リスプによる OWL-Full プロセッサ

OWL-Full Processor on Lisp

小出 誠二　　武田英明
Seiji Koide　　Hideaki Takeda

国立情報学研究所 総研大
National Institute of Informatics, Sokendai

This paper describes an RDF(S) and OWL-Full modeling language called SWCLOS, which is built on top of Common Lisp Object System (CLOS). The axioms and entailment rules of RDF(S) are implemented and the RDF(S) semantics is realized using CLOS Meta-Object Protocol. The OWL-Full style compatibility is adopted in joining the OWL universe to the RDF universe, then the structural subsumption algorithm in FL_0 is extended with many OWL entailment rules so as to include the intersection, union, disjointness, negation, and equivalency of concepts, the equivalent property, the functional and inverse-functional property, the filler restriction, the value restriction, the existential quantification, and number restriction. We discuss metamodeling problems related to RDF(S) and OWL-Full, and introduce new criteria for metamodeling. The reflection in RDF(S) is also discussed from the Object-Oriented perspective. Finally, we demonstrate some examples of the OWL-Full metamodeling in SWCLOS.

1. はじめに

オブジェクト指向プログラミングにおけるクラス設計は，オントロジー開発におけるオブジェクト指向モデリング[Borgida03]に重ね合わせることができる。OWL をオブジェクト指向プログラミングに統合することで，OWL や記述論理のベースの上に，正当で誤りの無いソフトウェアプログラムを得ることが可能になる．W3C の Software Engineering Task Force(SETF)は，セマンティックウェブ技術をソフトウェア技術関連領域に応用する活動を開始した[Knublauch06]．彼らの目的とするオントロジー駆動型ソフトウェア開発により，曖昧さの無いドメインモデル，モデルの整合性チェック機構，正しいドメインモデルの共有，半自動のコード生成などの効果を期待することができる．SETF の提唱するソフトウェア開発におけるオントロジー駆動アーキテクチュア（ODA）を可能にするためには，オブジェクト指向プログラミングにおけるクラス設計をセマンティックウェブとりわけ OWL の枠組みに組み替えることが必要である．そのためには，オブジェクト指向言語と OWL との間の意味論が同一であることが望ましいが，現在そのような言語はない．一般に用いられている Java や C#は静的なオブジェクト指向言語であり，それらの言語の意味論を OWL に合わせて変更することは難しい．我々は，動的かつリフレクティブなオブジェクト指向言語 Common Lisp Object System(CLOS)を用いて，OWL を実装し，OWL 意味論によるオブジェクト指向プログラミングを可能にした．

W3C の OWL ガイド[Smith04]は，OWL-DL あるいは OWL-Full の選択はユーザが RDFS におけるメタモデリング機能をどれほど必要とするかに依存すると述べているが，ソフトウェア技術の立場から言えば，あるモデルにおいてエンティティをクラスとするかインスタンスとするかは，アプリケーションの性質と技術者の態度如何にかかっている．たとえば，ワインのエキスパートシステムでは ElyseZinfandel をインスタンスとしてよいが，ワインの物流ソフトウェアにおいては，それはクラスでなければならない．したがって，一般のオントロジーモデリング言語においてメタモデリング機能は必須である．

連絡先：小出誠二，国立情報学研究所，101-8430 東京都千代田区一ツ橋 2-1-2，koide@grad.nii.ac.jp

オントロジー技術の立場から見て，OWL-DL においてクラスと個体がはっきりと分離されていることは問題である．Borgida ら[Borgida03]は述語論理におけるこの種の問題を解決するために“meta-individual”なるものを導入した．しかし，タブロー法のような典型的な述語論理に基づく OWL 推論において，未だに OWL-Full 推論を可能にするアイデアは生まれていない．CLOS はもともとメタモデリング機能を有するオブジェクト指向言語であり，CLOS のメタモデリング機能を用いて OWL-Full の性能を実現できる可能性がある．

我々は動的かつリフレクティブなオブジェクト指向言語 CLOS を用いて，RDF(S)と OWL-Full の意味論を実現し，OWL-Full 処理可能なモデリング言語 SWCLOS を開発した．CLOS ではクラスはインスタンスのためのスキーマのみならず，それ自身 *metaobject* と呼ばれるオブジェクトである．CLOS プログラマーは SWCLOS を用いることで，OWL-Full のメタモデリングを実行できる．実際，SWCLOS ではすべての OWL クラスを owl:Thing のサブクラスとすることにより，OWL-DL では個体にのみ適応される owl:sameAs プロパティをクラスにも適応可能になっている．SWCLOS ではクラスを個体と同様に扱うことができる．

一方，述語論理におけるタブロー法に基づかないことにより，包摂推論計算において推論の完全性が満足されない[Nardi03]．OWL 推論の実現にあたり，我々は FL_0 レベルの構造的包摂推論アルゴリズム[Baarder03]を注意深く OWL 仕様レベル（概念の合成･合併や関数プロパティ･逆関数プロパティ，全称限量子･存在限量子などを含む）にまで拡張したが，特に owl:someValuesFrom においてまだ不完全となっている．

本論文では，第 2 章において静的オブジェクト指向言語と OWL との意味論的ミスマッチを指摘して CLOS の動的言語の特徴を述べ，第 3 章において RDF(S)と CLOS の同一性と相違を述べて，CLOS における RDF(S)の実現について記述する．第 4 章では RDF ユニバースと OWL ユニバースの結合方法について議論し，我々の採用した OWL ユニバース実現方法について説明し，OWL 推論の実現について述べる．第 5 章において実装とベンチマーク結果について報告する．第 6 章にて SWCLOS におけるメタモデリングのための新しい基準を紹介し，第 7 章にてその基準を用いたメタモデリングの実例を紹介する．最後に，まとめを述べる．

2. OWL とオブジェクト指向言語の意味論の乖離

W3C の SETF は OWL/RDF の特徴と Java や C#のようなオブジェクト指向言語(OOPL)との特徴を比較して，意味論的な違いを次のように指摘した．ここで OOPL のクラスは OWL のクラスに，OOPL のインスタンスは OWL の個体に対応し，OOPL のメンバー変数あるいはスロット値は，OWL のプロパティとその値に対応している．

- OOPL におけるクラスはインスタンスが分類される集合ではなく，ある種のインスタンスに共通する型である．
- OOPL におけるインスタンスは一つのクラスのみに型付けされるが，OWL の個体は複数のクラスに分類されてよい．
- OOPL におけるインスタンスは実行中にその型を変更できないが，OWL の個体は宣言と伴意にしたがってより詳細なクラスに変更される．
- OOPL におけるクラスリストはコンパイル時にすべて確定されていなければならないが，OWL では順次追加定義されてよい．
- OOPL ではコンパイル時あるいはビルド時にクラス分類したり，整合性をチェックしたりする機能がない．
- OOPL におけるスロットは局所的にクラスに定義され，単独には存在しないが，RDF のプロパティは大域的であり，プロパティが定義するドメインのクラスとは独立に存在する．
- OOPL におけるインスタンスはそのクラスに定義されたスロットにのみ値を持つことができるが，OWL の個体はクラス定義の如何にかかわらず，どんなプロパティについても任意の値を持つことができる．

しかしながら，これらの指摘の一部は CLOS のような動的な OOPL にはあてはまらない．以下に CLOS における動的な特徴を示す．

- 多重継承：メソッドとスロット定義は複数のクラスから継承される．
- 動的プログラミング：CLOS では実行中にクラスを変更できる．
- メタオブジェクト：CLOS におけるクラスは第1級のエンティティであり，オブジェクトとして存在する．
- メタクラス：メタクラスすなわちクラスのクラスによりメタオブジェクト・プロトコル(MOP)を用いて，システム組み込みのメソッドも含めて変更できる．
- リフレクティブ・プログラミング：MOP を用いて，メタクラスの挙動を変更できる．プログラマーは CLOS システムの挙動と仕様を変更できる．

