Meta-Circularity and MOP in Common Lisp for OWL Full

Seiji Koide*
The Graduate University for Advanced Studies (SOKENDAI)
2-1-2 Hitotsubashi
Chiyoda-ku, Tokyo 101-8430
koide@nii.ac.jp

Hideaki Takeda†
National Institute of Informatics and SOKENDAI
2-1-2 Hitotsubashi
Chiyoda-ku, Tokyo 101-8430
takeda@nii.ac.jp

ABSTRACT

We have developed an OWL (Web Ontology Language) Full language processor, SWCLOS, for processing semantic web pages on top of the Common Lisp Object System (CLOS). To implement the OWL Full level of capability, we leveraged the dynamic and reflective features of CLOS. The metamodeling capability of CLOS is utilized to realize the metamodeling capability of Resource Description Framework (RDF) and OWL Full. The native computational model of CLOS is changed into the model of RDF and OWL by using the Meta-Object Protocol (MOP) in CLOS. Although the metamodeling specifications in CLOS are firmly established in the Common Lisp community, the semantics is not yet fully developed, since Common Lisp does not have formal semantics. In this paper, we focus on metamodeling in CLOS. We point out that the architecture of CLOS in the metamodeling is the same as in RDF and clarify the denotational semantics of CLOS in comparison with the RDF semantics.

Categories and Subject Descriptors

F.3.2 [**LOGICS AND MEANINGS OF PROGRAMS**]: Semantics of Programming Languages—*Denotational semantics*

General Terms

Design, Standardization, Languages, Theory

Keywords

CLOS, Semantic Web, metacircularity, membership circularity, denotational semantics, extensional semantics, RDF, RDFS, OWL, OWL Full, metamodeling, meta-programming, MOP

*The first author also works for IHI Corporation in Japan.

†The second author is also the Sumitomo endowed professor at Research into Artifacts, Center of Engineering (RACE) in the University of Tokyo.

ELW'09, July 6, 2009 Genova, Italy

1. INTRODUCTION

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web [8]. The special syntax in XML called RDF/XML [1] and the model-theoretic semantics of RDF [3] have been issued as W3C Recommendations. The RDF Schema (RDFS) is a semantic extension of RDF that provides a minimal set for describing classes of objects in an ontology [3]. The Web Ontology Language (OWL) is also W3C Recommendation [9]. It is designed for representing the content of information as an ontology; it facilitates greater machine interpretability of web content than is supported by XML, RDF, or RDFS, by providing additional vocabulary along with formal semantics. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full. The specifications of OWL Lite are considerably restricted in order to facilitate easy implementation. The computations and realization of OWL DL are underpinned by Description Logics (DLs), but the specifications are more conditioned than those of OWL Full. OWL Full is aimed at full compatibility with RDF.

The model theoretic RDF semantics is specified by denotational semantics, and RDF allows for *membership circularity* of the class extensions in the universe of discourse [3]. In particular, the class extension of the denotation of rdfs:Class in the RDFS vocabulary includes the denotation of rdfs:Class itself, and this membership circularity has caused confusion and even arguments in the OWL community, especially in the OWL DL community [13, 14, 11], since such a membership loop seems to violate the axiom of foundation, one of the axioms of standard (Zermelo-Fraenkel) set theory.

On the other hand, in the Common Lisp Object System (CLOS), all Common Lisp objects, including classes, are an instance of a class [4]. Hence, the class objects are called *metaobjects* [2], and we also see *metacircularity* of the class-instance at cl:standard-class in CLOS just like the membership circularity at rdfs:Class in the RDF universe. Therefore, there is no bewilderment in the Lisp community with respect to the membership-loop. We previously implemented an RDF and OWL Full language processor on top of CLOS [6, 5, 7], through which we bridged the semantic gap between CLOS and RDF or OWL Full by leveraging the metamodeling capability of CLOS and using the Meta-Object Protocol (MOP).

Although there are as yet no formal semantic specifications of Common Lisp and CLOS, it is obvious that a discussion of languages based on formal semantics is useful to disambiguate the specifications of languages and help-

ful for implementing language processors. In this paper, we focus on the denotational semantics of metacircularity in Common Lisp in comparison with those in RDF and OWL. This paper revisits the type and subtype semantics of Common Lisp in the context of denotational and extensional semantics and redefines metacircularity at cl:standard-class in CLOS without the notions of methods or inheritance. The discussion brings with it the new perspective of CLOS and meta-programming in CLOS.

The remainder of this paper is structured as follows. Section 2 discusses the denotational semantics between RDF and CLOS and points out the similarity of metacircularity in each. The disambiguation for compound CLOS types by the denotational semantics is also discussed. Section 3 demonstrates metamodeling in CLOS and OWL ontologies and shows metamodeling disorders in several ontologies that are regarded as standard. We address an idea of CLOS-clean metamodeling. Section 4 extends the discussion to the reflective tower in metamodeling.

