

軽量オントロジによるセマンティックプログラミング

A Lightweight Ontology with Semantic Programming

倉光 君郎
Kimio Kuramitsu

横浜国立大学大学院
Yokohama National University
kimio@ynu.ac.jp, <http://www.ubicg.ynu.ac.jp/>

keywords: knowledge representation, semantic web, lightweight ontology, information integration

Summary

近年、セマンティック情報が付加された Web データが増えており、それらの情報を統合したアプリケーション開発の需要も高まっている。本研究では、軽量オントロジをオブジェクト指向モデルに統合することで、ISA や意味的な同義性を簡単に扱える Konoha スクリプティング言語を開発している。本発表では、Konoha スクリプティング言語の設計と実装を通し、セマンティックプログラミングの実現について紹介する。

1. はじめに

オントロジ技術は、Web 上の意味付けされたデータや知的処理システムの知識表現 [11, 20] を管理する中心的な技法として広く認知されている。特に近年ではセマンティック Web への関心の高まりから Web 上でオントロジを共有するためのツールやライブラリの開発 [12] も盛んになっている。しかしこれらの関心の高まりに対して、一般的なプログラマにはオントロジは複雑な技術体系であり、オントロジ技術が有効であると考えられるデータ指向の Web アプリケーション開発においても手軽に応用されることは少ない。

ひとつの考えられる理由は、オントロジ技術の専門用語 (Concepts, Individuals, Roles) が、今日のプログラマが親しんだオブジェクト指向プログラミングパラダイムの語彙 (Class, Instance, Field) と異なることがあげられる。また同時に、既存のプログラミング技術からオントロジの利用を考えたとき、ライブラリやツールであっても論理コンストラクタ (forall や exists) を利用しなければならない。これらは知識表現の専門家にはわかりやすくても、現代の標準的なプログラマには敷居が高すぎるといえる。

本研究の目的な、新しいマップベースのアプローチを提案し、オブジェクト指向プログラミング言語の一般的な演算子や意味論からオントロジの利用を可能にすることである。そのため、まず複雑なオントロジの概念や関係に関する記述や推論を意味的な同義 (*equivalence*) や包摂 (*subsumption*) に軽量化し、それを `===` や `isa` のような演算子によって処理することを目指す。結果、プログラミング言語から次のようにオントロジ知識を利用

したプログラムを簡単に書くことが可能になる。

```
Medicine m = "Amoxillin";
if(m isa Antibiotics || m === "Penicillin") ..
```

マップベースの統合アプローチの特徴は、オントロジ記述中立性である。我々は、セマンティックマッピングを導入することで、オントロジ記述の複雑な概念関係を単一的に再定義した。これにより、キャスト演算子を透過的に拡張したマップキャスト演算子によって、オントロジ推論による意味知識にアクセス可能になった。

本論文のもうひとつの貢献は、我々が開発を進める Konoha スクリプティング言語によって提案する手法を実装し、いくつかのプログラミングコーディングによって検証を行ったことである。Konoha 言語は、次のサイトからオープンソース製品としてダウンロード可能である。

```
begincenter http://konoha.sourceforge.jp/ endcenter
```

本論文では、Konoha 言語を通してオントロジとオブジェクト指向プログラミング言語パラダイムのマップベースの統合を論じる。論文は以下のとおり進む。第 2 節では、まず Konoha 言語によるセマンティックプログラミングの概要を述べる。第 3 節では、オントロジとプログラミング言語の共通モデルを構築する。第 4 節では、我々の言語設計に関して簡単に紹介し、第 5 節では言語設計の妥当性に関して論じる。第 6 節では関連研究を紹介し、第 7 節において論文を総括する。

2. セマンティックプログラミング

Konoha 言語は、我々の研究チームがオープンソースで開発している次世代のスクリプティング言語である。特徴は、静的に型付けされたオブジェクト指向モデルを採用している点にある。本節では、Konoha 言語を用いて、我々の提案するセマンティックプログラミングを紹介する。

