OWL vs. Object Oriented Programming

Seiji Koide[1], Jans Aasman[2], and Steve Haflich

[1] Galaxy Express Corporation, 18-16, Hamamatsu-cho 1-chome, Minato-ku, Tokyo, JAPAN,
koide@galaxy-express.co.jp,
WWW home page: http://www.galaxy-express.co.jp/
[2] Franz Inc., 555 12th Street, Suite 1450, Oakland, CA 94607, USA.
{ja, smh}@franz.com,
WWW home page: http://www.franz.com/

Abstract. The integration of Object-Oriented Programming (OOP) and Description Logic seems to be so desirable that it produces valid, sound, and reusable software on the firm base of Logic. Whereas we believe the possibility of the integration from our experience up to now on SWCLOS, which is a Semantic Web description language developed on top of Common Lisp Object System (CLOS), we have still an open issue to establish SWCLOS as not only ontology description language but also OOP language with OWL. In this paper, after an introduction of SWCLOS, we demostrate the possibility of the integration of OWL and OOP, and discuss the open question of the OOP with OWL, that is, the method inheritance for OWL objects in OOP. Several ideas of solutions are addressed.

1 Introduction

On one hand, OWL is an ontology description language annotating World Wide Webs, and it is not deemed to be a software programming language. On the other hand, Object Oriented Programming(OOP) is a software programming method dominant today, and it is not captured as the way of ontology construction. However, the ontological representation of objects in OWL is, not only syntactically but also semantically, very similar to the description of object classes and instances in OOP. In practice, programmers tends to explain building ontologies with the analogy of system-analysis in software engineering. It is partly true but not correct as a whole. However, such a view of regarding the system-analysis in software engineering process as the ontology building process invites us to an idea of system development based on Description Logic (DL), in other words, the software development based on the formalized ontological description. Out of the formal description of system specifications we may expect fruitful results of software sharing, validity, reusability, adaptability, and so on.

We have developed an OWL processor, SWCLOS[3], which is developed on top of Common Lisp Object System (CLOS). CLOS allows lisp programmers

[3] It is available from http://pegasus.agent.galaxy-express.co.jp/galexinfo/indexe.htm

to develop Object-Oriented systems, and SWCLOS allows lisp programmers to construct domain and task ontologies in software application fields. Therefore, using SWCLOS, a lisp programmer may unify system development process and ontology building process in application fields. However, we have some open issues to enjoy this happy marriage.

In this paper, at first we introduce CLOS as ontology description language with an introductory example for ontology, and then explain the similarity between CLOS and OWL in SWCLOS with examples in Wine Ontology and others. Then, we discuss the semantic gap between CLOS and OWL and show the solutions. Finally, the question in the integration of OOP and OWL, that is, the method inheritance in OWL objects is discussed, and explore the diverse directions to solve the question.

2 Common Lisp Object System

The Common Lisp Object System is a set of operators for doing Object-Oriented Programming in ANSI Common Lisp, and it has following features.

- Multiple Class Inheritance: methods and slots are inherited from multiple classes.
- Dynamic Programming: CLOS provides the means to redefine class definitions in program run time.
- Meta-Object: a class is the first object in CLOS as an instance, so a class in CLOS is called *metaobject*.
- Meta-Class: a meta-class in CLOS, or a class of classes, allows us to modify the methods of classes including system methods using the Meta-Object Protocol [Kiczales1992].
- Reflecting Programming: the behavior of meta-classes including system methods is changeable using the Meta-Object Protocol. A programmer can modify behaviors of lisp systems. For example, so-called NEW method can be customized adapting for applications by programmers.

2.1 CLOS for Ontology Description

Fig. 1 illustrates the introductory example of RDF graph that was published at the draft of RDF Schema documentation[4]. We can encode this RDF graph using CLOS as follows.

```
(defpackage rdfs
  (:export "Resource")
  (:documentation "http://www.w3.org/2000/01/rdf-schema"))
(defpackage rdf
  (:export "about")
  (:documentation "http://www.w3.org/1999/02/22-rdf-syntax-ns"))
(defpackage eg
```