2. DENOTATIONAL SEMANTICS

2.1 Introduction of Denotational Semantics

In the denotational semantics of computer languages, it is important to distinguish lexical components in syntactical expressions from their denotations. While the expressions in languages have certain structures (syntaxes), the meaning of a component is recursively described by the sub-components of the syntactical structure. For example, consider the following syntactic expression of ontology[1].

```
ClassAssertion(a:Dog a:Brian)
ClassAssertion(a:Species a:Dog)
```

In these sentences, the meaning of each sentence is defined by components 'ClassAssertion', 'a:Dog', 'a:Brain', 'a:Species' and the structures. The first sentence expresses that Brian is a dog, and the second expresses that a dog is a species. The token or word 'a:Dog' appears twice in two sentences, but the first token denotes the dog as a class and the second denotes the dog as an instance of species. In the RDF and OWL Full semantics, these two sentences are interpreted as follows.

$$\begin{aligned} a{:}Brian^{\mathcal{I}} &\in CEXT^{\mathcal{I}}(a{:}Dog^{\mathcal{I}}) \\ a{:}Dog^{\mathcal{I}} &\in CEXT^{\mathcal{I}}(a{:}Species^{\mathcal{I}}) \end{aligned}$$

Here, $a{:}Dog^{\mathcal{I}}$ represents the denotation of the token 'a:Dog', and it is indicated through a mapping $S^{\mathcal{I}}$ from lexical tokens to entities in the *universe of discourse*.

$$\begin{aligned} a{:}Brian^{\mathcal{I}} &= S^{\mathcal{I}}(\text{a : Brian}) \\ a{:}Dog^{\mathcal{I}} &= S^{\mathcal{I}}(\text{a : Dog}) \\ a{:}Species^{\mathcal{I}} &= S^{\mathcal{I}}(\text{a : Species}) \end{aligned}$$

$CEXT^{\mathcal{I}}(x)$ is called the class extension of x, and it represents a set of instances of x. Here, x is called a class.

2.2 Semantics in the RDF Universe

The RDF semantics is specified using the denotational semantics that is based on model-theoretic semantics and first order logic with equality [3]. The RDF semantics is described as follows with the RDF simple interpretation $\mathcal{I}$ of vocabulary $\mathcal{V}$.

1. A non-empty set $\boldsymbol{R}^{\mathcal{I}}$ of entities, called the domain or universe of $\mathcal{I}$.
2. A set $\boldsymbol{P}^{\mathcal{I}}$, called the set of properties of $\mathcal{I}$.
3. A mapping $EXT^{\mathcal{I}}$ from $\boldsymbol{P}^{\mathcal{I}}$ into the powerset of $\boldsymbol{R}^{\mathcal{I}} \times \boldsymbol{R}^{\mathcal{I}}$, i.e., the set of sets of pairs $\langle x, y\rangle$ with x and y in $\boldsymbol{R}^{\mathcal{I}}$.
4. A mapping $S^{\mathcal{I}}$ from URI references in $\mathcal{V}$ into $\boldsymbol{R}^{\mathcal{I}} \cup \boldsymbol{P}^{\mathcal{I}}$.
5. A mapping $L^{\mathcal{I}}$ from typed literals in $\mathcal{V}$ into $\boldsymbol{R}^{\mathcal{I}}$.
6. A distinguished subset LV of $\boldsymbol{R}^{\mathcal{I}}$, called the set of literal values, which contains all the plain literals in $\mathcal{V}$.

Here, $EXT^{\mathcal{I}}(p)$ is called the property extension of p, and it represents a set of sets of pairs $\langle x, y\rangle$, where x and y are entities in the RDF universe. In other words, a property makes a set of the binary relation between entities in the RDF universe. Note that the order of a pair $\langle x, y\rangle$ is important so that $\langle x, y\rangle$ must be distinguished from $\langle y, x\rangle$.

The notion of property is introduced with rdf:Property and rdf:type in rdf vocabulary as follows.

Axiom 1. If an entity is a member of the set of properties of $\mathcal{I}$, then the entity makes a pair with $rdf{:}Property^{\mathcal{I}}$ and the pair is a member of property extension of $rdf:type^{\mathcal{I}}$, and vice versa:

$$x \in \boldsymbol{P}^{\mathcal{I}} \quad \textit{iff} \quad \langle x, rdf{:}Property^{\mathcal{I}}\rangle \in EXT^{\mathcal{I}}(rdf{:}type^{\mathcal{I}})$$

A particular pair of $\langle x, y\rangle$ for property p is also called a triple in infix notation or $x\ p\ y$. In this context, x is called subject, y is called object, and p is called predicate. A set of triples is called an RDF graph in Semantic Web. An RDF graph may include blank nodes. A blank node has no URI reference and may be designated by a *nodeID* instead of a URI reference. An RDF graph that does not include blank nodes is called a ground graph. The denotation of a ground RDF graph in $\mathcal{I}$ is given recursively by the semantic conditions for ground triples in RDF semantics, and the semantics of ungrounded graphs is extended from the ground graphs. (See [3] for details.)

