リスプによる RDF プロセッサ
Lisp-based RDF Processor

小出 誠二 Seiji Koide
川村 正則 Masanori Kawamura

株式会社ギャラクシーエクスプレス
Galaxy Express Corporation

Semantic Web is a promising application area for Lisp programming language as well as Prolog, because basic requirements for semantic web processing such as axiom, entailment, and constraint are also basic vocabularies and infrastructure of inference in AI. Roughly speaking, from the viewpoint of the implementation, semantic web processors fall into a few categories; statement-based (XML-based), graph-based (triple-based), and frame-based or object-based. Despite that there are diverse implements on semantic web processors and toolkits in various programming languages; there is only one implement in Lisp, that is subject-predicate-object triple based. We have developed an object-based RDFS processor on CLOS using the Meta-Object Protocol, with straightforward mapping of RDFS class to CLOS class and RDFS individual to CLOS instance. By this mapping, we have obtained an efficient performance of processing instead of the difficulty of implementation. This paper describes an implementation of the processor on CLOS, and discusses ideas to extend it to OWL. We claim that the ends of Semantic Web processing will be attained through this approach, namely meta-object reflective approach, since CLOS is one of reflective languages and the processor is expected to provide meta-modeling facilities in OWL Full level.

1. はじめに

次世代ウェブ技術として，セマンティックウェブが注目されている．RDF はウェブ上の情報についてメタ記述するための表記方法であり，RDFS は RDF 表記によるスキーマ記述のための最小限のボキャブラリである．OWL は RDF, RDFS を用いたオントロジ記述用マークアップ言語であり，OWL-S は OWL を用いたウェブサービスに関するオントロジ記述のための規約である．

オントロジ記述における RDFS の rdf:type 包摂律(subsumption rule)や rdfs:subClassOf 推移律(transitivity rule), あるいは rdfs:domain と rdfs:range の制約や OWL の局所プロパティ制約に代表されるような公理(axiom)とその伴意ルール(entailment rule)を実装しようとすると，Prolog やリスプのような人工知能用言語を用いてセマンティックウェブの処理系を開発[Wielemaker 2003] [Lassila 2002]するのが便利である．

セマンティックウェブ処理系は，用いる計算機言語にかかわらず基礎となるデータ表現に何を用いるかによって，大きくわけて宣言ベース，グラフベース，フレームベースあるいはオブジェクトベースに分けられる．宣言ベースは DOM のような RDF/XML 表記の素直な計算機内表現であり，グラフベースは N-Triple 表記の計算機内表現である．グラフベースの実装では，ノードとエッジをエンティティとして計算機内部に有向グラフを生成する．N-Triple の1行(サブジェクト，プレディケイト，オブジェクト)はグラフ上のノード･エッジ･ノードの一組である．一方，グラフ中の一つのノードを中心に，それから放射するエッジとその先のエンティティまでの情報をまとめて一つにしたものをフレームベースあるいはオブジェクトベースの実装と考える．RDF モデル論では RDF グラフがモデル論の理論的基礎となっていて，グラフベースの実装では RDF あるいは RDFS の意味論を実装するのは容易であるが，オブジェクト指向的な OWL（オブジェクト内部のプロパティに局所的制約を持つ）の効率的な実装には困難が予想される．一方，フレームベースあるいはオブジェクトベースの実装では RDF あるいは RDFS の意味論を実装することはグラフベースに比べて難しいが，OWL においては効率的で自然な実装が比較的容易と考える．オブジェクト機能を持たない Prolog では N-Triple の宣言的表現が Prolog のファクトと本質的に同等であることから RDF の実装は容易であるが，rdfs:subClassOf や rdfs:subPropertyOf の効率的な実装が必要[Wielemaker 2003]となり，OWL の意味論の実装は難しいと思われる．[Lassila 2002]は Common Lisp Object System (CLOS) を用いているがそのエンティティはグラフベースであり，表面上フレーム指向的な API を提供している[1]．

我々は文科省 IT プロジェクト「IT を活用した大規模運用システムの支援システムの構築」[小出 2002][Koide 2003]において，ロケット打上げを題材として大規模運用支援システムを開発しているが，ここではセマンティックウェブ技術とウェブサービス技術の活用を目指している．OWL 記述ドメインオントロジと OWL-S 記述タスクオントロジの処理を最終目標に，今回 RDF/XML 記述を読み，RDFS 公理と伴意関係を処理する RDFS プロセッサを，リスプを用いて開発した．この処理系では Common Lisp Object System (CLOS)によるオブジェクトを基本エンティティとし，CLOS メタプログラミングのための Meta-Object Protocol(MOP) を用いて RDFS の意味論をオブジェクトベースで実装した．

2. RDF/S 意味論

2.1 RDF/S のメタクラス，クラス，インスタンス

RDF/S に関する W3C Recommendation ドキュメントに記載された RDF/S エンティティについて，rdf:type, rdfs:subClassOf, rdfs:subPropertyOf 関係を図示すると Figure 1 のようになる．ここで特徴的なことは rdfs:Class は自身のインスタンスであり，

連絡先：小出誠二，（株）ギャラクシーエクスプレス，港区浜松町 1-18-16, Fax03-5733-7190, koide@galaxy-express.co.jp

