MetaModeling in OOP, MOF, RDFS, and OWL

Seiji Koide[1,2] and Hideaki Takeda[1]

[1] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan,
koide@grad.nii.ac.jp, takeda@nii.ac.jp,
WWW home page: http://www.nii.ac.jp/

[2] Galaxy Express Corporation, 1-18-16 Hamamatsu-cho, Minato-ku, Tokyo, Japan,
koide@galaxy-express.co.jp,
WWW home page: http://www.galaxy-express.co.jp/

Abstract. Metamodeling is the act of describing the model of a modeling language using another language, namely metamodeling language. When a language and its meta-language are the same, the language is called reflective. Reflective modeling languages enable reflective modeling or self-descriptive modeling. RDF(S) and OWL are reflective in nature. MOF aims to provide the reflective modeling capability. Therefore, MOF needs reflective modeling machineries to embody the reflection on RDF(S) and OWL. We developed a modeling language for RDF(S) and OWL on top of reflective OOP language, Common Lisp Object System. In this paper, we devote the discussion to provide deeper insights on metamodeling and the reflection in RDF(S) and OWL from the viewpoint of Object-Oriented metamodeling. We address the OWL-Full connection of the RDF universe and the OWL universe. Finally, we remark the formalization of ontological metamodeling.

1 Introduction

The metamodeling is a common feature of Object-Oriented Programming (OOP), Meta Object Facility (MOF), RDF(S), and OWL. There are many discussions on metamodeling at each of the domains. W3C Software Engineering Task Force (SETF) envisioned how Semantic Web technologies can be applied in and beneficial to the software engineering.[3] However, the divergence of discussion does not converge yet, whereas both Ontology Definition Metamodel (ODM) by OMG and Ontology Driven Architecture (ODA) by SETF talk a common interdisciplinary field of the ontology and the software engineering. We have developed a Semantic Web modeling language called SWCLOS [Koide2006][4] on top of Common Lisp Object System (CLOS). SWCLOS is an amalgam of OOP and OWL/RDF(S). In this paper, we discuss various issues of metamodeling from our experiences in the SWCLOS development. We aim to clarify the discussions on metamodeling in the interdisciplinary field of OOP, MOF, and Semantic Web. At Section 2, we

[3] http://www.w3.org/2001/sw/BestPractices/SE/ODA/

[4] It is available from http://pegasus.agent.galaxy-express.co.jp/SemanticWeb-swclos-en.htm

discuss the metamodel layers in ODM and point out that the layers in Model Driven Architecture (MDA) is different from the ontological metamodel layers in RDF(S). At Section 3, we discuss the reflection in modeling. At Section 4, we discuss the integration of the universe of RDF and OWL. At Section 5, we discuss related works and discussions. Finally, we summarize the discussions and propose several criteria for metamodeling.

2 Metamodeling

2.1 Metamodeling in MOF and ODM

Metamodeling is the act of describing the model of a modeling language using another language, namely metamodeling language. MDA[5] by OMG addressed metamodeling in UML and proposed *four layered metamodel architecture* [Mellor2004]. Although MOF itself allows any number of layers in metamodeling[6], ODM insists on the four layered metamodel architecture as follows.[7]

- M3 - the MOF
- M2 - a MOF class model, specifying the classes and associations of the system being modeled, the structure of OWL for example.
- M1 - an instance of an M2 model, describing a particular instance of the system being modeled, a particular OWL ontology, for example.
- M0 - ground individuals. A population of instances of the classes in a paricular OWL ontology, for example.

Java Metadata Interface (JMI) that is a MOF mapping to Java also insists on the four layered metamodel archtecture as follows.[8]

- The meta-metamodel (M3) layer defines the metamodel layer, describing the structure and semantics of the meta-metadata. It is the common “language” that describes all other models of information. Typically, the meta-metamodel is defined by the system that supports the metamodeling environment.
- The metamodel layer (also known as the M2 or meta-metadata layer) defines the model layer, describing the structure and semantics of the metadata. The metamodel specifies, for example, a database system table that describes the format of a table definition. A metamodel can also be thought of as a modeling language for describing different kinds of data. The M2 layer represents abstractions of software systems modeled using the MOF Model. Typically, metamodels describe technologies such as relational databases, vertical domains, etc.

