OWL-Full Reasoning from an Object Oriented Perspective

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

[1] National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 1-1-0051
koide@grad.nii.ac.jp, takeda@nii.ac.jp
http://www.nii.ac.jp/
[2] Galaxy Express Corporation, 1-18-16, Hamamatsu-cho, Minato-ku, Tokyo 105-0013
koide@galaxy-express.co.jp
http://www.galaxy-express.co.jp/

Abstract. Bridging the gap between OWL and Object-Oriented Programming (OOP) languages is an indispensable condition to enable the Object-Oriented Modeling in Software Engineering by OWL. However it is very difficult in case of static OOP languages like Java and C#. We have developed SWCLOS, which is an OWL processor seamlessly built on top of Common Lisp Object System (CLOS), a dynamic OOP language. SWCLOS allows programmers to develop application domain models by OWL and enables OOP upon the models. In this paper, we explain the semantic gap between OWL and OOP languages, introduce the RDFS and OWL realization at SWCLOS, and discuss the OWL features from OOP perspectives. Finally we demonstrate the OWL-Full level performance in SWCLOS.

1 Introduction

It is natural to combine the domain modeling in Object-Oriented Programming (OOP) with the idea of *object-centered modeling* in ontology development. Recently, the Software Engineering Task Force (SETF) in W3C Semantic Web Best Practices and Deployment Working Group has started to promote synergies between the Semantic Web and the domains associated with Software Engineering[1]. One of the objectives is the Ontology Driven Software Engineering, in which ones expect benefits of unambiguous domain models, consistency checking facilities, validated model sharing, and semi-automatic code generation in software development. The realization of the Ontology Driven Architecture (ODA) by SETF requires to reorganize the object-oriented modeling for Software Engineering on the framework of Semantic Web, in particular, OWL. In order to enable the OOP upon OWL, we should bridge the semantic gap between OOP languages and OWL. However it is difficult in case of static OOP languages like Java and C#. Rather we need dynamic OOP languages.

The problem of OWL-DL is the separation between the class and the individual from the viewpoint of Software Engineering. In reality, the decision whether

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

R. Mizoguchi, Z. Shi, and F. Giunchiglia (Eds.): ASWC 2006, LNCS 4185, pp. 263–277, 2006.

we capture an entity in a model as class or individual depends upon the characteristics of the application domain and the attitudes of human modelers. For example, a wine product such as ElyseZinfandel should be an individual for wine expert systems, but should be a class in logistics for wine wholesalers. Borgida, et al. [Borgida2003] pointed out that one must create a "meta-individual" in order to work around such problems in Description Logic. Still, there are no dominant ideas to compute OWL-Full by means of Description Logics such as Tableau Algorithms.

We developed an OWL processor called SWCLOS[2] [Koide2004, Koide2005] that is built on top of Common Lisp Object System (CLOS), a dynamic OOP language of Lisp. In CLOS, the class is not only an object schema to define instances but also an object per se called *metaobject*. CLOS programmers can encode meta-modeling using the CLOS reflective programming facilities and the Meta-Object Protocol [Kiczales1991]. Therefore, the OWL-Full performance can be obtained by CLOS meta-programming facilities using SWCLOS. In fact, the property owl:sameAs, of which domain is owl:Thing, can be attached to OWL classes in SWCLOS, because OWL classes are individuals of owl:Thing in SWCLOS. Thus, the lisp predicate `owl-same-p` is applicable to not only OWL individuals but also OWL classes.

On the contrary, the loss of Tableau Algorithms from Description Logic inference brings the incompleteness to the subsumption calculation [Nardi2003]. We carefully implemented the *extended structural subsumption algorithm* in SWCLOS. However, the completeness is not obtained yet.

In this paper, at Section 2 we explain the problem of OOP languages with the comparison to OWL/RDF, and the dynamic features of CLOS language that enable RDFS/RDF semantics. At Section 3, we introduce OWL specific features in SWCLOS and the *extended structural subsumption algorithm*. At Section 4, we demonstrate OWL-Full meta-modeling in SWCLOS, then we conclude at Section 5.

2 A Comparison of OWL/RDF and Object-Oriented Programming Languages

The Software Engineering Task Force compared OWL/RDF features to ordinary Object-Oriented Programming Languages (OOPLs) such as Java and C# [SETF2006]. They pointed out serious discrepancies between OOPLs and OWL/RDF as follows. Note that the class in OOPLs is compared to the OWL class, and the instance is compared to the OWL individual. The property and value pair (or role and filler pair) in OWL/RDF is compared with the slot or the member variable of OOPLs.

- Classes in OOPLs are regarded not as sets to which instances belong but as types for instances.
- Each instance in OOPLs belongs one class as its type's instance.