[1] http://wilbur-rdf.sourceforge.net/

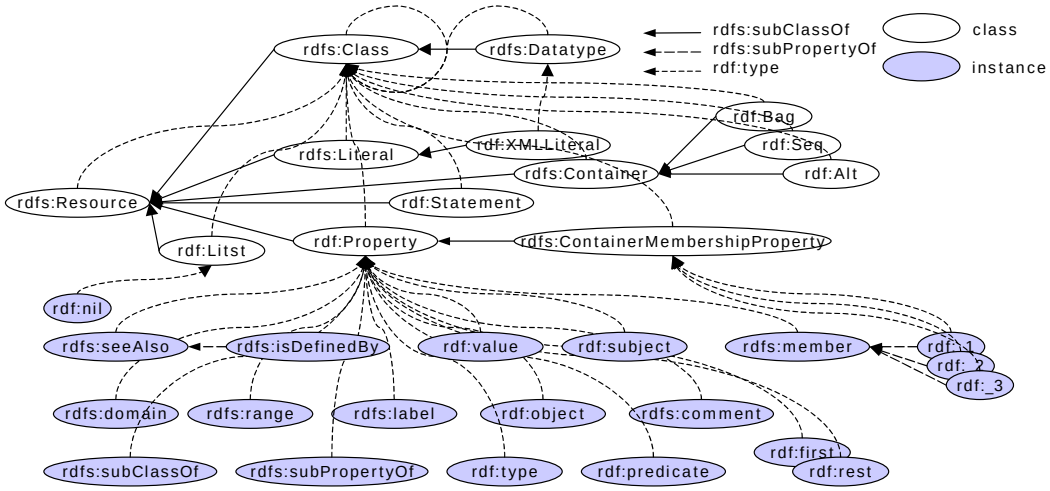


Figure 1. Hierarchy of RDF/S Resources and Properties

rdfs:Datatype は rdfs:Class の rdfs:subClassOf であると同時にインスタンスでもあることである. このようなメタモデリングの構造は一見わかりにくく, 一部に混乱を引き起こしているが[Pan 2003], このような自己定義は CLOS のようなリフレクティブなプログラミング言語ではあたりまえのこととされている[Kiczales 1992] [Paepcke 1993]. すなわち, リフレクティブ・プログラミングでは実行時にシステム定義を変更したり, プログラミング言語仕様の変更をユーザに開放したりすることを目的に, インスタンスとそれを制御するクラスの関係と同様に, システムクラスとそれを制御するメタクラスを設定し, このメタクラスのプログラム機能をユーザに開放している. 理論的にはこのリフレクティブなレイヤー階層は何段にもなり得るが, 実際にはメタクラスのクラスは自分自身として無限に階層が伸びていくことを終端させている. RDF/S もあきらかにこのリフレクティブ・プログラミングの成果を取り入れており, rdfs:Class は自分の型は自分自身であるとして, リフレクティブタワーが無限に伸びることを終端させている. CLOS における standard-class は(自身と standard-object も含め)すべてのクラスをインスタンスとするクラスであり, standard-object は(standard-class も含め)すべてのクラスの上位クラスとなっているが, rdfs:Class (すべてのリソースクラスのクラス)と rdfs:Resource (すべてのリソースクラスの上位クラス)の関係はそれと全く同一である.

CLOS を用いて Figure 1 に示されたクラスとプロパティを実装するにあたり, すべての RDF/S エンティティを CLOS オブジェクトまたはメタオブジェクトと捉え, RDF/S エンティティ間の rdf:type 関係を CLOS クラス・インスタンス関係に, rdfs:subClassOf 関係を CLOS 上位クラス・下位クラス関係に写像した. rdfs:Class と rdfs:Datatype は CLOS メタクラスに, そのほかの RDF/S クラスは CLOS クラスに, RDF/S プロパティは rdf:Property のインスタンスに写像される. そのような素直な写像により, 後で述べるように, リスプの機能を用いて rdf:type の包摂関係や rdfs:subClassOf 推移律関係の推論が自然に実現される. 以下に本システムにおける RDF/S 公理の確認例を示す.

```
gx(2): (typep rdfs:Class rdfs:Class) → t
gx(3): (typep rdfs:Class rdfs:Resource) → t
gx(4): (typep rdfs:Resource rdfs:Class) → t
gx(5): (typep rdfs:Datatype rdfs:Class) → t
gx(6): (typep rdfs:seeAlso rdf:Property) → t
```

ここで, typep はリスプネイティブの型判定述語である.

2.2 rdf:type 関係

(1) rdf:type の単一性前提

現在の RDF/S 仕様では, ある RDF/S エンティティについて rdf:type は一つとは限らない. たとえば, RDF グラフにおいて一つのエンティティから複数の rdf:type エッジがあってもよく, N-Triple ファイルにおいて, 一つの共通サブジェクトがプレディケイトに rdf:type を含む複数の行があってもよい. しかし, 本システムの現在の実現では, ある RDF/S エンティティについてその rdf:type はたった一つであるという前提を置いている. 実際に複数の rdf:type が RDF/S 記述において現れることはまれであり, 単一性を前提としてもあまり問題とならないであろう.

(2) 包摂関係推論機能

下記のような N-Triple 表記「Wine は PotableLiquid であり, Zinfandel は Wine であり, ElyseZinfandel は Zinfandel である」という知識があったとき(Figure 2 参照), 包摂関係推論(subsumption reasoning)とは「ElyseZinfandel は Wine であり, PotableLiquid でもある」というような推論を言う.

```
vin:Wine rdfs:subClassOf food:PotableLiquid .
vin:Zinfandel  rdfs:subClassOf  vin:Wine .
vin:ElyseZinfandel  rdf:type  vin:Zinfandel .
#<Subsumption, RDFS entailment rule rdfs9>
vin:ElyseZinfandel rdf:type vin:Wine .
vin:ElyseZinfandel rdf:type food:PotableLiquid .
```

すなわち, RDF/S においては, あるインスタンスはその rdf:type に定義されたクラスのインスタンスであるばかりでなく, そのクラスの rdf:subClassOf 関係にある上位クラスのインスタンスでもあるというように推論されなければならない(RDFS 伴意ルール rdfs9, すなわちこの推論機能は RDF 意味論[1]における伴意ルール rdfs9 に相当する, 以下同様). 本プロセッサにおいては, この包摂関係推論機能は rdf:type を CLOS クラス／インスタンス関係に, rdfs:subClassOf を CLOS スーパ／下位クラスに写像することにより, CLOS の機能により自然に実現される. たとえばこの場合, 以下のように推論結果を得ることができる.

```
gx(7): (typep vin:ElyseZinfandel vin:Wine)→ t
gx(9): (typep vin:ElyseZinfandel
food:PotableLiquid) → t
```

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

2.3 rdfs:subClassOf 関係

(1) 複数の rdfs:subClassOf

RDF グラフにおいて, あるノードから複数の rdfs:subClassOf エッジが出ていれば, それは複数の上位クラスを持つということである. 複数の上位クラスを許すということはあるインスタンスが複数の異なるクラスに所属することを許すということである. さいわい, CLOS クラスは複数の上位クラスを持つことができて, 前述の包摂関係推論があてはまるので, 複数の rdfs:subClassOf は CLOS において保証されている.

(2) rdfs:subClassOf の推移律推論

推移律推論(transitivity reasoning)とは, 以下に示すように, 「Wine は PotableLiquid の下位クラスであり, Zinfandel は Wine の下位クラス」であるならば, 「Zinfandel は PotableLiquid の下位クラスである」というような推論を言う.

```
vin:Wine rdfs:subClassOf  food:PotableLiquid .
vin:Zinfandel  rdfs:subClassOf  vin:Wine .
#<Transitivity, RDFS entailment rule rdfs11>
vin:Zinfandel  rdfs:subClassOf
food:PotableLiquid .
```

すなわち, RDF/S においてあるクラスの rdfs:subClassOf 関係にあるクラスについて, それがさらに三番目のクラスにも rdfs:subClassOf 関係にあるならば, 最初のクラスは三番目のクラスとも rdfs:subClassOf 関係にある(RDFS 伴意ルール rdfs11). この推移律はもちろん何重にも重ねられる.