[4] http://www.w3.org/TR/2002/WD-rdf-schema-20021112/

```
  (:export "Work" "Document" "Agent" "Person" "author" "Proposal" "name")
  (:documentation "http://somewhere-for-eg/eg"))
(defpackage dc
  (:export "title")
  (:documentation "http://dublincore.org/2002/08/13/dces"))

(defclass rdfs:Resource ()
  ((rdf:about :initarg :about :type net.uri:uri)))
(defclass eg:Work (rdfs:Resource) ())
(defclass eg:Agent (rdfs:Resource) ())
(defclass eg:Person (eg:Agent)
  ((eg:name :initarg :name :type string)))
(defclass eg:Document (eg:Work)
  ((eg:author :initarg :author :type eg:Person)
   (dc:title :initarg :title :type string)))
(defparameter eg:Proposal
  (make-instance 'eg:Document
    :author (make-instance 'eg:Person :name "Tim Berners-Lee")
    :title "Information Management: A Proposal"
    :about (net.uri:parse-uri "http:/.../Proposal/")))
```

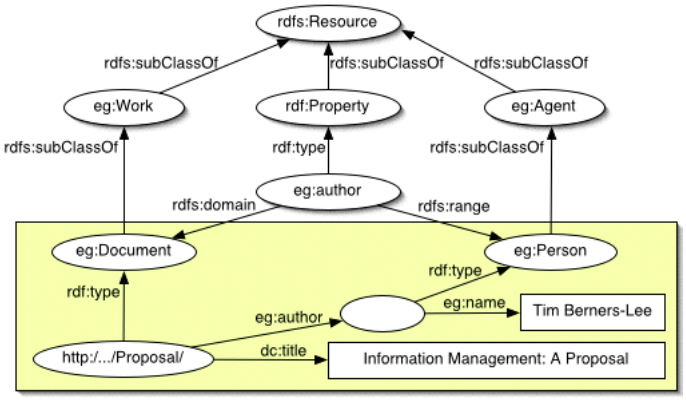

Fig. 1. An Introductory Example of RDF Graph
from http://www.w3.org/TR/2002/WD-rdf-schema-20021112/

Where rdfs:subClassOf is replaced with superclass relation in CLOS and rdf:type is replaced with class-instance relation. Note that we use here the lisp system that is case sensitive, and a namespace of QName is mapped onto a lisp package. The blank node in Fig. 1 is realized as an instance object of class `eg:Person`, which is stored in the author slot of `eg:Proposal` that is an instance of `eg:Document`. The object `eg:Proposal` is bound to the QName symbol `eg:Proposal`, but the blank node, that has no QName or URI, is not bound to any symbol.

After loading the above lisp forms, a lisp programmer can test the class of `eg:Proposal` and the subsumption, and retrieve the fact data from this simple ontology using CLOS APIs.

```
gx(11): eg:Proposal                         => #<eg:Document @ #x20f95aba>
gx(12): (typep eg:Proposal 'eg:Document)    => t
gx(13): (typep eg:Proposal 'eg:Work)        => t
gx(14): (typep eg:Proposal 'rdfs:Resource) => t
gx(15): (slot-value (slot-value eg:Proposal 'eg:author) 'eg:name)
=> "Tim Berners-Lee"
gx(16): (slot-value eg:Proposal 'dc:title)
=> "Information Management: A Proposal"
```

3 SWCLOS: A Semantic Web Processor on CLOS

In the previous section, we demonstrated the possibility of CLOS as ontology description language. In this section, we introduce SWCLOS [Koide2004,Koide2005], a Semantic Web Processor developed on top of CLOS, and demonstrate the similarity between CLOS and OWL in syntax and semantics. The semantic gaps between CLOS and OWL and the solutions in SWCLOS are discussed in the next section.

3.1 RDF/XML Syntax and S-expression

OWL ontology is usually encoded in RDF/XML. Since the XML nested structure is very similar to the S-expression nested structure, it is theoretically easy to transform XML to S-expression. The following shows the syntax transformation from RDF/XML to S-expression on the example from the RDF documentation[5]. Note that an attribute in XML representation is basically treated like a property.

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

(rdf:Description
  (rdf:about "http://www.w3.org/TR/rdf-syntax-grammar")
  (ex:editor
    (rdf:Description
      (ex:homePage
```

[5] http://www.w3.org/TR/rdf-syntax-grammar/#example2

```
        (rdf:Description
          (rdf:about "http://purl.org/net/dajobe/"))))))
```

We have developed an RDF/XML parser that makes the same nested structures as RDF/XML representation with lisp structure `Description` and `property`. The following is an simple example for the RDF/XML parser in SWCLOS. The print functions of lisp structures for `Description` structure and `property` structure are customized so that the lisp printer prints not standard structure expressions but XML forms. In this example, SWCLOS reads an RDF/XML file that describes the ontology shown in Fig.1, and the return value is printed. Note that the returned value is a list that includes three elements, which are an `XMLDecl` structure, a `doctypedecl` structure, and a nested structure composed by `Description` and `property` structures.