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

- Instances in OOPLs cannot change their type at runtime.
- The list of classes in OOPLs must be fully known at compile-time and cannot change after that.
- There is no reasoner in OOPLs that can be used for classification and consistency checking at runtime or build-time.
- Properties in OOPLs are defined locally to a class and not stand-alone entities.
- Instances in OOPLs cannot have arbitrary values for any property without the definition in its class, and no domain constraint.

However, some of these items are not properly applied to CLOS. We summarized the dynamic features of CLOS in Object-Oriented Programming as follows.

- Multiple Class Inheritance: Methods and slots are inherited from multiple classes.
- Dynamic Programming: CLOS provides the means to redefine class definitions in program runtime.
- Meta-Object: A class is the first-class entity as object in CLOS, so a class in CLOS is called *metaobject*.
- Meta-Class: A meta-class or a class of classes allows ones to modify methods for classes including system intrinsic methods using the Meta-Object Protocol [Kiczales1991].
- Reflective Programming: The behavior of meta-classes including system methods is alterable using the Meta-Object Protocol. A programmer can modify behaviors of lisp systems. For example, so-called NEW method can be customized adapting for the features of applications by programmers.

We have implemented OWL/RDF semantics with CLOS by leveraging such dynamic and reflective language features. In the rest of this section, we explain the implementation of basic OWL/RDF semantics and RDFS/RDF axioms and entailments in SWCLOS through CLOS features. OWL specific semantics and the implementation are explained in the next section.

2.1 The Type in CLOS and the Membership in RDF

A class in OWL/RDF is a set of some individuals (called an *extension*), and 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, the semantics of class-instance in CLOS is different from OWL/RDF. A class in CLOS is a thing of which instances share methods and slot structure definitions. The semantics of CLOS class is built on the frame of slot structures and methods. However, the class-subclass relation and class-instance relation in CLOS work upon the transitivity and subsumption just same as RDFS. In practice, the RDF entailment rule **rdfs9**[3] (subsumption rule) and **rdfs11**[4] (transitivity rule

[3] http://www.w3.org/TR/rdf-mt/#rulerdfs9
[4] http://www.w3.org/TR/rdf-mt/#rulerdfs11

on rdfs:subClassOf) are natively realized in the CLOS class-subclass relation. Therefore, OWL individuals are straightforwardly mapped to CLOS instances and OWL classes are mapped to CLOS classes. Thus, rdfs:subClassOf is replaced with class-subclass relation in CLOS and rdf:type is replaced with class-instance relation.

2.2 Multiple Types by Invisible Classes

In the semantics of OWL/RDF, an instance can belong to multiple classes. For example, a vintage wine vin:SaucelitoCanyonZinfandel1998 in Wine Ontology[5] is an instance of both vin:Vintage and vin:Zinfandel. However, a CLOS class is a prototype to create its instances, then instances must inevitably belong to a single class. To solve this problem, 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.

2.3 Forward Reference by Proactive Entailments

In order to enable forward-referencing, CLOS automatically creates an undefined but referred class as a class under `forward-referenced-class`. However, an attempt to make an instance of a forward referenced class causes an alarm in CLOS. The forward referenced class must be defined by the time of its instance creation. This function is insufficient for RDF forward reference. Fortunately, there are explicitly a number of RDF and RDFS entailment rules, in addition to the monotonicity principle in Semantic Web. Therefore, if we encounter an undefined class reference in reading an OWL file, we can create it as the most abstract concept in the context by applying various RDF and RDFS entailment rules for the context without the contradiction in definitions that will appear later on. For instance, **rdf1**[6] can be utilized for an undefined predicate to be created as an instance of rdf:Property, and **rdfs4**[7] assures for a subject and an object in triple to be defined as a resource object. The definition afterwards may be used to refine forward-referencing definitions precisely. The dynamic OOP features of CLOS such as class-change and reinitialization in runtime enable the implementation upon the forward reference by means of such *proactive entailments*.

2.4 The Realization of RDFS/RDF Axioms and Entailments

We implemented all RDFS/RDF axioms and entailment rules[8] in SWCLOS by exploiting the CLOS potential with Meta-Object Protocol. Most of entailments are realized only by mapping RDFS classes to CLOS classes and RDFS

[5] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf
[6] http://www.w3.org/TR/rdf-mt/#rulerdf1
[7] http://www.w3.org/TR/rdf-mt/#rulerdfs4
[8] http://www.w3.org/TR/rdf-mt/

metaclasses (rdfs:Class and rdfs:Datatype) to CLOS metaclasses. In RDFS, rdfs:Class is an instance of itself. Such membership loop is the source of reflective systems and `cl:standard-class` in CLOS is also an instance of itself. However, CLOS does not allow to include other membership loops except `cl:standard-class`, so we worked around this problem by setting another class of rdfs:Class and making customized `typep` that pretends the membership loop upon rdfs:Class.

```
(cl:typep rdfs:Class rdfs:Class)    -> common-lisp:nil
(cl:subtypep (class-of rdfs:Class) rdfs:Resource) -> t
(typep rdfs:Class rdfs:Class)       -> t
```

Where `t` means boolean true in Lisp, and `typep` is a type testing function that is almost same as Common Lisp native predicate `cl:typep` except on `rdfs:Class`.

