記　号　処　理　36－7
プログラミング言語　5－7
（１９８６．３．１０）

# Common Lisp へのオブジェクト指向機能導入の動向

石田　亨（ＮＴＴ）　井田昌之（青山学院大学）　内田智史（青山学院大学）　大久保清貴（パナファコム）
岡本泰次（富士通）　佐治信之（日本電気）　湯浦克彦（日立製作所）　西田大治（富士ゼロックス）

## 1．まえがき

1984年 8月に開かれた米国Common Lisp 委員会の決定を受けて、同年秋よりCommon Lisp （以下CLと略す）にオブジェクト指向機能の導入を検討するオブジェクト指向専門委員会が、活動を開始した。初期のメンバは、議長を務めるXerox 社 Palo Alto Research Center の Kenneth M. Kahnを含め42名であった。(ARPA Net 内の電子掲示板を利用して議事が進行しているために、メンバと呼ぶのは不適切かもしれない。)　当初は、専門委員会の方向付けに関する議論が中心となった。10ヵ月あまりの間、方針に関する議論や、 Smalltalk, Loops, Flavors をベースにした種々の技術的討論が行われた後、1985年 8月に、オブジェクト指向専門委員会のミーティングが 150名程度の参加者を集めて開かれ、以下の 3つのプロポーザルが提案された。

① Xerox 社によるCommon Loops [1]

② LMI 社によるObject Lisp [2]

③ HP社Alan Snyder による提案[3]

討議の結果、①を主に、②③の良い点を加味して標準化案をまとめるという予定が示されたが、同年12月のCommon Lisp 委員会では、標準化案をまとめるには至らなかった模様である。また、12月の委員会ではSymbolics 社が、標準化のプロポーザルではないと断りながらも新しい Flavors システム（以降new Flavors と呼ぶ。）[4] の紹介を行っている。

本報告は、1984年 8月以来のCommon Lisp へのオブジェクト指向機能導入の経過をまとめたものである。 2章で専門委員会の方針に関する討議を述べ、 3章以降で 3つのプロポーザルと、今後の標準化に影響を与えると思われるnew Flavors の概要を紹介する。各提案の全内容を紹介することはできないため、考え方の相違が明確に現れると思われるクラス、メソッド、多重継承等の基本構成に絞って報告する。

## 2．専門委員会の方針

専門委員会の方向付けに関する意見は大きく以下の２種に分かれた。

① 複数のオブジェクトシステムを効率良く実現するための基本機構(hook)だけをCLの言語仕様として標準化しようという意見。

② 標準的なオブジェクトシステムを決定しようという意見。

(1) hookの標準化を主張する意見

Fahlman, Handerson(CMU),Moon, Weinreb(Symbolics),Kahn (Xerox) 等がこの意見を主張している。概要を以下にまとめる。

① CLは言語であり、プログラミングシステムではない。CL環境下に複数のオブジェクト指向プログラミングシステムを共存できるようにすべきである。

② CL処理系と密接に関係する、即ち効率に影響する基本機構(hook)のみを標準化すべきであって、CLを用いてポータブルなコードで記述できる所は、標準化する必要はない。『メカニズムは核に、ポリシーは核の外側にあるべきである。』

③ オブジェクトシステムの標準化は、多くのオブジェクトシステムができて問題化すればその時にやればよい。その前にや

るべきではない。議論の余地のある機能は決して同意は得られない。

(2) オブジェクトシステムの標準化を主張する意見

Snyder(HP)がこの意見を主張している。

① hookの提案はオブジェクトシステムの開発者には便利だが、共通のオブジェクトシステムを使いたいというCLユーザの要求は満足しない。ユーザはオブジェクトシステムのドキュメント、教科書、および問題が生じた時のサポートを必要としている。

② hookを標準化するだけでは、あるオブジェクトシステムが普及によって実質的標準となるまで、多くの計算機メーカは非標準なオブジェクトシステムをサポートしようとはしないだろう。したがって、ここで標準を決定できなければFlavorsが実質的な標準となっていくだろう。(注：方向付けの議論の中でFlavors に言及する意見が多く出されている。“Flavors が使えなければCLとは呼べない”という意見や“Flavors を標準にしてはならない”という意見等がみられる。)