本システムでは rdfs:subClassOf の推移律は, RDF/S クラスを CLOS クラスに写像することにより自然に実現される. たとえばこの場合, リスプネイティブな型に関する述語 subtypep を用いて以下に示すような結果を得ることができる.

```
gx(10): (subtypep vin:Zinfandel
food:PotableLiquid) → t
```

2.4 rdfs:subPropertyOf 関係

RDF/S プロパティの rdfs:domain や rdfs:range のグローバルな制約条件などプロパティの性質は, RDF/S プロパティ定義時にプロパティオブジェクトのスロット情報として保存されるが, プロパティの rdfs:subPropertyOf 関係は, この制約条件に関する包摂概念をもたらす. もしあるプロパティ P が別のプロパティ P' の下位プロパティならば, P で成立する N-Triple 表現を P'に入れ替えても成立する(RDFS 伴意ルール rdfs7). ここでは下位プロパティにおける記述が上位プロパティでも成り立つことに注意されたい. たとえば, 以下のように「ワインの色属性はワイン記述属性であり, ElyseZinfandel の色属性は赤」ということであれば, 「ElyseZinfandel は赤というワイン記述属性を持つ」ということが推論される.

```
vin:hasColor rdfs:subPropertyOf
vin:hasWineDescriptor .
vin:ElyseZinfandel vin:hasColor vin:Red .
#<RDFS entailment rule rdfs7>
vin:ElyseZinfandel vin:hasWineDescriptor vin:Red.
```

上記推論機能を手続き的に実装するのは難しく, 以下に示すように Prolog のようなクエリ言語により実行できるようにする予定である.

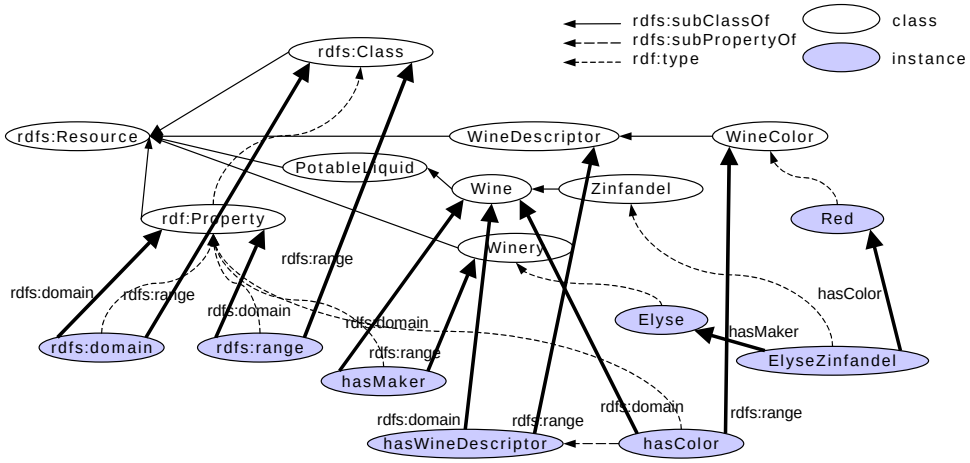
```
gx(11): (?- (rdfs:hasWineDescriptor
vin:ElyseZinfandel vin:Red)) → t
```

スーパプロパティが定義済みの制約を持つなら, その制約は下位プロパティに反映されなければならない. プロパティは通常ドメインとレンジを持つがその属性は下位プロパティに継承される(RDFS 伴意ルール ext3, ext4) . 以下は rdfs:subPropertyOf 関係によるプロパティのドメイン情報とレンジ情報の継承についての例である(Figure 2 参照).

```
vin:hasWineDescriptor  rdfs:domain  vin:Wine  .
vin:hasColor  rdfs:subPropertyOf
vin:hasWineDescriptor  .
#<Domain rule, Extensional entailment rule ext3>
vin:hasColor rdfs:domain vin:Wine  .

vin:hasWineDescriptor  rdfs:range
vin:WineDescriptor  .
vin:hasColor  rdfs:subPropertyOf
vin:hasWineDescriptor  .
#<Range rule, Extensional entailment rule ext4>
vin:hasColor  rdfs:range  vin:WineDescriptor  .
```

本システムでは, あるプロパティに定義されたドメイン情報やレンジ情報を rdfs:domain, rdfs:range アクセッサや domain-value, range-value 関数で取り出すことができるが, 当該プロパティのみならず, そのスーパプロパティに定義されたドメイン情報やレンジ情報を取り出すために, 以下に示すように, get-



rdfs:domain：class that is reachable from a start node of a property edge via rdf:type→rdfs:subClassOf
rdfs:range：class that is reachable from a target node of a property edge via rdf:type→rdfs:subClassOf

Figure 2. An RDF/S Example from Wine Ontology

domain, get-range 関数を用意した. これらの関数はあるプロパティとそのスーパプロパティに存在するドメインやレンジのうち, 最も特殊な値を, すなわち rdfs:subClassOf 関係で最も下位のものを取り出す. また, 上記 N-Triple 表記に対して以下のような, domainp および rangep 判別関数を用意した.

```
gx(12): (rdfs:domain vin:hasColor)
Error: The slot rdfs:domain is unbound in the
object
       #<rdf:Property vin:hasColor> of class
#<rdfs:Class rdf:Property>.
[condition type: unbound-slot]
gx(13): (domain-value vin:hasColor)
common-lisp:nil
gx(14): (get-domain vin:hasColor)
#<rdfs:Class vin:Wine>
gx(15): (get-range vin:hasColor)
#<rdfs:Class vin:WineDescriptor>
gx(16): (domainp vin:hasColor vin:Wine)
t
gx(17): (rangep vin:hasColor vin:WineDescriptor)
t
```

2.5 プロパティのドメインとレンジ

プロパティのドメインは N-Triple 表記におけるサブジェクトの型の制約を定義し, レンジはオブジェクトの型の制約を定義する. ワインオントロジの一部を RDF/S グラフ表現したものを Figure 2 に示した. 図に照らしてドメインとレンジを説明すれば, あるエッジの根のノードから rdf:type の階段を1段上がり, 上位クラスの方向に遡及していくと, そのエッジのドメイン定義にたどりつく. レンジについてはエッジの先から同様にトラバースしてレンジ定義にたどりつく.

あるプロパティに既に定義済みのドメイン・レンジ制約があって, そのプロパティをプレディケイトに持つ N-Triple が入力されたとき, 入力サブジェクトとオブジェクトが制約に違反しないか, チェックされなければならないし, もしそのサブジェクトやオブジェクトが未定義であれば, それは制約となるべきクラスのインスタンスとして定義されなければならない (RDFS 伴意ルール rdfs2, rdfs3). たとえば以下のような N-Triple 入力が行われたとき, もし vin:ElyseZinfandel や vin:Red が未定義ならば, システムによってそれらが自動的に制約を満足するように定義される.

```
vin:hasColor  rdfs:domain  vin:Wine  .
vin:hasColor  rdfs:range   vin:WineColor  .
vin:ElyseZinfandel  vin:hasColor  vin:Red  .
# vin:ElyseZinfandel is undefined so far.
#<RDFS entailment rule rdfs2>
vin:ElyseZinfandel  rdfs:type  vin:Wine  .
# vin:Red is undefined so far.
#<RDFS entailment rule rdfs3>
vin:Red  rdfs:type  vin:WineColor  .
```

また, このドメインとレンジの制約は, その上位クラスについても成立する. すなわち, 以下に示すように, あるプロパティのドメインあるいはレンジ制約が成り立てば, そのドメインあるいはレンジの上位クラスについても制約が成り立つ (RDFS 伴意ルール ext1, ext2).

```
vin:hasColor  rdfs:domain  vin:Wine  .
vin:hasColor  rdfs:range   vin:WineColor  .
vin:Wine  rdfs:subClassOf  food:PotableLiquid .
vin:WineColor rdfs:subClassOf
vin:WineDescriptor .
#<Extensional entailment rule ext1>
vin:hasColor  rdfs:domain  food:PotableLiquid  .
#<Extensional entailment rule ext2>
vin:hasColor  rdfs:range  vin:WineDescriptor  .
```

本システムでは, この伴意ルールをドメインとレンジの判別関数 domainp および rangep で実行するようにした. 実行例を以下に示す.

```
gx(18): (domainp vin:hasColor food:PotableLiquid)
t
gx(19): (rangep vin:hasColor vin:WineDescriptor)
t
```

3. 簡単な実行例

Figure 3 は旧 RDF/S ドキュメント[1]に掲載された簡単な RDF グラフ例である.