3 OWL Reasoning in SWCLOS

3.1 OWL Axioms over RDFS Axioms

In theory, OWL is an extension of RDFS/RDF. Therefore, SWCLOS syntactically and semantically reads the OWL definition file[9] as RDFS/RDF, and keeps RDFS/RDF semantics among OWL vocabularies in RDFS vocabularies. The followings demonstrate the relation that is defined in the OWL definition between rdfs:Class and owl:Class.

```
(typep owl:Class rdfs:Class)    -> t
(subtypep owl:Class rdfs:Class) -> t
```

In the CLOS perspective, owl:Class is also a metaclass as well as rdfs:Class, because it is a subclass of rdfs:Class. The class in CLOS defines slot structures or the role existence in its instances. Thus, the above axioms involve that owl:Class inherits the roles for rdfs:Class instances and rdfs:Resource instances (rdfs:Resource is a superclass of rdfs:Class). Namely, the roles such as rdfs:comment, rdfs:label, and rdfs:subClassOf can be attached to instances of owl:Class.

However, there exist some ambiguities to include OWL vocabularies among RDFS vocabularies. We set several axioms in addition to the defined ones in the OWL definition file. See Table 1 in Appendix. **Axiom1** is a compromise between OWL theory and the reality. In OWL-Full theory, owl:Thing is unified to rdfs:Resource, and then we cannot distinguish them. However, **axiom1** is needed in reality, just same as owl:Class is identified to a subclass of rdfs:Class in the OWL definition file. **Axiom2** is crucial for OWL-Full. The instances of owl:Class inherits the roles of owl:Thing (and rdfs:Resource). Thus, every class in OWL can have role owl:sameAs, owl:differentFrom, etc., as OWL individuals.

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

3.2 Anonymous Restriction Classes for Properties

The OWL object-centered expressions look like objects rather than RDF graphs, while they still obey RDF syntax and semantics. Therefore, the property restrictions in OWL/RDF turn out anonymous classes as instances of owl:Restriction. Then, the subjective CLOS object in the expression is defined as a subclass of the restriction classes that appears within rdfs:subClassOf or owl:intersectionOf representation forms.

In the CLOS perspective, a subclass inherits roles that exist in its superclasses, so it is reasonable that an anonymous restriction class, which provides the information of property value constraint, is placed at the superclass position of the subjective CLOS object. The information of restriction is inherited and shared by all instances of the subclasses. The slot information for instance such as value restriction (owl:allValuesFrom) for CLOS slots are defined in the direct class or its superclasses of an instance, and the information is stored into the CLOS *slot definition objects* that belong to the defined class.

The CLOS native type facet in the slot definition is utilized to realize the value restriction (owl:allValuesFrom) and the existential restriction (owl:someValuesFrom) in OWL. On the other hand, in order to implement cardinality restrictions for property value (owl:maxCardinality, owl:minCardinality, and owl:cardinality), we have introduced new slot facets, `mincardinality` and `maxcardinality`, into the *slot definitions*. If there exist multiple pieces of information upon a property with same restriction but different values among superclasses, they are collected and reduced to the most special one according to the monotonicity principle. For instance, the most special concepts are computed for the value restriction or the existential restriction and the maximum `mincardinality` and minimum `maxcardinality` are calculated for the cardinality restriction. When SWCLOS creates new instances, those constraints stored in the effective slot definition does work as constraints in instance creation. Thus, the satisfiability-checking for slot-value is performed in instance creation.

3.3 Axiomatic Complete Relations

Among many properties in OWL, only `owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`, and `owl:oneOf` make axiomatic assertions. In other words, these properties define the complete equivalency upon the binary relation of concepts. For example, the following asserts the definition of `WhiteBordeaux` from the right-hand side to the left-hand side, and if something is a `Bordeaux` and `WhiteWine`, it is concluded to be a `WhiteBordeaux`.

$$\texttt{WhiteBordeaux} \equiv \texttt{Bordeaux} \sqcap \texttt{WhiteWine}$$

Similarly, the following assertion defines `WineColor`, which has the enumerative membership of `White`, `Rose`, and `Red`, so that the instance of `WineColor` is exactly one of the three, and not to be the others.

$$\texttt{WineColor} \equiv \{\texttt{White Rose Red}\}$$

Therefore, it is not necessary to mind the open world assumption upon such axiomatic complete relation properties. If we find the right-hand side of such equation matches the database, then we may conclude the left-hand side without worry about other statements.

