1D4-03

MOP3: セマンティックウェブプロセッサ

MOP3: A Semantic Web Processor

小出 誠二 Seiji Koide
北村 幸雄 Yukio Kitamura

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

Memory Organization Package (MOP) is a case memory that developed by Schank *et al*. The MOP system has been developed on top of the Common Lisp Object System (CLOS) using the Meta-Object Protocol (MOP). Then it has been expanded into a Semantic Web processor. While the MOP may be conceived of as a kind of frame system, it can be deemed as a classifier or similarity learner differently than a frame system. Thus, extending the MOP to the Semantic Webs deserves consideration for application of a memory system of agents that crawl on the Semantic Webs. In this paper, we show what the MOP is, how it is expanded into the Semantic Webs, and how it is appropriate to represent knowledge. This paper is prepared with the contract under the Japanese IT Program of the Ministry of Education, Culture, Sports, and Technology.

1. はじめに

セマンティックウェブを扱うためには，RDF (Resource Description Framework)，RDFS (RDF Schema)，OWL (Web Ontology Language)，DAML-S (Semantic Markup Language for Web Services) などの表記を解釈処理するためのプロセッサが必要となる．MOP (Memory Organization Package)[Riesbeck 1989]はシャンクらによって開発された事例ベース推論 (CBR, Case-Based Reasoning) のための記憶機構であるが，MOP では MOP 表記によるオントロジを処理することができるので，これらセマンティックウェブ用オントロジ処理のためのプロセッサとして MOP を拡張することは考慮に値する．

MOP はフレームシステムの一種と見なすことができるが，それればかりでなくオントロジ構築に適した，制約と類似に関する推論機能を有している．すなわち，MOP で構成されたオントロジの上位概念のスロット集合は，それが包摂する下位概念の保有するスロット集合の制約となる．また，同一の事例はシステム中で同一のエンティティとなり，類似事例は分類木の中で隣り合う．事例ベース記憶である MOP 機構は，分類器，学習器でもあり，セマンティックウェブにおいて表された情報を収集し記憶するエージェントの記憶機構として有利な特徴を備えている．

本報告では，シャンクらの MOP を CLOS(Common Lisp Object System)[CLtL2 1990][Paepcke 1993]上で開発し，それをさらに RDF/RDFS プロセッサとして拡張した結果について述べる．なお，本研究開発は，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」において行われたものである．

2. MOP (Memory Organization Package)

事例ベース推論には様々な形態があるが，Mop システムはフレームシステムの一種と見なすことができる．すなわち，mop エンティティには抽象 mop と具体 mop があり，抽象 mop には抽象・特殊関係があって，最上位の「Root」mop から末端の具体 mop まで階層構造を成す．また mop は属性とその値のペア（スロット）を有して上位 mop からスロットを継承することができる．しかし，Mop システムには単なるフレームシステム以上の機能がある．すなわち，ある事例に関わる属性とその属性値の束が入力されたとき，システムはそれらをスロットとする新しい mop を生成し，それを mop の階層構造の適切な位置にインストールしようとする．ここで「適切な」とは，その属性値は生成された mop が含まれる階層構造の上下関係に整合的であるという意味である．もっと正確に言えば，より特殊な mop や具体 mop のスロットにある属性値は，その上位にある抽象 mop の該当する属性値よりも特殊である．ここで，抽象 mop の属性値は特殊 mop あるいは具体 mop の該当する属性値に対する制約となる．

この MOP の属性値制約機能は，人間の経験からの知識獲得の観点から見て，非常に興味深い．言語学習に関連して，なぜ子供がほんのわずかな正例のみから効率よく言語を獲得することができるかについての議論において，それは子供が生得的なオントロジと学習バイアスを持っているからだという主張がされている[今井 1997][今井 2000][Pinker 1994]．すなわち，乳児は生まれながらに生物，非生物，動作主体(agent)，動作主体から影響を受けるもの(affected theme)について区別することができ，できごと(event)，動き(motion)，方向性(direction)などを認識することができる．これらは最上位のオントロジを構成するが，加えて乳幼児が言葉とその言葉が指す対象に出会ったときに，言葉と概念を結びつけるための各種のバイアスがある．乳幼児はこれらの上位オントロジを制約として，バイアスを利用することで効率よく言葉と概念獲得を行っているらしい．チョムスキ革命以来，どのようにして幼児が母語の文法を獲得するかについても議論があるが，ピンカーは動詞の句構造と意味論的なプリミティブ単語との関係において生得的な連関規則があり，それが統語ルールの獲得を可能にする「ブートストラップ」モデルを主張している[今井 1997][橋田 1999]．