なお、Konoha 言語は、Java 風の文法を採用しており、本論文で紹介する例は、特に断りがないかぎり、Java 言語仕様の文法にしたがって解釈できる。また、>>> は、Konoha 対話モードのコマンドプロンプトであり、本論文で紹介する機能は、一部をのぞいてそのまま試すことができる。

2.1 意味的に拡張された型

全てのプログラミング言語は、整数や文字列を表現するため、int や String など基本データ型をサポートしている。しかし、これらの型は、そのデータ値の内容に関して意味を一切あらわすことはない。例えば、String は、名前や電子メール、ISBN など様々な意味をもったテキストを表現することができるが、そのテキストのクラスを識別する意味を持たない。

セマンティックプログラミングの最初のステップは、世界に意味を導入することにある。Konoha 言語では、using 文を用いることで、基本データ型と URN (Universal Resource Name^{*1}) で識別される意味を関連付けて、新しい型 (クラス) を作ることができる。

次は、Float に摂氏 (Celsius) の意味を与えた例である。

```
>>> using Float:C http://unit/Celsius
>>> Float:C t = 20C;
>>> t
20[C]
>>> t.class
Float{http://unit/Celsius}
```

新しい型は、Float:C という新しい名前で行うことができる。ここで、:C は、現在のプログラミング名前空間でのみ有効な (ローカルの) 意味タグであり、リテラルの識別子 (20C) にも利用することができる。ただし、プログラミング言語処理系としては、クラス名には URN を追加して識別される。

次は、体感温度に関するボキャブラリ (freezing, chilly, cool, comfortable, hot, ...) などを考えてみる。文字列 String も同じく、URN を意味識別子にして新しい意味付けされた型を作ることができる。次の例は、体感温度のボキャブラリ (FeelTemp) を表す String:feel 型である。

'feel:... ' は、feel をタグとして意味付けされた文字列を表現する拡張されたリテラルである。

```
>>> using String:feel http://vocabulary/FeelTemp
>>> String:feel ft = 'feel:chilly';
>>> ft = "hello,world";
(Type!!: 異なる型を代入しようとした)
```

「意味的に拡張された型」の重要な点は、単純に型に意味をアノテートするだけでなく、プログラミング言語の型システムとして型安全性を保証する点にある。上記の例では、http://vocabulary/FeelTemp において定義されたボキャブラリのみ代入を認められる。

i. 注意

本論文では、URN が先でどのようなフォーマットでボキャブラリや意味が定義されているかに関しては詳細を述べない。現在、Web 上には様々なボキャブラリを定義する規格があり、それらを Konoha 言語へ接続することは本論文のセマンティックプログラミング機構の実現に比べ、些細なことと考えるためである。

2.2 意味推論

意味的に拡張された型は、プログラマには意味を提供して便利になるかもしれないが、コンピュータ処理的には依然として意味がないままである。つまり、Float:C と String:feel で表現されるデータは、型 (クラス名) で識別されるが、10C と 'feel:chilly' の意味的な関係は不明なままである。

このような疑問に答えるため、オントロジ技術は意味的な推論系を提供する。両者の意味の違いを論理学記号を用いて表現するのである。Konoha 言語は、推論系自体をサポートしないが、推論系の出力した意味的な関係をマッピングとして取り込むことができる。詳細は、後節に譲り、ここでは、セマンティックプログラミングという観点から、プログラマはどのように 2 つのクラスの意味的な関係を扱えるか紹介するにとどめる。

次は、10C は、どのような体感温度なのか問い合わせた例である。単純に、しかもよく知られているキャスト演算子で扱うことができる。

```
>>> t = 20C;
>>> (String:feel)t
'feel:comfortable'
```

Konoha では、従来のキャスト演算子と区別するため、マップキャスト演算子と呼ぶ。マップキャスト演算子が実行されたとき、外部のオントロジ推論システムやデータベースシステムに対して、?: -20C ↦ String:feel のようなマップベースのクエリを発行する。その結果を受けて、マップキャスト演算子の結果がユーザに返される。ただし、プログラマから見たとき、マップキャスト演算子と通常のキャスト演算子は、完全に透過である。