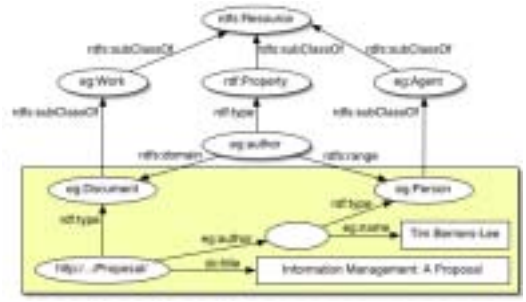


Figure 3. The Introductory Example

本システムのS式による N-Triple 定義マクロ defTriple による入力例を以下に示す. defTriple の文法を付録に記した. この場合エッジに着目して図から三つ組みを拾い出し入力する. ただし最初の二つは図示されていない. この二つの N-Triple 入力では rdfs:range プロパティのドメイン制約により eg:name と dc:title が rdf:Property であるように伴意され, 三番目以下の N-Triple 入力では rdfs:subClassOf プロパティのドメイン制約により各サブジェクトが rdfs:Class のインスタンスであるように伴意されている.

```
gx(5): (defTriple eg:name rdfs:range rdfs:Literal)
Warning: Entail in eg:name #<rdf:Property rdfs:range>
#<rdfs:Class rdfs:Literal>:
..... eg:name rdf:type rdf:Property.
#<rdf:Property eg:name>
gx(6): (defTriple dc:title rdfs:range rdfs:Literal)
Warning: Entail in dc:title #<rdf:Property rdfs:range>
#<rdfs:Class rdfs:Literal>:
..... dc:title rdf:type rdf:Property.
#<rdf:Property dc:title>
gx(7): (defTriple eg:Person rdfs:subClassOf eg:Agent)
Warning: Entail in eg:Person #<rdf:Property rdfs:subClassOf>
eg:Agent:
..... eg:Person rdf:type rdfs:Class.
Warning: Entail in #<rdfs:Class eg:Person> #<rdf:Property
rdfs:subClassOf> eg:Agent:
..... eg:Agent rdf:type rdfs:Class.
#<rdfs:Class eg:Person>
gx(8): (defTriple eg:Document rdfs:subClassOf eg:Work)
Warning: Entail in eg:Document #<rdf:Property rdfs:subClassOf>
eg:Work:
..... eg:Document rdf:type rdfs:Class.
Warning: Entail in #<rdfs:Class eg:Document> #<rdf:Property
rdfs:subClassOf> eg:Work:
..... eg:Work rdf:type rdfs:Class.
#<rdfs:Class eg:Document>
gx(9): (defTriple eg:author rdf:type rdf:Property)
#<rdf:Property eg:author>
gx(10): (defTriple eg:author rdfs:domain eg:Document)
#<rdf:Property eg:author>
gx(11): (defTriple eg:author rdfs:range eg:Person)
#<rdf:Property eg:author>
gx(12): (defTriple _:a rdf:type eg:Person)
#<eg:Person _:a>
gx(13): (defTriple _:a eg:name "Tim Berners-Lee")
#<eg:Person _:a>
gx(14): (defTriple eg:Proposal rdf:type eg:Document)
#<eg:Document eg:Proposal>
gx(15): (defTriple eg:Proposal eg:author _:a)
#<eg:Document eg:Proposal>
```

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

```
gx(16): (defTriple eg:Proposal dc:title "Information Management:
A Proposal")
#<eg:Document eg:Proposal>
```

同じ RDF グラフを本システムのS式によるオブジェクト表現として入力すると以下のようになる．この場合，ノードに着目して図から情報を拾い出し，defResource あるいは defIndividual マクロで入力する．Figure 3 中のブランクノードがここでは eg:author スロットのフィラーの無名オブジェクト記述として自然に表現されていることに注意されたい．

```
gx(4): (defProperty eg:name
  (rdfs:domain eg:Person)
  (rdfs:range rdfs:Literal))
Warning: Entail: eg:Person rdf:type rdfs:Class.
eg:name
gx(5): (defProperty dc:title
  (rdfs:domain eg:Document)
  (rdfs:range rdfs:Literal))
Warning: Entail: eg:Document rdf:type rdfs:Class.
dc:title
gx(6): (defProperty eg:author
  (rdfs:domain eg:Document)
  (rdfs:range eg:Person))
eg:author
gx(7): (defResource eg:Person
  (rdfs:subClassOf eg:Agent))
Warning: Entail: eg:Agent rdf:type rdfs:Class.
eg:Person
gx(8): (defResource eg:Document
  (rdfs:subClassOf eg:Work))
Warning: Entail: eg:Work rdf:type rdfs:Class.
eg:Document
gx(9): (defIndividual eg:Proposal
  (rdf:type eg:Document)
  (eg:author (eg:Person (eg:name "Tim Berners-Lee")))
  (dc:title "Information Management: A Proposal")
  (rdf:about "http:/…/Proposal/"))
eg:Proposal
```

このような入力のあと，ユーザは任意の情報を取り出すことができる．Figure 1 に掲げられた RDF/S エンティティもユーザ定義のエンティティも，それは CLOS オブジェクトであり，任意のオブジェクトのプロパティ値を取り出すのに，CLOS の slot-value を利用することができる．一方，Figure 1 に掲げられた RDF/S プロパティについては，そのプロパティ名をメソッド名とするアクセッサを用意した．そのほかに，プロパティのパスを与えてパスに沿った値の取り出し関数を用意した．これは RDF/S グラフで言えば与えられたパスに従ってトラバースする行為に相当する．

```
gx(10): (print-form eg:Proposal)
(eg:Document eg:Proposal (rdf:about "http:/…/Proposal/")
(eg:author (eg:Person (eg:name "Tim Berners-Lee")))
 (dc:title "Information Management: A Proposal"))
gx(11): (<- eg:Proposal 'eg:author 'eg:name)
"Tim Berners-Lee"
gx(12): (<- eg:Proposal 'dc:title)
"Information Management: A Proposal"
gx(13): (typep (<- eg:Proposal 'eg:author) eg:Agent)
t
```