一方，意味論における内包と外延の関係から見ても，MOP の属性値制約機能は興味深い．外延とは，言ってみれば，指示対象のある集合を表現するものであり，その指示対象の種々の集合は現実世界における事物を区別し概念化したものである．この概念化された集合／部分集合の関係は，集合の包摂関係に従って木構造に表現され，タキソノミと呼ばれる．それでは，何がその概念を区別するのか．それが内包すなわち指示対象に関する属性である．人は素朴理論によって飛ぶ動物を鳥と認知する．すなわち，内包が外延を決定する．事例を属性値の集合として与えたとき MOP の行うことは，この内包と外延に関する意味論の示すところとよく似ている．

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，Fax03-5733-7190，koide@galaxy-express.co.jp

3. メタオブジェクトプロトコルによる MOP 実装

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]を開始するにあたって, CLOS を用いて Mop システムを開発した. CLOS ではクラス継承関係やスロット値の継承においてキャッシングが行われているので, 開発にあたっては高速化を期待して, 抽象 mop を CLOS のクラスに, 具体 mop を CLOS の実現体に相当させ, mop スロットをオブジェクトスロットに対応させた. しかし, そのような素直な実装を行うにあたり, 大きな問題点があった. それは, 具体 mop は複数の抽象 mop にリンクされ得るが, CLOS 実現体は一つのクラスの実現体であり, しかも具体 mop のオブジェクト構造は抽象 mop と同じであるのに CLOS 実現体のオブジェクト構造はクラスのオブジェクト構造と全く異なっているということであった. この差異を吸収するため, 一見表面には現れてこないが具体 mop と抽象 mop の間に介在する影の mop (クラス)なるものを導入した. このクラスは, 親の抽象 mop についてそれを共有しスロット構造が共通である具体 mop の間で共有されるものである.

CLOS のメタオブジェクトプロトコルは, クラス生成や継承ルールなどのオブジェクトシステムの挙動をプログラマが自由にカスタマイズするための機能であるが, CLOS 上の MOP 開発においてもこの機能が有効に用いられた. すなわちメタオブジェクトプロトコルの API が用いられたことは当然であるが, すべての抽象 mop(クラス)のクラスであるメタクラス *Mop* が定義され, そのサブクラスにすべての影の mop(クラス)のクラスであるメタクラス *ShadowMop* が定義された. 抽象 mop(クラス)はメタクラス *Mop* の実現体であり, 影の mop(クラス)はメタクラス *ShadowMop* の実現体であり, 具体 mop は一見抽象 mop の実現体であるが実装上は影の mop の実現体となっている.

4. RDF, RDFS と MOP

RDF および RDFS はセマンティックウェブ実現を目的としたオントロジ表記法で最初の基礎となるものである[Lassila 1999][Brickley 2003]. RDF の数学的な根拠はモデル論としてはっきりしているが[Hayes 2003], 一方その標準準拠の処理系として満足できるものはない. 特にリスプ言語の処理系は見あたらない. 本章では, RDF および RDFS と MOP の関係について述べ, MOP を使用して RDF および RDFS の処理系開発が可能であることを明らかにする. RDF の仕様は確定しているが, RDFS の仕様は未だ確定されておらず, これまで MOP3 開発中にも仕様の変更があったが, これからもあり得るかも知れない.

