass` instances in which one instance is a sub class of the other. It specifies a subset/superset relation between `MMClass` instances.
- `MMDomain`. `MMDomain` is a concrete implementation of the `MMRelationship` class. `MMDomain` defines a relationship between two `MMClass` instances in which the **source** requires a domain on **input** value, which will be restricted to constructs of the destination type.

- `MMRange`. `MMRange` is a concrete implementation of the `MMRelationship` class. `MMRange` defines a relationship between two `MMClass` instances in which the **source** produces **output** values whose type should be restricted constructs of the destination type.

To model the above construct, we use UML Class constructs to model each of the `MMClass`, `MMRelationship`, `MMDomain`, `MMRange`, and `MMSubClassOf` classes. We use UML inheritance construct to represent the `subclassing` mechanism between `MMRelationship` and `MMDomain`, `MMRange`, and `MMSubClassOf`. Finally, `MMRelationship` has two UML associations with the `MMClass` to represent the source and destination references. Figure 3 illustrates the UML class diagram that captures the meta-meta modeling layer.

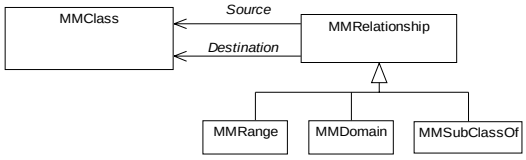

Fig. 3 The UML class diagram for the Meta-Meta Modeling layer

4.2 UML Presentation Notation for the Next Layer

Figure 3 defines the modeling constructs and their relationships at the meta-meta modeling layer using UML class diagram and using the UML constructs: Class, Inheritance, and Association and their representation as described in figure 2.

Instances of those constructs will be used to develop the models for the next layer. We will use UML modeling notation (similar to the notations previously illustrated in Figure 3) to develop a model for the next layer. To be able to develop those models we have to define how instances of the constructs defined at this layer will look like in the next layer. This means we need to define presentation notation for instances of `MMClass`, `MMDomain`, `MMRange`, and `MMSubClassOf`. To be consistent, instances of `MMClass` will have the same presentation, instances of the `MMDomain` will have the same presentations, and so on.

Since we adopted the UML as our modeling language, we look for presentation symbols in UML to present those instances. UML has predefined set of elements that constitute the core of the UML language. However, UML also provides extension mechanisms by which one can define his own elements and give them different notations. This extension mechanism is called Stereotype. Stereotypes are used by designer to adapt the meaning and the look of some UML modeling construct to suit specific application domain. We use UML Stereotypes to define instances of the meta-meta modeling layer constructs as follows.

- Instance of the `MMClass` will be the basic building blocks of the meta modeling layer. We use a UML class to represent an instance of `MMClass`. In this case, we find a UML construct that best represents our needs. We will add the stereotype label "`MMClass`" to make it explicit that this is a class from the previous meta modeling layer and not a user UML class.

- An instance of `MMSubClassOf` relationship has an inheritance meaning. Therefore we use the UML inheritance symbol but we will add the stereotype label "`MMSubClassOf`" to make it explicit that this is a relationship defined in a previous meta modeling layer and not a user-defined inheritance relationship.
- An instance of `MMDomain` relationship defines a relationship between two `MMClass` instances. We use a UML dependency relationship stereotyped by the label "`MMDomain`" to model instances of the `MMDomain` construct.
- An instance of `MMRange` relationship defines a relationship between two `MMClass` instances. We use a UML association relationship stereotyped by the label "`MMRange`" to model instances of the `MMRange` construct. We used association for `MMRange` instance and dependencies for `MMDomain` to have distinct presentation.

The following figure illustrates the legend that we use in modeling instances of the `MMClass`, `MMSubClassOf`, `MMDomain`, and `MMRange` constructs. This notation will be used to construct the UML class diagram for the meta modeling layer (the layer below).