The notion of class-instance is introduced as an rdfs-extension of the RDF universe. For rdfs-vocabulary, i.e., rdfs:Class, rdfs:Resource, rdfs:subClassOf, etc., in $\mathcal{V}$, rdfs-interpretation satisfies the extra conditions for RDFS [3]. In the following axiom and definitions, $\boldsymbol{C}^{\mathcal{I}}$ represents a class extension of the denotation of rdfs:Class in the RDF universe, and $\boldsymbol{R}^{\mathcal{I}}$, which is initially defined as the universe of discourse in the rdf simple interpretation, is realized as a class extension of the denotation of rdfs:Resource.

Axiom 2. If an entity is a member of class extension of another entity, then a pair of both becomes a member of property extension of $rdf{:}type$, and vice versa.

$$x \in CEXT^{\mathcal{I}}(y) \quad \textit{iff} \quad \langle x, y\rangle \in EXT^{\mathcal{I}}(rdf{:}type^{\mathcal{I}})$$

This axiom displays the semantics of the class and instance. Here, x is called an instance of class y, and the class-instance relation is expressed through the property rdf:type.

[1]This example is taken from http://www.w3.org/TR/owl2-syntax/#Metamodeling.

We define several classes and their extensions of rdfs-vocabulary in the RDF universe as follows.

$$\begin{aligned} \boldsymbol{C}^{\mathcal{I}} &= CEXT^{\mathcal{I}}(rdfs:Class^{\mathcal{I}}) \\ \boldsymbol{DC}^{\mathcal{I}} &= CEXT^{\mathcal{I}}(rdfs:Datatype^{\mathcal{I}}) \\ \boldsymbol{R}^{\mathcal{I}} &= CEXT^{\mathcal{I}}(rdfs:Resource^{\mathcal{I}}) \\ LV &= CEXT^{\mathcal{I}}(rdfs:Literal^{\mathcal{I}}) \end{aligned}$$

The class-superclass relation in the RDF universe is specified with the property rdfs:subClassOf as the inclusiveness of the class extensions of class and its superclass as follows.

Axiom 3. If a pair of two entities is a member of property extensions of $rdfs{:}subClassOf^{\mathcal{I}}$, then the both entities are instances of $rdfs:Class^{\mathcal{I}}$ and the class extension of the predecessor in the pair is included by the class extension of the successor.

$$\begin{aligned} &\langle x, y\rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}}) \Rightarrow \\ &\quad x, y \in \boldsymbol{C}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x) \subseteq CEXT^{\mathcal{I}}(y) \end{aligned}$$

Namely, the class extension of a subclass x is included by the class extension of its superclass y. This is called *subsumption*. Thus, every instance of class x is also an instance of the superclass y.

The property rdfs:subClassOf in rdfs-vocabulary is transitive and reflexive on the property extension.

$$\begin{aligned} &\langle x, y\rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}}) \wedge \\ &\quad \langle y, z\rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}}) \\ &\quad \Rightarrow \langle x, z\rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}}) \end{aligned}$$

$$x \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow \langle x, x\rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}})$$

Furthermore, the final axiom with respect to an instance of rdfs:Class is introduced as follows.

Axiom 4. Every instance of $rdfs:Class^{\mathcal{I}}$ is a subclass of $rdfs:Resource^{\mathcal{I}}$.

$$x \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow \langle x, rdfs:Resource^{\mathcal{I}}\rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}})$$

2.3 Meta-Circularity of rdfs:Class

Here, let us introduce a new symbol $\sqsubseteq$ to simplify the description of rdfs:subClassOf relation.

$$\langle x, y\rangle \in EXT^{\mathcal{I}}(rdfs:subClassOf^{\mathcal{I}}) \rightarrow x \sqsubseteq y$$

Then, the last two axioms on subsumption of rdfs:subClassOf are rephrased as follows.

$$x \in \boldsymbol{C}^{\mathcal{I}} \Rightarrow x \sqsubseteq rdfs:Resource^{\mathcal{I}} \quad (1)$$

$$x \sqsubseteq y \Rightarrow x, y \in \boldsymbol{C}^{\mathcal{I}} \wedge CEXT^{\mathcal{I}}(x) \subseteq CEXT^{\mathcal{I}}(y) \quad (2)$$

Axiom (1) means every class that is an instance of the denotation of rdfs:Class is a subclass of the denotation of rdfs:Resource. Axiom (2) means any class in rdfs:subClassOf relation is an instance of the denotation rdfs:Class.

Hence, axiom (1) and (2) lead to the following lemma in the combination.

Lemma 1.

$$x \sqsubseteq rdfs:Resource^{\mathcal{I}} \Leftrightarrow x \in \boldsymbol{C}^{\mathcal{I}} \quad (3)$$

Namely, any subclass of rdfs:Resource is an instance of (the denotation of) rdfs:Class or a member of the class extension of (the denotation of) rdfs:Class.

Let us set an intuitively true axiom for the RDF universe in order to discuss the metacircularity in the RDF universe.