しかしそれでもまだ CLOS と RDF(S)／OWL の間には，意味論上の深刻な相違が存在している．我々は上記のような CLOS の柔軟な機能を用いて，CLOS 上に RDF(S)と OWL の意味論を実現した．次章では，CLOS の特徴と RDF(S)意味論について述べ，SWCLOS における RDF(S)の意味論の実現について報告する．OWL の意味論とその実現については，第4章以降に記述する．

3. RDF(S)と Common Lisp Object System

3.1 CLOS における型と RDF メンバーシップ

CLOS におけるクラス・インスタンス関係は RDF(S)のそれとは意味論的に異なる．CLOS におけるクラスはインスタンスの型であり，インスタンスはクラスに定義されたスロット定義とメソッドを共有する．CLOS の意味論はオブジェクトのスロット定義とメソッド継承の枠組みによって支えられている．一方，RDF(S)におけるクラスはそこにインスタンスが分類される集合概念であり，その集合はクラス外延と呼ばれる．RDF(S)におけるクラス・サブクラス概念は，クラス外延の包含関係である．すなわち，クラス C がクラス D のサブクラスであるとは，C のクラス外延 $CEXT'(C')$ が D のクラス外延 $CEXT'(D')$ に集合として含まれるという意味である ($CEXT'(C') \subseteq CEXT'(D')$)．ここでクラス C のすべてのインスタンスは同時にクラス D のインスタンスでもある．このように，CLOS のクラス概念と RDF(S)のクラス概念は意味的に異なるが，CLOS のクラス・サブクラス推移律とクラス・インスタンス包摂律は RDF(S)のそれと全く同一である．実際に，RDF 伴意ルール **rdfs9**（クラス推移律）[1]および **rdfs11**（包摂律）[2]は CLOS においてネイティブに実現されている．それゆえ，我々は RDFS のインスタンスを CLOS インスタンスに，RDFS のクラスを CLOS のクラスにマッピングした．以下に CLOS そのもので実行したクラス推移律と包摂律を示す．**rdfs9** と **rdfs11** が実現されていることを確認されたい．

```
(defclass xxx () ())     -> #<standard-class xxx>
(defclass vvv (xxx) ()) -> #<standard-class vvv>
(defclass uuu (vvv) ()) -> #<standard-class uuu>
(cl:subtypep 'uuu 'xxx) -> t

(defclass xxx () ())     -> #<standard-class xxx>
(defclass uuu (xxx) ()) -> #<standard-class uuu>
(setq vvv (make-instance 'uuu))
                         -> #<uuu @ #x2126cc9a>
(cl:typep vvv 'uuu)      -> t
(cl:typep vvv 'xxx)      -> t
```

3.2 RDF(S)における多重クラス

RDF(S)におけるクラス概念はインスタンス集合であるため，インスタンスは同時に互いに独立な複数のクラスに所属することができる．例えば，vin:SaucelitoCanyonZinfandel1998 はワインオントロジー[3]において vin:Vintage クラスと vin:Zinfandel クラスのインスタンスである．しかし，CLOS においてクラス概念はインスタンス生成のための母型となるものであり，必然的に CLOS においてはあるインスタンスのクラスは単一である．この問題を処理するために，我々は CLOS 上では存在するが RDF(S)の観点では見えないクラス，invisible class を導入した．システムは複数のクラス所属が必要なとき，それら複数のクラスをスーパクラスとする invisible class を自動的に生成し，CLOS 上はその invisible class の元にインスタンスを生成する．例えば，vin:SaucelitoCanyonZinfandel1998 の直接のクラス vin:Zinfandel.15 は RDF(S) 上は invisible であり，vin:Zinfandel.15 はスーパクラスとして vin:Vintage クラスと vin:Zinfandel クラスを持つ．

3.3 スロット定義なしのスロット追加

CLOS において実行中にクラス変更は可能であるが，それでもインスタンスに必要なすべてのスロットはインスタンス生成前にクラスに定義されていなければならない．一方，RDF におけるプロパティとその値は，RDF グラフに追加されるものであり，なんら言語依存なしに，いつでもどこでも自由に追加できる．グラフベースの実装ではなく，ここで採用したように RDF(S)クラス／インスタンスを素直に CLOS クラス／インスタンスに写像した場合，この問題は深刻であったが，我々はこれを解決し，必要に応じて個々のスロット定義をクラスに追加することで，インスタンスにおいてスロット追加を自由に行うことを可能にした．

[1] http://www.w3.org/TR/rdf-mt/#rulerdfs9
[2] http://www.w3.org/TR/rdf-mt/#rulerdfs11
[3] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf

3.4 プロパティ・インスタンスとそのエクステンション

サブジェクト, プレディケイト, オブジェクトの三つ組み〈*s, p, o*〉において, あるプロパティ *p* に関するすべての三つ組みの集合を, そのプロパティ *p* のエクステンションと呼ぶ($EXT'(p')$). SWCLOS では〈*s, p, o*〉の三つ組みは, オブジェクト *s* のスロット〈*p, o*〉に相当する. ここで *p* はスロット名, *o* はスロット値である. 一般に, インスタンスのクラスにはそのインスタンスに対するスロット定義があるが, このスロット定義はプロパティ *p* のエクステンションのうち, 主題となるオブジェクトのクラスごとに分割されたプロパティ *p* のエクステンションの部分集合を代表するものと捉えることができる. この特徴を利用して, 我々はプロパティ *p* からそのエクステンションのすべての要素を集めてくるような仕組みを実装した.

3.5 プロパティ包摂概念と定義域／値域

RDF(S) クラス／インスタンスを CLOS クラス／インスタンスに写像した結果, rdfs:label や rdfs:comment のようなプロパティは CLOS において rdf:Property のインスタンスとなる. RDF(S) には rdfs:subPropertyOf によるプロパティの包摂概念があるが, CLOS ではその他のオブジェクト指向言語と同様に, インスタンスにはクラスのような包摂概念がない. そこで新たにプロパティにクラスとは独立に包摂概念を実装し, プロパティの定義域／値域の属性継承を実現した. 以下に定義域に関する RDFS 伴意ルール, **rdfs5**[1], **rdfs7**, **rdfs2** の SWCLOS 実行例を示す.

```
(defProperty aaa (rdfs:subPropertyOf bbb))
                          -> #<rdf:Property aaa>
(defProperty bbb (rdfs:domain xxx))
                          -> #<rdf:Property bbb>
(defIndividual uuu (aaa yyy)) -> #<xxx uuu>
(typep uuu xxx)               -> t
```

プロパティの値域情報は, プロパティ定義時あるいはクラス定義時に定義域クラスにおけるスロット定義オブジェクトに移され, そのクラスのインスタンス生成やスロット生成の際に, スロット値の制約として用いられる. 以下は値域に関する RDFS 伴意ルール, **rdfs5**, **rdfs7**, **rdfs3** による実行例である.

```
(defProperty aaa (rdfs:subPropertyOf bbb))
                           -> #<rdf:Property aaa>
(defProperty bbb (rdfs:range zzz))
                           -> #<rdf:Property bbb>
(defIndividual uuu (aaa yyy)) -> #<xxx uuu>
(typep yyy zzz)               -> t
```

3.6 RDFS のメタクラスと rdfs:Class のメンバーシップ巡回

RDF(S) クラス／インスタンスから CLOS クラス／インスタンスへの写像の結果, rdfs:Class と rdfs:Datatype はメタクラスすなわちクラスのクラスとなる. CLOS はメタクラスの機能を有しているため, これ自体は何の問題も無い. CLOS のメタプログラミングの作法では, クラスをメタクラスとするためには, そのクラスを `cl:standard-class` のサブクラスとする. すなわち rdfs:Class を `cl:standard-class` のサブクラスとすることで, rdfs:Class と rdfs:Datatype は CLOS 上でもメタクラスとなる〈rdfs:Datatype は rdfs:Class のサブクラスだから, これもメタクラスになる〉.