*1 URN は、URL や URI の一種であるが、内部的に一意に識別される名前の拡張も含まれ、必ずしもリソースの入手先をあらわすとは限らない。

次の節からオントロジとオブジェクト指向プログラミングモデルの観点から、2つの世界を融合する方法に関して述べる。

3. 二つの世界の融合

我々のマップベースの統合アプローチについてよりフォーマルな定義を行う。従来、オブジェクト指向プログラミングの型理論とオントロジ記述の論理学 [1, 19] は異なる形式モデルを用いてきたため、本節は可能なかぎり集合論に基づく単純な定義で記述を試みる。

3.1 クラスとコンセプト

オブジェクト指向世界のクラスとインスタンス (instances) は、知識表現世界のコンセプトと個体 (individuals) と非常によく似た表現単位であるが、その定義の順番において重要に異なる。つまり、クラスは先に定義され、インスタンスはインスタンス化によってクラスから生成される。これに対し、個体は原則的に先にあり、個体のクラスフィケーションによってコンセプトは導出される。

我々は、まず両者を統合するスタート地点としてクラス先な世界観の上にコンセプトを追加する方法を採用した。つまり、全ての個体はひとつの先に存在するクラス (コンセプト) のインスタンスである。ここで、 C をコンセプト名とする。そのコンセプト C のインスタンス集合を C^I と書く。ある個体 t が C のインスタンスであるならば、 $t \in C^I$ といえる。

次は、2つのコンセプト AmericanSeason and BritishSeason の例である。

AmericanSeason^I = {spring, summer, fall, winter}
 BritishSeason^I = {spring, summer, autumn, winter}

さて、この2つのコンセプトは非常によく似ている。つまり、共通のインスタンス (spring, summer, and winter) を含んでいる。しかし、これらのインスタンスは異なる個体であると扱い、(次節で説明する関係が定義されない限り、意味的な関係はないものとするという意味において) 同音異義語 (homonyms) と仮定する。オブジェクト指向世界では、例えば `t.getClass()` を取り出すことができるが、異なるコンセプトの同音異義語の扱いは、表記上不便である。本論文では、コンセプトを接頭辞につけて個体 (インスタンス) を表現する。

【定義 1】(個体) $C.t$ は、クラス C のインスタンス t である。プログラミング言語上では、`t.getClass() == C` が常に真となる。

3.2 セマンティックマッピング

オブジェクト指向モデルでは、サブタイプシステムによってクラス間のインスタンスに意味的な関係が存在する。しかし、本節ではオントロジがもたらす意味的な関

係と同じ議論の土台で議論するため、一旦、クラス=サブクラスを忘れ、クラス間には何の意味的な関係がないものと想定する。

我々の形式モデルでは、クラス間の意味的な関係はセマンティックマッピング (semantic mapping) を与えることで追加できるものとする。(サブタイプシステム系とセマンティックマッピングの互換性は、次節で述べる。)

はじめに、まず二つの異なるクラスのインスタンス $C.x$ と $D.y$ を考える。もし $C.x$ は $D.y$ と解釈が可能であれば、我々はセマンティックマッピングを $C.x \mapsto D.y$ のように与えることができる。ここで、セマンティックマッピングのより数学的な定義は、相対的な情報許容量の理論 [17] によるものとする。つまり、直感的に言えば、 $C.x$ は、 $D.y$ と比べて、より多くの情報を含まれており、(情報が増える方向にはマッピングはできないが) 減少する方向に変換できるといえる。簡単な例は、Aircraft.”ボーイング 777” \mapsto Transportation.”飛行機”であるが、逆は言えない。意味的な同義性は、 $C.x \mapsto D.y$ かつ $D.y \mapsto C.x$ のときいえ、我々は $C.x \equiv D.y$ と書く。

個体間のマッピングは、自然にクラス間のマッピングに拡張可能であり、それによりクラス間の意味的な関係をセマンティックマッピングで記述することができる。

【定義 2】(クラス間の関係)

$$\frac{\forall x \exists y C.x \mapsto D.y}{C \mapsto D}, \quad \frac{C \mapsto D \quad D \mapsto C}{C \equiv D} \quad (1)$$

partial mapping. We say no mapping if $C.x \mapsto \text{null}$, and we write $C \not\mapsto D$ if for each $x \in C^I C.x \mapsto D.\text{null}$. The class C, D are disjoint if $C \not\mapsto D$ and $D \not\mapsto C$.