Axiom 5. $rdfs:Resource^{\mathcal{I}}$ is a superclass of every class in the RDF universe, including the denotation of rdfs:Class, because $rdfs:Resource^{\mathcal{I}}$ is the top class in the RDF universe and the denotation of rdfs:Class is in the RDF universe:

$$rdfs:Class^{\mathcal{I}} \sqsubseteq rdfs:Resource^{\mathcal{I}} \quad (4)$$

Thus, we can obtain metacircularity of rdfs:Class by replacing x in (3) with rdfs:Class as follows.

Lemma 2. If $rdfs:Class^{\mathcal{I}}$ is in the RDF universe and a subclass of $rdfs:Resource^{\mathcal{I}}$, then a membership loop occurs in $rdfs{:}Class^{\mathcal{I}}$.

$$rdfs:Class^{\mathcal{I}} \sqsubseteq rdfs:Resource^{\mathcal{I}} \Leftrightarrow rdfs:Class^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \quad (5)$$

Conversely, if we set the membership loop of rdfs:Class, lemma (5) produces rdfs:Class as a subclass of rdfs:Resource. In a nutshell, the membership loop of rdfs:Class is equivalent to the subsumption of rdfs:Class into rdfs:Resource under axioms (1) and (2).

We often call rdfs:Class a 'universal' class, of which the extension contains all classes, including itself, in the domain, and call rdfs:Resource the top class, which takes its place at the top of all classes, including the universal class, in the class-superclass or subsumption hierarchy.

Let us call the specialized consequent of (2) for rdfs:Class and rdfs:Resource a twisted relation of the universal class and top class.

$$rdfs:Resource^{\mathcal{I}} \in \boldsymbol{C}^{\mathcal{I}} \wedge rdfs:Class^{\mathcal{I}} \sqsubseteq rdfs:Resource^{\mathcal{I}} \quad (6)$$

In this case, every class in the domain is an instance of the 'universal' class, and every class which is an instance of 'universal' class belongs to the domain. If the membership loop is admitted into a 'universal' class, the 'universal' class also belongs to the domain. Thus, we can regard the domain as closed by the universal class and the top class.

2.4 Mapping CLOS to RDF

Regarding the class-instance and the class-superclass relations, the semantics of these relations in CLOS are the same as membership and subsumption in the RDF universe. Therefore, by capturing CLOS objects as entities in the RDF universe, we can map the class-instance relation in CLOS to an rdf:type relation in RDF semantics and a class-superclass relation in CLOS to rdfs:subClassOf relation in RDF semantics, and vice versa. Note that CLOS retains the transitive and reflexive properties of the class-superclass relation.

Note that the relationship between cl:standard-class and cl:standard-object is the same as the relationship between rdfs:Class and rdfs:Resource in the RDF specification. That is, cl:standard-class is the universal class and cl:standard-object is the top class in CLOS. Therefore, the topological structure of the twisted relationship of cl:standard-class and cl:standard-object forms the domain of CLOS objects.

$$\begin{aligned} &\text{cl:standard-object}^{\mathcal{I}} \in CEXT^{\mathcal{I}}(\text{cl:standard-class}^{\mathcal{I}}) \wedge \\ &\qquad \text{cl:standard-class}^{\mathcal{I}} \sqsubseteq \text{cl:standard-object}^{\mathcal{I}} \end{aligned} \quad (7)$$

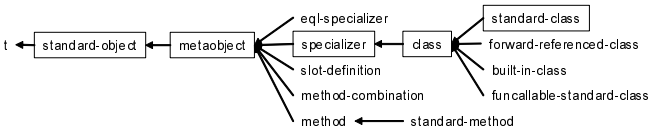

Figure 1: Part of the Class Hierarchy in CLOS.

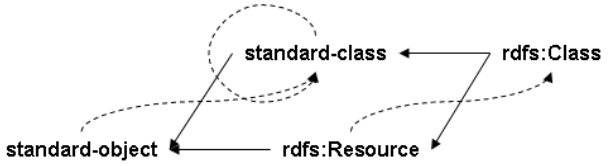

Figure 2: The Twisted Relationship of rdfs:Class and rdfs:Resource in CLOS.

Furthermore, mop:metaobject, mop:specializer, and cl:class create sub-domains inside the CLOS object domain, because they also make twisted relationships with respect to cl:standard-class. **Figure 1** shows part of the class hierarchy of CLOS [4]. Every entity in the figure except `t` is an instance of cl:standard-class.

To form the RDF universe in CLOS objects, we implemented the twisted relationship of rdfs:Class and rdfs:Resource under the twisted relationship of cl:standard-class and cl:standard-object as shown in **Figure 2**. However, we needed a trick to implement the membership loop for rdfs:Class in CLOS, because CLOS inhibits any direct and indirect membership loops except cl:standard-class. SWCLOS users are recommended to use gx:typep instead of cl:typep. The functionality of gx:typep is almost the same as that of cl:typep, but it pretends that there is metacircularity in rdfs:Class.