```
gx(21): (with-open-file (p "Example.rdf") (parse-rdf p))
(<?xml version="1.0" ?> :doctypedecl...
<rdf:RDF xmlns="http://galaxy-express.co.jp/semantic-web/example#"
         xmlns:eg="http://galaxy-express.co.jp/semantic-web/example#"
         xmlns:dc="http://dublincore.org/documents/2003/06/02/dces#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Property rdf:ID="name">
    <rdfs:domain rdf:resource="#Person" />
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  </rdf:Property>
  <rdf:Property rdf:about="http://dublincore.org/documents/2003/06/02/dces#title">
    <rdfs:domain rdf:resource="#Document" />
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  </rdf:Property>
  <rdf:Property rdf:ID="author">
    <rdfs:domain rdf:resource="#Document" />
    <rdfs:range rdf:resource="#Person" />
  </rdf:Property>
  <rdfs:Class rdf:ID="Person">
    <rdfs:subClassOf rdf:resource="#Agent" />
  </rdfs:Class>
  <rdfs:Class rdf:ID="Document">
    <rdfs:subClassOf rdf:resource="#Work" />
  </rdfs:Class>
  <eg:Document rdf:about="http:/.../Proposal/">
    <eg:author>
      <eg:Person>
        <eg:name>Tim Berners-Lee</eg:name>
      </eg:Person>
    </eg:author>
    <dc:title>Information Management: A Proposal</dc:title>
  </eg:Document>
</rdf:RDF>)
```

SWCLOS internal function `Description-form` and `prop-form` translate the lisp structure `Description` and `property` to the corresponding S-expression, respectively. Thus, SWCLOS function `addRdfXml`, which invokes these functions, converts RDF structures into S-expressions and adds the assertions in S-expression into memory with function `addForm`, which adds general RDF expression in S-expression into lisp memory.

3.2 RDF Graph and OWL Semantics

In RDF, a directed graph with nodes and labeled arcs is a theoretical base model for semantic representation. In SWCLOS, a resource node in RDF graph is represented by a CLOS object, and a labeled arc from a node to another is represented by a slot that belongs to an arrow-tail node and has an arrow-head node as slot value, but rdf:type relation is replaced with instance-class relation and rdfs:subClassOf relation is replaced with class-superclass relation in CLOS. A class in CLOS is also an object (*metaobject*), so it may have arbitrary slots in the same way as an instance object, in addition to a few special slots dedicated to the classes.

All axioms and entailment rules in RDFS[6] are implemented in SWCLOS. The subsumption calculation in CLOS conveys the subsumption among RDFS classes upon the class-instance and the superclass-class relationship. However, there is no super-sub concept of slots in CLOS, whereas RDFS involves it on property. Therefore, the subsumption on RDF property is programmed in addition to the implementation of the domain and range constraint on the property.

The OWL representation is much more likely for objects than RDF graphs. Especially, the property restrictions that provide the local constraints on property values for a specific domain may be straightforwardly implemented by CLOS *slot definitions* that belong to a class. The CLOS native type facet of slot definition is utilized to realize the value type restriction in OWL. On the other hand, in order to implement cardinality restrictions for property value, we have introduced new slot facets, mincardinality and maxcardinality, into the *slot definitions*. For example, after reading Wine Ontology[7], in addition to the following assertional data upon OWL class vin:Zinfandel, CLOS metaobject vin:Zinfandel holds the *effective slot definitions* for instances of vin:Zinfandel in the metaobject structure. The following shows a description of the effective slot definition for vin:hasColor slot of vin:Zinfandel, i.e., the type constraint, maxcardinality and mincardinality constraint for vin:hasColor in vin:Zinfandel are stored in the effective slot definition of vin:hasColor in vin:Zinfandel metaobject. Note that the cardinality constraint for vin:hasColor here is inherited from vin:Wine. When SWCLOS creates new instances of vin:Zinfandel, those constraints stored in the effective slot definition does work as constraints in instance creation.