本論文は、単純化のためセマンティックマッピングは、全射 (total) であると仮定する。ただし、実際のマップキャスト演算では、部分写像 (partial mapping) であっても実施されることが一般的にある。これは、null ポインタへのマッピングとして実装することが可能である。

3.3 サブタイプシステム

サブタイプシステム (subtyping) は、オブジェクト指向プログラミング言語の中心的な機構であり、プログラマに対し、クラスを階層的な手法で構築することを可能にする。一般に、クラスとサブクラスの関係は半順序で表現できるが、プログラミング言語コミュニティでよく用いられる $<$: を用いてクラス=サブクラス間の半順序を書く。

サブタイプシステムは、Featherweight Java 形式論 [?] によれば、次のとおり定義することができる。(Konoha も Java と同様に class-extends でクラス継承を宣言することができる。)

$$\frac{\text{class } C \text{ extends } D \{ \dots \}}{C <: D}, \quad \frac{C <: D \quad D <: E}{C <: E} \quad (2)$$

さて、サブタイムシステムとセマンティックマッピングには、次の定理が成り立つ。

[定理 1] セマンティックマッピングによるクラス関係は、サブタイプシステムを包含する。

(Proof) $C <: D$ のとき、クラス C 上の全てのインスタンス $C.x$ に対し、 $C.x \in D^I$ がいえる。これは、 $C^I \subset D^I$ である。したがって、 C 上の全てのインスタンスに対し、 D 上の自分自身の写像がいえ、これは同じ個体であるため、セマンティックマッピングである。加えて、 $C^I \subset D^I$ かつ $D^I \subset E^I$ のとき、 $C^I \subset E^I$ がいえるのは自明である。

オントロジ記述とセマンティックマッピングの表現力による互換性に関する議論は、第 3 節の設計合理性の箇所でも証明する。

4. 言語設計

Konoha 言語の設計は、Java 言語をベースにして、スク립ティング言語として公平に単純化した言語である。ここでは、オントロジ利用に関する拡張のみに焦点を当てて言語設計を述べる。

4.1 マップキャスト演算子

Konoha 言語では、セマンティックマッピングに関する操作は全てマップキャスト演算子で行う。これは、C プログラマから Java プログラマまで広く知られた演算子と透過的な記法である。今、クラス C のインスタンス $C.o$ を考える。マップキャスト演算 $(D)C.o$ は、クラス C と D の関係によって、次のとおり変化する。

- upcast $C <: D$ のとき、 $C.o$,
- downcast $D <: C$ のとき、 $C.o$,
- mapcast $C.o \mapsto D.o'$ かつ $C <: D$ でないとき、 $D.o'$
- stupidcast 上記以外の場合、NoSuchMapping 例外が発生

Konoha 言語では、いくつかの場合、暗示的なキャストを認めている。ただし、本論文では暗示的なキャスト演算に関しては考慮に入れない。

加えて、Konoha 言語ではセマンティックマッピングを応用する形式で、いくつかの意味的な比較演算子を導入している。

| | |
|-----------------------------|-----------------------------|
| $C === D$ | $C \equiv D$ |
| $C.o === D.o'$ | $C.o \equiv D.o'$ |
| $C \text{ to? } D$ | $C \mapsto D$ |
| $C.o \text{ instanceof } D$ | $C == D \text{ or } C <: D$ |
| $C.o \text{ isa } D$ | $C.o \mapsto D.o'$ |