[5] http://www.omg.org/docs/omg/03-06-01.pdf
[6] http://www.omg.org/docs/formal/06-01-01.pdf
[7] http://www.omg.org/docs/ad/05-08-01.pdf
[8] http://www.jcp.org/en/jsr/detail?id=40

- The model layer (also known as the M1 or metadata layer) defines the information layer, describing the format and semantics of the data. The metadata specifies, for example, a table definition in a database schema that describes the format of the M0 level instances. A complete database schema combines many metadata definitions to construct a database model. The M1 layer represents instances (or realizations) of one or more metamodels.
- The information layer (also known as the M0 or data layer) refers to actual instances of information. These are not shown in the figure, but would be instances of a particular database, application objects, etc.

The ODM Document seems to intend to position RDFS and OWL vocabularies somewhere in the four layers and specified that some of RDFS and OWL vocabularies are located in M2 and some of them are placed in M1. Eventually the Document stated that OWL does not make clear distinction between M3, M2 and M1 objects. However, we argue that the ontological layers that are constructed by the membership and subsumption relation of objects do not need to coincide with the MDA metamodeling layers. The ontological metamodel layers and the metamodeling language layers are different things. It is important to understand that the idea of the MDA layered architecture originally does not express the ontological modeling layers. Rather, MDA layers represent metamodeling layers by modeling languages. In fact, the automatic compilation from Platform Independent Models (PIM) to Platform Specific Models (PSM) is expected in MDA, in which UML is expected as PIM and the platform means Java environments, CORBA, .NET, etc.

In consideration of the ontological metamodel layers and the metamodeling language layers, we may capture two extreme different approaches. One is the usage of a single modeling language across all ontological layers. The other is the usage of different modeling languages at different metamodel layers. In the former approach, the modeling language must embody the semantics of ontological models in all ontological layers. In the latter approach, the upper modeling language should be more universal like MOF and the semantics of the lower modeling languages must satisfy the semantics of ontological models and must be underpinned by additional axioms or constraints. In this approach, same built-in vocabularies of RDFS and OWL may appear in M2, M1, and MO.

We developed a modeling language SWCLOS with the former approach [Koide2006]. Hereafter, we focus the discussion of metamodeling on the former approach.

2.2 Metamodeling in RDF(S) and OWL

Figure 1 shows all RDFS vocabularies and the hierarchical structure. In the figure, rdfs:Class and rdfs:Datatype are metaclasses, namely a class of other classes. Properties such as rdfs:seeAlso and rdfs:member are instances of rdf:Property and rdf:nil is an instance of rdf:List. Thus, we capture RDFS vocabularies as three metamodel layers, i.e., metaclass layer, class layer, and instance layer. Note that this ontological metamodel structure is ascribed to the membership and the subsumption by rdf:type and rdfs:subClassOf.