```
gx(7): (get-form vin:Zinfandel)
```

[6] http://www.w3.org/TR/rdf-mt/
[7] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf

```
(owl:Class vin:Zinfandel (rdf:about "#Zinfandel")
 (rdfs:subClassOf (owl:Restriction (owl:onProperty vin:hasColor)
                   (owl:hasValue vin:Red))
                  (owl:Restriction (owl:onProperty vin:hasSugar)
                   (owl:hasValue vin:Dry))
                  (owl:Restriction (owl:onProperty vin:hasBody)
                   (owl:allValuesFrom
                    (owl:Class (owl:oneOf vin:Full vin:Medium))))
                  (owl:Restriction (owl:onProperty vin:hasFlavor)
                   (owl:allValuesFrom
                    (owl:Class
                     (owl:oneOf vin:Moderate vin:Strong)))))
 (owl:intersectionOf vin:Wine
  (owl:Restriction (owl:onProperty vin:madeFromGrape)
   (owl:hasValue vin:ZinfandelGrape))
  (owl:Restriction (owl:onProperty vin:madeFromGrape)
   (owl:maxCardinality 1))))
gx(8): (describe (find 'vin:hasColor (mop:class-slots vin:Zinfandel)
         :key #'name))
#<OwlProperty-effective-slot-definition vin:hasColor @ #x20bd8032> is an
    instance of #<standard-class OwlProperty-effective-slot-definition>:
 The following slots have :instance allocation:
  name              vin:hasColor
  type              #<owl:Class vin:WineColor>
  documentation     common-lisp:nil
  initform          #<vin:WineColor vin:Red>
  initfunction      #<Closure (:internal constantly 0) @ #x20bac0ca>
  initargs          (vin:hasColor)
  allocation        :instance
  subject           #<owl:Class vin:Zinfandel>
  maxcardinality    1
  mincardinality    1
```

3.3 Satisfiability Check and Proactive Entailment

For RDFS, the domain and range constraint works effectively at the instance definition. Namely, if programmers attempt to define or set a slot value that violates the range constraint for the slot, SWCLOS signals an alarm. When programmers attempt to define an object ambiguously (to define an object to a more abstract class), if the domain definition is available, then SWCLOS defines an object more precisely (defines an object to a more special class).

Such satisfiablility check works in OWL, too. It prevents programmers from importing bugs into ontologies. The followings demonstrate an example of satisfiability checking, in which a programmer defined class BlendedWine and tried to create an instance that has two wine colors, `vin:Red` and `vin:White`, then SWCLOS signaled an un-satisfiability alarm, because the number of wine color was restricted to one at `vin:Wine` definition that is a superclass of BlendedWine. Note that the most strict and precise information is inherited and effective in SWCLOS, according to the principle of the monotonicity.

```
gx(2): (defResource BlendedWine (rdf:type owl:Class)
  (rdfs:subClassOf
    vin:Wine
    (owl:Restriction (owl:onProperty vin:madeFromGrape)
                     (owl:minCardinality 2))
    (owl:Restriction (owl:onProperty vin:hasColor)
                     (owl:minCardinality 2))))
#<owl:Class BlendedWine>
gx(3): (defIndividual MyBlendedWine (rdf:type BlendedWine)
  (vin:hasColor vin:Red vin:White))
Error: Unsatisfiability by cardinality for BlendedWine vin:hasColor
```

Furthermore, the combination of owl:intersectionOf and the property constraints brings not only the entailment on slot values but also the powerful entailment on objects in domain. For example, the following example shows a number of entailments in the adding process and finally SWCLOS entails that `QueenElizabethII` should be a woman, because it is asserted that a person who has gender female is a woman, and it is asserted that `QueenElizabethII` is an instance of `Person` and `hasGender Female`.