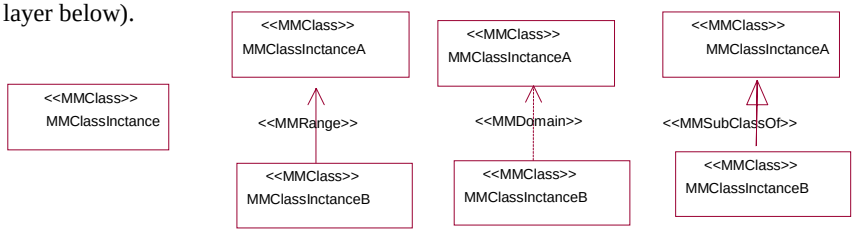

Fig. 4 Legend for instance of the meta-meta modeling layer constructs to be used in the meta modeling layer

5 Layer 3: The Meta Modeling Layer

In this layer we define constructs that we will use to build modeling elements of the RDFS and RDF languages (the modeling layer). To build this layer, we use instances of the four elements that we defined in the previous layer; i.e. instances of `MMClass`, `MMDomain`, `MMRange`, and `MMSubClassOf`. We also use their presentation notation that we developed in Figure 4.

5.1 UML Class Diagram for the Meta Modeling Layer

In this layer, we capture all the elements that are used in defining the RDFS and the RDF languages and their language constructs. To define those meta modeling construct we abstract elements from the RDFS and RDF specifications [1,2] and identify the relationships between the elements at the meta-modeling layer rather than the language layer. Every element that we use in this layer should be an instance of

MMClass and every relationship should be an instance of one of the predefined relationships: MMDomain, MMRange, and MMSubClassOf.

We identify the following elements for the meta modeling layer:

- MResource. MResource is the top super class for this layer.
- MClass. Captures an abstraction of the every Class concept used in the RDFS and RDF languages. It is a subclass of MResource. Since the only valid relationships at this layer should come from the layer above, we will use the MMSubClassOf relationship to implement sub classing.
- MProperty. MProperty captures an abstraction of every property used in the RDFS and RDF languages. It is an MMSubClassOf the MResource construct.
- MConstrainedProperty. This has a direct map to the ConstrainedProperty defined in the RDFS language.
- MSubClassOf and MSubPropoertyOf. These two MMClass instances represent abstraction of the inheritance mechanism used in the RDFS and the RDF languages. MSubClassOf is used to define inheritance between elements in the RDFS and RDF languages that are instances of the MClass construct. MSubPropoertyOf is used to define inheritance between elements in the RDFS and RDF languages that are instances of the MProperty construct.
- MDomain and MRange. Define two constrained relationship between RDFS and RDF language constructs. Note that these two are different from the rdfs:domain and rdfs:range. Whereas MDomain and MRange constraints are applied to the constructs used to develop the RDFS and RDF languages, rdfs:domain and rdfs:range are actual constructs of the RDFS and RDF languages that can be used by the user.

If we investigate the relationships between these constructs, we find the following (note that any relationship defined here should either be MMSubClassOf, MMDomain, or MMRange):

- MClass, MProperty, and MConstrianedProperty classes are MMSubClassOf of the MResource class.
- MConstrainedProperty is MMSubClassOf of both MProperty and MconstrainedResource.
- The MMDomain and MMRange for the MSubClassOf are of type MClass.
- The MMDomain and MMRange for the MSubPropoertyOf are of type MProperty.
- The MMDomain of the MDomain construct or the MRange construct is of type MProperty.
- The MMRange of the MDomain construct or the MRange construct is of type MClass.

We defined the modeling constructs and their relationships at this layer. The following UML class diagram illustrates the meta modeling layer constructs and relationships. We used the presentation notations defined in Figure 4 to create this UML class diagram.