2.5 Disjointness and Compound Type Specifiers in CLOS

In the specification of ANSI Common Lisp, it is said that any two distinct classes created by defclass in CLOS are disjoint unless they have a common subclass or one class is a subclass of the other[2]. This agreement is supported by the premise that an object in CLOS is typed to only one class. However, in the RDF universe, an entity can be typed to multiple classes. So, the nature of disjointness in CLOS is not applicable in the RDF universe in theory. However, SWCLOS implements pseudo multiple-classing machinery, by using the CLOS class and the multiple-inheritance mechanism. Therefore, the algorithm of disjointness for CLOS is still valid in SWCLOS by virtue of CLOS.

The specifications of ANSI Common Lisp also describe the compound type specifiers 'and', 'or', and 'not' as follows:

And denotes the set of all objects of the type determined by the intersection of the typespecs[3].

Or denotes the set of all objects of the type determined by the union of the typespecs. For example, the type list by definition is the same as (or null cons)[4].

[2]http://www.lispworks.com/documentation/HyperSpec/Body/04_bb.htm

[3]http://www.lispworks.com/documentation/HyperSpec/Body/t_and.htm

[4]http://www.lispworks.com/documentation/HyperSpec/Body/t_or.htm

Table 1: An Example of Ambiguity in CLOS

proc	Q1	Q2	Q3	Q4	Q5	Q6
A	$\langle t,t\rangle$	$\langle f,t\rangle$	$\langle f,t\rangle$	$\langle t,t\rangle$	$\langle f,f\rangle$	$\langle f,f\rangle$
B	$\langle f,f\rangle$	$\langle f,t\rangle$	$\langle f,t\rangle$	$\langle t,t\rangle$	$\langle f,t\rangle$	$\langle f,t\rangle$
C	$\langle t,t\rangle$	$\langle f,t\rangle$	$\langle f,t\rangle$	$\langle t,t\rangle$	$\langle f,t\rangle$	$\langle f,t\rangle$
D	$\langle t,t\rangle$	$\langle f,t\rangle$	$\langle f,t\rangle$	$\langle t,t\rangle$	$\langle f,t\rangle$	$\langle f,t\rangle$

Not denotes the set of all objects that are not of the type typespec[5].

However, it seems that ANSI Common Lisp does not clearly express the extensional semantics such as shown in Subsection 2.2. As shown in **Table 1**, some Common Lisp processors return different values for the following very simple subsumption test. Note that `subtypep` may return two values of `t` and `nil`. The expression $\langle t,t\rangle$ in Table 1 means true, $\langle f,t\rangle$ or $\langle nil,t\rangle$ means false, and $\langle f,f\rangle$ or $\langle nil,nil\rangle$ means unknown.

```
(defparameter a (defclass a ()()))
(defparameter b (defclass b ()()))

Query 1: (subtypep '(and a b) a)
Query 2: (subtypep a '(and a b))
Query 3: (subtypep '(or a b) a)
Query 4: (subtypep a '(or a b))
Query 5: (subtypep '(not a) a)
Query 6: (subtypep a '(not a))
```

The ambiguity also comes from the ANSI specification that subtypep is permitted to return two values, false and false, only when at least one argument involves one of these type specifiers: and, eql, the list form of function, member, not, or, satisfies, or values. Therefore, the answers listed above are permissible in ANSI Common Lisp. However, the correct answer is obvious from the extensional semantics of 'and', 'or', and 'not', as follows.

$$
\begin{aligned}
CEXT^{\mathcal{I}}(x \wedge y) &\equiv CEXT^{\mathcal{I}}(x) \cap CEXT^{\mathcal{I}}(y) \\
CEXT^{\mathcal{I}}(x \vee y) &\equiv CEXT^{\mathcal{I}}(x) \cup CEXT^{\mathcal{I}}(y) \\
CEXT^{\mathcal{I}}(x) &\equiv CEXT^{\mathcal{I}}(\text{cl:standard-object}^{\mathcal{I}}) - CEXT^{\mathcal{I}}(x)
\end{aligned}
$$

To disambiguate the semantics and realize a procedural `cl:subtypep` algorithm in Common Lisp for compound types of CLOS classes, ternary truth value logics are required. In SWCLOS2 (revised version), a truth table shown in **Table 2** is implemented in `gx:subtypep`. In the table, T means true (represented by $\langle t,t\rangle$), F means false ($\langle nil,t\rangle$), and U means unknown ($\langle nil,nil\rangle$) in RDFS and OWL semantics. Furthermore, to resolve the compound type and compute the truth value, we adopted rewriting rules for inclusiveness of class extensions, as shown in **Table 3**

3. METAMODELING

3.1 Metamodeling in CLOS

CLOS provides the capability of metamodeling, so a class can be treated as an instance of metaclasses, such as follows:

[5]http://www.lispworks.com/documentation/HyperSpec/Body/t_not.htm

Table 2: Ternary Truth Table

conjunction

	T	F	U	Ex.
T	T	F	U	T ∧ U → U
F	F	F	F	F ∧ U → F
U	U	F	U	

disjunction

	T	F	U	Ex.
T	T	T	T	T ∨ U → T
F	T	F	U	F ∨ U → U
U	T	U	U	

negation

x	T	F	U	Ex.
$\neg x$	F	T	U	$\neg U \rightarrow$ U

Table 3: Rewriting Rules for Inclusiveness

$C \subseteq (A \wedge B) \Leftrightarrow (C \subseteq A) \wedge (C \subseteq A)$
$C \subseteq (A \vee B) \Leftrightarrow (C \subseteq A) \vee (C \subseteq B)$
$(A \wedge B) \subseteq C \Leftrightarrow (A \subseteq C) \vee (B \subseteq C)$
$(A \vee B) \subseteq C \Leftrightarrow (A \subseteq C) \wedge (B \subseteq C)$
$(A \vee B) \subseteq (C \wedge D) \Leftrightarrow (A \subseteq C) \wedge (A \subseteq D) \wedge (B \subseteq C) \wedge (B \subseteq D)$
$(A \wedge B) \subseteq (C \vee D) \Leftrightarrow (A \subseteq C) \vee (A \subseteq D) \vee (B \subseteq C) \vee (B \subseteq D)$
$(A \wedge B) \subseteq (C \wedge D) \Leftrightarrow ((A \subseteq C) \wedge (A \subseteq D)) \vee ((B \subseteq C) \wedge (B \subseteq D))$
$(A \vee B) \subseteq (C \vee D) \Leftrightarrow ((A \subseteq D) \vee (A \subseteq D)) \wedge ((B \subseteq C) \vee (B \subseteq D))$
$\neg A \subseteq \neg B \Leftrightarrow B \subseteq A$

```
(defpackage :a
  (:export "Dog" "Brian" "Species"))
(defparameter a:Species
  (defclass a:Species (cl:standard-class)()
    (:metaclass cl:standard-class)))
(defparameter a:Dog
  (defclass a:Dog () ()
    (:metaclass a:Species)))
(defparameter a:Brian
  (make-instance 'a:Dog))
```

After these expressions in pure CLOS, the following demonstrates that a:Species is a class of class a:Dog; that is, it is a metaclass.

```
(cl:typep a:Brian a:Dog) → t
(cl:typep a:Dog a:Species) → t
```

We can interpret that these definitions denote the following conditions in the notation of Section 2.

$$a{:}Brian^{\mathcal{I}} \in CEXT^{\mathcal{I}}(a{:}Dog^{\mathcal{I}})$$
$$a{:}Dog^{\mathcal{I}} \in CEXT^{\mathcal{I}}(a{:}Species^{\mathcal{I}})$$

In the above CLOS class definitions, the metaclass of a:Species is designated to cl:standard-class and the direct superclass of a:Species is also designated to cl:standard-class. Such a condition is described as follows.

$$a{:}Species^{\mathcal{I}} \in CEXT^{\mathcal{I}}(\text{cl:standard-class}^{\mathcal{I}}) \wedge a{:}Species^{\mathcal{I}} \sqsubseteq \text{cl:standard-class}^{\mathcal{I}}$$

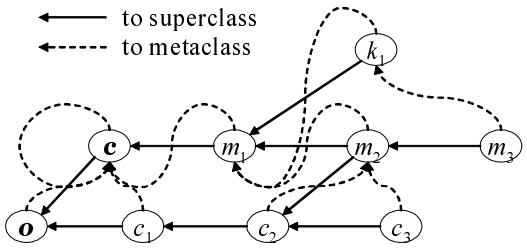

Figure 3: Typical Patterns of Metamodeling in CLOS.

This condition might seem puzzling, but it is the usual way of defining a metaclass in CLOS. Here, let us call a:Brian a base object, a:Dog a strict class (not a metaclass), and a:Species a metaclass (a class of classes). The definitions of base object, strict class, and metaclass are described as follows.

$$\begin{aligned} baseObject \in CEXT^{\mathcal{I}}(\text{cl:standard-object}^{\mathcal{I}}) \wedge \\ baseObject \notin CCEXT^{\mathcal{I}}(\text{cl:standard-class}^{\mathcal{I}}) \wedge \\ baseObject \not\sqsubseteq \text{cl:standard-class}^{\mathcal{I}} \end{aligned}$$

$$\begin{aligned} strictClass \in CEXT^{\mathcal{I}}(\text{cl:standard-object}^{\mathcal{I}}) \wedge \\ strictClass \in CEXT^{\mathcal{I}}(\text{cl:standard-class}^{\mathcal{I}}) \wedge \\ strictClass \not\sqsubseteq \text{cl:standard-class}^{\mathcal{I}} \end{aligned}$$

$$\begin{aligned} metaClass \in CEXT^{\mathcal{I}}(\text{cl:standard-object}^{\mathcal{I}}) \wedge \\ metaClass \in CEXT^{\mathcal{I}}(\text{cl:standard-class}^{\mathcal{I}}) \wedge \\ metaClass \sqsubseteq \text{cl:standard-class}^{\mathcal{I}} \end{aligned}$$

The rationale behind metacircularity of cl:standard-class in CLOS can be clarified by providing a meta-programming capability. We can modify the behavior of classes that are subclasses of cl:standard-object and instances of cl:standard-class by creating a metaclass as a subclass of cl:standard-class and redefining the standard methods defined for cl:standard-class.