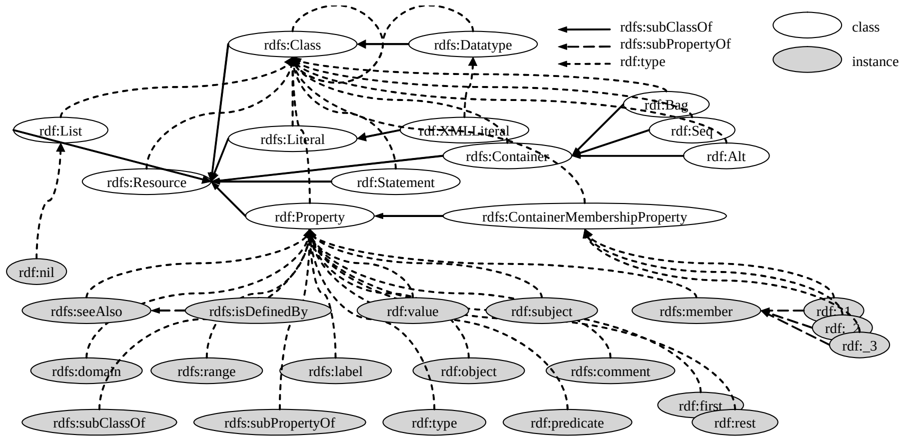

Fig. 1. Layered Hierarchy of RDFS

Since OWL is an extension of RDF(S), OWL vocabularies inherit RDFS characteristics and the metamodeling of RDF(S) is involved in OWL. OWL-Full provides the capability to capture a class as individual. Therefore, the exect extension of RDF(S) metamodeling is realized in OWL-Full precisely. However, it does not mean no rules or no principles allow to deal with classes as individual in OWL-Full. Distinguishing objects between instances, classes, and metaclasses is crucial to metamodeling. In SUMO ontology[9], `sumo:Meter`, which is an instance of `sumo:SystemeInternationalUnit`, is a subclass of `sumo:PhysicalQuantity`. However, `sumo:SystemeInternationalUnit` is also a subclass of `sumo:PhysicalQuantity`. Namely, `sumo:PhysicalQuantity` is simultaneously both a metaclass and superclass of `sumo:Meter`. The `sumo:PhysicalQuantity` is not categorized onto either the metaclass layer or the class layer in ontology. We argue that we need more precise formalization on metamodeling in RDF(S) and OWL-Full, whereas such ad hoc classification of `sumo:PhysicalQuantity` originated from the EngMath Ontology [Gruber1994] by KIF. The problem of the formalization of metamodeling is challenging and almost not tackled yet in Semantic Web.

2.3 Metamodeling from Object-Oriented Perspective

The semantics in Object-Oriented models is underpinned by the behavior of objects. So, we discuss the semantics of OOP through the method definition and invocation mechanism. A class in OOP is a computational notion that rules a set of objects and controls the behavior of them. The methods defined at a class establish the behavior of instances of the class. Note that methods at superclasses are inherited by the subclasses. This semantics of class-instance and inheritance in OOP is consistent with the semantics of the membership and the subsumption in RDF(S) and OWL. W3C Software Engineering Task Force

[9] http://www.ontologyportal.org/

(SETF) compared OWL features to OOP language features and pointed out several serious discrepancies between OOP semantics and OWL semantics. We solved the discrepancies in the development of SWCLOS [Koide2006] using Meta-Object Protocol (MOP) [Kiczales1992]. RDFS and OWL vocabularies are CLOS objects in SWCLOS. Note that a class in CLOS is an object called *metaobject*. In this subsection, we discuss the metamodeling in RDF(S) from the viewpoint of OOP method definition and invocation.

The metamodeling is a basic framework of dynamic OOP languages such as CLOS. In order to capture an object as instance, a class of an object must be established in OOP. This principle is extended to classes and metaclasses in OOP metamodeling. Namely, in order to capture a class as individual, we must establish a class of the class (metaclass). For instance, in order to capture `sumo:PhysicalQuantity` as individual, the class of `sumo:PhysicalQuantity` must be established as metaclass. So far, no difficulty exists. `rdfs:Class` is a metaclass in RDFS vocabularies.

Then, let us suppose we define the concept of old Japanese length measure unit `Shaku` as follows.

```
<rdfs:Class rdf:ID= "Shaku">
  <rdfs:subClassOf rdf:resource ="#LengthMeasure"/>
  <rdf:type rdf:resource = "#OldJapaneseUnitClass"/>
  <rdfs:comment>This example is for the demonstration of meta-metaclasses.
  Shaku is an old Japanese length measure unit.
  </rdfs:comment>
</rdfs:Class>
```

Here, `Shaku` is a user-defined class. In non-metamodeling language like Java or C#, the language system interprets and compiles class definitions. In meta-modeling language like MDA, the machinery in the metaclass layer (M2) processes such class definitions and a user-defined class populates a class layer (M1). In CLOS, a class is a kind of object in CLOS system and a user-defined class also populates the computational environment in runtime.

The above definition of `Shaku` seems to be unconcerned about the lifecycle of objects. However, lifecycle functions such as CREATE and DELETE are requisite in actual modeling languages, and it is inevitable for the single modeling language approach. Even an abstract modeling language MOF is equipped with CREATE and DELETE functions. In the rest of this section, we explain the semantics of RDF(S) with the object creation method and method invocation mechanism. However, the objective of discussion is not on the OOP, rather on the clarity of RDF(S) semantic model.

From the dynamic OO perspective, so-called CREATE method is applied to rdfs:Class in order to make a `Shaku` object as an instance of rdfs:Class. Then, in case of a full-bodied dynamic OO language in which all procedures including CREATE are implemented as method, the method of creating a new object must be defined at a class of the receiver of CREATE message. Namely, the CREATE method for `Shaku` must be defined at a class of rdfs:Class in this case, that is, a class of metaclass or a meta-metaclass. Note that the class of rdfs:Class is rdfs:Class itself in RDFS. Figure 2 illustrates such message passing and the method definition and invocation mechanism.