上記 2種の意見に結論が出されたわけではないが、大勢は以下に示す手順をとる方向であると思われる。

① 種々のオブジェクトシステムを構築するための最小のhookを定義する。

② Flavors をはじめ、種々のオブジェクトシステムをその上に開発し、その内いくつかのシステムをyellow page のライブラリとして提供する。

③ 普及したものをwhite pageに入れる。(注：CL仕様書では、言語仕様をwhite page, 機種独立のツール等のライブラリ仕様をyellow page,システム依存の仕様を red page 、システムのガイダンスをblue page と呼んでいる。)

## 3. Common Loops

Common Loops (a COMMON Lisp Object Oriented Programming System) は複数のオブジェクトシステムの共通の核となるものとして、Xerox 社によって提案されたものである。

(1) メタクラス、クラス、インスタンス

Common Loopsはクラスを定義するのにdefstruct を用いる。クラスのクラスはメタクラスと呼ばれ、 :class オプションを用いて指定できる。 :class オプションはCLの :typeオプションを汎用化したものである。例を以下に示す。

```
1:  (defstruct point(x-position 0)(y-position 0))
2:  (defstruct(point(:class flavor))(x-position 0)(y-position 0))
3:  (defstruct(point(:class class))
4:    (all-points() :allocation class)
5:    (x-position 0)(y-position 0))
```

1:はpoint という構造体を表わすCLの定義である。2:以下はCommon Loopsの機能を用いたクラスの定義である。2:はFlavors の方法で、3:はLoops の方法でそれぞれ多重継承が行われることを示している。4:の:allocation オプションは変数all-pointsがクラスに割付けられることを表わしている(クラス変数)。特に指定が無い場合には、変数はインスタンスに割付けられる(インスタンス変数)。このようにCommon Loopsでは、異なるオブジェクトシステムをメタクラスを用いて実現することができるようにしている。

(2) メソッド

メソッドはdefun の構文を拡張したdefmethod によって定義される。

(defmethod name-and-options lambda-list [doc-string]{form}* )

ここでlambda-list には任意の引数を、(var type-specifier)という形式で書いても良いようにCLの仕様が拡張されている。

(defmethod move((obj block) x y) code-for-moving-a-block)

これは最初の引数 objが block型のオブジェクトであることを表わしている。moveはこのメソッドのセレクタと呼ばれる。

(defmethod insidep ((w window)(x integer)(y integer))...)

は最初の引数がwindowで 2,3番目の引数がinteger であるメソッドinsidep を定義している。メソッドの呼出しはsend構文ではなく、関数呼出しの形式で行われる。Common Loopsが、他のオブジェクト指向言語と異なる点は、このメソッド呼び出し時に第1の引数だけでなく、2番目以降の引数の型をも検査し、全ての型指定子(type-specifier)が一致したメソッドを実行する所にある。この時、全ての型が一致するメソッド定義が複数個存在する場合には、どのメソッドを選択するかをメソッド順位リスト(method-precedence list)と呼ばれるリストによって決定する。一般に、左の引数から型指定子を調べていき、より特殊化された型指定子を持つメソッドが選択される。どの型指定子がより特殊化されたものであるかは、後述のクラス順位リスト(class-precedence list) によって決定される。(注:ここで型が一致するとは、実引数の型が、仮引数の型が等しいか仮引数の型を特殊化したものである場合をいう。)

(3) 多重継承

Common Loopsでは多重継承を許すために、defstruct の :include オプションを以下のように拡張している。

(defstruct (titled-window (:include (window titled-thing))))

は、titled-window がwindowとtitled-thingを含む構造となることを示している。include オプションに書かれたクラス(スーパクラスと呼ぶ)の順位は、局所順位(local- precedence) と呼ばれ、左に記述されたものほど優先度が高い。

局所的な順位から全体のクラス順位リストを作成するアルゴリズムはメタクラスによって与えられる。省略時のアルゴリズムはLoops やObject Lisp と同一である(後述)。スーパクラスのインスタンス変数は継承されるが、もし同名のインスタンス変数が複数のスーパクラスに存在する場合には、最も優先度の高いクラスのインスタンス変数が継承される。

(4) メソッド結合

Common Loopsでメソッド結合を実現する基本機構はrun-super である。run-super によって、メソッド順位リスト中でrun-super を発行したメソッドの1つ前に位置するメソッドが起動される。今、bounded-windowのスーパクラスをwindowとすると、

```
(defmethod move((w bounded-window)(x integer)(y integer))
      (cond ((in-bounds-p w x y)(run-super))
            (t(signal-errror ...))))
```