RDFS において, rdfs:Class はクラス・インスタンス階層において最上位に位置するクラスであり, しかも rdfs:Class のクラスは rdfs:Class というメンバーシップ巡回の性質を有する. このようなメンバーシップ巡回は, 第6章で述べるように, リフレクティブなシステムを実現する鍵となる重要な性質であり, RDFS も明らかにリフレクティブな構造を有している. しかもリフレクティブなオブジェクトシステムである CLOS 自身, `cl:standard-class` に rdfs:Class と同様なメンバーシップ巡回を有しており, CLOS では `cl:standard-class` 以外にシステム中にメンバーシップ巡回を有することを許していない. そこで実装上は rdfs:Class のクラスを別途設け, rdfs:Class のメンバーシップ巡回を見かけ上成立させるために, CLOS ネイティブなタイプ判定述語 `cl:typep` の代わりに RDFS 用のタイプ判定述語 typep を用いることとした.

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

3.7 前方参照とプロアクティブな伴意

オントロジー記述における前方参照を可能にするために, 宣言において参照される未定義なエンティティを, 伴意ルールを用いて自動的にオブジェクトとして定義する機能を開発した. 元来 CLOS には未定義で参照されたクラスを一時的に `forward-referenced-class` として定義する機能があるが, 実際に未定義クラスのインスタンスを生成する時点までには `forward-referenced-class` は定義済みにならなければならず, 我々の目的には不十分である. RDF(S)においては二つの RDF 伴意ルールと 13 個の RDFS 伴意ルールがあり, 幸いなことに知識の単調増加性というセマンティックウェブにおける原則がある. そこで, これらの伴意ルールと知識の単調増加性を利用して, 未定義クラスやオブジェクトをその場で可容な範囲で最も特殊な概念で定義し, 後からのより詳細な定義において, 正しく再定義する. 例えば, 伴意ルール **rdf1** を使えば, 三つ組みのプレディケイトに現れたエンティティ〈スロット名〉を rdf:Property のインスタンスとして定義できるし, 伴意ルール **rdf4** を用いれば, 三つ組みのサブジェクトとオブジェクトに現れたエンティティ〈主題となるオブジェクトとスロット値として引用されるオブジェクト〉を, rdfs:Resource のインスタンスとして定義できる. 3.5 節に示したようにプロパティの定義域と値域の制約も利用できることは言うまでもない. 実行時におけるクラス変更や再イニシャライズなどの CLOS の動的なオブジェクト指向言語の特徴が, そのような前方参照とプロアクティブな伴意を可能にした. ここでプロアクティブな伴意とは, ユーザからの要求によらずに, その時の文脈において適応可能な伴意ルールを積極的に適応し, 伴意ルールを実行することを言う. 以下の実行例を参照されたい. ここでは未定義な三つのエンティティを三つ組みと宣言するだけで, **rdf1**, **rdfs4a**, **rdfs4b** を適応して, 一つのプロパティと二つのリソースが定義されている.

```
(defIndividual uuu (aaa yyy))
                -> #<rdfs:Resource uuu>
aaa              -> #<rdf:Property aaa>
uuu              -> #<rdfs:Resource uuu>
yyy              -> #<rdfs:Resource yyy>
```

4. RDF(S)上の OWL

4.1 OWL の RDF 適合性

W3C では OWL ユニバースと RDF ユニバースの適合性について二つのスタイルがあると述べられている[Schneider04]. OWL-Full スタイルでは, OWL の三つの部分〈個体 OT', クラス OC', プロパティ OP'〉は RDF ユニバースのそれぞれ相当部分

[1] http://www.w3.org/TR/rdf-mt/#rulerdfs5**, 以下同様**

〈rdfs:Resource のクラス外延 R', rdfs:Class のクラス外延 C', rdf:Property のクラス外延 P'〉と同一とされる. 一方, OWL-DL スタイルでは, OWL ユニバースにおける三つの部分はそれぞれ RDF ユニバースの相当部分とは別物であり, さらにそれら三つも互いに disjoint とされている. 我々は OWL の実現にあたって, 第三のスタイル, すなわち OWL ユニバースは RDF ユニバースの一部であり, 意味的機能的には RDF(S)の拡張であるというスタイルを採用した. すなわち, OWL 個体は rdfs:Resource クラス外延の一部, OWL クラスは rdfs:Class クラス外延の一部, OWL プロパティは rdf:Property クラス外延の一部である.

$$(OT' \subseteq R') \wedge (OC' \subseteq C') \wedge (OP' \subseteq P')$$

V を RDF と RDFS におけるボキャブラリの URI およびリテラルの集合とし, D はデータタイプへの写像とする. RDF(S)における解釈 I はタプル〈R', P', EXT', S', L', LV'〉で表現され, ここで R' は V における URI およびリテラルの指示を含む空でない集合, P' はプロパティを表す R' の部分集合であり, EXT' は P' からべき集合 $R' \times R'$ への写像, S' は V における URI 参照から R' におけるその指示への写像であり, L' は V におけるリテラルから R' におけるその指示への写像であり, リテラル値 LV' はユニコード文字列の集合と, ユニコード文字列と言語タグのペアの集合と, D の各データタイプについての値空間を含む集合である.

さて, RDF ユニバースでは, 以下のようになっている.

$$R' = EXT'(rdfs:Resource') \quad (1)$$
$$C' = EXT'(rdfs:Class') \quad (2)$$
$$P' = EXT'(rdf:Property') \quad (3)$$
$$LV' = EXT'(rdfs:Literal') \quad (4)$$

ただし, RDF ユニバース中の任意の x と y の2項関係〈x, y〉について, 〈x, y〉$\in EXT'(rdf:type')$ のとき, そのときに限って x は y のクラス外延の要素であり $x \in CEXT'(y)$, また $rdfs:Resource'$ は rdfs:Resource の URI の R' におけるその指示への写像 $rdfs:Resource' = S'$(rdfs:Resource), 以下同様に $rdfs:Class' = S'$ (rdfs:Class), $rdf:Property' = S'$ (rdf:Property), $rdfs:Literal' = S'$ (rdfs:Literal) である.

RDF ユニバース中に OWL ユニバースを構築するために, RDF/S におけるボキャブラリ V を OWL のボキャブラリを含むように拡張し, RDF ユニバースが OWL ユニバースを含むように以下のようにする.

- owl:Class の URI の指示は RDF におけるクラスに所属する.
  $owl:Class' \in CEXT'(rdfs:Class')$ (5)
- owl:Class の URI の指示のクラス外延は RDF のクラスに包含される.
  $CEXT'(owl:Class') \subseteq CEXT'(rdfs:Class')$ (6)
- owl:Restriction の URI の指示は RDF におけるクラスに所属する.
  $owl:Restriction' \in CEXT'(rdfs:Class')$ (7)
- owl:Restriction の URI の指示のクラス外延は owl:Class の URI の指示のクラス外延に包含される.
  $CEXT'(owl:Restriction') \subseteq CEXT'(owl:Class')$ (8)
- owl:Thing の URI の指示は owl:Class の URI の指示のクラス外延に所属する.
  $owl:Thing' \in CEXT'(owl:Class')$ (9)
- owl:Thing の URI の指示のクラス外延は RDF ユニバースに包含される.
  $CEXT'(owl:Thing') \subseteq CEXT'(rdfs:Resource')$ (10)

上記(5)から(9)は OWL 定義ファイル[1]に記述してある内容である. 我々の OWL/RDF 適合性のスタイルでは, OWL ユニバースは RDF ユニバースと別物ではなく, OWL ユニバースにおいても RDF(S)の統語論と意味論が成立するとする. そのように考えると, 上記(10)は上記 OWL 定義ファイルを RDF ユニバース中に読み込むことで, 伴意ルール **rdfs4a** により, 自然に成立する. 以下に実際に RDF(S)を実装した SWCLOS に OWL 定義ファイルを読み込んだのちの実行例を示す. 上記公理が実現されていることを確認されたい.