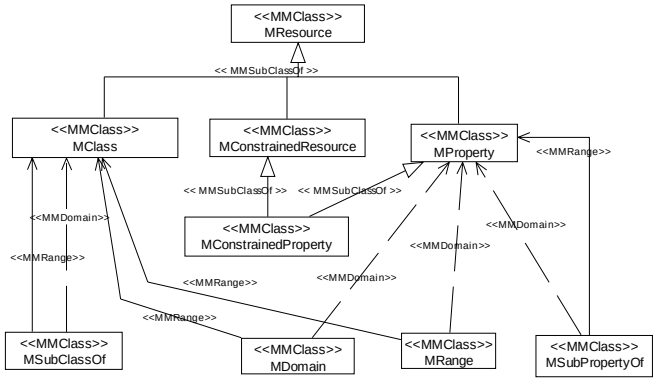

Fig. 5 The UML Class diagram for the Meta Modeling layer

5.2 UML Presentation Notation for the Next Layer

Figure 5 defines the modeling constructs and their relationships at the meta modeling layer using UML class diagram and using the stereotyped UML constructs: `MMClass`, `MMSubClassOf`, `MMDomain`, and `MMRange`.

Instances of constructs defined in Figure 5 will be used to develop the models for the next layer. We will use UML modeling notations (similar to those notations defined in Figure 5) to develop a UML model for the RDFS and RDF languages. To be able to develop those models we have to define how instances of the constructs defined at this layer will look like in the next layer. For the purpose of the RDFS and RDF languages, we need to define presentation notations for instances of the following constructs: `MClass`, `MProperty`, `MConstrainedProperty`, `MDomain`, `MRange`, `MSubClassOf`, and `MSubPropertyOf`.

Since we adopt the UML as our modeling language, we look for model elements in UML to present those instances. Similar to the discussion in section 4.2, we use UML stereotype mechanisms to define instances of the meta modeling layer constructs.

- Instance of the `MClass` will be presented using a UML class stereotyped with the label `MClass`.
- For `MProperty` we create a new stereotype shaped as a solid ellipse.
- For `MConstrainedProperty` we create a new stereotype shaped as a dotted ellipse.
- `MRange` translates into a relationship between two instances of `MClass`. For `MRange` we create a UML associations with a stereotype labeled as "`MRange`".
- `MDomain` translates into a relationship between two instances of `MClass`. For `MDomain` we use a UML dependency with a stereotype labeled with "`MDomain`".
- `MSubClassOf` translated into a relationship between two instances of `MClass`. For `MSubClassOf` we use the UML inheritance symbol labeled with "`MSubTypeOf`".
- `MDomain` translates into a relationship between two instances of `MProperty`. For `MSubProportyOf` we create a symbol similar to inheritance symbol but has a dotted boundary and labeled with "`MSubPropertyOf`"

The following figure illustrates the legend that we will use in modeling instances of `MClass`, `MProperty`, `MConstrainedProperty`, `MSubClassOf`, `MSubPropertyOf`, `MDomain`, and `MRange`. This notation will be used to construct the UML class diagram for the modeling layer (the layer below).

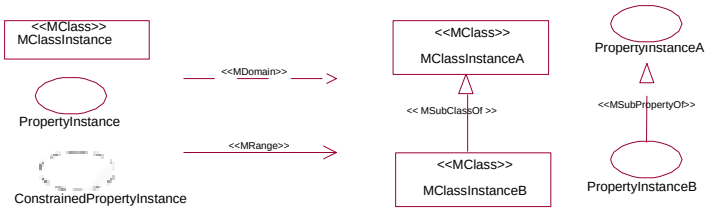

Fig. 6 Legend for instance of the meta modeling layer constructs to be used in developing the modeling Layer.

Figure 6 resembles a toolkit of tools that will be used to define the RDFS and the RDF languages while Figure 5 defines the relationship between these tools.

6 Layer 2: Modeling Layer

In this layer we define constructs that are part of the RDFS and RDF languages by instantiating elements from the meta modeling layer. To build this layer, we use instances of the elements that we defined in the previous layer; i.e. instances of `MClass`, `MProperty`, `MConstrainedProperty`, `MDomain`, `MRange`, `MSubClassOf`, and `MSubPropertyOf`. We also use their presentation notation that we developed in Figure 6.