4.1 RDF, RDFS のクラスとプロパティ

図1に現在の仕様案[Brickey 2003]に従った RDF および RDFS のクラス構造を示す. オブジェクト指向の観点からは, *rdfs:Class* と *rdfs:Datatype* はメタクラスと捉えることができる. その他の白抜きの楕円はそれらメタクラスの実現体であるクラスである. *rdf:List* 以外のクラスはすべてリソースであり, *rdfs:subClassOf* 関係により *rdfs:Resource* につながっている. 図示していないが, *rdfs:Class* も *rdfs:Resource* につながっている. 影がつけられた楕円はすべて実現体であり, *rdf:nil* は *rdf:List* の, その他はすべて *rdf:Property* あるいはそのサブクラスの実現体である. RDF/S を用いてオントロジを記述する場合には, 図1に存在する定義済みのリソースやプロパティを出発点として, 個別分野向きのリソースやプロパティを定義し, それらを用いて XML や N-triple の表記法でオントロジを記述する.

ところが実際に RDF/S の処理系をプログラムしようとすると, 図1にはオブジェクト指向の観点からは問題がある. 一つは *rdfs:subPropertyOf* である. *rdfs:isDefinedBy* は *rdf:Property* の実現体であるにもかかわらず, *rdfs:subPropertyOf* 関係で *rdfs:seeAlso* に結合されており, 同様に *rdf:_1*, *rdf:_2*, *rdf:_3* などは *rdf:ContainerMembershipProperty* の実現体であるにもかかわらず *rdfs:member* に結合されている. 今, 任意のオブジェクト指向言語を使って *rdfs:subClassOf* 関係も *rdfs:subPropertyOf* 関係も同じスーパー／サブクラス関係で記述しようとすると, 実現体である *rdfs:isDefinedBy* や *rdf:_1* にスーパー／サブクラス関係を持たせることはできず, かといってこれらを実現体ではなくクラスとすると *rdf:Property* はメタクラスでなければならなくなる.

もう一つの問題は, *rdfs:Class* は自分自身をクラスとしており, *rdfs:Datatype* は *rdfs:Class* のサブクラスであると同時にそれをクラスとしていることである. CLOS は別として, 通常のオブジェクト指向言語でこのようなクラス／インスタンス関係のサーキュレーションは実現されていない. またたとえ CLOS を用いても, このようなサーキュレーションをカスタマイズすることはできない.

そこでオブジェクト指向プログラミングでの処理系実現を目的に, 図1のクラス階層関係を**図2**のように変更した. 図からわかるように, メタクラスサーキュレーションは取り除かれ, *rdfs:seeAlso* は *rdf:Property* をスーパークラスとするクラスに変更され, *rdfs:member* はクラスに, *rdf:_1*, *rdf:_2*, *rdf:_3* などはその実現体とされた. こうすれば Java や.NET, CLOS などのオブジェクト指向プログラミング言語を用いて, 仕様に極力忠実に従った RDF/S 処理系が開発可能となる. もちろん, 既存のオブジェクト指向プログラミング言語にたよらず, 最初から *rdfs:subClassOf*

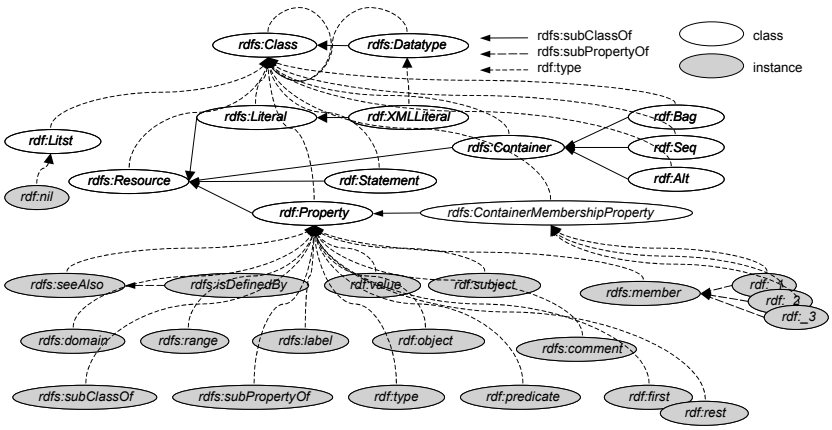

図1　RDF/Sにおけるクラスとプロパティの階層構造

関係や *rdfs:subProperty* 関係を持つ処理系をスクラッチで開発すれば図1に従った厳密な処理系を開発することは可能である. たとえば, メタクラス定義のサーキュレーションについては CLOS のメタクラス standard class の実現法[Kiczales 1991]などを参考にすればよい. しかし今ここでは CLOS 上に実現された Mop システムを用いて RDF/S 処理系を開発しようとしており, 図1に示された仕様に厳密に従うことは不可能であると同時に, 図2に示された些細な仕様変更により特に不都合は生じないと思われる.