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

CLOS ではクラスはそのスーパクラスのスロット定義を継承する. owl:Class は rdfs:Class のサブクラスであるため, すべての OWL クラスは rdfs:subClassOf プロパティを持つことができて, rdfsClass は rdfs:Resource のサブクラスであるため, owl:Thing も含めてすべての OWL クラスと個体には rdf:type, rdfs:comment, rdfs:label など, RDF ユニバースにおいて定義されたすべてのプロパティが有効である. また, 上記公理において, owl:Restriction と owl:Class は OWL ユニバースにはなく, RDF ユニバース中にあることに注意されたい.

上記公理に加えて, OWL ユニバースを OWL-Full 可能にするために, 次の公理を追加した.

- owl:Class の URI の指示のクラス外延は owl:Thing のクラス外延に包含される.
  $CEXT'(owl:Class') \subseteq CEXT'(owl:Thing')$ (11)

すなわち, SWCLOS の書式では以下のようである.

```
(defResource owl:Class
  (rdfs:subClassOf owl:Thing)
```

この公理は, オブジェクト指向の視点から言えば, OWL ユニバースにおけるすべてのクラスは, 個体の持つプロパティ〈owl:sameAs や owl:differentFrom など〉を持つことができるということを意味している.

4.2 プロパティ値制約のクラス

OWL/RDF におけるオブジェクト中心の表現は, 一見 RDF グラフ表現よりもオブジェクト指向言語における表現と似ているが, 我々の OWL-Full スタイルでは, 依然として RDF の統語論と意味論に従う. その結果, rdfs:subClassOf や owl:intersectionOf の値によく出現するプロパティ値の制約は, システム中では owl:Restriction のインスタンスである匿名クラスとなる.

CLOS の観点で言えば, スーパクラス中に現れるスロット定義はサブクラスに継承されるので, 主題となる CLOS クラスのスーパクラスに存在するプロパティ値に関する制約〈スロット値制約〉が主題となるクラスに継承されて, そのインスタンスにおいて制

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

約を受けるということは, OWL 意味論上においても合理的である.

CLOS クラスに存在するスロット定義には, スロット値の型を指定するための CLOS ネイティブな type ファセットがあるが, プロパティ値制約の全称限量子と存在限量子の情報を保存するために, この type ファセットを有効に用いることにした. 一方, カージナリティ制約にはそのような CLOS ネイティブな設備が無く, スロット定義中に新たに mincardinality と maxcardinality のファセットを設けることとした. CLOS では, インスタンス生成時にはそのクラスのスーパクラスまで含めてすべてのスロット定義が集められて有効なスロット定義集合が計算されるが, その計算過程において, もし複数の type ファセット定義やカージナリティ定義があれば, それらを通算して最も特殊な概念や最大の mincardinality と最小の maxcardinality を計算し, 主題となるオブジェクトの有効なスロット定義とするようにした. かくして, インスタンス生成時にスロット値の制約条件の充足をチェックすることができる.

以下の実行例では, ワインオントロジーとフードオントロジーをロードしたあと, プログラマーが独自の特別料理を定義したが, そのインスタンス生成時に制約条件が満足されないためにエラーが発報されている.

```
(defResource TheSpecialCourse
  (rdf:type owl:Class)
  (owl:intersectionOf
    food:RedMeatCourse
    (owl:Restriction
     (owl:onProperty food:hasFood)
     (owl:allValuesFrom food:Fruit))))

(defIndividual No1SpecialCourse
  (rdf:type TheSpecialCourse)
  (food:hasFood food:Meat food:Bananas))

Error: Unsatisfiable by disjoint pair in
(#<owl:Class food:Fruit>
 #<owl:Class food:RedMeat>) for TheSpecialCourse
food:hasFood
```

ここで food:hasFood に対する制約 food:RedMeat はスーパクラス food:RedMeatCourse から継承されており, この特別料理の定義において定義された food:Fruit とは互いに disjoint であるために充足不能となった.

4.3 公理的完全関係

多くの OWL プロパティの中でも, 四つのプロパティ, すなわち owl:intersectionOf , owl:unionOf , owl:complementOf , owl:oneOf だけは, セマンティックウェブの原則である開世界原則に縛られない, 公理的完全関係として知られている[1]. すなわち, これらの四つのプロパティについては, 宣言において「もし･･･ならば, そしてその時に限り･･･」という完全な関係が定義されるため, プログラマーは宣言に遭遇したときに, それ以外の場所で何が宣言されているかを気にする必要がない. たとえば, 次の宣言は vin:WhiteBordeaux の定義であり, 人はもし何かが Bordeaux でかつ WhiteWine であれば, それを WhiteBordeaux と定義してよいということを意味している.

```
<owl:Class rdf:ID="WhiteBordeaux">
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Bordeaux" />
  <owl:Class rdf:about="#WhiteWine" />
  </owl:intersectionOf>
</owl:Class>
```

[1] http://www.w3.org/TR/owl-ref/#DescriptionAxiom

同様に, 次の宣言は vin:WineColor を定義し, そのインスタンスは White, Rose, Red であり, それ以外にはないということを意味している.

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf
    rdf:resource="#WineDescriptor" />
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#White" />
    <owl:Thing rdf:about="#Rose" />
    <owl:Thing rdf:about="#Red" />
  </owl:oneOf>
</owl:Class>
```

もしデータベース中にあるものが vin:White であるということがわかれば, それを自動的に vin:WineColor に分類してよいし, もしデータベース中にあるものが vin:WineColor であるということがわかれば, それは White, Rose, Red のどれかであると推論してよい.

SWCLOS においてはプロアクティブな伴意の実行機能としてこのような公理的完全関係を積極的に利用している. 以下の実行例ではシステムは自動的に QueenElizabethII は Woman であると伴意した. なぜならば, Femail という性別を持つ人は Woman であると宣言されていて, QueenElizabethII の性別は Femail だからである.