```
gx(4): (defIndividual Female (rdf:type Gender)
  (owl:differentFrom Male))
Warning: Special Entailing with domain: Gender is a owl:Class.
Warning: Entail by range: Male rdf:type owl:Thing.
#<Gender Female>
gx(5): (defResource Person (rdf:type owl:Class)
  (owl:intersectionOf
    Human
    (owl:Restriction (owl:onProperty hasGender)
                     (owl:cardinality 1))))
Warning: Entail by range: hasGender rdf:type rdf:Property.
Warning: Simple entailment: Human rdf:type owl:Class.
#<owl:Class Person>
gx(6): (defResource Woman (rdf:type owl:Class)
  (owl:intersectionOf
    Person
    (owl:Restriction (owl:onProperty hasGender)
                     (owl:hasValue Female))))
#<owl:Class Woman>
gx(7): (defIndividual QueenElizabethII (rdf:type Person)
  (hasGender Female))
Warning: Entailed in refining: #<Person QueenElizabethII> to Woman.
#<Woman QueenElizabethII>
```

Such proactive entailment as demonstrated above, namely effective entailing on the fly without queries is very helpful to construct consisitent ontologies and to build application systems.

3.4 Description Logic and Software Development

The validity by satisfiability checking and entailing in OWL is underpinned by DL. Therefore, it is possible that we obtain more valid and bug-less software systems by introducing OWL descriptions for system-analysis in system development process. Sattler, et al. [Sattler2003] discussed DL and other formalisms that include the ER model and the Object-Oriented model, but they did not extend the discussion to OOP. Recently, the fusion of UML and DL has been propelled. Brockmans, et al. [Brockmans2004] utilized UML for visualized modeling language for ontologies. Berardi [Berardi2003] formalized UML class diagrams from DL, and Kaneiwa and Satoh [Kaneiwa2005a] has cleared the validation algorithm for UML using First Order Logic and counting quantifieres. It is clear that the integration of OOP and DL reasoner will be very useful. However, we emphasize that the work of the DL reasoner is just to reply queries on the classification of instances and the subsumption of classes. Precisely, it involves satisfiability check of concepts with the refutation and the Tableau Algorithm. We must set up right queries in the relevant situations. The work of the OWL reasoner is to perform appropriately the entailments in the situation. However, the complete set of OWL entailment rules are not known [Horst2004]. Therefore, the complete proactive entailments, in other words, setting up right queries for right entailments in the situation is still open in OWL. In the next section, we introduce a few examples of OWL entailments and discuss the difference of semantics in CLOS and OWL, which we have solved to realize OWL functions on top of CLOS.

4 Semantic Gaps between CLOS and OWL

4.1 Classification and Subsumption

A class in OWL is a set of some individuals (called an extension), the class-subclass relation in OWL is the inclusiveness of the extensions. Namely, the statement that a class C_2 includes a class C_1 ($C_1 \sqsubseteq C_2$) means that all individuals of class C_1 are concurrently individuals of class C_2. On the other hand, class semantics of CLOS is different from RDF/OWL. A class in CLOS is one whose instances share methods and slot structure definitions. A CLOS class is a prototype to create CLOS instances, and the class is required to make a CLOS instance. However, the class-subclass relation and class-instance relation in CLOS work upon the transitivity and subsumption of class/instance like OWL. In practice, the RDF entailment rule **rdfs9**[8] (subsumption rule) and **rdfs11** (transitivity rule on rdfs:subClassOf) are natively realized in the CLOS class-subclass relation.

In SWCLOS, we have introduced appropriate subsumption among classes for owl:intersectionOf and owl:unionOf assertion. For instance, owl:intersectionOf results the class-subclass entailment as follows.

[8] http://www.w3.org/TR/rdf-mt/#rulerdfs9

owl:intersectionOf. An extension of class A is denoted by $CEXT(A)$. A is an intersection of C_i (where i = 1 ...n), if and only if $CEXT(A) = CEXT(C_1) \sqcap \ldots \sqcap CEXT(C_n)$. Thus, $A \sqsubseteq C_i$ (A is a subclass of C_i) is entailed, since $CEXT(A)$ is included $CEXT(C_i)$ for every C_i.

In practice, SWCLOS adds super-subclass information from owl:intersectionOf and owl:unionOf assertions. Thus, `vin:MariettaOldVinesRed` in Wine Ontology is entailed to be an instance of `vin:Wine` in addition to `vin:TableWine` as follows.

```
gx(11): (get-form vin:MariettaOldVinesRed)
(vin:RedTableWine vin:MariettaOldVinesRed (vin:hasMaker vin:Marietta)
 (vin:hasFlavor vin:Moderate) (vin:hasBody vin:Medium)
 (vin:locatedIn vin:SonomaRegion) (vin:hasColor vin:Red)
 (vin:hasSugar vin:Dry))
gx(12): (typep vin:MariettaOldVinesRed vin:RedTableWine)     => t
gx(13): (typep vin:MariettaOldVinesRed vin:TableWine)        => t
gx(14): (typep vin:MariettaOldVinesRed vin:Wine)             => t
```

In addition to the simple subsumption entailment of owl:intersectionOf demonstrated above and owl:unionOf, we have realized several entailments for the property constraints in SWCLOS. Fig. 2 depicts a part of Wine Ontology, where the assertion that `vin:RedWine` is an intersection of `vin:Wine` and the property constraint that the property `vin:hasColor` must have `vin:Red` on `vin:RedWine`. The interest thing here is the effect of owl:intersectionOf. SWCLOS entails that `vin:RedTableWine` is a subclass of `vin:RedWine` and `vin:MariettaOldVinesRed` is typed to `vin:RedWine`, whereas those relationships are not explicitly asserted.

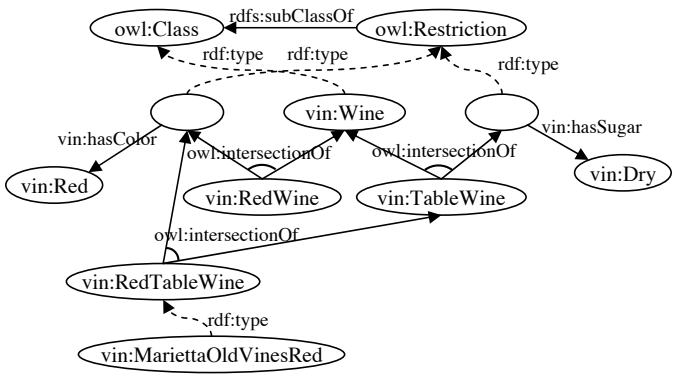

Fig. 2. An Example of Entailment by owl:intersectionOf and Property Constraint