See the following example. SWCLOS concludes that `QueenElizabethII` should be a woman, because it is asserted that a person who has gender female is a woman, and it is also asserted that `QueenElizabethII` is an instance of `Person` and `hasGender Female`. Here note that SWCLOS proactively made the entailment without demand or query from users.

```
(defIndividual Female (rdf:type Gender) (owl:differentFrom Male))
                                                    -> #<Gender Female>
(defResource Person (rdf:type owl:Class)
  (owl:intersectionOf
    Human
    (owl:Restriction (owl:onProperty hasGender)
                     (owl:cardinality 1))))         -> #<owl:Class Person>
(defResource Woman (rdf:type owl:Class)
  (owl:intersectionOf
    Person
    (owl:Restriction (owl:onProperty hasGender)
                     (owl:hasValue Female))))       -> #<owl:Class Woman>
(defIndividual QueenElizabethII (rdf:type Person)
  (hasGender Female))                          -> #<Woman QueenElizabethII>
```

3.4 Substantial Properties and Non-substantial Properties

There are many properties that rule the inclusiveness of concepts, i.e., rdfs:subClassOf, owl:intersectionOf, owl:unionOf, owl:equivalentClass, owl:equivalentProperty, etc. From the viewpoint of DL, they have same strength for subsumption decidability. However, from the viewpoint of Ontology Engineering and Software Engineering, we have to discriminate substantial ones and non-substantial ones for ruling subsumption. Borgida [Borgida2003] argued that ones should deal with individual objects that remain related rather than volatile references. Mizoguchi [Mizoguchi2004] has claimed that the IS-A relation (the substantial sorts) should comply with single inheritance from the viewpoint of Ontology Engineering, whereas an object may have multiple roles (the non-substantial sorts). Kaneiwa and Mizoguchi [Kaneiwa2005] developed the formal ontology on property classification and extended Order-Sorted Logic onto the property classification.

It is also important from the ontology and database maintainability to distinguish persistent relations and temporal relations. In SWCLOS, rdfs:subClassOf relation is mapped onto class-subclass relation, and a CLOS object as rdfs:subClassOf property value is additionally placed in the direct-superclasses-list slot of the class object. However, in case that a property `p1` is a subproperty of or a equivalent property of rdfs:subClassOf, whether should we place the `p1`'s value into the direct-superclasses slot in the class or not? In other words, what property

in OWL should cause the structural variation in the CLOS class-subclass relation, and what property should cause subsumption reasoning without the structural variation? In SWCLOS, we specified that rdfs:subClass, owl:intersectionOf, and owl:unionOf should cause the variation, but owl:equivalentClass, owl:equivalentProperty and other properties, including subproperties and equivalent properties of rdfs:subClass, owl:intersectionOf, or owl:unionOf, should affect the inference but not the structural variation.

Conversely, we should define the substantial and persistent subsumption with rdfs:subClassOf, owl:intersectionOf, and owl:unionOf, and the non-substantial subsumption should be defined through other properties. The substantial subsumption may cause the proactive entailment, but the non-substantial subsumption should not cause any structural variation in the entailment. Thus, such discrimination of substantial and non-substantial subsumption allows us to add and delete relations and keeps it easy to maintain ontologies.

3.5 Extended Structural Subsumption Algorithm

The structural subsumption algorithm is described as follows [Baarder2003] for the $\mathcal{FL}_0$ level, which allows only conjunction $(C \sqcap D)$ and value restriction $(\forall R.C)$.

Let

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \ldots \sqcap \forall R_n.C_n$$

be the normal form of the $\mathcal{FL}_0$-concept description C, and let

$$B_1 \sqcap \ldots \sqcap B_k \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_l.D_l$$

be the normal form of the $\mathcal{FL}_0$-concept description D, then $C \sqsubseteq D$ iff the following two conditions hold:
(1) For all i, $1 \leq i \leq k$, there exists j, $1 \leq j \leq m$ such that $A_j \sqsubseteq B_i$
(2) For all i, $1 \leq i \leq l$, there exists j, $1 \leq j \leq n$ such that $S_i = R_j$ and $C_j \sqsubseteq D_i$

The substantial inclusiveness is computed through the CLOS class-subclass relationship, and the non-substantial inclusiveness is deduced by the extended structural subsumption algorithm. The top concept $\top$ (owl:Thing) substantially subsumes every concept in the CLOS class-subclass relation, but the bottom concept $\bot$ (owl:Nothing) is virtually subsumed by other concepts through this extended structural subsumption algorithm. We extended the above structural subsumption algorithm in the $\mathcal{FL}_0$ level to what includes disjointness(owl:disjointWith), negation(owl:complementOf), equivalency(owl:sameAs, owl:equivalentClass, owl:equivalentProperty), functional and inverse-functional relation-(owl:Functional-Property and owl:InverseFunctionalProperty), full existential-restriction(owl:some-ValuesFrom), filler restriction(owl:hasValue), and number restriction(owl:max-Cardinality, owl:minCardinality, and owl:cardinality) as follows.