4.2 MOP 上の RDF および RDFS

(1) コアとなるリソースの定義

MOP3 ではすべての抽象 mop のクラスとしてメタクラス *Mop* があるが, そのサブクラスとして *rdfs:Class* を定義し, すべての mop の最上位である *Root* の直下の抽象 mop として *rdfs:Class* の実現体である *rdfs:Resource* を定義した. このようにして, 実現体の生成など, mop のオブジェクト機能のすべてを RDF/S のプログラミングに利用することができるようになる. ただし *rdf:List* はリソースではなく *rdfs:Resource* と同様に *rdfs:Class* の実現体として *Root* 直下に定義した. この最小限の三つのクラス, メタクラスを定義したのち, 以下のようにコアとなるリソースを定義した. ただし, ここで `defResource` は CLOS で言えば, クラス定義にメタクラス *rdfs:Class* を指定するものである.

```
(defResource rdf:Property
             (rdfs:Resource gx:Role) ())
(definstance rdfs:domain (rdf:Property)
  ((rdfs:range rdfs:Class)
   (rdfs:domain rdf:Property)))
(definstance rdfs:range (rdf:Property)
  ((rdfs:range rdfs:Class)
   (rdfs:domain rdf:Property)))
(defResource rdfs:seeAlso (rdf:Property)
  ((rdfs:range rdfs:Resource)
   (rdfs:domain rdfs:Resource)))
(definstance isDefinedBy (rdfs:seeAlso)
  ((rdfs:range rdfs:Resource)
   (rdfs:domain rdfs:Resource)))
(defResource rdfs:Container (rdfs:Resource) ())
(defResource rdf:Bag (rdfs:Container) ())
(defResource rdf:Seq (rdfs:Container) ())
(defResource rdf:Alt (rdfs:Container) ())
(defResource rdfs:ContainerMembershipProperty
             (rdf:Property) ())
(defResource rdfs:member
             (rdfs:ContainerMembershipProperty)
  ((rdfs:domain rdfs:Resource)
   (rdfs:range rdfs:Resource)))
(defResource rdf:Statement (rdfs:Resource) ())
(definstance rdf:nil (rdf:List) ())
(definstance rdf:first (rdf:Property)
  ((rdfs:domain rdf:List)
   (rdfs:range rdfs:Resource)))
(definstance rdf:rest (rdf:Property)
  ((rdfs:domain rdf:List)
   (rdfs:range rdf:List)))
(definstance rdfs:subClassOf (rdf:Property)
  ((domain rdfs:Class)
   (range rdfs:Class)))
(definstance rdfs:subPropertyOf (rdf:Property)
  ((domain rdf:Property)
   (range rdf:Property)))
(definstance rdfs:comment (rdf:Property)
  ((rdfs:domain rdfs:Resource)
   (rdfs:range rdfs:Literal)))
(definstance rdfs:label (rdf:Property)
  ((rdfs:domain rdfs:Resource)
   (rdfs:range rdfs:Literal)))
(definstance rdf:subject (rdf:Property)
  ((rdfs:domain rdf:Statement)
   (rdfs:range rdfs:Resource)))
(definstance rdf:predicate (rdf:Property)
  ((rdfs:domain rdf:Statement)
   (rdfs:range rdf:Property)))
(definstance rdf:object (rdf:Property)
  ((rdfs:domain rdf:Statement)
   (rdfs:range rdfs:Resource)))
(definstance rdf:value (rdf:Property)
  ((rdfs:domain rdfs:Resource)
   (rdfs:range rdfs:Resource)))
```