```
gx(14): (subtypep vin:RedTableWine vin:RedWine)          => t
gx(15): (typep vin:MariettaOldVinesRed vin:RedWine)      => t
```

4.2 Multiple Classing

In the semantics of OWL (and RDFS), an instance may belong to multiple classes. In fact, a vintage wine vin:SaucelitoCanyonZinfandel1998 is an instance of vin:Vintage and vin:Zinfandel in Wine Ontology. However, a CLOS class is a prototype to create its instances and eventually instances must belong to a single class. To solve this question, we have introduced the invisible class that may be a subclass of visible multiple classes. For example, vin:SaucelitoCanyonZinfandel1998 is an instance of `vin:Zinfandel.15` that is invisible in OWL and a subclass of vin:Vintage and vin:Zinfandel in CLOS.

4.3 Slot Restriction Inheritance and Default Slot Value

Although there is no word of *inheritance* in the specification documentations of Semantic Webs, the domain and range constraint in RDFS and the local property constraint in OWL are clearly inherited from superclasses to subclasses. In contrast, we have no way to designate the default slot value that is often used in frame systems and some of object systems. The reason is that DL cannot infer anything about the slot default value. Only if we set the property constraint of owl:oneOf with the cardinality one, the slot value of instances is entailed as if it is a default value, although we cannot set a different value to the slot.

5 Reflective Programming or Metamodeling

In RDFS model theory, some discussions on metamodeling happened. As explained in RDF Semantics document[9], the 'membership loops' of rdfs:Class might seem to violate the axiom of foundation of set theory. However, this is the mechanism for the reflective knowledge modelling which originated to McCarthy [McCarthy1968], in which he aimed a common language that describe reasoning procedures and huristics knowledge in order to improve the system behavior with adding assertions afterward. Weyhrauch's First-Order Logic program FOL [Weyhrauch1980] attempted to establish 'reflection principles' between theory and meta-theory. He also mentioned the meta-metatheory. 3-Lisp [Smith1984] was the first implementation of reflective lisp programming language, where `eval` and `apply` in lisp was effectively untilized for the reflection. However, it was not an OOP language. CLOS is a language designed to standardize reflective programming in lisp, namely, describing the language implementation by the language. This paradigm produces great flexibility in programming. We can modify the language specification and implementation with the language.

Pan and Horrocks [Pan2001] argued that RDFS metamodeling architecture was difficult and proposed to separate the architecture into modeling layers (Instance Layer and Ontology Layer) and language layers (Language Layer and Metalanguage Layer) in the same way as UML. However, this idea contradicts the object-oriented reflection. In fact, the architecture of RDFS, that is, "the

[9] http://www.w3.org/TR/rdf-mt/

class rdfs:Resource is a superclass and instance of rdfs:Class at the same time" is quite same as CLOS (the standard-object is a superclass and instance of the standard-class). Actually there is no obstacle, even if every concept is also an object, to add missing objects into the domain when required. The following shows a simple example taken from [Pan2003]. SWCLOS adds required objects through a simple proactive entailment (a referent by owl:intersectionOf should be an instance of owl:Class).