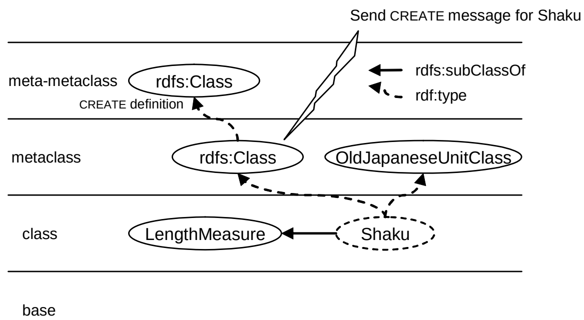

Fig. 2. CREATE method definition and invocation for a new class creation

3 Reflection in RDF(S) and OOP

3.1 Self-referencing Meta-circularity

RDF(S) embraces not only the metamodeling mechanism but also the reflection in metamodeling. We discuss the reflection in RDF(S) from the perspective of dynamic OOP languages.

In order to complete the `Shaku` definition, we must define `OldJapaneseUnitClass`, supposing that `LengthMeasure` is already defined beforehand in the same way as `sumo:LengthMeasure` in SUMO ontology.

The following is an example of the definition.

```
<rdfs:Class rdf:ID= "OldJapaneseUnitClass">
  <rdfs:comment>OldJapaneseUnitClass is a metaclass for old Japanese
  mesurement unit classes.
  </rdfs:comment>
</rdfs:Class>
```

A grance at this definition gives the same impression as the definition of `Shaku`. However, both are different from the viewpoint of metamodeling. The `OldJapaneseUnitClass` in the `Shaku` definition should be a metaclass, because it is a class of `Shaku` through rdf:type. If so, what is the `rdfs:Class` in the `OldJapaneseUnitClass` definition? Is it a metaclass or meta-metaclass?

In order to create `OldJapaneseUnitClass` (metaclass) or to apply a CREATE method to the class of `OldJapaneseUnitClass`, that is `rdfs:Class` (meta-metaclass) in this case, we must define the CREATE method at a class of the

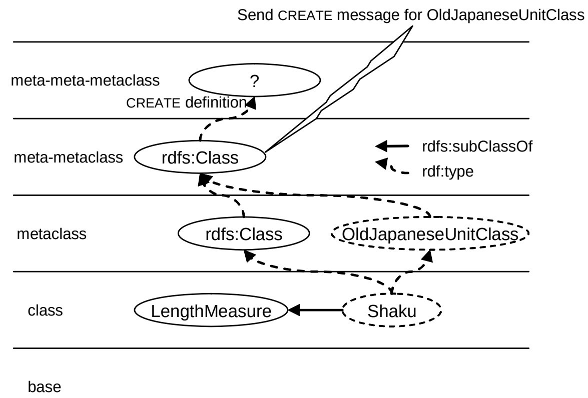

Fig. 3. Ontological metamodel tower concerning CREATE method definition and invocation

`rdfs:Class` (meta-metaclass), namely we need a meta-meta-metaclass in metaclass creation. Figure 3 shows such mechanism of metamodeling and the ontological metamodel tower.

Such a metamodel tower from base-level to class-level, metaclass-level, meta-metaclass level, meta-meta-metaclass level will be infinite in principle, if we desire to mandate the perfect freedom to the language system. Therefore, people usually abandon the perfection and accept the limited flexibility of system. However, the *reflection* in programming language systems provides the ability to modify the language's implementation without leaving the realm of the language [Paepcke1993]. Historically *reflective Knowledge Representation* and *reflective programming* has been researched and developed over two decades. According to *reflection principle* [Feferman1962], Weyhrauch presented the reflection mechanism in the first-order logic system FOL [Weyhrauch1980]. Bowen [Bowen1986] invented `demo` predicate in Prolog, which simulates the behavior of Prolog system.

From the viewpoint of Knowledge Representation, *self-reference* was the key issue required for conducting meta-theory and coping with cognitive overflow. From the viewpoint of programming, *meta-circularity* was a key technology to enable reflection. 3-Lisp [Smith1984] was the first reflective programming language in Lisp, extending basic lisp machinery `eval` and `apply` to reflective computing. According to the idea of reflection, CLOS was designed to specify a model for the language implementation and to standardize it in OO metamodeling and reflective computing [Kiczales1992]. A programmer can manipulate the internal working mechanism in language systems by using CLOS Meta-Object Protocol (MOP). In CLOS, `standard-class` is the class of all other classes including both `standard-class` itself and `standard-object`, and `standard-object` is the top class of all other classes including `standard-class`. This morphology is the same graph-structure as `rdfs:Class` and `rdfs:Resource` in RDFS vocabularies.