```
(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>
```

4.4 本質的プロパティと非本質的プロパティ

OWL には, rdfs:subClassOf, owl:intersectionOf, owl:unionOf, owl:equivalentProperty など, 概念(クラス)の包含関係を左右する多くのプロパティが存在する. 述語論理の観点から言えば, これらのプロパティは包含関係を決定する強度において何の違いもない. しかしオントロジー工学とソフトウェア工学の観点から言えば, 概念の本質的関係と非本質的関係を区別することは重要である. Borgida[Borgida03]は移ろいやすい参照関係ではなく, 固定的な個体オブジェクトを扱わなければならないと主張した. 溝口[溝口 05, 溝口 06]は本質的関係である IS-A 関係のみについて単一継承によりオントロジー構築し, その他の関係は part-of 関係やロール概念として整理すべきであると主張している. 兼岩と溝口[兼岩 05]は拡張ソート論理を用いて本質･非本質的などの観点でプロパティを分類した.

データベースの保守性の観点からも永続的関係と非永続的関係を区別することは重要である. RDF ユニバースにおいて(そして OWL ユニバースでも), SWCLOS では rdfs:subClassOf は CLOS のクラス･サブクラス関係にマップされる. あるクラス定義における rdfs:subClassOf プロパティ値であるクラスは, 単純にその主題のクラスの direct-superclasses-list という CLOS スロッ

トに置くだけで, クラス包摂関係の推論が CLOS において実行される. しかし OWL ユニバースにおいて, プロパティ p1 が rdfs:subPropertyOf の equivalent なプロパティであったり, サブプロパティであったりした場合, その p1 の値は direct-superclasses-list に置くべきであろうか, それとも置かざるべきであろうか. CLOS スロットの direct-superclasses-list に置いた場合には, それは CLOS の操作においてスロット定義などの継承によりオブジェクトの構造的変化が引き起こされる. より一般化して言えば, OWL のどのプロパティが CLOS の構造的変化を起こしてもよく, どのプロパティは構造的変化を起こすことなく, OWL の推論機能として実装されるべきであるか, という問題である. 我々は rdfs:subClassOf, owl:intersectionOf, owl:unionOf は構造的変化を引き起こしてよく, その他のプロパティによる関係は OWL 推論によるべきであるとした. 逆に言えば, SWCLOS においては本質的・永続的関係は rdfs:subClassOf, owl:intersectionOf, owl:unionOf により記述され, 非本質的関係・非永続的関係はその他のプロパティにより記述されるべきであると考えた. そして, 本質的・永続的関係はプロアクティブな伴意を適応されるが, 非本質的関係・非永続的関係にはプロアクティブな伴意は適応されず, OWL 推論が適応される.

4.5 拡張構造的包摂計算アルゴリズム

我々は概念の合成(conjunction)と全称限量子のみを許す $\mathcal{FL}_0$ レベルの構造的包摂計算アルゴリズム[Baarder03]を, owl:disjointWith, owl:complementOf, owl:sameAs, owl:equivalentClass, owl:equivalentProperty, owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:someValuesFrom, owl:hasValue, owl:maxCardinality, owl:minCardinality, owl:cardinality, rdfs:subClassOf, owl:intersectionOf, owl:unionOf に対応するように, 次のように拡張した. ただし, 前述したように, rdfs:subClassOf, owl:intersectionOf, owl:unionOf に関しては CLOS のクラス・サブクラス関係に置き換えられている. このアルゴリズムでは, owl:Thing は CLOS クラス・サブクラス関係においても OWL クラスのトップに位置しているが, owl:Nothing は CLOS オブジェクトとしては存在していても, 本アルゴリズムのみにおいて仮想的にすべての OWL クラスのボトムとなっている.

CEXT' (*owl:Class'*) の要素である *C* と *D* に関して
`subsumed`(*CEXT'* (*owl:Class'*) *C*, *CEXT'* (*owl:Class'*) *D*)
if *C* = *owl:Nothing'* **then** return true ; Proc. 1
if *D* = *owl:Thing'* **then** return true ; Proc. 2
if *C* = *owl:Thing'* **then** return false ; Proc. 3
if *C* is a subtype of *D* in CLOS **then** return true ; Proc. 4
for some C_x **in** `equivalent-classes-of`(*C*) ; Proc. 5
  for some D_x **in** `equivalent-classes-of`(*D*) ; Proc. 6
    if `complement`(C_x , D_x) **then** return false ; Proc. 7
    for every D_{xx} **in** `intersect-classes-of`(D_x) ; Proc. 8
    and every S_{xx} **in** `intersect-restrictions-of`(D_x) ; Proc. 9
      for some C_{xx} **in** `all-superclasses-of`(C_x) ; Proc. 10
      and some R_{xx} **in** `all-superrestrictions-of`(C_x) ; 11
        if `subsumed`(C_{xx} , D_{xx}) **then**
          if `equivalent-property`(`onproperty-of`(R_{xx}), `onproperty-of`(S_{xx}))
          then
          if `qualification-subsumed`(R_{xx}, S_{xx}) **and** `cardinality-subsumed`(R_{xx}, S_{xx})
          then return true
          else return false
        else return false
      else return false

`qualification-subsumed`(*CEXT'* (*owl:Restriction'*) R_{xx} , *CEXT'* (*owl: Restriction'*) S_{xx})
if (`subsumed`(`hasvalue-of`(R_{xx}), `hasvalue-of`(S_{xx}))
  then return true
else if (`subsumed`(`allvaluefrom-of`(R_{xx}), `allvaluefrom-of`(S_{xx}))
  then return true
else if (`typed`(`hasvalue-of`(R_{xx}), `allvaluefrom-of`(S_{xx}))
  then return true
else if (*`subsumed`(`somevaluefrom-of`(R_{xx}), `allvaluefrom-of`(S_{xx}))
  then return true
else if (*`subsumed`(`somevaluefrom-of`(R_{xx}), `somevaluefrom-of`(S_{xx}))
  then return true
else if (`subsumed`(`allvaluefrom-of`(R_{xx}), `somevaluefrom-of`(S_{xx}))
  then return true
else if (`subsumed`(`hasvalue-of`(R_{xx}), `somevaluefrom-of`(S_{xx}))
  then return true
else return false

`cardinality-subsumed`(*CEXT'* (*owl:Restriction'*) R_{xx} , *CEXT'* (*owl: Restriction'*) S_{xx})
if `mincardinality-of`(R_{xx}) ≧ `mincardinality-of`(S_{xx}) **and**
  `maxcardinality-of`(R_{xx}) ≦ `maxcardinality-of`(S_{xx})
  then return true
else return false

ここで Proc. 1 において owl:Nothing のボトム概念がサポートされ, Proc. 2 と 3 において, owl:Thing のトップ概念がサポートされている. Proc. 4 には, rdfs:subClassOf に加えて owl:intersectionOf と owl:unionOf の概念関係における包摂計算が含まれる.

我々のシステムでは, クラスも含めてすべてのオブジェクトが owl:sameAs と推移プロパティに関して, 個体としての包摂関係の計算を受ける. すなわち, 以下のとおり.

CEXT' (*owl:Thing'*) の要素である *C* と *D* に関して
`subsumed`(*CEXT'* (*owl:Thing'*) *C*, *CEXT'* (*owl:Thing'*) *D*)
if `sameas`(*C*, *D*) **then** return true
else
for some prop
  in `intersection`(`all-transitive-props-of`(*C*), `all-transitive-props-of`(*D*))
  for some D_x **in** `same-things-of`(*D*)
    for some C_x **in** `same-things-of`(*C*)
      if `transitively-sub-on`(prop, C_x , D_x) **then** true **else** false

上記アルゴリズムは明らかに再帰を含んでいる. しかし SWCLOS においてこの計算は停止する. なぜならば, CLOS はそのクラス・サブクラス関係においてターミノロジカルな巡回定義(例えば *C* が直接間接に *D* のサブクラスであり, *D* も直接あるいは間接に *C* のサブクラスである)を許していないし, rdfs:subClassOf と owl:unionOf の組み合わせ(例えば *B* は *C* のサブクラスで *C* は *A* と *B* の合併概念)において定義を順にトレースしたときに生じる巡回の生起(*B*→*C*→*B*)なども Proc.10 における手続き(*C* のすべてのスーパクラスを一括して取り出す)を用いることにより, 生じないからである.

この拡張構造的包摂計算アルゴリズムは, アルゴリズム中において*をつけた owl:someValueFrom に起因する部分と, owl:someValueFrom の意味とカージナリティ制約の組み合わせによって生じる制約の計算について不完全であるが, 実用的にはほとんどの場合に有効である.

4.6 充足性のチェック

プロアクティブな伴意ルールの実行は, 結果的に充足性のチェックの負荷を大いに低減し, 曖昧なオントロジーモデリングを許さないが, それでもオントロジーのバグを防ぐために充足性のチェックは必要である. 我々はシステム中に, プロパティの定義域・値域チェック, カージナリティチェック, disjoint 関係のチェックなどを組み込んだ. それに加えて, 表1のような非充足条件を追加した.

表1 OWL における追加非充足条件

非充足	条件
unsatisfiability1	*C* oneOf {x_1 …} *y* type *C* *y* differentFrom x_i
unsatisfiability2	*x* differentFrom *y* *x* sameAs *y*
unsatisfiability3	*C* disjointWith *D* *D* equivalentClass *C*
unsatisfiability4	*C* disjointWith *D* *x* type *C* *x* type *D*

4.7 OWL 伴意ルール

RDF および RDFS における伴意ルールは完全に分かっている[Hayes04]. 一方で, OWL における伴意ルールはまだ完全には分かっていなくて, ter Horst[Horst05]によりこれまで完全化への努力が進められてきている. ルールベースの OWL 推論システムや SWCLOS のような手続き型の OWL 推論システムでは伴意ルールを必要とする. タブロー法によれば一見伴意ルールは必要ないように思われるかもしれないが, そうではない. タブロー法では他の論理型推論システムと同様に反駁法を用いている. すなわち, ある表明が正しいかどうかを証明するためにその否定を知識ベースに追加して矛盾を導出する. ある宣言を知識ベースに追加したとき, それがモデルを非充足にするかどうかはタブロー法により判明するが, 宣言の追加によって何が導出できるかを明らかにするためには伴意ルールが必要である.

SWCLOS では, ter Horst による OWL 伴意ルールのほか, 独自の伴意ルールも追加実装している. 本節ではこれらの伴意ルールとその SWCLOS における実装方法について説明する. 表2には追加された公理を, 表3には追加された伴意ルールを示す. 本節において, **rdf*** および **rdfs**** という記述はそれぞれ RDF 伴意ルールおよび RDFS 伴意ルール [Hayes04]を指し, **rdfp**** という記述は ter Horst による P-entailment ルールを指している. **rule*** という記述は表3に示したルールである.

表2 追加された OWL 公理

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