4. 処理系の実装

4.1 Entailment による RDF/S エンティティの前方参照

RDF/XML あるいは N-Triple ファイルは前方参照可能であることが要求されている．幸いなことに，RDF/S は知識の単調性(monotonicity)が前提となっている．すなわち，知識は増加方向に一方的に追加され，詳細化されることはあってもそれまでの知識が新しい知識で否定されることはない．言い換えれば，同一ファイル内で前方参照があったとしても，最初の前方参照時に最も汎用の知識を仮定してオブジェクト実体を実現しておき(たとえば，クラスならそのメタクラスを rdfs:Class とし rdfs:Resource を上位クラスとする，インスタンスなら rdfs:Resouce のインスタンスとし，プロパティなら rdf:Property のインスタンスとする)，後方で実際の定義が現れたときにその定義に従った詳細化を行えばよい．もし最初に詳細な知識が入力され，その後抽象的な知識がたとえ入力されても，関連する rdfs:subClassOf 関係にあればそれは無視し，独立した rdfs:subClassOf 関係にあればその情報を追加すればよい．

前方参照を前提として未定義なエンティティが引用されたときどのような定義を行えばよいかという問題解決の助けとなるのが，RDF/S の伴意ルールである．その多くは本実装において自明のことであったり，CLOS によって自然に実現されたりするが，いくつかは特別にプログラムされなければならなかった．たとえば，RDF 伴意ルール rdf1 ではプロパティとして引用されたエンティティはそれが未定義であれば rdf:Property のインスタンスとして定義する．RDFS 伴意ルール rdfs2 と rdfs3 では 2.5 節で述べたように rdfs:domain と rdfs:range 制約に従って定義する．唯一問題となるのが，RDFS 伴意ルール rdfs4a と rdfs4b である．これはプロパティではないすべての引用は rdfs:Resource のインスタンスであるという伴意であるが，ここで問題となるのが，実際にそれを rdfs:Class のインスタンス(すなわちクラス)とするべきなのか rdfs:Resource のインスタンス(すなわちインスタンス)とするべきなのかという問題である(どちらの場合でも伴意は満足される)．その場合はインスタンスとして仮に実現しておき，その後それがクラスと判明した時点で change-class することとした．

4.2 rdfs:Class の自己定義

2.1 節で rdfs:Class に見られるような自己定義がリフレクティブ･プログラミングでは普通のことであることを述べた．しかし，実装上はこれは困難な問題を引き起こす．通常は母型となる型が先にあって，その情報を用いてインスタンスが定義されるが，rdfs:Class においてはこれから定義しようとするときに自分は存在していない．この問題は CLOS の実装などでも同様であり(standard-class について)ブートストラッピングの手法が研究開発されている[Kiczales 1992] [Paepcke 1993]．もし RDF/S 処理システムをスクラッチで開発する場合には，リフレクティブ言語に関する過去の経験に学んで RDF/S 処理システムをプログラミングすることになるが，本システムでは CLOS システム上に実装したため，既存のリフレクティブシステム上にどのように RDF/S リフレクティブを実装するかという問題になる．そこで問題となるのが CLOS システム上の rdfs:Class の取り扱いである．CLOS では standard-class の下位クラスとして定義されるクラスがメタクラスとなるが，現在の実装言語である Allegro Common Lisp では standard-class 以外の場所で自己参照的に型サーキュレーションを行おうとするとエラーとなる．そこでやむを得ず，rdfs:Class の型は内部の実装としては standard-class とし，外部から型判定文やメソッドで見たときには RDF/S の仕様通りになるようにした．ほかにも実装上の細かい工夫があったが，詳細は省略し結果として既存 CLOS クラスと RDF/S 実装との関係を Figure 4 に示す．さらに Common Lisp パッケージからの subtypep, type-of, typep の輸入を隠蔽し，rdfs:Class と rdfs:Literal のみについて型判定が RDF/S 仕様に従うような gx:subtypep，gx:type-of，gx:typep を導入した．

4.3 プロパティの rdfs:subPropertyOf

通常のオブジェクトシステムでは，クラスに上位／下位関係があるがインスタンスにはない．ところが，RDF/S ではプロパティ(rdf:Property インスタンス)が rdfs:subPropertyOf を有する．この機能によってたとえば rdfs:member の下位プロパティである rdf:_1 や rdf:_2 のように，実行時に動的にプロパティが生成されるような場合に，あらかじめそのようなプロパティに属性を定義するようなことが可能になる．[Wielemaker 2003]はこの機能を利

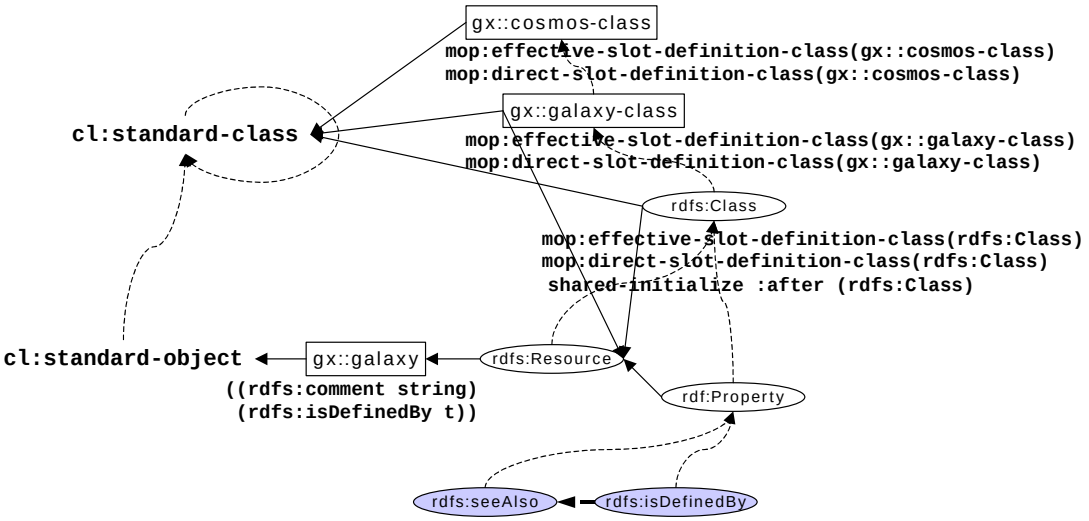


Figure 4. An Implement of RDF/S on CLOS

用して, WordNet における wns:hyponymOf を rdfs:subClassOf の下位プロパティとして定義した.

rdf:Property オブジェクトは rdfs:domain, rdfs:range スロットのほかに rdfs:subPropertyOf, rdfs:superPropertyOf スロットを有し, rdfs:subClassOf 関係と同様に上下関係の管理が行われるようにし, get-domain 関数や get-range 関数はこの情報を使ってスーパプロパティのドメイン・レンジ情報を取り出すようにした.