The class of rdfs:Class is rdfs:Class itself in RDFS. The infinite metamodeling is terminated with the self-referencing meta-circularity. Note that `rdfs:Class` plays multiple roles as metaclass, meta-metaclass, meta-meta-metaclass, and so on. Thus, the infinite metaclass layers are folded into one layer by the meta-circularity of `rdfs:Class`. Therefore, in order to make the metaclass `OldJapaneseUnitClass` as an instance of meta-metaclass `rdfs:Class`, a user applies a CREATE method that is defined at `rdfs:Class` as meta-meta-metaclass to `rdfs:Class` as meta-metaclass, whereas the CREATE method shall be same to the definition at `rdfs:Class` as metaclass at first.

3.2 Undecidability in Metaclass Layers

Objects can be discriminated between instances, classes, and metaclasses. However, we cannot distinguish metaclass objects between internal metaclass layers. Suppose the CREATE method is intrinsically defined at rdfs:Class, the method is applicable in creating objects at base level, class level, metaclass level, meta-metaclass level, and meta-meta-metaclasse level because of rdfs:Class meta-circularity. Then, how we expand or customize the CREATE method for `OldJapaneseUnitClass` in order to attach specialties to `Shaku`? For example, the CREATE method might signal an alarm or make some special structure dedicated to `Shaku`. To do so, we let `OldJapaneseUnitClass` be a subclass of meta-metaclass `rdfs:Class` rather than metaclass `rdfs:Class`. As `OldJapaneseUnitClass` inherits the CREATE method at `rdfs:Class`, we can modify the inherited method and encode a special CREATE method that is dedicated to create an instance of instances of `OldJapaneseUnitClass` at the meta-metaclass layer. Figure 4 illustrates the mechanism of the special CREATE method to create `Shaku`. The followings describe the definition of `OldJapaneseUnitClass` and an instance of `Shaku` in RDF(S).

```
<rdfs:Class rdf:ID= "OldJapaneseUnitClass">
  <rdfs:subClassOf rdf:resource = "&rdfs;Class">
  <rdfs:comment>OldJapaneseUnitClass is a metaclass for
  old Japanese mesurement unit classes.
  </rdfs:comment>
</rdfs:Class>

<Shaku rdf:ID= "10_shaku">
  <rdfs:comment>ten times of one unit of shaku</rdfs:comment>
</rdfs:Class>
```

Note that `OldJapaneseUnitClass` in the metaclass layer and the meta-metaclass layer is identical as well as `rdfs:Class` is identical in the two layers. Therefore, we cannot distinguish between the metaclass `OldJapaneseUnitClass` and the meta-metaclass one as is. Then, we need to distinguish the case of instance creation and class creation in CREATE method. Similarly, we need to distinguish property value setting in the case of a property to classes (the method should be defined at a metaclass) and a property to metaclasses (the method

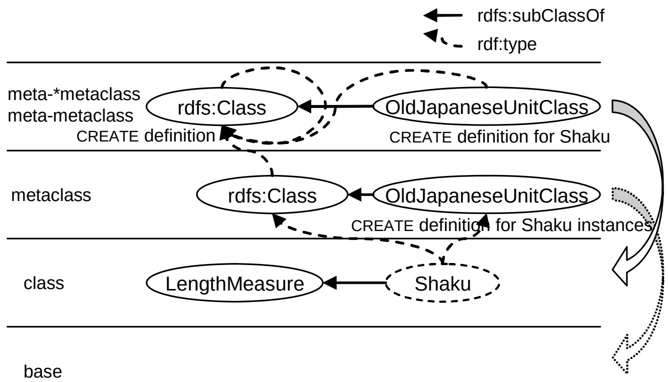

Fig. 4. Unfolded RDF graph structure in CREATE invocation for class creation

should be defined at a meta-metaclass) in SET method, which is identical between a metaclass and a meta-metaclass.

Figure 5 shows the final RDF graph around `OldJapaneseUnitClass`.

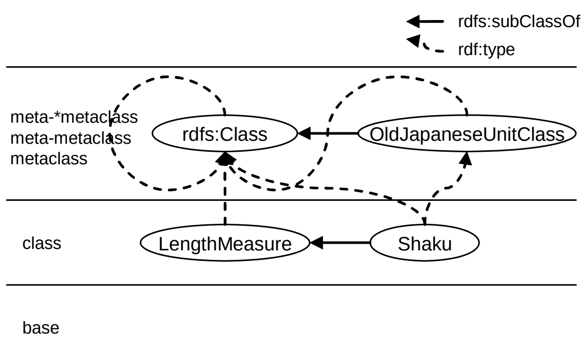

Fig. 5. An example of folded RDF graph structure of a metaclass

4 Connection of RDF Universe and OWL Universe

The document on OWL semantics [OWLRDFS] states that there are two different styles on the connection between the RDF universe and the OWL universe. In OWL-Full style, elements of the OWL universe are identified to the elements in RDF universe. In OWL-DL style, elements of the OWL universe are different from their RDF counterparts. In this section, we discuss the OWL-Full style connection according to OO modeling principle.