We will capture all elements defined in the RDFS and the RDF languages. To capture those constructs, we inspect the RDF and the RDFS language specification [1,2]. Every element that we use in this layer should be an instance of one of the predefined classes: `MClass`, `MProperty`, or `MConstrinedProperty`. Every relationship should be an instance of one of the predefined relationships: `MDomain`, `MRange`, `MSubClassOf`, or `MSubPropertyOf`.

6.1 UML Model for the RDFS Language

We start first with the RDFS language. We capture the following classes.

- We extract RDFS constructs that are instances of `MClass`. These classes will be `rdfs:Resource`, `rdfs:Class`, `rdfs:Container`, `rdfs:Literal`, and `rdf:Property`. Note that `rdf:Property` is defined within the RDF scope but is used in defining the RDFS language.
- We extract RDFS constructs that are instance of `MProperty`. The following RDFS properties are instances of `MProperty`: `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:comment`, `rdfs:label`, `rdfs:seeAlso`, and `rdfs:isDefinedBy`.

- Then, we extract RDFS constructs that are instances of `MConstrainedProperty`. The following are RDFS properties that are instance of `MConstrianted-Property: rdfs:domain` and `rdfs:range`

We then capture the relationships between RDFS constructs using one of the relations predefined in the meta modeling layer (i.e. `MDomain`, `MRange`, `MSubClassOf`, or `MSubPropertyOf`)

- `rdfs:Class`, `rdfs:Container`, and `rdf:Property` classes are `MSub-ClassOf` of the `rdfs:Resource` class.
- `rdfs:isDefinedBy` property is an `MSubProprtyOf` of the `rdfs:seeAlso` property.
- We then define the constraints (`MDomain` and `MRange`) on the various RDFS properties. For instance:
  - The `MDomain` constraint and the `MRange` constraints for the `rdfs:seeAlso` property is an `rdfs:Resource` class.
  - The `MDomain` constraint for the properties `rdfs:label` and `rdfs:comment` is the `rdfs:Resource` class and their `MRange` is `rdfs:Literal`.
  - The property `SubPropertyOf` has `MDomain` and `MRange` as the `rdfs:Property` class.

The following UML class diagram illustrates the modeling layer constructs and relationships for the RDFS language. We use the modeling notations for the classes and relationships as defined in Figure 6.

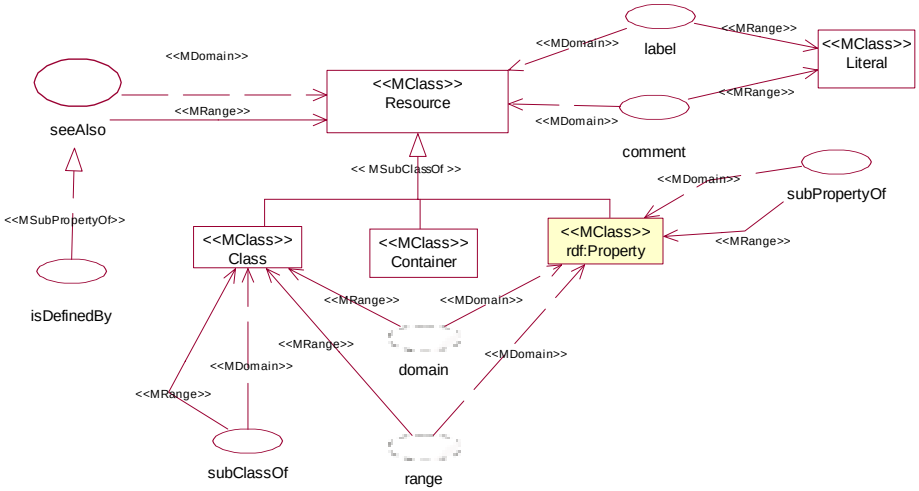

Fig. 7 The UML class diagram for the RDFS language

6.2 UML Model for the RDF Language

We can also define the RDF language using instances form the meta modeling constructs. We capture the following elements for the RDF language.

- We extract RDF constructs that are instances of `MClass`. These classes are `rdf:Property`, `rdf:Statement`, `rdf:Seq`, `rdf:Bag`, and `rdf:Alt`.
- We extract RDF constructs that are instance of `MProperty`. The following RDF properties are instances of `MProperty`: `rdf:object`, `rdf:subject`, `rdf:predicate`, `rdf:type`, and `rdf:value`.