Moreover, the metacircularity of cl:standard-class allows us to build a virtually infinite metamodeling tower.

3.2 Ontological Metamodeling

Whereas there are many ontologies over the WWW, some of them include direct or indirect self-referential metacircularities, and some of them involve ill-structured metaclasses from the CLOS perspective. For example, CollectionType in OpenCyc ontology[6] has an indirect membership loop via VariedOrderCollection. In the SUMO ontology, the strict class sumo:Meter is simultaneously an indirect subclass and instance of strict class sumo:PhysicalQuantity. (See Figure 4.) CLOS and SWCLOS cannot treat such ill-structured metaclass conditions.

Basically, CLOS does not accept any cycle in the class-subclass relation nor class-instance relation except the metacircularity of cl:standard-class. The well-structured metaclass hierarchy in the CLOS perspective is depicted as follows. We call this condition *CLOS-clean metamodeling*. **Figure 3** shows several patterns that are acceptable in metamodeling in the CLOS perspective.

In paticular, the figure reflects that

[6]http://www.opencyc.org/

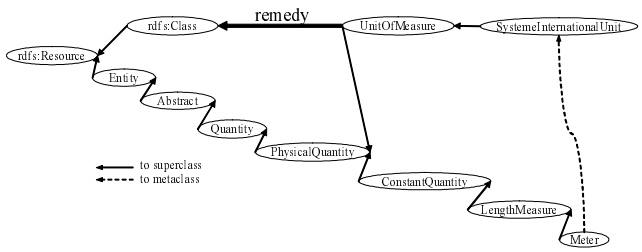

Figure 4: The Remedy for Ill-structured SUMO Ontology.

1. No class can have any direct and indirect cyclic loop in subclass relation.

2. No class can have any direct and indirect membership loop but the direct loop of cl:standard-class (**c** in Figure 3).

3. A parallel relation of subclass and membership can exists anywhere among metaclasses. See, m_1 and m_2, and m_1 and k_1 in Figure 3. Note that the parallel relation can make a branch of metaclass layer surface (m_1 and k_1), but it does not make its own universe, e.g., m_1 and its subclasses belong in the universe of the superclass.

4. A twisted relation of subclass and membership can exists anywhere among classes and metaclasses. See, c_2 and m_2 in Figure 3. The twisted relation makes a sub-universe in which the top class in the sub-universe sits as a superclass of every class and every metaclass in its universe.

Thus, the defect in SUMO ontology with respect to metamodeling can be overcome, as shown in **Figure 4**.

4. REFLECTION IN RDFS AND CLOS

RDFS contains not only a metamodeling mechanism but also a reflection mechanism like in CLOS languages. In this section, we discuss reflection in RDFS and CLOS.

Consider the following examples in the SUMO Ontology with SWCLOS: Meter is a strict class and LengthMeasure is also a strict class, but SystemeInternationalUnit and UnitOfMeasure must be a metaclass with the remedy as described in the previous subsection.

```
(defConcept UnitOfMeasure
  (rdf:type rdfs:Class)
  (rdfs:subClassOf PhysicalQuantity rdfs:Class))
(defConcept SystemeInternationalUnit
  (rdf:type rdfs:Class)
  (rdfs:subClassOf UnitOfMeasure))
(defConcept Meter
  (rdf:type SystemeInternationalUnit)
  (rdfs:subClassOf LengthMeasure)))
```

Here, rdfs:Class occurs twice in UnitOfMeasure definition form, one is as a type and the other as a superclass of UnitOfMeasure. The membership loop at cl:standard-class enables such double roles of metaclasses, i.e., rdfs:Class is just the same as the cl:standard-class.

Let us discuss metamodeling further. Suppose that we treat UnitOfMeasure as an individual; for example, imagine that it has some slot and value as follows.

```
(setf (slot-value UnitOfMeasure 'p1) Value1)
```

To do so, slot `p1` must be defined at the class of UnitOfMeasure. Thus, we define its class as follows.

```
(defConcept UnitOfMeasureClass
  (rdfs:type rdfs:Class)
  (rdfs:subClassOf rdfs:Class))
(defProperty p1
  (rdfs:domain UnitOfMeasureClass))
(defConcept UnitOfMeasure
  (rdf:type UnitOfMeasureClass)
  (rdfs:subClassOf PhysicalQuantity rdfs:Class))
```

In such encoding, UnitOfMeasureClass must be a meta-metaclass, because UnitOfMeasure is a metaclass. Hence, for two occurrences of rdfs:Class in the definition of UnitOfMeasureClass, one must be a meta-metaclass (on rdfs:subClassOf) and the other must be a meta-meta-metaclass (on rdfs:type). This reminds us again of the membership loop in cl:standard-class. The rdfs:Class inherits the virtue of cl:standard-class.

In short, cl:standard-class plays multiple roles, as a metaclass, meta-metaclass, meta-meta-metaclass, and so forth. The membership loop of the cl:standard-class enables multiple roles. Each of the layers in class modeling, i.e., strict class layer, metaclass layer, meta-metaclass layer, etc., that links directly or indirectly to cl:standard-class with the class-superclass relation. Virtually infinite metaclass layers are folded into one layer by the membership loop of cl:standard-class.