注意: 演算子 instanceof は、Java 言語の文法論を維持するために残されている。

4.2 マッピング透過性

我々は、意味的に拡張されたクラス間のセマンティックマッピングに関して議論をしてきた。しかし、Konoha 言語は、任意のクラス間に対してマッピングを定義して、マップキャスト演算子を行うことを認めている。sf String から sf InputStream へのマップキャストも原理的に可能である。

```
in = (InputStream)"file.txt";
```

しかし、上記のコーディングには意味的な曖昧さが伴う。つまり、文字列がファイル名を表すときはよいが、文字列は任意のテキストを表現できるため、必ずしもファイル名でない場合もありえる。現実として「使えるマッピング」を定義することができない。

Konoha では、意味的に拡張されたクラスを用いることで、基本型であっても意味を制約することができる。そのため、文字リテラルにおいて verb—'file:' を与えることで、明示的にファイル名であることを宣言できる。そこで、曖昧さを除去してマップキャストを行うことができる。

```
in = (InputStream)'file:file.txt';
```

このマップキャストは、ファイル名から sf InputStream を得るためには、sf FileInputStream というサブクラスを用いるという次のようなプログラミング知識を機械推論から導出したものといえる。

```
new FileInputStream("file.txt");
```

クラス名を覚える方が楽か、データに意味を与える方が楽か、議論の余地があるが、Konoha 言語は、セマンティックプログラミングという新しいプログラミングパラダイムを提案している。

5. 設計合理性

この節では、我々の提案するセマンティックマッピングによる統合手法によって、どの程度のオントロジ記述が利用可能になるか、その設計合理性に関して議論する。

5.1 オントロジとの相互運用性

セマンティックマッピングは、ISA 関係とほぼ同義に置き換えられるため、ISA 関係のみで構造化された軽量オントロジ [16] と同等の記述力といえる。ISA 関係は、現在、Web 上のフォークソノミーなど、厳密に定義しにくい世界でも広く利用されており、これ自体でも十分な有意義な利用シナリオは多い。しかし、一般的なオントロジに対してどうなるかという疑問もある。

我々の研究において、オントロジは「構造化された」ボキャブラリの集合と定義している。ここで、構造は数学的な関係で与えられ、それは単項関係 $C(t)$ と 2 項関係 $R(t, t2)$ と想定している。これらは、それぞれコンセプトやロールと呼ばれるものに相当する。しかし、具体的なオントロジ記述を見ると、普遍的なオントロジの記述言語は存在しない。現在、記述論理をベースにした OWL は、ひとつの標準として受け入れられているが、現在の Web 上では、RDF や RDF/S, ISA 関係だけの軽量オントロジも存在する。

我々は、本研究においてオントロジ記述より、プログラミング言語システムにおけるその利用に関して関心をもってきた。オントロジ記述に関しては、様々なバリエーション [1, 4, 9] が存在するが、その利用（推論系）においては次の共通した推論が定義されている。

- (equivalence) $C \equiv D$,
- (subsumption) $C \sqsubseteq D$
- (disjointness) $C \sqcap D = \perp$

オントロジは、推論システムを通して記述を利用するという点を考慮に入れれば、これらの推論システムとセマンティックマッピングが等価であれば、様々なオントロジ記述の知識を利用することができる。

[定理 2] セマンティックマッピングは、 $C \equiv D$ や $C \sqsubseteq D$ 、 $C \sqcap D = \perp$ の関係を十分に表現できる。

(Proof) Let Δ be a finite set of terms in an ontology system. Suppose $t \in \Delta$. If a unary relation $C(t)$ is true, then we make a new instance $C.t$ in C^I . We always say $C.t \equiv D.t$ because t is identical on Δ . On the other hand, $C(t)$ is said to be true if $C \sqsubseteq D$ and $D(t)$ is true. Accordingly, we say $C.t \mapsto D.t$ for all t that satisfies both $C(t)$ and $C \sqsubseteq D$ (, i.e., $D(t)$ is true).