はwindowのメソッドmoveを特殊化したメソッドを定義したことを意味する。

## 4. Object Lisp

Object Lisp はLMI から提案されたオブジェクトシステムである。

(1) クラスとインスタンス

Object Lisp の大きな特徴はクラスとインスタンスの区別が基本的には存在しないことである。言い換えれば、スーパの階層とクラス/インスタンスの階層が同一のものとして扱われている。実際、インスタンスを生成する関数make-objは、スーパクラスの階層を作り出す関数kindofの別名であるとされている。(しかし、クラス/インスタンスの区別は有用であることは

認められていて、表現上、区別して書く方法が提供されている。）例を以下に示す。

```
(setq icon (make-obj))
(setq icon1 (kindof icon))  (setq icon2 (kindof icon))
```

icon1, icon2はiconを特殊化したオブジェクトである。

(2)　オブジェクトの関数

Object Lisp でメソッドに相当するものはオブジェクトの関数と呼ばれている。オブジェクトの関数はdefobfunによって定義する。以下はiconの関数goto-xy の定義である。

```
(defobfun (goto-xy icon)(x y)
  (undraw-self x-coord y-coord) (draw-self x y)
  (setq x-coord x)(setq y-coord y))
```

関数の呼出しはask による。例を以下に示す。

```
(ask icon (goto-xy 100 100))
```

ask はメッセージ送信とは異なり、ブロックを形成して残りの式を順次評価する。まずgoto-xy がiconの関数として定義されているか否かが調べられる。もし定義されていればそれを実行し、なければグローバルな関数定義が実行される。

(3)　オブジェクトの変数

オブジェクトの変数にはhaveを用いて値を設定する。

```
(ask icon1  (have 'x-coord 0) (have 'y-coord 25)
            (have 'icon-array circular-design))
```

このように、オブジェクトの変数はオブジェクトを生成した後に定義される。

(4)　継承

オブジェクトの関数は継承される。（オブジェクトの変数に関しては継承という考え方がない。後述のスコープを参照。）

継承の規則は、Common Loopsの省略時の規則と同じである。（Flavors とは異なる。）まず以下の例を考える。

```
1:  (setq boundary-icon (kindof icon))
2:  (setq icon-with-trail (kindof icon))
3:  (setq boundary-icon-with-trail (kindof boundary-icon icon-with-trail))
```

1:, 2:によって、iconの特殊化されたオブジェクトであるboundary-icon とicon-with-trail は、iconの全関数を継承する。但し、同名の関数を定義することによってスーパの関数をshadowすることもできる。boundary-icon-with-trailを例にとると、オブジェクトの順位リストは以下の規則に従って作成される。

①　オブジェクトは常にそのスーパオブジェクトより前に位置する。

②　重複したオブジェクトは順位リスト中の後のものが残される。

重複したオブジェクトは後のものが残される。

したがって、boundary-icon-with-trailの順位リストは以下のようになる。

```
(boundary-icon-with-trail  boundary-icon icon-with-trail icon)
```

（注：後のものが残される理由は、各オブジェクトの局所順位をboundary-icon-with-trailの順位の中でも保存するためであ

ると考えられるが、この方法は完全ではない。）

(5)　関数の特殊化

Object Lisp ではスーパのオブジェクトの関数を特殊化する機能を備えている。以下は、icon-with-trail の関数goto-xy を、iconの関数goto-xy を特殊化することによって定義した例である。

```
1:  (defobfun (goto-xy icon-with-trail)(x y)
2:     (draw-line-on-graphics-screen x-coord y-coord x y)
3:     (shadowed-goto-xy x y))
```

3:のshadowed-goto-xyは、icon-with-trail のスーパオブジェクトであるiconのgoto-xy を指している。（Object Lisp ではメソッド結合という用語は用いられていないが、shadowedはCommon Loopsのrun-super に相当するものである。）

(6)　スコープ

Object Lisp のスコープルールは特徴的である。Object Lisp ではlexical binding （以下の例ではlet ）がobject binding （ask ）より優先する。いま、オブジェクトobj が((a 10)(b 10))というbinding を持つとする。すると、

```
1:  (let ((a 3))  (ask obj
2:                (setq a (1+ a)) (print a)
3:                (setq b (1+ b)) (print b)))
```

2:は 4を出力し、3:は11を出力する。もう１つ例をあげる。

```
(ask icon1 (let ((x x-coord)(y y-coord)) (ask icon2 (goto-xy x y))))
```

このようにすればicon2 はicon1 と同じ位置に来る。しかし、

```
(ask icon1 (ask icon2 (goto-xy x-coord y-coord)))
```