4.4 rdfs:domain と rdfs:range の実現

N-Triple の入力や RDF/S オブジェクトの入力においては, プロパティのドメイン・レンジ情報により, サブジェクトとオブジェクトのチェックやオブジェクト生成が行われる. 第3章で示したS式オブジェクト表現による入力では, そのオブジェクト生成時にイニシャライズメソッドによってドメイン・レンジのチェックやオブジェクト生成を行うようにした. ここではメタオブジェクト・プログラミングの機能が最大限活用された. 詳細は省略するが, 実装の結果の一部は Figure 4 にも現れている. 通常の CLOS スロットと RDF/S プロパティのスロットを識別するために standard-direct-slot-definition の下位クラスに gx::Property-direct-slot-definition を, standard-effective-slot-definition の下位クラスに gx::Property-effective-slot-definition を設け, これらのクラスに対する特殊なメソッドをいくつか作成した.

4.5 クラス変数とインスタンス変数

RDF/S のプロパティとその値を, 本システムでは CLOS のスロットとして実現した. RDF/S インスタンスに所属するプロパティとその値をインスタンス変数, RDF/S クラスに所属するプロパティとその値をクラス変数と呼ぶとき, クラス変数を CLOS で実現するには, 一見スロットオプションである:allocation オプションに:class を指定すればよいように思われるが, それは誤りである. CLOS におけるそれは, そのクラスに所属してインスタンスで共有される変数を意味する. RDF/S におけるクラス変数は共有変数ではなく, そのクラスに固有の局所変数である. クラスに局所変数を用意するにはそのメタクラスにスロット定義がなければならない.

前述の defResource, defIndividual, defProperty における定義では, 与えられるプロパティとその値はそのクラスあるいはインスタンスの局所変数を指定する. そのため, システムはメタクラスまたはクラスに該当のスロット定義があるかどうか調べ, なければその定義を自動的に追加して自分自身のスロットも設定するようにした.

5. RDF リーダ

これまではリスプの特徴を生かして, リスプトップレベルで対話的に RDF/S エンティティの関連を調べたり取り出したり, あるいは入力したりする機能について述べた. defTriple や defResource のような形式の記述をファイルからロードすることもできるが, N-Triple フォーマットや RDF/XML フォーマットのファイルからもロードできるように, それぞれのパーザを開発した.

5.1 RDF/S エンティティのリスプ実現

ティム・バーナーズ＝リーによるセマンティックウェブ・レイヤーケーキでは, その最下層はグローバルな文字識別のための Unicode と, リソースの唯一性をグローバルに保証するための URI(Uniform Resource Indicator)になっている. 幸いにして市販のリスプ処理系においてすでにその内部文字表現は Unicode になっており, URI 処理のためのライブラリも整理され, これらを有効に用いることができる.

名前空間はセマンティックウェブのグラウンディングにとって非常に重要な機能である. すなわち, グローバルな URI の唯一性の特徴を利用して, XML タグのグローバルな唯一性を保証している. 一方, リスプのシンボルに関する唯一性のための仕組みとしてパッケージがある. 幸いなことに, リスプシンボルをエキスポートすると見かけ上 QName のように「gx:foo」などと印刷されて見やすい. そこで URI からリスプシンボルへ変換する機能を開発した. RDF/XML 記述においては xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"など名前空間が入力されたときに必要なら該当パッケージを生成して URI と関連付け, フラグメント付き URI 入力のたびに該当パッケージ内にて正しくリスプシンボルをインターンかつエキスポートするようにした. リスプは記号処理に向いた言語であるが, URI とリスプシンボルの1対1対応をとることにより, リスプで URI の知的処理を行う. ただし, URI フラグメントのない URI については URI のままとしている.

本システム中でリテラルや XML スキーマデータ以外のすべての RDF/S エンティティについて, リソース URI はその QName に相当するリスプシンボルに変換され, そのシンボルにリソースのオブジェクト実体がバインドされる.

5.2 XML スキーマ･データタイプ

XML データ表現において XML のスキーマ機能は一部のソフトウェア技術者にとっては重要かもしれないが, 代表的なデータ種類さえそろえばセマンティックウェブ開発にとって実用上それほど決定的ではない. リスプでは任意の型を定義することができる. 今回とりあえずの処置として, xsd:nonPositiveInteger や xsd:nonNegativeInteger などの基礎的な XML スキーマについてリスプの型宣言をし, RDF/XML 入力において整数, 数, 文字列など対応するリスプのプリミティブデータに変換すると同時に型チェックをほどこすようにした. プリミティブなリスプデータは, さらに NonPositiveInteger や NonNegativeInteger など該当のクラスオブジェクトに変更される. これらのオブジェクトの印刷関数もカスタマイズされ, 1^^nonNegativeInteger などと印刷される.

5.3 RDF/XML パーザ

XML は汎用を目的とした, 機械可読な記法であるが, その本質は開始タグと終了タグをセットとするエレメントの入れ子構造により, ものごとを表現することである. そのテキスト表現がリスプのS式と類似であることから, リスプで XML 処理系を開発することは比較的容易である. リスプによる XML パーザには, CLOCC[1]のライブラリ CLLIB 中の xml.lisp, CL-HTTP の一環としての CL-XML[2], Franz 社の提供する XML パーザ[3]など, このほかにもいくつかある. しかしいずれも汎用の XML 処理を目的としており, DTD 処理や妥当性検証の実現が課題となっている. 我々は RDF/XML 表記の XML 文書の処理を目的としており, その点では DTD 処理や妥当性検証は RDF/XML 表記に限ったものでよく, RDF/XML 専用パーザであれば処理系に埋め込むこともできる.

XML パーザとしてここでは XML 一般ではなく, RDF シンタックスに特化した RDF/XML パーザを考えた. XML エレメントの入れ子構造を DOM のようなノードの構成する木構造ではなく, RDF 構造体やその内部に格納される Description 構造体の入れ子構造として内部表現した. リスプ構造体はその印刷関数をユーザが自由にカスタマイズすることができる. RDF 構造体や Description 構造体の印刷関数をカスタマイズすることにより, XML シリアライズ表現として構造体を印刷させ, RDF/XML 入力結果を, 入力したとおりの XML 表記とすることを可能とした. リスプのリードマクロ機能を用いれば, 山形括弧をリーダマクロ文字として, 山形括弧を読み込むと同時にその開始タグからそれに対応する終了タグ読み込みまでを XML パーザで処理して, 終了タグ読み込み完了とともにリスプのリーダに戻るというようなことも可能となる. すなわち, リスプのトップレベルにおいて丸型括弧でS式を入力し, 山形括弧で XML 記述を入力することも可能となる.

当初は Franz 社の提供する XML パーザの利用を考え, 我々の目的には以下のような不具合があったため, パーザの一部を改造し改良をほどこした.

- 汎用の XML パーザを目的として, スペースや改行などの空白文字も厳密にその情報を保持するため, パーズされた結果が見にくく煩雑である. 空白文字は削除するようにした.
- パーズ結果を LXML(Lisp XML)フォーマットで表記するため, FIPA ACL 仕様にあるような S 式と RDF/XML 表記混在の記述ができない. パーズ結果を前述のように RDF 構造体や Description 構造体で表現し, 印刷関数をカスタマイズすることにより, S 式と RDF/XML 表記の混在を可能とした.
- 名前空間の取扱いが不十分なので, RDF/XML に沿った改良を行った.