```
gx(2): (defResource EuropeanEmployeeStudent (rdf:type owl:Class)
         (owl:intersectionOf Student Employee European))
Warning: Simple entailment: Student rdf:type owl:Class.
Warning: Simple entailment: Employee rdf:type owl:Class.
Warning: Simple entailment: European rdf:type owl:Class.
#<owl:Class EuropeanEmployeeStudent>
gx(3): (defIndividual John (rdf:type EuropeanEmployeeStudent))
#<EuropeanEmployeeStudent John>
gx(4): (typep John Student)   => t
gx(5): (typep John Employee)  => t
gx(6): (typep John European)  => t
```

6 Methods and Method Inheritance

For example, in case that we need to translate a legacy ontology described by CLOS to OWL/XML, SWCLOS provides the best way to solve it with defining a set of CLOS methods that translate legacy knowledge representation to OWL.

Furthermore, we can directly combine OOP and OWL much more. So far, we have discussed SWCLOS as ontology description language, then have demonstrated that DL entailing in SWCLOS reduces inconsistency in ontology. It suggests that it may be also useful to reduce bugs in software by means of software specifications in formal logic description. Namely, system engineers describe program specifications out of system-analysis with OWL. The system description in OWL is a sort of meta program that defines programs to be instantiated and properly guides programmers to encode the programs. The advantage of SWCLOS is the continuity from the description language to the programming language. We can perform OOP in the envionment that is underpinned by DL.

However, in case that we attemt to perform OOP directly on top of SWCLOS, a delicate problem is revealed upon the method inheritance for OWL object parameters.

6.1 Method Selection in CLOS

A method in CLOS can be selected for a set of arguments when each required argument satisfies its corresponding parameter specializer [DeMichiel1993]. Actually, the *most specific applicable method* in applicable methods for the parameters is selected by consulting the class subsumption for the parameters. However, the method selection in CLOS depends on the *class precedence list*, that is, a totally ordered superclass list from the viewpoint of each class. It implies that this

view may be different among classes, even if two classes have same superclasses as set. It easily misleads people at the method combination.

To prevent the mutual inconsistency among class definitions, CLOS detects the inconsistency at making the class precedence list, and signals an alarm. For the example from [DeMichiel1993], CoopStudent is a subclass of Employee and Student, and ResearchStudent is also a subclass of the same both, but Employee precedes to Student in CoopStudent and inversely in ResearchStudent. Then CLOS signals the inconsistency when a programmer attempts to create an instance of a mixed class of Coop- and Research-Student as a subclass of CoopStudent and ResearchStudent, because CLOS cannot decide the total ordered class precedence list.

Even if CLOS does not detect the total order inconsistency, we have still a delicate question on the method inheritance. In the case that we set a class-subclass relation that is shown in Fig. 3 [Kiczales1992], there are a few possibilities of total ordering from given local orders, depending on the method for computing the class precedence list. CLOS adopts the *topological sort* for making the class precedence list. The digits in the figure mean the local order in the local superclass definitions. By the *topological sorting* for total ordering in the class precedence list, we have different class precedence lists for ColoredNoisyWindow and NoisyColoredWindow as shown in the figure. In such a situation, if

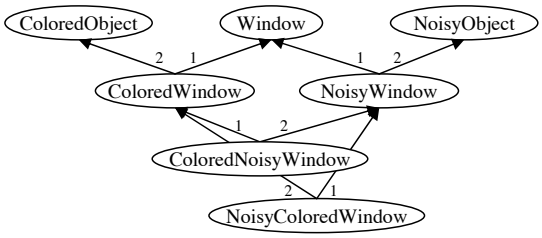

Fig. 3. A Delicate Example on Class Precedence List
ColoredNoisyWindow with superclasses (ColoredWindow NoisyWindow) →
(ColoredWindow NoisyWindow Window NoisyObject ColoredObject)
NoisyColoredWindow with superclasses (NoisyWindow ColoredWindow) →
(NoisyWindow ColoredWindow Window ColoredObject NoisyObject)

you define different methods with an identical name, the selected methods vary through the class precedence list, and those methods could contingently cause unexpected effects on method overloading, or may interfere with each other on before- and after-method execution. The problem is in the ordering between non-subsuming (independent) classes and an identical name, even though there is no ordering among independent classes in OWL and DL. It is possible to think diverse strategies to cope with this problem as follows.

– Single Inheritance: Mizoguchi [Mizoguchi2004] has claimed that the IS-A relation (the substantial sorts) should be single inheritance from the viewpoint

of Ontology Engineering, whereas an object may have multiple roles (the non-substantial sorts), and Kaneiwa and Mizoguchi [Kaneiwa2005b] developed formal ontology on property classification and extended Order-Sorted Logic onto the property classification. This claim suggests us to extend Ontology Engineering from properties and property values to object behaviors, namely what behavior of object should be defined and inherited as substantial method upon IS-A relation, and what behavior should be defined and inherited as interface for a role.