と書いたのでは、x-coord, y-coordはicon2 の変数を指すのでicon2 の位置は変わらない。

## 5.　Snyderの提案

この章で述べるオブジェクトシステムは、HPのAlan Snyder の提案によるものである。

(1)　クラス、インスタンス、メソッド

Snyder案では、クラス、インスタンス、メソッドはそれぞれ関数define-type,make-instance, define-methodで定義する。(Snyder 案ではクラスを型（type）と呼んでいる。）メッセージの送信は、

```
(=> type method-name arguments... )
```

で表わす。この関数はsetfでも使える。

```
(setf (=> x :foo a b c) v)
```

(2)　インスタンス変数の継承

Snyder案の最大の特徴はカプセル化(Encapsulation) の重視である。継承によって名前が衝突し、どちらか一方を何らかの規則で選択する（例えばCommon Loopsのクラス順位リスト）という事態を避ける工夫がされている。

まずインスタンス変数の例を示す。

```
1:  (define-type parent
2:     (:var a)(:var b))
```

```
3:  (define-type child
4:     (:inherit-from parent)
5:     (:var a)(:var c))
```

4:はchild のスーパ型がparentであることを表わしている。この場合、child のインスタンス変数は、a, b, c という 3個の変数を持つのではない。 4個の変数を持つのである。parentで定義されたa, bは、parentで定義されたメソッドでのみアクセス可能である。child で定義されたa, cはchild で定義されたメソッドでのみアクセス可能である。

もう一つ例を挙げる。例えば、child がparent1 とparent2 をスーパ型として持ち、parent2 がparent1 をスーパ型として持つと、child はparent1 のインスタンス変数を2個ずつ持つことになる。上記の仕様は明らかに他のオブジェクトシステムの仕様とは異なっている。これは、カプセル化（即ち、他のオブジェクトと独立にオブジェクトを定義できる）を追求した結果である。逆にこの仕様に従うならば、継承によって不必要な重複が生じないように型の階層を構成しなければならない。

(3)　メソッドの継承

メソッドの起動には、スーパ型の名前を陽に指定するcall-method,apply-methodがある。call-method,apply-method中で起動されたメソッドのselfは、呼び出し元のselfと同じものとなる。スーパ型のメソッドは全て継承される。今、 child がスーパ型であるparentからメソッド :editを継承するとは、次のメソッドが定義されたことと等価である。

```
(define-method (child :edit)(&rest args ) (apply-method (parent :edit)args))
```

snyder案では多重継承により、同名のメソッドが複数継承されるとエラーとなる。このような場合には、メソッドの継承を制御する次のような手段がある。

① 継承するメソッドを制限する。

define-type の :inherit-fromオプションで制限を記述する。

```
(define-type child(:inherit-from parent (:methods [:except] { symbols})))
```

ここで、 :exceptの指定がない場合には、symbols で指定されたメソッドが継承される。逆に、 :exceptの指定がある場合には、指定されたメソッドは継承されない。

② 継承するメソッドを再定義する。

define-type の :redefined-methods オプションで指定する。

```
(define-type child(:redefined-methods {symbols})... )
```

この場合には、symbols で指定されたメソッドがこの型で再定義される。

## 6.　new Flavors

new Flavors は、Symbolics によって85年12月のCommon Lisp 委員会に報告されている。

(1)　flavor, インスタンスflavor

Flavors では、クラスという用語の代りにflavorが用いられている。flavorは defflavor で定義され、そのインスタンスはmake-instance で生成される。例を示す。

```
1:  (defflavor 3-d-moving-object (x-velocity y-velocity z-velocity)
2:        ()                          ; component flavor
3:        :initable-instance-variables)
```

```
4:  (defflavor comet (percent-iron estimated-mass)
5:          (3-d-moving-object)       ; component flavor
6:         :initable-instance-variables)
7:  (make-instance 'comet :estimated-mass 27  :x-velocity 12  :y-velocity 44  :z-velocity 87)
```

1:は3-d-moving-object, 4: はcomet の定義を表わしている。5:はcomet のスーパが3-d-moving-object であることを示している。7:はcomet のインスタンスを生成し、変数に値を初期設定している。

(2) メソッド

new Flavors ではLisp構文との親和性を高めるために、従来使用してきたsend構文をとりやめ、以下に示す関数呼び出しの形式でメソッドを起動することにしている。(したがって、 Common Loops との親和性も良くなっている。)

```
(function-name  arguments...  )
```