ところが, このパーザはファイル内容のすべてをシステム内部に取り込んでから処理を行うため, 処理効率が悪く, WordNet の RDF ファイル(10MB, 15MB のサイズ)を処理することができなかった. そこで, スクラッチであらたに RDF/XML パーザを開発した. 高速化の手法として, コンティニュエーションによるコルーチンの技法を用いた. このパーザは 15MB の RDF ファイルを数十秒で処理することができる. 以下に開発したシステムによる実行例を示す. 一見 RDF/XML 表記そのものを印刷したように見えるが実際はそうではなく, RDF/XML 記述のファイル内容をパーザが読み込み, パーズした結果として構造体を生成し, リスプトップレベルにその構造体を返しているが, リスプ処理系がその構造体を印刷するときに, あたかも RDF/XML 記述であるかのごとく印刷している. 以下の例ではパーザの結果として, リスプの括弧の中に XMLDecl 構造体と RDF 構造体の二つが返されている.

開発されたパーザを addObject 関数と組み合わせることで, defResource や defInstance と同等の結果を得ることができる.

```
cg-user(10): (with-open-file (p "example07.rdf") (rdf::parse->rdf
p))
(<?xml version="1.0" ?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ex="http://example.org/stuff/1.0/">
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-
grammar"
dc:title="RDF/XML Syntax Specification (Revised)">
  <ex:editor>
    <rdf:Description ex:fullName="Dave Beckett">
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/" />
    </rdf:Description>
  </ex:editor>
</rdf:Description>
</rdf:RDF>)
```

5.4 N-Triple パーザ

N-Triple フォーマットファイルを読み込んで解釈できる N-Triple パーザを開発した. パーザと addTriple 関数を組み合わせることで, defTriple と同等の結果を得ることができる.

6. OWL への展望

現在の RDF/S の拡張として OWL 処理系を開発する. Figure 5 に OWL のクラス階層を示す. オントロジプロパティとアノテーションプロパティは省略している. RDF/S にない OWL の特徴は, owl:Restriction と owl:onProperty を用いてオブジェクトのプロパティについてそのオブジェクトに特有の局所的な制約を記述することができることである. OWL におけるS式のオブジェクト表現については今後検討するが, たとえばワインオントロジ[4]の vin:Wine について次のような表記が考えられる.

```
(defResource Wine (rdf:type owl:Class)
  (rdfs:subClassOf food:PotableLiquid)
  (hasMaker owl:restriction
```

[1] http://clocc.sourceforge.net/
[2] http://www.cl-xml.org/
[3] http://opensource.franz.com/xmlutils/index.html
[4] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf

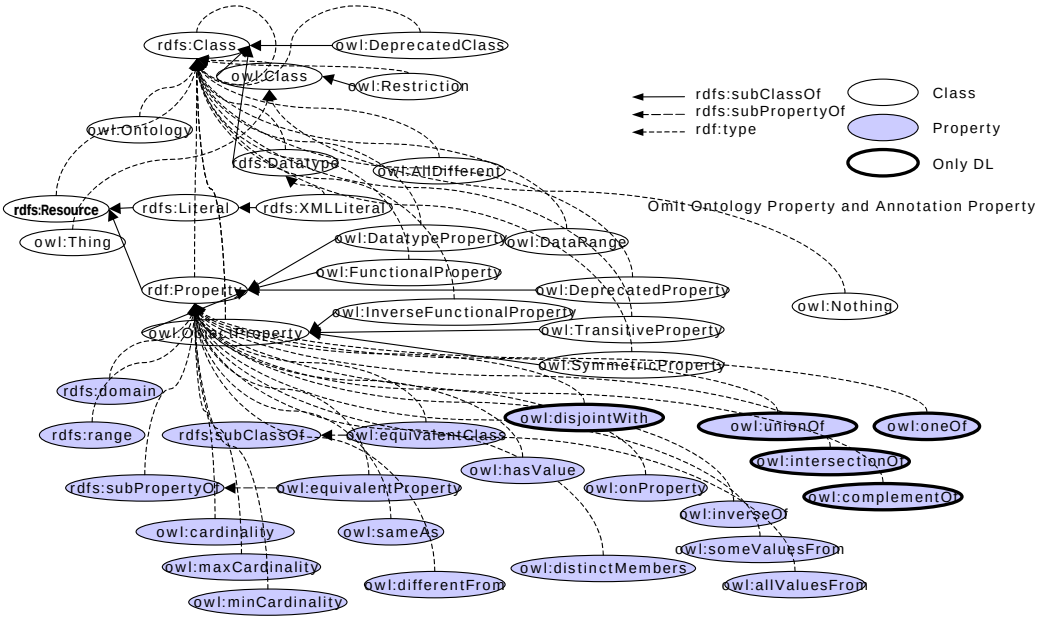


Figure 5. Hierarchy of OWL DL and OWL Lite Classes and Properties

```
          owl:cardinality 1
          owl:allValuesFrom Winery)
  (madeFromGrape owl:restriction
                 owl:minCardinality 1)
  (hasSugar owl:restriction owl:cardinality 1)
  (hasFlavor owl:restriction owl:cardinality 1)
  (hasBody owl:restriction owl:cardinality 1)
  (hasColor owl:restriction owl:cardinality 1)
  (locatedIn owl:restriction
             owl:someValuesFrom Region)
  (hasColor (en "wine") (fr "vin")))
```

ここで owl:restriction フラグは Wine のインスタンスはこのプロパティを有するがその値に対する局所制約があることを示し, owl:cardinality は値のオカレンス数（1は1個の値のみ持つ）の制約, owl:minCardinality はそれ以上の数のオカレンスである制約を表す. owl:allValuesFrom はこの場合, hasMaker プロパティの値はすべて Winery のインスタンスでなければならないことを意味し, owl:someValuesFrom はこの場合, locatedIn プロパティの値は少なくとも一つは Region のインスタンスでなければならないことを意味している. これらの制約は異なるオブジェクトではまた異なるように定義できなければならない. このようなプロパティの局所制約は, オブジェクトのクラス定義内にこれらの情報を保存するようにし, インスタンス生成時に CLOS スロットのイニシャライズメソッドにおいてこれらの制約を満足させるようにすることで実装できると考える.

owl:TransitiveProperty は推移律を持つプロパティを定義するためのプロパティクラスである. owl:inverseOf はそのプロパティが指定プロパティの逆関係にあることを表す. CLOS のメタオブジェクト・プロトコルを用いてこれらの機能を実装する.