5.2 論理学演算子

Konoha は、論理型プログラミング言語が提供するような論理学演算子やロールが定義する演算子を拡張することなく、オントロジが提供する概念の包含や同値を扱えるように拡張した点が特徴である。これは、知識表現に慣れていないプログラマにとってわかりやすさを提供するが、逆に論理記述による知識表現に慣れたプログラマには機能不足に感じる点になるかも知れない。

ここで、Konoha 言語によるオントロジ統合の位置づけを把握するため、次のような論理学演算子を用いてより複雑な知識を問い合わせるプログラムを書くシナリオを想定してみる。

```
Person x;
if (x isa Mother and
    (exists Parent y; (x hasChild y))) ...
```

もちろん、現在の Konoha では直接、上記のようなこ

とを書くことはできない。しかし、驚くべきことに、同等の処理をちょっとした工夫で書くことができる。それは、新しいコンセプトをインポートするで対応できる。

```
using String:GrandMother
    "dl://(and (isa Mother) ...)";
String:Person x;
if (x isa String:GrandMother) ...
```

GrandMother のコンセプトは、プログラミング言語の中で定義しなくても、オントロジシステムの上で使い慣れた記述言語を用いて定義すればよい。そして、オントロジシステムは、そのコンセプトのインスタンス集合のみ Konoha に提供してくれれば、Konoha の方から利用することが可能になる。

GrandMother \equiv Mother \sqcap \exists hasChild.Parent

これは、RDBMS 上のデータを利用するため、SQL 文をプログラミング言語がサポートしなくてもよいことに似ている。そして、オントロジシステムとの独立性が保たれ、ソースコード上では明確に名付けられたコンセプトのみ登場するため、可読性が高かまるメリットがある。

6. 関連研究

知識表現（オントロジ）をプログラミング言語システムへ統合することは、プログラミング言語と知識表現と独立したコミュニティで研究されてきたアイデアや技法を両方必要とする。ここでは、過去に行われて来た同様の試みに関して紹介する。

LISP は、コンピュータに知的な処理を行わせる試みの過程で創造され、それ以来、LISP スタイルのシンタックスによる知識表現の技術 [7, 9] の歴史は長い。そのため、LISP ベースの論理言語、と LISP シンタックスの知識表現を融合させる研究は自然な流れと言える。最近では、Go! [5] 言語が設計され、オブジェクト指向 Prolog とオントロジ記述の統合が実現されている。しかし、論理型プログラミング言語とオントロジを融合するのと、手続き型オブジェクト指向プログラミング言語を統合するのは全く異なる技術的な努力が必要といえる。

最近では、Semantic Web の人気の高まりにより、主流的オブジェクト指向プログラミング言語 (Java や C#) のためのクラスライブラリが整備されている。これらの多くは、例えば、Jena [12] のように、オントロジ技術の概念を Concept や Role のようにプログラミング言語のクラス設計で実現し、これらのクラスを通してオントロジを操作するように設計されている。そのため、オントロジとオブジェクト指向言語のモデル透過性は失われ、オントロジの世界で定義された意味がプログラミングの演算子に影響を与えることはない。

モデル透過な統合もいくつか試みられている。ひとつは、Goldman ら [8] によるコード生成手法によるアプロー

チであり、彼らは OWL で記述された概念から C# クラスを生成することで、オントロジとオブジェクト指向クラス設計の透過性を実現している。もうひとつは、ActiveRDF [18] が示すとおり、ORM(Object-Relational Mapping) スタイルのアプローチであり、SPARQL クエーリによって Ruby のクラスを生成している。これは、ダイナミックプログラミング言語の特性を生かし、RDF/S 記述と Ruby クラスの透過性を実現しているが、既存のオブジェクト指向プログラミング言語を利用しているため、クラス階層システムの限界により、包摂の関係しか表すことはできていない。

7. 結 論

オントロジ技法は、データを意味付けして効率よく管理する手法として認知されている。しかし、現在もなお、情報を大量に扱うアプリケーション開発においてオントロジを適用することに大きな困難が残されている。本論文は、オブジェクト指向プログラミング言語に対するマップベースのオントロジ推論の統合を提案し、論理的なコンストラクタを用いることなく、オントロジ技術の利点をプログラミングに活かせる手法を開発した。我々の方法は、オントロジによる意味的な関係とオブジェクト指向プログラミングのモデルを透過的に統合し、新しくセマンティックプログラミングの実現を可能することを示唆した。