この関数はgeneric functionと呼ばれる。generic functionの呼び出しは第 1引数のflavorのメソッドを起動することを意味する。メソッドは以下のように定義される。

```
(defmethod (speed 3-d-moving-object) ( )
        (sqrt ( + (expt x-velocity 2) (expt y-velocity 2) (expt z-velocity 2))))
```

これは3-d-moving-object のメソッドspeed を定義している。このようにして定義されたメソッドをprimary mothodと呼ぶ。

new Flavors では、gereric functionと一般のLisp関数が、構文的にも意味的にも互換性を有している。即ち、①packageはgeneric functionを通常のfunctionと同様に扱う、②funcall やmapcarの第1引数となる、③trace のようなプログラム開発用のツールを適用できる。

また種々のオプションを指定してgeneric functionを定義する場合のために、defmethod 以外にdefgenericが用意されている。

(3) メソッド結合

new Flavors のクラス順位リストの作成方法はCommon LoopsやObject Lisp 、従来のFlavors とは異なり、以下の規則が守られるよう改められている。

① flavorは、常にそのconponent flavorより前の位置に置かれる。

② 全体の順位リストが局所的な順位を保存する。

③ 重複したflavorは除去される。

Flavors の最大の特徴はメソッド結合にある。Flavors ではprimary methodの他にbefore method とafter method を定義できる。例を以下に示す。

```
(defmethod (launch rocket) () (setq position "Space") ;primary method
(defmethod (launch rocket :before) ()                 ;before method
        ;; add fuel before launching
        (setq tank "Full"))
(defmethod (launch rocket :after) ()                  ;after method
        ;; start radar tracking
```

```
(setq *radar-tracking* t))
```

典型的なメソッド結合(:daemon型のメソッド結合と呼ばれている)は、以下のように行われる。

① 全てのbefore method がbase-flavor-last-order (クラス順位リストの順) で実行される。

② primary methodが何らかの規準で1つ選択されて実行される。(例えば base-flavor-last-order で選択される。)

③ 全てのafter methodがbase-flavor-first-order (クラス順位リストの逆順) で実行される。

Flavors は、daemon型以外にも、さまざまなメソッド結合の方法を組み込みで提供している。new Flavors の1つの特徴は、メソッド結合の方法を定義するdefine-method-combination を提供したことである。(この機能はCommon Loopsのメタクラスの機能を思い起こさせる。) システム提供のメソッド結合の型は全て、この関数により定義されている。以下の例は、daemon型のメソッド結合を定義した例である。

```
(define-method-combination :daemon (&optinal (order ':base-flavor-last))
  ((before "before" :every :base-flavor-Last (:before))
          (primary "primary" :first order () :default)
          (after "after" :every :base-flavor-first(:after)))
  `(flavor:multiple-value-prog2  (flavor:call-component-methods ,before)
                                 (flavor:call-component-method ,primary)
                                 (flavor:call-component-methods ,after)))
```

## 7. むすび

最近の米国Common Lisp 委員会でのオブジェクト指向機能導入の動向について述べた。現在も活発な討論が、主として電子掲示板を通じて継続されており、Common Loopsを用いて、Loops, Flavors, Object Lisp 等複数のオブジェクトシステムを記述する試みも行われている模様である。

本報告は、60年度の日本電子工業振興協会Common Lisp 動向専門委員会の調査結果[5] をもとにまとめたものである。最後に、調査活動をサポートして戴いた電子協の鈴木　博氏、資料を提供して戴いた日本シンボリックス社の西本博明氏に感謝する。

## 参考文献

[1] D.G.Bobrow, K.Kahn, G.Kiczales, L.Masinter, M.Stefik and F. Zdybel : COMMONLOOPS : Merging COMMON LISP and Object-Oriented Programming,Xerox Palo Alto Research Center, ISL-85-8, Aug., 1985.

[2] G.L.Drescher : ObjectLISP for Experienced LISP Programmers, LMI documentation, Aug, 1985.

[3] A.Snyder : Object-Oriented Programming for Common Lisp, Application Technology Center,Hewlett-Packard Laboratories, ATC-85-1, July, 1985.

[4] S.E.Keene and D.A.Moon : Flavors :Object-oriented Programming on Symbolics Computers, Symbolics Inc., Dec., 1985.

[5] 昭和60年度日本電子工業振興協会CommonLisp動向専門委員会調査報告.