OWL では, クラス定義にインスタンス列挙（owl:oneOf）, クラス定義の共通集合（owl:intersectionOf, AND に相当）, 和集合（owl:unionOf, OR に相当）, 補集合（owl:complementOf, NOT に相当）を用いることができる. CLOS オブジェクトはリスプの型システムに統合され, リスプには型演算機能がある. したがって, これらの集合演算は CLOS クラスの演算として実行できる可能性があるが, この集合演算の要素がプロパティの制約の場合もあり, 詳細な検討が必要である.

7. おわりに

RDF/S の公理を実現し, 伴意ルールを実装した RDF/S プロセッサを, リスプオブジェクトシステム CLOS を用いて開発した. ユーザは本システムを用いることで, RDF/S の意味論に従ったアプリケーションを開発できる.

8. 謝辞

本報告は, 文科省ITプログラム「ITを活用した大規模システムの運用支援システムの構築」の一部として実施されたものである. 本プロジェクト実施では大須賀節雄東京大学名誉教授に技術評価委員長をお願いし, オントロジ構築に関して大阪大学溝口理一郎教授にご協力戴いている. 記して感謝の意を表する.

参考文献

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

[小出 2002] 小出ほか,「ITを活用した大規模システムの運用支援システム」, SICE システムインテグレーション部門講演会, pp.399-400, 2002.

[Koide 2003] Koide, S., et al., “Operation-Support System for Large-Scale System Using Information Technology”, Proceedings of 5th Int. Conf. Enterprise Information Systems (ICEIS2003), Vol.4, pp.430-437, ESEO, Anger, 2003.

[Lassila 2002] Lassila, O., Taking the RDF Model Theory Out for a Spin, ISWC2002, pp.307-317, 2002.

[Paepcke 1993] Paepke, A. (ed.), Object-Oriented Programming – The CLOS Perspective, MIT Press, 1993.

[Pan 2003] Pan, J.Z. and I. Horrocks, RDFS(FA) and RDF MT: Two Semantics for RDFS, ISWC2003, pp.30-40, 2003.

[Wielemaker 2003] Wielmaker, J., Guus Schreiber, and B. Weilinga, Prolog-Based Infrastructure for RDF: Scalability and Performance, ISWC2003, pp.644-658, 2003.

付録 1.1 N-Triple の入力

RDF/S モデル論では RDF グラフが理論的基礎となっている. これをそのままテキスト表現したものが N-Triple フォーマットである. N-Triple フォーマットのファイルの読み込み機能とは別に, リスプのS式として N-Triple を読み込ませるために, defTriple マクロを用いる.

```
(defTriple subject  predicate  object )

subject  ::=  シンボル
predicate  ::=  シンボル
object  ::=  シンボル
```

この defTriple マクロは即座に addTriple メソッドを呼び出す. addTriple はサブジェクト, プレディケイト, オブジェクトそれぞれが, シンボルであったり, 過去に定義済みのオブジェクトであったりしてよく, いずれも前方参照可能であり, 入力される度にその知識を単調増加的にシステムに追加していく. AddTriple は関数であるためシンボル入力にクォートが必要であるが, この defTriple を用いればシンボル入力にクォートを用いなくてすむ. ただし, あらかじめ該当パッケージ中でエキスポートシンボルとしてインターンされていなければならない.

付録 1.2 RDF/S クラス定義

RDF/XML ファイルの読み込みとは別に, リスプから RDF/S クラスを定義するのに便利なように, 以下のように defResource マクロを用意した.

```
(defResource  ID  slot-specifier* )

ID  ::=  シンボル
slot-specifier  ::=  (rdf:about URI )  |
                     (rdf:type class-name )  |
                     (rdfs:label label+ )  |
           (rdfs:subClassOf superclass-name* )  |
                     ( role filler-form )
class-name  ::=  シンボル
label  ::=  simple-label  |  label-with-lang
label-with-lang  ::=  (lang simple-label)
simple-label  ::=  文字列またはシンボル
lang  ::=  xml-langにおける言語に相当するシンボル
superclass-name  ::=  シンボル
role  ::=  シンボル
filler-form  ::=  文字列  |  数  |  シンボル  |
                  obj-sexpr
obj-sexpr  ::=
           (class-name  ID?  slot-specifier* )
```

ここでアスタリスクは 0 個以上の繰返しを表し, クエスチョンは 1 個または 0 個の存在を表している. ロールにはプロパティをシンボルとして指定する. フィラーにシンボルを用いた場合は, それは QName に同じものである. rdfs:subClassOf スロットを省略した場合には, rdfs:Resource がデフォールト値となる. また, 通常はスロット指定子に rdf:type スロットを指定しないが (その場合は rdfs:Class がデフォールト値), あえて指定するときは rdfs:Class あるいは rdfs:Datatype あるいはその下位クラスのクラス名をフィラーに指定することができる.

フィラーフォームには, リスプ文字列あるいはリスプ数 (これらは rdfs:Literal に相当), あるいは RDF/S エンティティに相当するシンボルのほか, RDF/S リソースを定義するリスト形式のフィラーフォームを指定できる. フィラーフォームの定義は再帰的であることに注意されたい. すなわち, ユーザは何重にも入れ子の形でフィラーの場所に RDF/S リソースを定義することができる. その場合, フィラーフォームの評価は入れ子の一番内側から外側に向かって行われる. defResource マクロの直下では, ID は必須であるが, フィラーフォームの中では ID を省略でき, その場合に定義された RDF/S リソースはアノニマスといわれる (RDF/S グラフのブランクノードに対応). 現在の仕様では, たとえスロット構成が同一の RDF/S リソースであっても, それがアノニマスとして別々の defResource マクロ中に定義されれば, 異なる RDF/S リソースとなる (エッジの先を共有する異なる複数のブランクノード). アノニマスでなければ, もちろん同一の RDF/S リソースとなる. なお, defResource, defIndividual, defPropety では前方参照が可能である.

付録 1.3 RDF/S インスタンス定義

リスプから RDF/S インスタンスを定義するのに便利なように, 以下のように defIndividual マクロを用意した.

```
(defIndividual  ID  slot-specifier* )

ID  ::=  シンボル
slot-specifier  ::=  (rdf:about URI )  |
                     (rdf:type class-name )  |
                     (rdfs:label label+ )  |
                     ( role filler-form )
```

defIndividual マクロでは rdfs:subClassOf スロットがあってはならない. 現在の版ではタイプ指定は1個だけである.

付録 1.4 RDF/S プロパティ定義

リスプから RDF/S プロパティを定義するのに便利なように, 以下のように defProperty マクロを用意した.

```
(defProperty  property-name  slot-specifier* )

property-name  ::=  シンボル
slot-specifier  ::=
      ( rdfs:subPropertyOf superproperty* ) |
      ( role filler-form )
superproperty  ::=  シンボル
```

ここで, rdfs:subPropertyOf スロットを省略した場合には, それは未定義 (リスプオブジェクトスロットが unbound) である. また, 通常はスロット指定子に rdf:type スロットを指定しないが (その場合は rdf:Property がデフォールト値), あえて指定するときは rdf:Property あるいは rdfs:ContainerMembershipProperty あるいはそれらの下位クラスのプロパティ名をフィラーに指定することができる.