このような RDF/S 各要素の定義ののち, Common Lisp のタイプ判定や Mop システムの抽象・特殊判定の関数を呼び出すことによって, 図2に示したとおりの結果が以下のように得られる.

```
> (typep rdf:subject rdfs:Resource)      -> t
> (typep rdf:subject rdf:Property)       -> t
> (typep rdfs:isDefinedBy rdfs:seeAlso)  -> t
> (typep rdfs:isDefinedBy rdf:Property)  -> t
> (typep rdfs:seeAlso rdf:Property)      -> nil
> (abstp rdfs:seeAlso rdf:Property)      -> nil
> (abstp rdf:Property rdfs:seeAlso)      -> t
> (abstp rdfs:ContainerMembershipProperty
rdfs:member)                             -> t
> (abstp rdf:Property rdfs:member)       -> t
  > (abstp rdfs:Resource rdfs:member)      -> t
```

(2) RDF/S 処理の実例

図3は RDFS 仕様に関する旧いドキュメント[Brickley 2002]に記載されていた RDF/S の導入例である. MOP3 でこれを実現するためには, 以下のように定義すればよい.

```
> (defpackage eg
    (:export "Work" "Document" "Agent" "Person"
             "author" "Proposal" "name")
    (:documentation
     "http://somewhere-for-eg/eg"))
#<The eg package>
> (defpackage dc
    (:export "title")
```

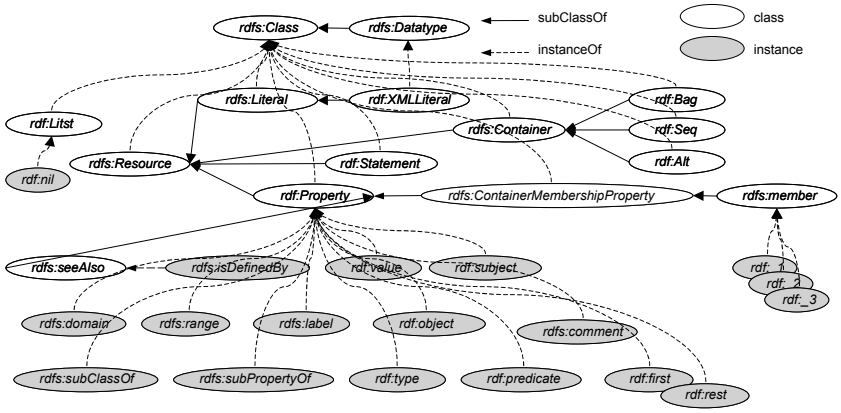

図2　オブジェクト指向プログラミングのためのRDF/S階層構造の変更

```
    (:documentation
     "http://dublincore.org/2002/08/13/dces"))
#<The dc package>
> (rdfs:defResource eg:Work (rdfs:Resource) ())
#<rdfs:Class eg:Work>

> (rdfs:defResource eg:Agent (rdfs:Resource) ())
#<rdfs:Class eg:Agent>
> (rdfs:defResource eg:Person (eg:Agent) ())
#<rdfs:Class eg:Person>
> (definstance eg:name (rdf:Property)
    ((rdfs:domain eg:Person)
     (rdfs:range rdfs:Literal)))
#<rdf::Property.1 eg:name>
> (rdfs:defResource eg:Document (eg:Work)
     ((eg:author eg:Person)
      (dc:title rdfs:Literal)))
#<rdfs:Class eg:Document>
> (definstance eg:author (rdf:Property)
    ((rdfs:domain eg:Document)
     (rdfs:range eg:Person)))
#<rdf::Property.1 eg:author>
> (definstance eg:Proposal (eg:Document)
   ((eg:author eg:Person
               (eg:name "Tim Berners-Lee"))
    (dc:title
      "Information Management: A Proposal"))
   rdf:about
    (net.uri:parse-uri "http:/…/Proposal/"))
  #<eg::Document.5 Proposal>
```