We then capture the relationships between RDF constructs using one of the relationships defined in the meta modeling layer (i.e. `MDomain`, `MRange`, `MSubClassOf`, or `MSubPropertyOf`)

- The `rdf:Statement` class is an `MSubClassOf` of the `rdfs:Class` class.
- The classes `rdf:Seq`, `rdf:Bag`, and `rdf:Alt` are `MSubClassOf` of the `rdfs:Container` class.
- We then define `MDomain` and `MRange` constrains on the various properties. For instance:
  - The `MDomain` constraint for the properties `rdf:object`, `rdf:subject`, and `rdf:predicate` is an `rdf:Statement` class.
  - The `MRange` constraint for `rdf:type` property is as `rdfs:Class` class.

The following UML class diagram illustrates the modeling layer constructs and relationships for the RDF language. We use the modeling notations defined in Figure 6.

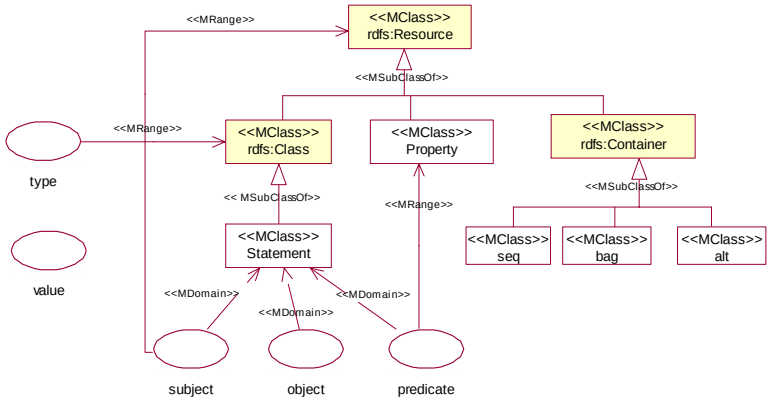

Fig. 8 The UML class diagram for the RDF language

6.3 UML Presentation Notation for the Next Layer

The elements of the RDFS and RDF languages will be instantiated to develop the ontology for a particular domain. There are several presentation options for an ontology. UML is one of these options [9,10]. To be able to present ontologies using UML

constructs we need to find presentation notations that complies with UML syntax and semantics and at the same time provide support for all the elements of the RDFS and RDF languages.

It is not the purpose of this paper to define a mapping between UML models and RDF and RDFS language constructs. A subset of this mapping is sufficient to illustrate the use of UML as an ontology modeling language. Therefore, we have selected the following UML elements to present a subset of the RDFS and RDF languages:

- An `rdfs:Class` is presented with a UML class and is stereotyped with "rdfs:Class" label.
- An `rdf:Property` is presented using a UML class which is stereotyped to a round ellipse graphical presentation.
- An `rdfs:subClassOf` is presented using a UML inheritance relationship and stereotypes with "rdfs:subClassOf"
- An "`rdf:subPropertyOf`" is presented using a UML inheritance relationship which is stereotyped to have a dotted presentation and a label "rdfs:subPropertyOf"
- An `rdfs:domain` will be presented by a UML dependency that is stereotyped with "rdfs:domain" label.
- An `rdfs:range` will be presented by a UML association relationship that is stereotyped with "rdfs:range" label.

The UML presentation for these constructs is shown in Figure 9.

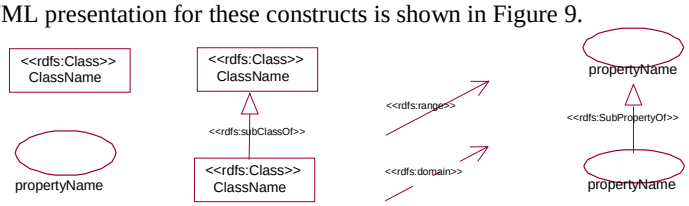

Fig. 9 Legend for instance of the RDFS and RDF language constructs

7 Ontology and Instance Layers

7.1 Layer 1: Ontology Layer

As mentioned in the previous section, it is not the purpose of this discussion to show that UML suffices for describing ontologies. We illustrate an example of modeling some elements of the RDFS and RDF language in UML and using them to develop a simple ontology.