All things in the RDF universe are instances of rdfs:Resource [RDFS]. The reason is described in the following list from OO perspective. the subsumption and transitivity rules in RDF(S) are shown at Table 1 [RDFMT].

Table 1. Subsumption and Trasitivity Rules

Rule	If Contains	Then Add
Subsumption	uuu rdfs:subClassOf xxx . vvv rdf:type uuu .	vvv rdf:type xxx .
Transitivity	uuu rdfs:subClassOf vvv . vvv rdfs:subClassOf xxx .	uuu rdfs:subClassOf xxx .

- All metaclasses, which are instances of rdfs:Class, are instances of rdfs:Resource, because rdfs:Class is a subclass of rdfs:Resource.
- rdfs:Class is also an instance of rdfs:Resource, because rdfs:Class (metaclass) is an instance of rdfs:Class (meta-metaclass) and rdfs:Class (meta-metaclass) is a subclass of rdfs:Resource.
- All classes, which are instances of metaclasses, are instances of rdfs:Resource, because all metaclasses are subclasses of rdfs:Resource with the premising condition that all metaclasses hold rdfs:Class as superclass. Thus, the premising condition is required for metaclasses in the RDF universe.
- All instances, which are instances of classes, are instances of rdfs:Resource, because all classes are subclasses of rdfs:Resource in the RDF universe.

In other words, the condition that all metaclasses must hold rdfs:Class as superclass, yields the metaclass layer. The aggregation of all classes, which are instances of rdfs:Class but do not hold rdfs:Class as superclass makes the class layer.

At first in the OWL universe discussion, all things in the OWL universe should be instances of owl:Thing as well as all things in the RDF universe are instances of rdfs:Resource. Namely, the following conditions are requried.

- All metaclasses in the OWL universe, which are instances of owl:Class, should be instances of owl:Thing. It implies that owl:Class should be a subclass of owl:Thing.
- To let owl:Class be in the OWL universe, owl:Class should be an instance of owl:Class itself, supposing owl:Class is a subclass of owl:Thing.
- All classes in the OWL universe, which are instances of OWL metaclasses, should be instances of owl:Thing, It implies that all metaclasses should hold owl:Class as superclass, supposing owl:Class is a subclass of owl:Thing.
- All instances in the OWL universe, which are instances of classes, should be instances of owl:Thing. It implies that all classes should be subclasses of owl:Thing.

However, those conditions are not satisfied in the OWL schema definition, in which owl:Class and owl:Thing is defined as follows.[10]

[10] http://www.w3.org/2002/07/owl.rdf

```
<rdfs:Class rdf:ID="Class">
  <rdfs:label>Class</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<Class rdf:ID="Thing">
  <rdfs:label>Thing</rdfs:label>
  <unionOf rdf:parseType="Collection">
    <Class rdf:about="#Nothing"/>
    <Class>
      <complementOf rdf:resource="#Nothing"/>
    </Class>
  </unionOf>
</Class>
```

In this definition, an instance of owl:Thing belongs to the OWL universe but an instance of owl:Class (class and metaclass except owl:Class itself) does not belong to the OWL universe. owl:Class should be a subclass of owl:Thing as well as rdfs:Class is a subclass of rdfs:Resource, in order that an instance of owl:Class is also an instance of owl:Thing.

In addition, in order to let owl:Class be an instance of owl:Thing, owl:Class must be an instance of itself, while the meta-circularity of owl:Class is not implemented in SWCLOS yet. We axiomatized the following in SWCLOS.

```
<rdfs:Class rdf:ID="Class">
  <rdf:type rdf:resource="#Class"> <!-- This line is not implemented. -->
  <rdfs:subClassOf rdf:resource="#Thing"/>
</rdfs:Class>
```

Second, the above conditions for establishing the OWL universe do not produce any contradiction in the connection of the OWL universe and the RDF universe in the sense of Object-Oriented modeling.

