6U－1　ISO規格ISLISP処理系における オブジェクトシステムの実装について

各務 寛之* 山田 雅彦* 高橋 順一* 五味 弘* 新谷 義弘** 長坂 篤** 梅村 恭司*** 湯浅 太一****

*(株) 沖テクノシステムズラボラトリ　** 沖電気工業 (株)
*** 豊橋技術科学大学　**** 京都大学

1　はじめに

我々は，Windows95/NT4.0およびUNIX上で動作するISLISP処理系の実装を行っており，オブジェクトシステムを除いたインタプリタの実装方式については既に報告した [1], [2] .

ISLISPのオブジェクトシステムは，COMMON LISPのオブジェクトシステムであるCLOS [6] のサブセットであり，CLOSと比較して機能が制限されているため，効率的な処理系の実装が可能となる.

また，ISLISPの仕様 [5] は処理系依存になっている部分が多いため，その部分をどのような仕様にするかによって実装に大きな影響を与える.

本稿では，本処理系におけるオブジェクトシステムの実装について述べる．特に効率的なクラスの再定義のために導入した，古いインスタンスの管理用のクラスやウィークポインタについて述べる.

2　ISLISPのオブジェクトシステム

ISLISPのオブジェクトシステムは，CLOSのサブセットになっており，包括関数 (CLOSでは総称関数) を動作の基本としたオブジェクトシステムであり，完全なオブジェクトシステムである.

ISLISPはほとんどのメタオブジェクトプロトコルが処理系依存になっているため，依存部をうまく定義することによりCLOSと比較してコンパクトになり，効率的な処理系の実装が可能となる．またISLISPの仕様では，クラスの再定義などの動的変更については処理系依存になっている.

本処理系では，ISLISPの言語仕様の処理系依存の主な部分を以下のように定義した.

1. ユーザ定義クラスのメタクラスは，
   `<standard-class>`のみ
2. ユーザ定義包括関数のメタクラスは，
   `<standard-generic-function>`のみ
3. クラスの再定義を可能にする
4. 関数，メソッドの再定義を可能にする

1と2により，メタクラスの新規作成を制限することにより，メタクラスのオーバヘッドをなくしている．これは，通常のLISPプログラムにおいては妥当な制限になっている.

このように定義することにより，コンパクトな処理系の実装ができる．我々はオブジェクトシステムを含む処理系のすべてをC言語で記述することにより，高速な処理系を作成した.

また，ISLISPは動的変更についての仕様が処理系依存になっているが，クラスの再定義等は必須な仕様と考え，3,4のように再定義可能にした.

3　クラスの再定義

第2節で述べたが，本処理系ではクラスの再定義を可能とした．ここでは，効率の良いクラスの再定義の実装について述べる.

3.1　インスタンス，サブクラスの変更

クラスが再定義されたとき，そのクラスのすべてのインスタンスとそのクラスのすべてのサブクラスのすべてのインスタンスの変更を行う必要がある．この変更の方法には以下のものがある.

- 一括型:
  クラスの再定義が行われたときに，そのクラスのインスタンスとそのクラスの全てのサブクラスのインスタンスの変更を行う．この方式では，再定義の処理には時間がかかるが，インスタンスの操作は高速である.

- 逐次型:
  変更されるクラスやインスタンスの操作が行われたときに変更を行う．この方式では，再定義の処理は高速であるが，インスタンスの操作は遅い.

本処理系では，実行時 (インスタンスアクセス時) の高速化のために一括型を採用した．また，再定義の処理を高速にするために，後述のウィークポインタとエラー処理用クラスを導入した.