1. If C is $\bot$, then $C \sqsubseteq D$ for any D, where $D \in$ owl:Class.
2. If D is $\top$, then $C \sqsubseteq D$ for any C, where $C \in$ owl:Class.
3. If D is $\bot$, then $\neg(C \sqsubseteq D)$ for any C, where $C \in$ owl:Class.
4. $\tilde{C}$ denotes a member of the equivalence group of C. If $\tilde{C} \sqsubseteq \tilde{D}$ (substantially), then $C \sqsubseteq D$ (inferred), where $\{C, D\} \in$ owl:Class.
5. $\not\cong$ denotes complement relation in `complementOf`. If $\tilde{C} \not\cong \tilde{D}$, then $\neg(C \sqsubseteq D)$.
6. Collect all substantially subsuming concepts and restrictions (all of CLOS superclasses) for each $\tilde{C}$ and each $\tilde{D}$, instead of C and D, and do the structural comparison as follows. Hereafter, use the notation such as $A_1 \sqcap \ldots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \ldots \sqcap \forall R_n.C_n$ for $\tilde{C}$ and $B_1 \sqcap \ldots \sqcap B_k \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_l.D_l$ for $\tilde{D}$, i.e. in case of value restrictions.
   $\bar{R}$ denotes a member of equivalent property group of R and $\approx$ denotes the equivalency in `equivalentProperty`. For all i, $1 \leq i \leq k$, if there exists j, $1 \leq j \leq m$ such that $A_j \sqsubseteq B_i$, and for all i, $1 \leq i \leq l$, there exists j, $1 \leq j \leq n$ such that $\bar{S}_i \approx \bar{R}_j$ and the followings hold, then $C \sqsubseteq D$.
   1) in case of value restriction ($\forall \bar{R}_j.C_j$ and $\forall \bar{S}_i.D_i$), $C_j \sqsubseteq D_i$.
   2) in case of full existential restrictions ($\exists \bar{R}_j.C_j$ and $\exists \bar{S}_i.D_i$), $C_j \sqsubseteq D_i$. This is incomplete but almost effective.
   3) in case of ($\forall \bar{R}_j.C_j$ and $\exists \bar{S}_i.D_i$), $C_j \sqsubseteq D_i$. This is incomplete but almost effective.
   4) in case of ($\exists \bar{R}_j.C_j$ and $\forall \bar{S}_i.D_i$), $C_j \sqsubseteq D_i$. This is incomplete but almost effective.
   5) in case of filler restrictions (owl:hasValue, $\bar{R}_j : a_j$ and $\bar{S}_i : b_i$), $a_j \sqsubseteq b_i$. Note that this is the inclusiveness among individuals. See Item 7.
   6) in case of ($\bar{R}_j : a_j$ and $\forall \bar{S}_i.D_i$), $a_j \in D_i$.
   7) in case of ($\bar{R}_j : a_j$ and $\exists \bar{S}_i.D_i$), $a_j \in D_i$. This overestimates the restriction but it is useful in most cases.
   8) in case of cardinality restriction ($\geq n\bar{R}_j$, $\leq nn\bar{R}_j$ and $\geq m\bar{S}_i$, $\leq mm\bar{S}_i$), $n \geq m$, $nn \leq mm$. This is incomplete with the combination of full existential restriction.
7. $\dot{C}$ denotes a member of `sameAs` group of C and $\doteq$ denotes the equivalency in `sameAs` relation. If $\dot{C} \doteq \dot{D}$, or $\dot{C}$ is transitive-lower than $\dot{D}$ on a shared transitive property, then $C \sqsubseteq D$, where $\{C, D\} \in$ owl:Thing. Note that the functional property entailment rule **rdfp1** and the inverse functional property entailment rule **rdf2** in [Horst2005] are used here.