(1) SameAs, EquivalentClass, EquivalentProperty

owl:sameAs, owl:equivalentClass, owl:equivalentProperty プロパティは反射的(**rdfp6**, **rdfp12a**, **rule10**)かつ推移的(**rdfp7**, **rdfp12c**, **rule11**)である. したがって, 関連するプロパティが連鎖的にすべて一緒になって, 一つのグループを形成する. 我々の実装ではこのグループ情報(グループに所属するすべてのエンティティ)がグループの各メンバーに記録されており, 前述のアルゴリズムにおける `same-things-of` 関数, `equivalent-class-of` 関数, `equivalent-property` 関数ではその情報を参照する.

(2) DifferentFrom ペアと DisjointWith ペア

一方, owl:differentFrom と owl:disjointWith は反射的ではあるが, 推移的ではない. したがってこれらの関係はペアを形成するが一つのグループとはならない. 前述のアルゴリズムにおける. 我々の実装ではこれらの関係が, ペアの両方に記録されており, 等価性の判定などに用いられる. 特に, 包摂性計算の否定との複合において disjoint 関係は重要であり, 上記 Proc4 で用いられている. また disjoint 関係はサブクラスに浸透的である. 上記 **rule4** を参照のこと.

表3 追加された伴意ルール

		Then
rule1a	*v p w* *v* type Class	*v* subtype Thing
rule1b	*v p w* *w* type Class	*w* subtype Thing
rule2a	*u* intersectionOf {v_j . . .}	v_j type Class
rule2b	*u* unionOf {v_j . . .}	v_j type Class
rule3	*x* distinctMembers {x_j . . .}	x_j type Thing
rule4	*u* disjointWith *v* u_0 subClassOf *u* v_0 subClassOf *v*	u_0 disjointWith v_0
rule5	*u* complementOf *v*	*v* complementOf *u*
rule6	*u* complementOf *v*	*v* disjointWith *u*
rule7	*u* oneOf {x_j . . .}	x_j type *u*
rule8	*v* allValuesFrom *w* *v* onProperty *p* *p* range *u*	*w* subtype *u*
rule9	*p* type SymmetricProperty *p* domain *C* *p* range *D*	*C* equivalentClass *D*
rule10	*p* equivalentProperty *q*	*q* equivalentProperty *p*
rule11	*p* equivalentProperty *q* *q* equivalentProperty *r*	*p* equivalentProperty *r*

(3) Functional Property および Inverse Functional Property

owl:FunctionalProperty と owl:InverseFunctionalProperty のインスタンスであるプロパティについてはそれぞれ **rdfp1**, **rdfp2** が成立するが, 我々の実装では互いの関係が互いに記録されており, このルールは等価性の判定述語や前述のアルゴリズム中では `same-things-of` 関数に反映されている.

(4) Symmetric Property

owl:SymetricProperty についても同様に, 互いの関係が互いに記録されている. また, **rule9** も参照のこと.

(5) 概念の合成(owl:intersectionOf)

これまで各所で述べたように, 概念の合成は各概念の下位概念を形成する. 我々の包摂計算の実装では, 合成概念の direct-superclasses-list に各概念を置くことで, CLOS により包摂計算が行われる. また, 合成概念を構成する各概念のサブクラスに共通する概念があれば, それを合成概念のサブクラスとする.

(6) 概念の合併(owl:unionOf)

反対に, 概念の合併は各概念の上位概念を形成する. われわれの実装では, 各概念の direct-superclasses-list に合併概念を置くことで, CLOS により包摂計算が行われる. また, 合併される各概念に共通のスーパクラスがあれば, それを合併概念のスーパクラスとする.

(7) 概念の否定(owl:complementOf)

概念の complement は反射的(**rule5**)かつ互いに disjoint (**rule6**)である. 我々の実装では互いにその情報を記録すると同時に, disjoint 関係の記録もする. 前述の包摂計算アルゴリズムでは Proc7 にて否定関係の情報による計算の短縮に用いられている.

5. 実装とベンチマーク性能

これまで述べたように, RDF のクラス／インスタンス関係を CLOS のクラス／インスタンス関係にマップし, RDF クラス-サブクラス関係を CLOS のクラス-サブクラス関係にマップして, RDF(S)と CLOS の意味論の違いを, CLOS の Meta-Object-Protocol を用いて解決した. プロパティの包摂関係が新たに実装された. rdfs:Class のメンバーシップ巡回は外見上そのような型判定述語 typep を実装することで解決した. RDF ユニバースの CLOS 上の詳細化として OWL ユニバースを実現し, OWL の包摂計算アルゴリズムと多くの OWL 伴意推論ルールを実装した.

Lisp の特徴を生かした S-式による定義式や各種 API のほかに, RDF/XML 記述のパーザを開発し, RDF/XML 記述のオントロジーを入力可能とした. また, SWCLOS 上でのオントロジー(その実態は CLOS オブジェクト)を RDF/XML としてシリアライズする機能も開発した. RDF/XML 出力機能開発にあたっては Common Lisp 仕様にあるプリティプリント機能が活用された. また, N トリプルの入力･出力機能も開発された.

MS-Windows2000 上の Allegro Common Lisp 8.0 を用いて, フードオントロジーとワインオントロジーを RDF/XML のまま, 合わせて 2 秒でロードする(Pentium 4, 2.6 GHz, 1GB RAM). ベンチマークテストとして, リーハイ大学ベンチマークテスト(LUBM(1, 0))[Guo05]を用いた. ロードに 4 分 14 秒(マシンは同上)であり, LUBM の 14 個のクエリ用に簡単な Lisp 関数をプログラムして実験した. 他のセマンティックウェブ用ツールと比較して, 遜色ない性能の結果であり, 14 個のクエリすべてについて正解を回答した[Koide06].

6. OWL-Full とメタモデリング

6.1 RDF(S)と OWL のメタモデリング

OWL-Full は RDF(S)と同様に, クラスを個体として扱うことのできる機能を提供するものとされている. しかし, だからと言って無原則的にクラスを個体として扱っていいということにはならない. オブジェクト指向の観点から言えば, クラスを個体として扱うとき, すなわちクラスをあるクラスのインスタンスとして扱うとき, そのクラスのクラスはメタクラスと呼ばれ, そのための特別な仕組みが用意されている. 本章では, SWCLOS において OWL-Full の機能を発揮するために, CLOS の視点からこれまでのオントロジー構築における問題点を整理し, メタモデリングのための明快な基準を提案する.

SUMO オントロジー[1]では, SystemeInternationalUnit のインスタンスである Meter は PhysicalQuantity のサブクラスであるが, SystemeInternationalUnit 自身もそのスーパクラスに UnitOfMeasure を経由して PhysicalQuantity を持っている. (図1参照) すなわち, 二者の間には以下の条件が成立する.

$$sumo:Meter^{I} \in CEXT^{I}(sumo:PhysicalQuantity^{I}) \quad (12)$$
$$CEXT^{I}(sumo:Meter^{I}) \subseteq CEXT^{I}(sumo:PhysicalQuantity^{I}) \quad (13)$$

SUMO オントロジーでは一見 OWL-Full の推論を要求するように見えるかもしれないが, このような無原則的なクラスとインスタンスの同一視はそもそも RDF(S)の, ひいては OWL-Full の意図するところではない. RDF(S)と OWL-Full のメタモデリングにおいて, そのようなアドホックなやり方ではなく, より厳密な形式化が必要である. OWL-Full はクラスを個体として扱うことができるとされながら, 実際にどのようなメタモデリング形式化を実施するかという問題は, これまでセマンティックウェブにおいてまともに取り組まれていなかった.

6.2 オブジェクト指向の視点からのメタモデリング基準

オブジェクト指向の視点から言えば, クラスを個体として扱うためにはそのクラス, すなわちメタクラスが必要である. 例えば, Meter を個体として扱うために, そのクラスである SystemeInternationalUnit がメタクラスでなければならない. オブジェクト指向の視点では, オブジェクトの性質はそのスーパクラスから継承される. すなわち, メタクラスがメタクラス足りうるためには, そのスーパクラスにメタクラスを持つことが必要である. RDF ユニバースにおけるメタクラスの源泉は rdfs:Class である. OWL ユニバースにおけるメタクラスの源泉は owl:Class であり, 4.1 節で述べたように, 我々の方式では owl:Class も rdfs:Class のサブクラスとしたために, メタクラスとなる. ところが先の SUMO オントロジーでは, SystemeInternationalUnit は一見メタクラスに見えながら, メタクラスとなっていない. 図1に SUMO オントロジーにおける SystemeInternationalUnit から最上位である Entity までの階層図を示す. SystemeInternationalUnit をメタクラス化するためには, 直接間接にこれを owl:Class あるいは rdfs:Class のサブクラスにしなければならない. それではどこで owl:Class あるいは rdfs:Class にサブクラスとして接続すべきであろうか.

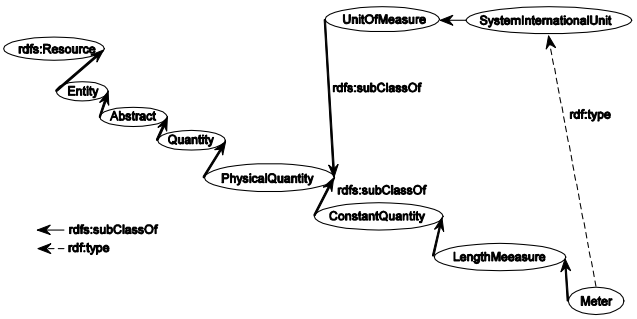

図 1 SUMO オントロジーにおける素朴なオントロジー階層

ここで, やはり二つのクラス間のメンバーシップ関係と包摂関係に注目する. もしあるオブジェクトがあるクラス外延の要素だとしても, それが同時にそのクラスのサブクラスになるとすれば, それを通常のメンバーシップと区別して, (14)式のように記述する. 同様に, もしあるクラスが他のクラスのサブクラスだとしても, それが同時にそのクラス外延に所属するならば, それを通常のクラス-サブクラス関係と区別して, (15)式のように記述する.

$$sumo:Meter^{I} \in_{\subseteq} CEXT^{I}(sumo:PhysicalQuantity^{I}) \quad (14)$$
$$CEXT^{I}(sumo:Meter^{I}) \subseteq_{\in} CEXT^{I}(sumo:PhysicalQuantity^{I}) \quad (15)$$

メタモデリングのための最低基準を以下のように設定する.

- もしクラス C がクラス D のインスタンスではあるが, サブクラスではないとき, D をメタクラスにしてよい.

$$C^{I} \in CEXT^{I}(D^{I}) \rightarrow CEXT^{I}(D^{I}) \subseteq CEXT^{I}(rdfs:Class^{I})$$

- もしクラス C がクラス D のインスタンスでかつサブクラスでもあるとき, D をメタクラスにできない.

$$C^{I} \in_{\subseteq} CEXT^{I}(D^{I}) \rightarrow \neg(CEXT^{I}(D^{I}) \subseteq CEXT^{I}(rdfs:Class^{I}))$$

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

この二つの最低基準から, 次の基準を導くことができる.

- もしクラス C がクラス D のインスタンスかつサブクラスであり, かつもしクラス C が D のサブクラスであるクラス B のインスタンスであるがサブクラスではないとき, B は C のメタクラスにできるが, D はできない.

$$C' \in_{\subseteq} CEXT'(D') \wedge C' \in CEXT'(B') \wedge CEXT'(B') \subseteq CEXT'(D') \rightarrow CEXT'(B') \subseteq CEXT'(rdfs:Class')$$

関係を図示すれば一目瞭然に理解できる.

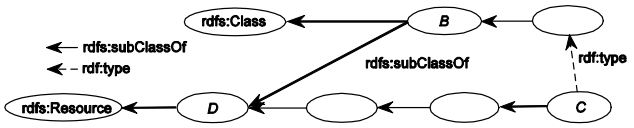

図2 メタモデリング基準の図示

先の SUMO オントロジーに戻して考えれば, Meter は PhysicalQuantity のサブクラスであるが UnitOfMeasure と SystemeInternationalUnit のサブクラスではない. それゆえ, UnitOfMeasure を rdfs:Class のサブクラスとすることで, 両者をメタクラス化することができ, 計算可能となる.

6.3 RDF(S)とオブジェクト指向におけるリフレクション

RDF(S)はメタモデリングの機構を有するのみならず, オブジェクト指向言語におけるリフレクションと同じ機構を有している. 本節では SUMO オントロジーを例にして, リフレクティブなオブジェクト指向言語の視点から RDF(S)におけるリフレクティブ機構を考察する.

SUMO オントロジーにおける Meter はクラスであり, LengthMeasure も同様にクラスであるが, 前述のように SystemeInternationalUnit はメタクラスである. SUMO オントロジーにおける RDF/XML 記述は以下のとおり.