As an example, we assume that we want to model the ontology of human race. We define a concept of "Human" which is a subset of another concept "Mammal". Using RDFS language, a Human and a Mammal will be instances of the `rdfs:Class` construct and a Human will be an `rdfs:subClassOf` a Mammal. We now add two properties for a Human, a "hasFriend" and a "hasCloseFriend". These two properties only apply to a Human and hasCloseFriend is a subset of hasFriend. Using RDFS and RDFS language, the two properties will be instances of `rdf:Property` and has-

CloseFriend will be an `rdfs:subPropertyOf` hasFriend. Moreover, we add the constraint that hasFriend has an `rdfs:domain` and `rdfs:range` of type Human and not Mammal. To present this example in UML, we use the UML constructs defined in Figure 9. Figure 10 illustrates the UML class diagram for the example.

7.2 Layer 0: Instance Layer

To instantiate an ontology, we use UML object diagrams to create instances of elements in Figure 10. For example, Figure 11 illustrates a simple example were we created the objects "John", "Mary", and "Sarrah" as instances of the "Human" ontology class and the two properties of "John" "hasFriend" and "hasCloseFriends" which point to "Mary" and "Sarrah" respectively. We emphasize here that we only use this for illustration and more work will be required to model the instantiation of RDFS and RDF constructs to model domain instances[9,10].

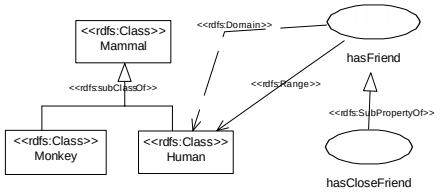

Fig. 10 The UML class diagram for the example

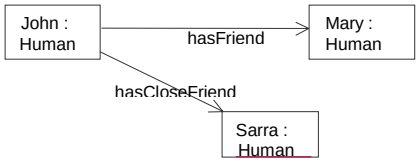

Fig. 11 UML object diagram to represnet an instance of the example ontology

8 Discussion

The UML meta modeling approach described in the previous sections has several benefits and applications. It reveals some properties about the Semantic Web languages.

a) The layered approach provides explicit separation between different levels of abstractions. The only relationship between elements form one layer and elements form the previous layer is that the lower level layer can only be an *instance of* elements from the upper layer (i.e. their types are defined by those at the upper layer). This separation of abstraction levels is useful in identifying how Semantic Web languages are represented and how they are used to drive other languages.

b) The UML modeling language is useful in bootstrapping the meta modeling layer. Without an initial top-level starting point, the layers could be infinite and hence sources of ambiguities would be more likely.

c) The UML-based meta modeling approach solves the problem of self referencing and instantiation. For example, the `rdfs:Class` can be an instance of itself in the original specification. In the meta modeling approach, `rdfs:Class` is an instance of `MClass` which is a construct at a the meta modeling layer and not the RDFS language layer.

d) The UML-based meta modeling approach solves the layer mistake problem for classes. For example, in the RDFS specification, the `rdfs:Class` is a subclass of `rdfs:Resrource` while `rdfs:Resrource` is an instance of `rdf:Class`. In the meta modeling approach, `rdfs:Class` is a subclass of `rdfs:Resoruce` but `rdfs:Resoruce` is an instance of `MClass` and not `rdfs:Class`.

e) The meta modeling approach solves the layer mistake problem for properties. For example, in the RDFS specification, the `rdfs:domain` and `rdfs:range` are used to describe constraints on the modeling elements of RDFS language and at the same time are used by the user in the ontology level. In the meta modeling approach, `rdfs:domain` and `rdfs:Class` are only used by the user to define constraints on elements of his ontology while `MDomain` and `MRange` are used to define constraints on elements of the RDFS and RDF languages.