Where Procedure 4 includes the performance of the subsumption test in RDFS semantics and the relation of owl:intersectionOf and owl:unionOf in OWL universe. Procedure 7 treats objects as individuals, including OWL classes. Obviously, this algorithm involves the recursion, but the calculation terminates. It is because CLOS prevents the terminological cycle in subsumption (ex. $C \sqsubseteq D$ and $D \sqsubseteq C$). While the occurrence cycle happens in chase of definitions with the combination of rdfs:subClassOf and owl:unionOf (ex. $B \sqsubseteq C$ and $C \equiv A \sqcup B$), where the chase along definition route causes ascending and descending movements in subsumption relation, the break down of owl:unionOf into the superclasses list

in Procedure 6 prevents to happen such infinite cycle calculation. This extended structural subsumption algorithm is incomplete for the existential restriction, but useful as OWL reasoning for most cases in practice.

3.6 Satisfiability Check

The proactive entailment reduces the load of satisfiability check. For example, when programmers attempt to define an object ambiguously (to define an object to a more abstract class), if the domain and range definition is available, then SWCLOS defines an object more specifically (defines an object to a more special class), with fitting the domain and range restriction. Nevertheless, the satisfiability check is useful to prevent programmers from importing bugs into ontologies. We implemented the domain and range checking, value restriction checking, filler restriction checking, cardinality checking, disjoint-pair checking, etc. Additional unsatisfiability rules to the OWL definition are summarized in Table 3 in Appendix.

3.7 OWL Entailment Rules

We note that the complete set of OWL entailment rules are not known, although ter Horst [Horst2004, Horst2005] has been investigating to make them clear. We emphasize that the prover of Tableau Algorithms is insufficient for the proactive entailment. The work of DL prover is to test the membership of individuals and the subsumption of classes. Precisely, it involves satisfiability check of concepts with the refutation. It implies to make a query for the prover. However, in order to perform proactive entailments, we need to sense the situation in which an entailment is deductive, and we must know what query is effective in the situation. In other words, if we know entailment rules, we can set an appropriate query to the prover in the situation, or we can proactively perform the entailment rules by properly applying the rules in the situation, or we can procedurally encode the entailment rules in software tools.

Hereafter in this subsection, many entailment rules in OWL are introduced and discussed how those rules are implemented in SWCLOS. The entailment rules that is denoted **rdfp**** represent P-entailment rules in [Horst2005]. The others denoted **rule*** are published here. See Table 2 in Appendix.

SameAs Group, EquivalentClass Group, EquivalentProperty Group: `sameAs` relation is reflexive (**rdfp6** in [Horst2005]) and transitive (**rdfp7**). So, all related individuals make one group upon `sameAs`. In SWCLOS, the group information that is a collection of related individuals is registered to each individual of group members. The `equivalentClass` is also reflexive (**rdfp12a**) and transitive (**rdfp12c**). Therefore, the same machinery is adopted for `equivalentClass` as `sameAs`. `equivalentProperty` is also the same. Such information is used in the subsumption calculation as explaned in Subsection 3.5.

DifferentFrom Pairs and DisjointWith Pairs: On the other hand, `differentFrom` is reflexive but not transitive. Therefore, the pairwise relation is not resolved into one group. In SWCLOS, the other member of a pair is registered to each individual. This is same for `disjointWith`.

If a class is disjoint with another class, the subclasses of each disjoint superclass are also disjoint each others. See **rule4** in Table 2, which is implemented in function `owl-disjoint-p`. If disjoint classes are specified as multiple classes in an instance definition, SWCLOS signals an alarm of unsatisfiability. See `unsatisfiability3` and `4` in Table 3.

FunctionalProperty: The entailment rule is described by **rdfp1** in [Horst2005]. SWCLOS maintains the bookkeeping of the inverse of `FunctionalProperty`. Then, `owl-same-p` infers this equality on individuals.

InverseFunctionalProperty: The semantics and the entailment rule is just inversely same as `functionalProperty`. See **rdfp2**. `owl-same-p` infers this equality on individuals.

Intersection of Concepts: If $A \equiv C_1 \sqcap \ldots \sqcap C_n$ (where $i = 1, \ldots, n$), then $A \sqsubseteq C_i$. SWCLOS adds every class C_i into the direct-superclasses list of class A from owl:intersectionOf assertions.

Union of Concepts: If $A \equiv C_1 \sqcup \ldots \sqcup C_n$ (where $i = 1, \ldots, n$), then $C_i \sqsubseteq A$. SWCLOS adds class A into the class-superclasses list of every class C_i from owl:unionOf assertions.

Complement Concepts: The complement relation is reflexive (see **rule5**) and the entails the disjointedness (**rule6**). SWCLOS registers one of the pair to both ones for complementness and disjointedness.