`owl:Thing` should be a subclass of `rdfs:Resource`. Otherwise, every instance of `owl:Thing` shall not be an instance of `rdfs:Resource`. Then, we axiomatized `owl:Thing` as follows in SWCLOS, although such an axiom is implemented in CLOS language level and invisible in RDF expression. Figure 6 illustrates the connection of RDF universe and OWL universe in the OWL-Full style.

```
<Class rdf:ID="Thing">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</Class>
```

All of the OWL vocabularies should be a division of three parts of the RDF universe, namely individuals, classes, and properties [OWLRDFS]. Note that this statement is satisfied because owl:Thing is a subclass of rdfs:Resource as discussed here, in addition that owl:Class is a subclass of rdfs:Class and owl:ObjectProperty/owl:DatatypeProperty is a subclass of rdf:Property in the definition of the OWL schema file.

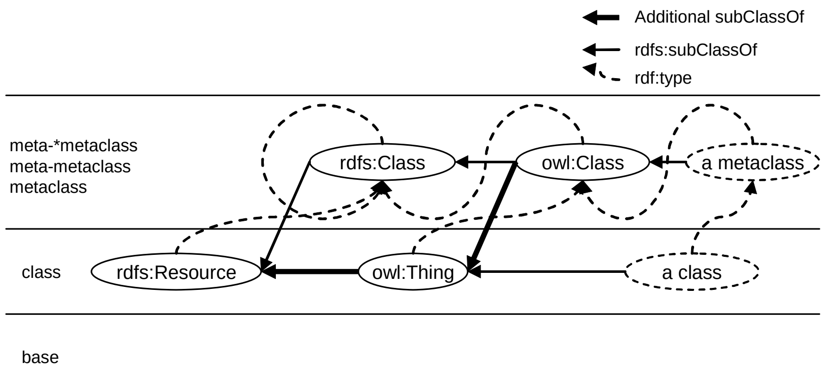

Fig. 6. The Connection of RDF Universe and OWL Universe

In practice, SWCLOS that embodies RDF(S) semantics read the OWL definition file and creates elements of OWL vocabularies as RDF elements, then additional axioms and functions for OWL are installed so that the OWL semantics is established in the SWCLOS booting process.

5 Discussion and Related Works

Harmelen, et al. [Harmelen1992] addressed a comprehensive discussion on the reflection. They emphasized the separated layered architecture in reflective knowledge systems in order to circumvent the self-referentiality for the purpose of the conceptual clarity and modularity. Pan and Horrocks have proposed the fixed layered metamodeling architecture for RDF [Pan2003] and OWL [Pan2005]. The motivation comes from DL-based implementation.

We argue that SWCLOS is a modeling language system that is not separated into metamodel layers, as it is implemented on top of CLOS, a reflective OO system, and the RDF(S) and OWL metamodeling structure is straight-forwardly mapped onto the CLOS structure of metaclasses, classes, and instances. Ontologies are separated into the base layer, class layer, and metaclass layer in the single object system. The rationale of the layered architecture is conveyed from the RDF(S) layered architecture. RDF(S) and OWL semantics are embodied by the built-in model of RDF(S) and OWL in SWCLOS. Ontologies modeled in RDF(S) and OWL by users also exist in SWCLOS. If we define an element of RDF universe using `rdfs:Class` and `rdfs:Resource` in SWCLOS, the defined element exists in the same universe of `rdfs:Class` and `rdfs:Resource` in the SWCLOS computational environment. There are no differences between user-defined elements and built-in elements such as `rdfs:Resource` in their software implementation. Every element in RDF(S) and OWL universe is an object in

CLOS. The causal connection[11][Maes1987] in reflection is involved in SWCLOS. If we change an object in RDF(S) and OWL universe, the change is identical in the base level, class level, and metaclass level. This language embodiment is different from modeling languages like ODM or DL-based languages.

The semantics of models that are represented by separated ontological modeling languages must be underpinned by another way, e.g., using OCL[12], SCL[13], or some modeling language outside that is attached or embedded to modeling systems. Widhalm and Mueck [Widhalm2003] used OCL in order to keep semantic constraints for merging Topics in Topic Map. Kaneiwa and Satoh [Kaneiwa2005] utilized First Order Logic and Counting Quantifieres in order to validate models represented by UML. The coverage of semantics and modeling capability of modeling languages is different each other. Therefore, interesting question is how ODM supports individual modeling language semantics in the common framework.

6 Concluding Remarks

In this paper, we discussed the metamodeling mechanism in OOP, MOF, RDF(S) and OWL, and the reflection of RDF(S) from the perspective of OOP. We addressed the clear image of the OWL-Full style connection between the RDF universe and the OWL universe from OO perspective.