f) Using the meta modeling approach, we are now able to construct valuators for a specific Semantic Web language. For example, assume we defined RDFS as illustrated in figure 7. We can check whether the language is a valid language or not by validating the language constructs and their relationships using the meta modeling elements define in the layer above. For example, a rule from the meta modeling layer (as illustrated in figure 5) states that an instance of the `MSubClassOf` defines a relationship between two instances of `MClass`. Hence, the `MSubClassOf` relationship between `rdfs:Class` and `rdfs:Resoruce` is a valid relationship (as shown in figure 7) while and `MSubClassOf` relationship between `rdfs:Class` and `rdfs:seeAlso` would have been **invalid** since the later if of type `MProperty`. As another example, from figure 5 we find that an instance of `MDomain` relationship should have a domain as an instance of type `MProperty`. Using this rule, we can check for every `MDomain` relationship in the RDFS language (figure 7) and make sure that the source is an instance of `MProperty`.

g) As discussed earlier, we find that there is no clear separation between the constructs used in the RDFS and the RDF languages. Some constructs from the RDFS are used to define other constructs in the RDF language such as `rdfs:Container` is the super class of `rdf:Seq` and `rdf:Bag`. Similarly, `rdf:Property` is used in defining the constructs used in the RDF language. Using the meta modeling approach, we can clearly define these elements and their abstractions for future treatment (if desired).

h) From a meta modeling data-oriented perspective, we find that the RDF language is used to describe the metadata and the RDFS language is used to describe the schema for the metadata which is considered to be a higher level in terms of the metadata model [7]. From a meta modeling language-oriented perspective, we find that RDF and RDFS languages are at the same level, the language level and they are both used in driving the ontology.

9 Related Work

The work presented in this paper is related to research in Semantic Web languages; namely the RDF [2] and RDF Schema [1] languages and other ontology languages building on top of them such as DAML[4], OIL[3] , and DAML+OIL [14].

The problems with RDF and RDF Schema and their formal definition are recognized by several researchers [5, 15,17]. The research in [16] provides an initiative to capture the intended semantics of RDF in first-order logic in an attempt to provide RDF with a formalization allowing its full exploitation as a key ingredient of the evolving Semantic Web.

There are also several ongoing researches on the relationship between UML and the Semantic web. An early synopsis of the relationship between UML and RDF Schema is presented in [7]. One research direction is concerned with using UML for Semantic Web languages. For instance, Cranefield *et.al.* [10] investigate using UML and its associated constraint language (the Object Constraint Language, OCL) to model ontologies and they proposed UML as an ontology language in [9]. Application of UML to Semantic Web activities is described in [11]. Another research direction is to extend UML to make it suitable to web activities. Melnik [12] discuss how to extend and represent UML in RDF syntax. Baclawski et.al. [18] extend UML to support ontology engineering for the Semantic Web. They compare DAML elements with UML modeling capabilities and identify similarities and differences between UML and DAML. For similarities, they define the mapping and for incompatibilities they highlight the differences and limitations.

However none of the work mentioned above addresses the problem of defining a meta model for the modeling languages as opposed to meta models for data. The idea and rational for meta modeling for Semantic Web languages is introduced by Kampman et.al. [19] and is further studied by Pan et.al. [5]. Pan et.al. proposed a fixed meta modeling architecture called RDFS(FA). We find the following advantages in the proposed meta modeling techniques as compared to RDFS(FA):

- The proposed meta modeling approach is bootstrapped using a well-formed language that has a well-defined semantics; i.e. the UML. RDFS(FA) does not have a bootstrap layer which would allow multi layering on top of its top level layer and hence would become a source of ambiguity.
- RDFS(FA) uses a Directed Label Graph (DLG) to describe layers. We used UML syntax and extension mechanisms to develop UML models for each layer. UML models are more expressive than DLGs and they are easier to understand and codify by software architect, designers, and developers. By virtue of using stereotyping one can easily distinguish the type of an element (i.e. which element in the above layer that it is instance of). DLG does not provide this visual effect.
- RDFS(FA) was developed based on RDFS and to solve problems in RDFS specification. The proposed meta modeling is based on RDF and RDFS and can be further used to identify the integration of other languages such as OIL or DAML.
- RDFS(FA) describes a portion of the RDFS specification. We describe all elements of the RDFS and RDF languages.