3.8 Calculation Efficiency of SWCLOS

The amount of loading time for Food Ontology and Wine Ontology is 2 seconds, and the amount of loading time for Lehigh University Benchmark (LUBM)[10] is 35 seconds for the data of 3235 persons + 659 courses + 6 departments + 759 university in Allegro Common Lisp 8.0 on MS-Windows 2000 with Pentium 4 (CPU 2.6GHz) and 1GB RAM. There is no stress to reply to the LUBM 14 queries for the above loaded data by lisp codes in ordinal programming manner.

4 OWL-Full and Meta-modeling

In this section, we demonstrate with two examples why the meta-class is needed and how it is used for OWL-Full.

[10] http://swat.cse.lehigh.edu/projects/lubm/

4.1 Meta-class for Role and Filler Attachment

Suppose that wine brands are ID-numbered by *International Wine Society*. Since there are mixed together brand wines such as vin:Zinfandel with non-brand wine concepts such as vin:CaliforniaWine in Wine Ontology, we must distinguish them at first. Even if we introduce two new classes as a subclass of vin:Wine, namely BrandWine of which instances have an ID-number and NonBrandWineConcept that does not provide ID-number, we cannot attach an ID-number to wine classes such as vin:Zinfandel (and can attach an ID-number to wine instances such as vin:ElyseZinfandel). Because a brand wine class should be a subclass of BrandWine but should not be an instance of BrandWine. In order to attach a role and filler to a class, a class of the class is required. The solution in SWCLOS is shown below.

```
(defResource BrandWine (rdf:type owl:Class)
  (rdfs:subClassOf vin:Wine owl:Class)) -> #<owl:Class BrandWine>
(defResource NonBrandWineConcept (rdf:type owl:Class)
  (rdfs:subClassOf vin:Wine owl:Class)) -> #<owl:Class NonBrandWineConcept>
(defProperty hasIDNumber (rdf:type owl:ObjectProperty)
  (rdfs:domain BrandWine)
  (rdfs:range xsd:positiveInteger))     -> #<owl:ObjectProperty hasIDNumber>
(defResource vin:Zinfandel (rdf:type BrandWine)
  (hasIDNumber 12345))                  -> #<BrandWine vin:Zinfandel>
(get-form vin:Zinfandel)
 -> (BrandWine vin:Zinfandel (rdf:about #<uri http://www.w3.org/TR ...
  (rdfs:subClassOf (owl:hasValueRestriction ...
                     ...
  (owl:intersectionOf vin:Wine
    (owl:hasValueRestriction ...
    (owl:cardinalityRestriction ...
  (hasIDNumber 12345))
```

4.2 Treatment of Instance as Class

In the OWL-S specification[11] for Semantic Web Services, the range of property `process:hasPrecondition` is `expr:Condition`, and an instance of `expr:Condition` may have a value of `expr:expressionBody`. Suppose that we have many kinds of conditions and need to classify actual conditions to one of these condition classes. For example, we have many operational modes in the rocket launch operation [Misono2005], and each operational mode selects applicable services through preconditions. Note that an `expr:expressionBody` is different each other by operational modes and it identifies each condition class. Here we need to attach `expr:expressionBody` value to class-like preconditions, in order to record actual conditions in operation and store them as instances of each operational conditions. Please recall that `expr:expressionBody` value may be attached to an instance of but cannot be attached to `expr:Condition` per se.

[11] http://www.daml.org/services/owl-s/1.1/

In SWCLOS, the problem is solved as follows. Here `gxprocess:Precondition` is a metaclass, since it is a subclass of owl:Class. Thus, `PipeCoolDownMode-`, `TankCoolDownMode-`, and `RocketTankingMode-Precondition` turn out classes within the boundary of the schema of OWL-S 1.1.

```
(defResource gxprocess::Precondition (rdf:type owl:Class)
  (rdfs:comment "This is a meta-class for precondition.")
  (rdfs:subClassOf owl:Class expr:Condition))
(defResource gxprocess::OperationModePrecondition
  (rdf:type gxprocess::Precondition)
  (rdfs:label :en "operation mode precondition")
  (rdfs:subClassOf expr:Condition gxdomain::OperationMode)
  (expr:expressionBody ... ))
(defResource gxprocess::PipeCoolDownModePrecondition
  (rdf:type gxprocess::Precondition)
  (rdfs:label :en "pipe cool-down mode precondition")
  (rdfs:subClassOf gxprocess::OperationModePrecondition
                   gxdomain::PipeCoolDownMode)
  (expr:expressionBody ... ))
(defResource gxprocess::TankCoolDownModePrecondition
  (rdf:type gxprocess::Precondition)
  (rdfs:label :en "tank cool-down mode precondition")
  (rdfs:subClassOf gxprocess::OperationModePrecondition
                   gxdomain::TankCoolDownMode)
  (expr:expressionBody ... ))
(defResource gxprocess::RocketTankingModePrecondition
  (rdf:type gxprocess::Precondition)
  (rdfs:label :en "rocket tanking mode precondition")
  (rdfs:subClassOf gxprocess::OperationModePrecondition
                   gxdomain::RocketTankingMode)
  (expr:expressionBody ... ))
```