このような定義ののち，Mop システムの関数を使って以下のように内部のデータを検索することができる．

```
> (dph eg:Proposal)
(eg:Proposal
 (eg:author eg::Person.4
            (eg:name "Tim Berners-Lee"))
 (dc:title "Information Management: A Proposal")
 )
> (<- eg:Proposal 'eg:author 'eg:name)
"Tim Berners-Lee"
> (<- eg:Proposal 'dc:title)
"Information Management: A Proposal"
> (typep (<- eg:Proposal 'eg:author) eg:Agent)
t
```

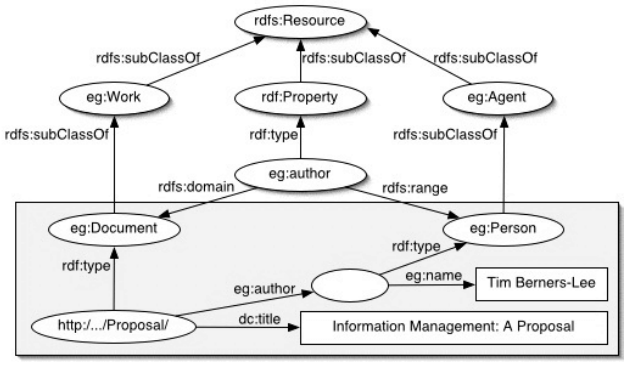

図3 RDF および RDFS の例
（RDFS 仕様書[Brickley 2002]より）

5. おわりに

CLOS(Common Lisp Object System)上でメタオブジェクトプロトコルを用いて MOP(Memory Organization Package)が開発された．これを MOP3 と呼ぶ．Memory Organization Package はシャンクらによって開発された事例ベース記憶システムであり，メタオブジェクトプロトコルは CLOS のシステム機能をカスタマイズするためのメタオブジェクトプログラミング機能である．実現性を調査したのち，RDF/S の仕様の些細な変更により MOP3 上で RDF および RDFS のプロセッサを開発した．今後はこれをさらに OWL や DAML-S の処理系に発展させる予定である．

6. 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．記して感謝の意を表する．本プロジェクト実施および MOP 開発では，先に通産省NEDOによって実施された「ヒューマンメディア」プロジェクトの「次世代プラントインタフェースの開発」によるところが大きい．ヒューマンメディアプロジェクト委託元である財団法人イメージ情報科学研究所，および次世代プラントインタフェース開発のリーダであった大阪大学溝口理一郎教授，サブリーダであった岡山大学五福明夫教授に感謝の意を表する．溝口教授および五福教授は現在も本プロジェクトにおける共同研究者として協力をいただいている．

参考文献

[Brickley 2002] Brickley, D., and R.V. Guha, “RDF Vocabulary Description Language 1.0: RDF Schema”, http://www.w3.org/TR/2002/WD-rdf-schema-20021112/, W3C, 2002.

[Brickley 2003] Brickley, D., and R.V. Guha, “RDF Vocabulary Description Language 1.0: RDF Schema”, http://www.w3.org/TR/rdf-schema/, W3C, 2003.

[CLtL2 1990] Steel Jr., G.L., *Common Lisp the Language Second Edition*, DEC, Bedford, 1990.

[橋田 1999] 橋田浩一ほか，言語の獲得と喪失，岩波，1999.

[Hayes 2001] Hayes, P., “RDF Model Theory”, http://blogspace.com/rdf/modeltheory/, W3C, 2001.

[Hayes 2003] Hayes, P., “RDF Model Theory”, http://www.w3.org/TR/rdf-mt/, W3C, 2003.

[今井 1997] 今井むつみ，ことばの学習のパラドックス，共立，1997.

[今井 2000] 今井むつみ(編)，心の生得性，共立，2000.

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

[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 1999] Lassila, O., and R.R. Swick, “Resource Description Framework (RDF) Model and Syntax Specification”, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222, W3C, 1999.

[Paepcke 1993] Paepcke, A. (ed.), *Object-Oriented Programming*, MIT, Cambridge, 1993.

[Pinker 1994] Pinker, S., *The Language Instinct*, William and Morrow, 1994.

[Riesbeck 1989] Riesbeck, C.K., and R.C. Schank, *Inside Case-Based Reasoning*, LEA, Hillsdale, 1989.