```
<rdfs:Class rdf:ID= "Meter">
  <rdfs:subClassOf rdf:resource ="#LengthMeasure"/>
  <rdf:type rdf:resource =
                    "#SystemeInternationalUnit"/>
    <rdfs:comment> ... </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID= "SystemeInternationalUnit">
  <rdfs:subClassOf rdf:resource ="#UnitOfMeasure"/>
  <rdfs:comment> ... </rdfs:comment>
</rdfs:Class>
```

一見, 両者の記述は同じように見えるかもしれないが, メタモデリングの観点では rdfs:Class の意味が異なる. RDF/XML に関する統語論における typed node element[1]記述方法によれば, 上記は以下と同等である.

```
<rdf:Description rdf:ID= "Meter">
  <rdfs:subClassOf rdf:resource ="#LengthMeasure"/>
  <rdf:type rdf:resource = "&rdfs;Class"/>
  <rdf:type rdf:resource = "#SystemeInternationalUnit"/>
  <rdfs:comment> ... </rdfs:comment>
</rdf:Description>

<rdf:Description rdf:ID= "SystemeInternationalUnit">
  <rdf:type rdf:resource = "&rdfs;Class"/>
  <rdfs:subClassOf rdf:resource ="#UnitOfMeasure"/>
  <rdfs:comment> ... </rdfs:comment>
</rdf:Description>
```

[1] http://www.w3.org/TR/rdf-syntax-grammar/#section-Syntax-typed-nodes

すなわち, Meter の記述に書かれた rdfs:Class はタイプ付けされた SystemeInternationalUnit と同じくメタクラスであるが, SystemeInternationalUnit の記述における rdfs:Class はメタクラスがタイプ付けされるところのクラス, すなわちメタ･メタクラスである. メタクラス rdfs:Class のクラスはメンバーシップ巡回によって rdfs:Class であることに注意されたい.

さて, SUMO オントロジーにおける UnitOfMeasure を OWL-Full 流に個体として扱うことにしよう. たとえば, 新しいプロパティ値を UnitOfMeasure に追加して, そのプロパティの定義域は UnitOfMeasure をインスタンスとする UnitOfMeasureClass とする.