5 Conclusion

Description Logics provide the means to formalize the application domain, and OWL becomes a modeling language for domain modeling in Software Engineering. SWCLOS is a language for ontology description in OWL, and simultaneously it is an Object-Oriented Programming language on Common Lisp. Therefore, programmers may exchange their idea on software systems on the firm base of Description Logic, and then they can instantiate the formalization and develop working lisp programs on the continuous ground of CLOS. In this paper, we introduced SWCLOS and explained OWL reasoning in SWCLOS. Strictly, the structural subsumption algorithm extended to OWL is still incomplete for the existential restriction, but the system works effectively in most cases of pratical use. SWCLOS provides OWL-Full performance with meta-modeling in CLOS, and we demonstrated some examples in OWL-Full programming. The incompleteness in OWL reasoning caused by the existential restriction will be solved in the future, by introducing First-Order Logic.

Acknowledgment

The most of the work was done 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 was a co-researcher in the project and Emeritus Prof. Ohsuga who was the chairperson of the Technology Evaluation Committee of the project. We also appreciate Dr. Kaneiwa, who discussed the entailments of Wine Ontology.

References

[Baarder2003] Baarder, F., W. Nutt: Basic Description Logics, The Description Logic Handbook (eds. Baader et al.). Chap. 2, Cambridge (2003) 43–95

[Borgida2003] Borgida, A., R. J. Brachman: Conceptual Modeling with Description Logics, The Description Logic Handbook (eds. Baader et al.). Chap. 10, Cambridge (2003) 349–372

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

[Horst2005] Horst, H. J. ter: Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. ISWC2005, (2005) 668–684

[Kaneiwa2005] 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

[Kiczales1991] 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., J. Aasman, S. Haflich: OWL vs. Object Oriented Programming, the 4th International Semantic Web Conference (ISWC 2005), Workshop on Semantic Web Enabled Software Engineering (SWESE), (2005) http://www.mel.nist.gov/msid/conferences/SWESE/repository/8owl-vs-OOP.pdf

[Misono2005] Misono, S., S. Koide, N. Shimada, M. Kawamura, and S. Nagano: Distributed Collaborative Decision Support System for Rocket Launch Operation, IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics (AIM2005), (2005)

[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

[Nardi2003] Nardi, D., R. J. Brackman: An Introduction to Description Logics, The Description Logic Handbook (eds. Baader et al.). Chap. 1, Cambridge (2003) 1–40

[SETF2006] A Semantic Web Primer for Object-Oriented Software Developers, http://www.w3.org/TR/2006/NOTE-sw-oosd-primer-20060309/, W3C (2006)

A OWL Axioms and Entailment Rules

Table 1. Additional OWL Axioms for SWCLOS

axiom1	`Thing subClassOf Resource`
axiom2	`Class subClassOf Thing`
axiom3	`FunctionalProperty type Class`
axiom4	`InverseFunctionalProperty type Class`
axiom5	`FunctionalProperty disjointWith InverseFunctionalProperty`

Table 2. Entailment Rules in OWL for SWCLOS

	If	Then
rule0	r `type Restriction`	r `subtype Resource`
rule1a	$v\ p\ w$	v `subtype Thing`
rule1b	$v\ p\ w$	w `subtype Thing`
rule2a	u `intersectionOf` $\{v_j \ldots\}$	v_j `type Class`
rule2b	u `unionOf` $\{v_j \ldots\}$	v_j `type Class`
rule3	x `distinctMembers` $\{x_j \ldots\}$	x_j `type Thing`
rule4	u `disjointWith` v u' `subClassOf` u v' `subClassOf` v	u' `disjointWith` v'
rule5	u `complementOf` v	v `complementOf` u
rule6	u `complementOf` v	v `disjointWith` u
rule7	u `oneOf` $\{x_j \ldots\}$	x_j `type` u
rule8	v `allValuesFrom` w v `onProperty` p p `range` u	w `subtype` u

Table 3. Unsatisfiability in OWL for SWCLOS

	Unsatisfiable Conditions
unsatisfiability1	u `oneOf` $\{\ x_i \ldots\ \}$ y `type` u y `differentFrom` x_i
unsatisfiability2	x `differentFrom` y x `sameAs` y
unsatisfiability3	u `disjointWith` v v `equivalentOf` u
unsatisfiability4	u `disjointWith` v x `type` u x `type` v