本論文において提案された技法は、Konoha スクリプティング言語においてプロトタイプ実装を進め、その開発成果はオープンソースとして一般に公開している。今後の課題は、外部リソースに対するマップキャスト演算子の効率評価と Konoha 言語の利用経験に基づくセマンティックプログラミングの評価である。

謝 辞

本研究の一部は、総務省 SCOPE-R「意味推論型システムを備えたユビキタス・パーチャルマシン技術の研究(062103013)」と文部科学省科学研究費補助金若手研究(B)「WEB オントロジを用いた意味型システムの研究」の補助を受けて行なわれた。また、Konoha プロジェクトへ参加し、オープンソースソフトウェア開発へ貢献して頂いた方々へ感謝の意を表したい。

◇ 参 考 文 献 ◇

- F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (eds). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2000.
- Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):28-37, 2001.
- Biezunski, M., Bryan, M., and Newcomb(eds), S. R. ISO/IEC 13250:2000 topic maps: Information technology

- document description and markup languages, 1999
- Ronald J. Brachman and James G. Schmolze. An overview of the KL-One Knowledge Representation System. *Cognitive Science*, 9(2):171-216, 1985.
- Keith L. Clark and Frank G. McCabe. Ontology Oriented Programming in Go! *Journal of Applied Intelligence*, 2005.
- Anhai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the International Conference on WWW*, 2002.
- M. R. Genesereth. Knowledge Interchange Format. In *Proceedings of the Conference of the Principles of Knowledge Representation and Reasoning*, pages 599-600, 1991.
- Neil.M. Goldman. Ontology oriented programming - static typing for the inconsistent programmer. In *Proc. of ISWC 2003*, LNCS 2870, pp.850-865, 2003.
- Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Aquisition*, 5(2):199-220, 1993.
- Nicola Guarino. Formal ontology and information systems. In *Proceedings of the 1st International Conference Formal Ontology in Information Systems - FOIS98*, pages 3-15, 1998.
- Michael Gruninger and Jintae Lee. Special issue: Ontology applications and design. *Communications of the ACM*, 45(2):39-41, 2002.
- Jena - A Semantic Web Framework for Java. <http://jena.sourceforge.net/>
- Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art, *The Knowledge Engineering Review*, 18(1):pp. 1-31, January 2003.
- 倉光君郎. 情報爆発時代のデータ翻訳に適したスクリプティング言語の設計. 情報処理学会データベースと Web 情報システムに関するシンポジウム, 2007.
- 倉光君郎. プログラミングとマッピング. 情報処理学会プログラミング言語研究会, PRO68, 1 月, 2008.
- K. Kuramitsu. Mappings As A Lightweight Ontology System for the World-Wide Web. In *Proc. of the Symposium on Professional Practice in AI / IFIP World Computer Congress (WCC2004)*, 2004.
- Renée J. Miller, Yannis E. Ioannidis, and Raghu Ramakrishnan. The use of information capacity in schema integration and translation. In *Proceedings of 19th International Conference on Very Large Data Bases*, pages 120-133. Morgan Kaufmann, 1993.
- Eyal Oren, Renaud Delbru, Sebastian Gerke, Armin Haller, and Stefan Decker. ActiveRDF: Object-Oriented Semantic Web Programming. In *Proc. of WWW2007*, 2007.
- John F. Sowa and David Dietz. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole, 1999.
- Aris M. Ouksel and Amit P. Sheth. Semantic interoperability in global information systems: A brief introduction to the research area and the special section. *SIGMOD Record*, 28(1):5-12, 1999.
- Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks(eds.) OWL Web Ontology Language: Semantics and Abstract Syntax, *W3C Recommendation*, 10 February, 2004.
- Steve Pepper and Graham Moore (eds.) XML Topic Maps (XTM) 1.0 *TopicMaps.Org Specification*, 2001.

[担当委員 : × ×]

19YY 年 MM 月 DD 日 受理