- Inhibition of Identical Name among Independent Classes: If a programmer does not use an identical name among independent classes, we have no problem, even though we may loose the advantage of method overloading upon mix-in objects.
- New Method Selection: In the current CLOS version, primary methods are overloaded by subsumption ordering, and before- and after-methods are invoked in the special-first (before-method) and special-last (after-method) manner. We may set any alternative method selection and combination algorithm for OWL classes using the Meta-Object Protocol. For example, in the case-based memory package, Memory Organization Package (MOP) by Schank et al., the most special slot value is obtained out of all inherited slot values collected from relevant superclasses. For instance, if we make relevant multiple methods being a taxonomy tree so that it is consistent to parameters' taxonomy trees, we can pick up the most specific applicable method in the tree, descending the taxonomy tree with the refining algorithm such as demonstrated at the last of section 3.3.

As those strategies above are orthogonal, we can adopt them multiply and concurrently. Pros and Cons among those strategies, which should be related to the characteristics of applications, will be cleared when we tackle a number of applications in real world in the near future.

7 Conclusion

Reorganizing software development process from the viewpoint of ontological analysis is plausible. Description Logic will provide means to formalize software specifications and definitions, and OWL will be the language for software description as well as UML. SWCLOS is a language for ontology description in OWL, and simultaneously it is an Object-Oriented Programming language on Common Lisp. Lisp programmers may exchange their idea on software systems on the firm base of Description Logic with SWCLOS, and then they can instantiate the formalization and develop working lisp programs on the continuous and reachable ground. In this paper, SWCLOS is briefly introduced, and some examples in OWL for SWCLOS are demonstrated. The method inheritance of OWL objects is issued as an open question, and several strategies to solve the question are discussed. Although the question is still open, it will be solved in the process of software development in real world in the future.

8 Acknowledgment

This paper has been prepared as a part of the Japanese IT project entitled 'Building a Support System for the Large-Scale Operation System using Information Technology' under the contract with the Ministry of Education, Culture, Sports, and Technology (MEXT). We thank Prof. Mizoguchi at Osaka University who is a co-researcher in the project and Emeritus Prof. Ohsuga who is the chairperson of the Technology Evaluation Committee of this project. We also appriciate Dr. Kaneiwa, who discussed the entailments of Wine Ontology.

References

[Berardi2003] Berardi, D., A. Calì, D. Calvanese, and G. De Giacomo: Reasoning on UML Class Diagrams. TR-11-2003, Dipartimento di Informatica e Sistemistica, Università di Roma, La Sapienza (2003), 11–03

[Brockmans2004] Brockmans, S., et al.: Visual modeling of OWL DL ontologies using UML. ISWC2004, (2004)

[DeMichiel1993] DeMichiel, L. G.: An Introduction to CLOS. Object-Oriented Programming The CLOS Perspective (ed. A. Paepcke), Chap. 1 (1993) 3–27

[Horst2004] Horst, H. J. ter: Extending the RDFS Entailment Lemma. ISWC2004, (2004) 79–91

[Kaneiwa2005a] Kaneiwa, K. and K. Satoh: Consistency Checking Algorithms for Restricted UML Class Diagrams. NII Technical Report, NII-2005-013E, National Institute of Informatics, (2005)

[Kaneiwa2005b] Kaneiwa, K. and R. Mizoguchi:Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005), LNCS 3702, Springer–Verlag, (2005) 169–184

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

[Koide2004] Koide, S., Kawamura M.: SWCLOS: A Semantic Web Processor on Common Lisp Object System. ISWC2004 Demos, http://iswc2004.semanticweb.org/demos/32/ (2004)

[Koide2005] Koide, S.: SWCLOS: Semantic Web Processing in CLOS. Int. Lisp Conf. Tutorials (2005)

[McCarthy1968] McCarthy, J.: Programs with Common Sense, Semantic Information Processing (ed. M. Minsky), MIT Press (1968) 404–418

[Mizoguchi2004] Mizoguchi, R.:Tutorial on Ontological Engineering - Part 2: Ontology Development, Tools and Languages, New Generation Computing, OhmSha and Springer, **22**-1, (2004) 61–96

[Pan2001] Pan, J. Z. and I. Horrocks: Metamodeling Architecture of Web Ontology Languages, Proc. 1st Semantic Web Working Symposium, Stanford (2001) 131–149

[Pan2003] Pan, J. Z. and I. Horrocks: RDFS(FA) and RDF MT: Two Semantics for RDFS, The Semantic Web - ISWC2003, Springer (2003) 30–46

[Sattler2003] Sattler, U., D. Calvanese, and R. Molitor: Relationships with Other Formalisms. The Description Logic Handbook (eds. Baader, F., et al.), Chap. 4, Cambridge (2003) 137–177

[Smith1984] Smith, B.: Reflection and Semantics in Lisp, Proc. 1984 ACM Principles of Programming Language Conference, ACM (1984) 23–35

[Weyhrauch1980] Weyhrauch, R. W.: Prolegomena to a Theory of Mechanized Formal Reasoning, Artificial Intelligence, **13**-1–2 (1980) 133–170