Reflection in programming language systems provides the ability to modify the language's implementation without leaving the realm of the language [12]. CLOS is a reflective OOP language system and such an infinite metamodel tower is virtually enabled through the membership loop of cl:standard-class. Thus, if we want to change functions in CLOS, we can do so by redefining methods in the language system with the Meta-Object Protocol [4].

RDF and OWL Full languages are also metamodeling languages. If we wish to use the ontology in practice, we must understand the reflective structure of rdfs:Class and owl:Class in our ontology metamodeling. We already demonstrated metamodeling examples in [7], whereby metamodeling is required to put brand wine IDs in relation to wine concepts by distinguishing brand wines like Zinfandel and non-brand wine concepts like California wine; and in another example, metamodeling is required to realize the OWL-S ontology in a decision support system for rocket launch operations [10].

5. CONCLUSIONS

In this paper, we explained metacircularity and metamodeling in CLOS in terms of denotational semantics obtained from the W3C document of RDF Semantics Recommendation [3]. The membership loop of the cl:standard-class and the twisted relation between cl:standard-class and cl:standard-object are similar to those of the rdfs:Class and rdfs:Resource in the RDF universe. We also gave illustrative examples of ontological metamodeling with reflection.

SWCLOS developed on top of CLOS is a language for ontology description in RDF and OWL, and simultaneously, it is an Object-Oriented Programming language by virtue of CLOS. SWCLOS has been successfully utilized in several projects. Through our experience, we have recognized the advantages of SWCLOS as an amalgamation of an OOP language with RDF and OWL logics.

One extensible direction of SWCLOS is software engineering with Semantic Technology. The Software Engineering Task Force (SETF) of the W3C Semantic Web Best Practices and Deployment Working Group (SWBPD)[7] and the OMG Ontology Working Group[8] are working to integrate object-oriented representation and ontological representation. They are exploring Model-Driven Architectures (MDA) augmented by RDF and OWL. From the viewpoint of OOP, the method is the key issue for software engineering. However, the semantics of methods, specifically as related to RDF and OWL semantics, are still unknown. We intend to tackle the semantics of methods for objects augmented in terms of Semantic Technology.

6. ACKNOWLEDGMENTS

This paper was inspired from our experience of developing of SWCLOS, an RDF and OWL Full language processor for Semantic Webs. Most of the initial work on SWCLOS was done as part of a Japanese IT project titled 'Building a Support System for the Large-Scale Operation System using Information Technology' under contract by the Ministry of Education, Culture, Sports, and Technology (MEXT) from 2002 through 2005. We thank IHI Corporation for supporting us in practically implementing and improving SWCLOS by applying it to applications in the real world.

7. REFERENCES

[1] D. Beckett and B. McBride. RDF/XML Syntax Specification (Revised). W3C Recommendation, February 2004.

[2] J. Guy L. Steele. *Common Lisp The Language 2nd edition*. Digital Press, 1990.

[3] P. Hayes and B. McBride. RDF Semantics. W3C Recommendation, Feb. 2004. http://www.w3.org/TR/rdf-mt/.

[4] G. Kiczales, J. des Rivières, and D. G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[5] S. Koide, J. Aasman, and S. Haflich. OWL vs. object oriented programming. In *Workshop on Semantic Web Enabled Software Engineering (SWESE) at the 4th International Semantic Web Conference (ISWC 2005)*, Galway, Ireland, November 2005.

[6] S. Koide and M. Kawamura. SWCLOS: A semantic web processor on common lisp object system. In *3rd International Semantic Web Conference (ISWC2004), Demos*, 2004.

[7] S. Koide and H. Takeda. OWL-Full reasoning from an object oriented perspectiv. In *Asian Semantic Web Conf., ASWC2006*, pages 263–277. Springer, 2006.

[8] F. Manola and E. Miller. RDF Primer. W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-primer/.

[7] http://www.w3.org/2001/sw/BestPractices/

[8] http://www.omg.org/ontology/

[9] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004, February 2004. http://www.w3.org/TR/owl-features/.

[10] S. Misono, S. Koide, N. Shimada, M. Kawamura, and S. Nagano. Distributed collaborative decision support system for rocket launch operation. In *IEEE/ASME International Conference on Advanced Intelligent Mechtronics (AIM2005), WB3-01*, pages 1318–1323, 2005.

[11] B. Motik. On the properties of metamodeling in OWL. In *ISWC 2005*, pages 548–562, Galway, 2005. Springer-Verlag.

[12] A. Paepcke, editor. *Object-Oriented Programming - The CLOS Perspective*. MIT Press, 1993.

[13] J. Z. Pan and I. Horrocks. RDFS(FA) and RDF MT: Two semantics for RDFS. In *ISWC2003*, pages 30–40, Sanibel Island, 2003.

[14] J. Z. Pan, I. Horrocks, and G. Schreiber. OWL FA: A metamodeling extension of OWL DL. In *Workshop OWL: Experiences and Directions*, Galway, 2005.