RDFS(FA) has an advantage over the proposed approach, it used a model theoretic semantic to interpret data model of RDFS(FA) which could be more formal than UML semantics.

10 Conclusion

The Semantic Web cannot possibly become a reality unless it has carefully perfected the necessary technologies for modeling information in a machine understandable format. Semantic Web modeling languages, in turn, cannot be successful and widely adopted unless we clearly adopt a meta modeling layered approach that defines: a) the scope of a Semantic Web language, b) formal or semi-formal definition of the language constructs, c) relationships between languages and their constructs, and d) a common meta modeling for languages and a common meta model for data.

In this paper, we presented a meta modeling approach for Semantic Web languages that is inspired by the UML meta modeling architecture. We described the constructs of RDFS and RDF: how they fit into the model and the elements used to create those languages. We used UML syntax and extension mechanisms to model the elements at each layer. We illustrated the benefits of using UML and the meta model to: solve the layer mistake of classes, solve the layer mistake of properties, describe element of Semantic Web languages such as RDF and RDFS, and solve the self instantiation and the ambiguous relationship problems.

In this paper, we focused on creating the meta model using UML and using the RDFS and RDF language for illustrating how Semantic Web languages can fit into the meta model. Future research work will include investigation of how other Semantic Web languages such as DAML+OIL could build on top of the meta modeling approach.

References

[1] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Speciation 1.0. W3C *Candidate Recommendation*, URL http://www.w3.org/TR/rdf-schema, Mar. 2000.

[2] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. *W3C Recommendation*, February 1999

[3] The *Ontology Information Language*, OIL. http://www.ontoknowledge.org/oil/, 2002.

[4] The *DARPA Agent Markup Language*, DAML. http://www.daml.org, 2002.

[5] J. Pan and I. Horrocks. Metamodeling architecture of web ontology languages. In *Proceedings of the First Semantic Web Working Symposium* (SWWS'01), pages 131-149. CEUR (http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/), 2001.

[6] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38-45, 2001

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

[8] 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.

[9] S. Cranefield and M. Purvis. UML as an Ontology Modeling Language. *In IJCAI Workshop on Intelligent Information Integration*, 1999.

[10] S. Cranefield. Networked Knowledge Representation and Exchange using UML and RDF. *Journal of Digital Information*, 1(8), Feb. 2001.

[11] S. Cranefield. UML and the Semantic Web, Feb. 2001. http://divcom.otago.ac.nz/infosci/pubs/publications.htm

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

[13] The Unified Modeling Language, resources homepage. http://omg.org/uml

[14] F. van Harmelen, P. Patel-Schneider, and I. Horrocks. Reference description of the DAML+OIL ontology markup language. March 2001, http://www.daml.org/2001/03/reference.html

[15] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling Knowledge Representation on the Web by Extending RDF Schema. *Proceedings of the tenth World Wide Web conference* WWW'10, pages 467-478, May 2001.

[16] W. Conen and R. Klapsing. A Logical Interpretation of RDF. In *Electronic Articles n Computer and Information Science*, 5(13), 2000. http://www.ep.liu.se/ea/cis/2000/013

[17] W. Nejdl, M. Wolpers, and C. Capelle. The RDF Schema Specification Revisited. *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik*, Modellierung 2000, April 2000.

[18] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, and M. Aronson. Extending UML to Support Ontology Engineering for the Semantic Web. The *Fourth International Conference on UML* (UML 2001), Toronto, October 1-5, 2001

[19] A. Kampman and F. van Harmelen. Sesame's interpretation of RDF Schema. Aidministrator Nederland Note, URL http://sesame.aidministrator.nl/doc/rdf-interpretation.html, Apr. 2001.