We formalize the metamodeling in RDF(S) and OWL-Full as follows from the discussion in this paper.

- RDF(S) and OWL-Full ontology should be separated into three layers, i.e., base layer, class layer, and metaclass layer.
- An object that is an instance and a subclass of rdfs:Class belongs to the metaclass layer.
- An object that is an instance but not a subclass of rdfs:Class belongs to the class layer.
- An object that is not an instance of rdfs:Class belongs to the base layer.
- An object in the class layer must not be a subclass of rdfs:Class, and is distinguished from objects in the metaclass layer.
- An object in the metaclass layer can be a metaclass, meta-metaclass, meta-meta-metaclass, and so on, but cannot be recognized which metaclass layer it is in. The role must be interpreted in the context.

We expect that the above formalization on metamodeling increases the decidability on RDF(S) and OWL-Full computation, although we cannot decide which internal metaclass layer an object in the metaclass layer belongs to.

[11] The object level condition must reflect the metalevel condition, and vice vase. In separated modeling languages, the change in one layer must flow up or down to another layer in runtime. In single modeling language that shares the computational environment, variables and objects are shared among separated ontological layers.

[12] http://www.omg.org/docs/ptc/03-10-14.pdf

[13] http://cl.tamu.edu/

References

[Bowen1986] Bowen, K.: Meta-level Techniques in Logic Programming. Int. Conf. AI and its Applications, Singapore (1986)

[Feferman1962] Feferman, S.: Transfinite Recursive Progressions of Axiomatic Theories. J. Symbolic Logic, **27**-3 (1962) 259–316

[Gruber1994] Gruber, T. R., Olsen, G. R.: An Ontology for Engineering Mathematics. 4th Int. Conf. Principles of Knowledge Representation and Reasoning, Bonn, Morgan Kaufmann (1994)

[Harmelen1992] Harmelen, F. van, et al.: Knowledge-level Reflection. Pate and Steels (eds.), Enhancing the Knowledge Engineering Process — Contributions from ESPRIT, Elsevier (1992) 175–204

[Kaneiwa2005] Kaneia, Satoh: Consistency Checking Algorithms for Restricted UML Class Diagrams. NII Technical Report, NII-2005-013E, National Institute of Informatics (2005)

[Kiczales1992] Kiczales, G., des Rivières, J., Bobrow, D.G.: The Art of the Metaobject Protocol, MIT Press, (1992)

[Koide2006] Koide S., Takeda H.: OWL-Full Reasoning from an Object Oriented Perspective. Asian Semantic Web Conf. ASWC2006, Springer (2006) 263–277

[Maes1987] Maes, P.: Computational Reflection. Technical Report **87**-2, Free university of Brussels (1987)

[Mellor2004] Mellor, S. J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley Professional (2004)

[OWLRDFS] Patel-Schneider, P. F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax Section 5. RDF-Compatible Model-Theoretic Semantics. W3C Recommendation (2004-2). http://www.w3.org/TR/owl-semantics/rdfs.html

[Paepcke1993] Paepcke, A.: User-Level Language Crafting Introducing the CLOS Metaobject Protocol. Paepcke (ed.) Object-Oriented Programming — The CLOS Perspective, MIT Press, (1993) 65–99

[Pan2003] Pan, J. Z., Horrocks, I.: RDFS(FA) and RDF MT: Two Semantics for RDFS. Proc. 2nd Int. Semantic Web Conf. (ISWC2003), Sanibel Island (2003) 30–46

[Pan2005] Pan, J. Z., Horrocks, I., Schreiber, G.: OWL FA: A Metamodeling Extension of OWL DL. Workshop OWL: Experiences and Directions, Galway (2005)

[RDFMT] Hayes, P., McBride, B.: RDF Semantics. W3C Recommendation (2004-2). http://www.w3.org/TR/rdf-mt/

[RDFS] Brickley, D., Guha, R. V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (2004-2). http://www.w3.org/TR/rdf-schema/

[Smith1984] Smith, B.: Reflection and Semantics in Lisp. POPL '84: 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (1984) 23–35

[Weyhrauch1980] Weyhrauch, R. W.: Prolegomena to a Theory of Mechanized Formal Reasoning. Artificial Intelligence, **13** (1980) 133–170

[Widhalm2003] Widhalm, R., Mueck, T. A.: Merging Topics in Well-Formed XML Topic Maps. Int. Semantic Web Conf., Sanibel Island (2003) 64–79