```
<rdfs:Class rdf:ID= "UnitOfMeasure">
  <ex:p1 rdf:resource ="&ex;Value1" />
</rdfs:Class>

<owl:ObjectProperty rdf:ID = "&ex;p1">
  <rdfs:domain rdf:resource ="&ex;UnitOfMeasureClass" />
</owl:ObjectProperty>
```

この定義は一見自然であるが, メタモデリングの観点からは注意が必要である. 前節で明らかなように, UnitOfMeasure はメタクラスであるから, UnitOfMeasureClass はメタ･メタクラスである. したがって次のように定義されなければならない (SWCLOS では上記宣言により自動的にそのように伴意される).

```
<rdfs:Class rdf:resource = "&ex;UnitOfMeasureClass">
  <rdfs:subClassOf rdf:resource ="&rdfs;Class" />
</rdfs:Class>
```

ここでは二つの rdfs:Class が現れているが, subClassOf の値である rdfs:Class は UnitOfMeasureClass がメタ･メタクラスのはずであるから, メタ･メタクラスであり, typed node element としての rdfs:Class は, したがってメタ･メタ･メタクラスのはずである. メタ･メタクラス rdfs:Class のクラスはメンバーシップ巡回によって rdfs:Class であるから, これは成立する.

端的に言って, rdfs:Class のメンバーシップ巡回によって rdfs:Class はメタクラス, メタ･メタクラス, メタ･メタ･メタクラス, 等々の多重な役割を担うことができ, rdfs:Class をスーパクラスとすることで, 任意のクラスをメタクラス, メタ･メタクラス, メタ･メタ･メタクラス, 等々として定義することができる. subClassOf によってどのメタレベルの rdfs:Class をスーパクラスとして持つかによって, メタクラスは層状に分化するが, rdfs:Class のメンバーシップ巡回によってそれが1階層に纏められているため, 外見上個別クラス定義を見ただけではメタ階層のどこに所属するかは判定できない. 我々の計算方式ではあるクラスがどの階層に所属するかは固定的であり, ボトム (インスタンス) 層, クラス層, メタクラス層を混合することは許されないし (あるインスタンスをクラスにしたり, クラスをメタクラスに変更したりすることはできる), メタクラス階層中のどの内部メタクラスに所属するかは曖昧である. メタクラス階層は上方には無限であるため, 理論上他のクラスとのクラス／インスタンス関係により波及的にボトムが確定したときに初めてそのメタクラスのレベルが確定する.

7. OWL-Full によるメタモデリング例

7.1 クラスにスロット追加のためのメタクラス

国際ワイン協会がすべてのワインブランドに ID ナンバーをふる事を考えたとしよう. ワインオントロジーにはクラスとしてワインブランドだけではなく, カリフォルニアワインというようなブランドではないワインクラスも定義してある. そこで BrandWine と NonBrandWineConcept とにこれを分けることにする. 両者を Wine のサブクラスとしてもそれだけでは, 次に示す

hasIDNumber 定義では, ブランドワインに ID ナンバーをつけられない. そこで BrandWine をメタクラスとして定義する.

```
(defProperty hasIDNumber (rdf:type owl:ObjectProperty)
  (rdfs:domain BrandWine)
  (rdfs:range xsd:positiveInteger))
 (defResource BrandWine (rdf:type owl:Class)
  (rdfs:subClassOf vin:Wine owl:Class))
(defResource NonBrandWineConcept (rdf:type owl:Class)
  (rdfs:subClassOf vin:Wine owl:Class))
```

これで以下のように定義することができる.

```
(defResource vin:Zinfandel (rdf:type BrandWine)
  (hasIDNumber 12345))
```

7.2 インスタンスのクラス扱い

ウェブサービスを利用したロケット打上げ支援用システム[小出 07]において, ウェブサービス前提条件として OWL-S[1]の expr:Condition と各種オペレーションモードの合成概念を考え, オペレーションモードに対応した expr:expressionBody で評価すると同時に, 実際のイベントをこれらの概念のインスタンスとして記録したい. ところが, OWL-S 仕様では, プロパティ process:hasPrecondition プロパティ値は expr:Condition のインスタンスでなければならず, クラスを取れない. そこで, expr:Condition を継承するメタクラスを準備することで, rocess:hasPrecondition プロパティ値にクラスを取ることを可能にする.

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

8. まとめ

記述論理を用いることにより, ソフトウェア工学のアプリケーションドメインを論理の力により定式化することができ, それにより多くの利益を得ることができる. OWL はオントロジー記述言語のみならず, ソフトウェア開発のためのモデリング言語となり得る. SWCLOS は RDF(S)および OWL によるオントロジー記述のための言語のみならず, CLOS を用いたオブジェクト指向プログラミングのためのモデリング言語でもあり, プログラマーは記述論理をベースに彼らのアイデアを交換し, しかもモデリング言語をそのまま用いてオブジェクト指向プログラミングを実行できる.

本報告では RDF(S)記述言語としての SWCLOS の実現について述べた後, オブジェクト指向の視点から OWL の実現について述べ, RDF(S)および OWL-Full におけるメタモデリングについて考察して, メタモデリングのための基準を新たに提案した. SWCLOS は存在限量子について不完全であるが, 実用上は十分な性能を有していることを, ベンチマークにおいて確認した. 既にいくつかの応用プログラムが SWCLOS を用いて開発されているが, ソースフリーとして公開[2]されているので, 日本発の OWL-Full 処理系として世界に広く普及することを期待している.

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

[2] http://pegasus.agent.galaxy-express.co.jp/SemanticWeb-swclos-ja.htm

参考文献

[Borgida03] Borgida, A and R. J. Brachman. *The Description Logic Handbook*, chapter 10. Conceptual Modeling with Description Logics, pp. 349–372. Cambridge, 2003.

[Knublauch06] Knublauch, H, D. Oberle, P. Tetlow, and E. Wallace. A semantic web primer for object-oriented software developers. W3C Working Group Note, http://www.w3.org/TR/sw-oosd-primer/, March 2006.

[Smith04] Smith, M. K., C.Welty, and D. L. McGuinness. OWL web ontology language guide. W3C Recommendation, http://www.w3.org/TR/owl-guide/, February 2004.

[Nardi03] Nardi, D. and R. J. Brackman. *The Description Logic Handbook*, chapter 1. An Introduction to Description Logics, pp. 1–40. Cambridge, 2003.

[Baarder03] Baarder, F. and Werner Nutt. *The Description Logic Handbook*, chapter 2. Basic Description Logics, pp. 43–95. Cambridge, 2003.

[Schneider04] Patel-Schneider, P. F., P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax section 5. RDF-compatible modeltheoretic semantics. W3C Recommendation, http://www.w3.org/TR/owlsemantics/rdfs.html, Feb. 2004.

[溝口 05] 溝口理一郎. オントロジー工学, オーム社, 2005.

[溝口 06] 溝口理一郎, 來村徳信, 笹島宗彦. オントロジー構築入門, オーム社, 2006.

[兼岩 05] 兼岩 憲, 溝口理一郎. 形式オントロジーと順序ソート論理の拡張, 人工知能学会論文誌, Vol.20 No.6 pp.387–395.

[Hayes04] Hayes, P. and B. McBride. RDF semantics. W3C Recommendation, http://www.w3.org/TR/rdf-mt/, Feb. 2004.

[Horst05] ter Horst, H. J.. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary, *Journal of Web Semantics*, Vol.3, pp.79–115, Elsevier, 2005.

[Guo05] Guo, Y., Z. Pan, and J. Hefin. LUBM: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, Vol. 3, No.2, pp.158–182, Elsevier, 2005.

[Koide06] Koide, S. H. Takeda. OWL-Full reasoning from an object oriented perspective. In Asian Semantic Web Conf., ASWC2006, pp. 263–277. Springer, 2006.

[小出 07] 電気学会･次世代の原子力運転保守技術調査専門委員会, 次世代のプラント運転支援技術, 5.3 節, pp.211–226, 大学教育出版, 2007.