"An implementation of ISO-standardized ISLISP object system"
Hiroyuki KAGAMI*, Masahiko YAMADA*, Junichi TAKAHASHI*, Hiroshi GOMI*, Yoshihiro SHINTANI**, Atsushi NAGASAKA**, Kyoji UMEMURA***, Taiichi YUASA****

*Oki Technosystems Laboratory, Inc.
**Oki Electric Industry Co., Ltd.
***Toyohashi University of Technology
****Kyoto University

3.2 エラー処理用クラス<invalid>

前述したように，本処理系ではクラスの再定義を可能にする仕様にした．クラスの再定義を可能にすることにより，以前のクラス定義で生成されたインスタンスをどのように扱うかを決めなければならない．

本処理系では，以前のクラス定義で生成されたインスタンスに対する操作はエラーを通知するようにした．そのために，特別なエラー処理用のクラス<invalid>を導入した．

本処理系では，クラスが再定義されたとき，以前のクラス定義で生成されたインスタンスはクラス<invalid>のインスタンスに変換するようにした．そして，クラス<invalid>のインスタンスに対する操作はエラーを通知するようにした．図1にクラス再定義時のエラー通知の例を示す．

```
;;; クラス circle を定義
ISLisp>(defclass circle (figure)
         ((radius :accessor circle-radius
                  :initarg radius)))
CIRCLE

;;; クラス circle のインスタンスを生成し，
;;; グローバル変数 fig1 に設定する．
ISLisp>(defglobal fig1 (create (class circle) 'radius 1))
FIG1

;;; fig1 の半径を調べる
ISLisp>(circle-radius fig1)
1

;;; クラス circle を再定義する．
;;; ここで，以前の定義で生成されたインスタンス fig1 は
;;; クラス<invalid>のインスタンスに変換される．
ISLisp>(defclass circle (figure)
         ((radius :accessor circle-radius
                  :initarg radius
                  :initform 1)))
CIRCLE

;;; fig1 の半径を調べる
;;; しかし，fig1 はクラス<invalid>のインスタンスなので
;;; これに対する操作はエラーが通知される．
ISLisp>(circle-radius fig1)
> Error at CIRCLE-RADIUS
> No applicable method:
```

図 1: クラス<invalid>のインスタンスの操作例

クラス<invalid>の導入により，インスタンスのアクセス時に古いインスタンスであるかどうかのチェックが不要となり，操作が高速に行うことができる．

4 ウィークポインタ

ウィークポインタのみで参照されているオブジェクトはゴミになり，ガーベッジコレクション (ゴミ集め) により領域の再利用ができる

本処理系では，全てのインスタンスをクラスからのウィークポインタで管理する．これによりクラスの再定義が行われたときは，クラスからウィークポインタをたどることにより，そのクラスに属するすべてのインスタンスをクラス<invalid>に変更する (一括型変更) 操作を高速に行うことができる．

このウィークポインタを通常のポインタで管理すると，インスタンスは不要になってもゴミにならないが，ウィークポインタを導入すると使われていないインスタンスはゴミになって回収され，効率的なメモリ利用が行われる (図 2)．

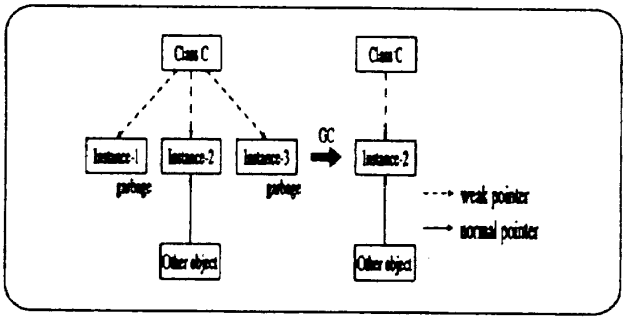

図 2: ウィークポインタ

5 おわりに

現在開発中の Windows95/NT4.0 および主な UNIX (Solaris, HP-UX, Linux, FreeBSD) で動作する ISLISP 処理系のオブジェクトシステムの実装について述べた．本実装では，

- C 言語によるオブジェクトシステムの記述
- ウィークポインタの導入
- エラー処理用クラス<invalid>の導入

により，実行効率の向上を実現している．

現在，バイトコードコンパイラを作成しており，オブジェクトシステムと同様に，全てを C 言語で記述している．

今後は本処理系に他システム，他言語との外部インタフェース機能を実装する予定であり，現在検討中である．

最後に，本開発は IPA の「独創的先進的情報技術に関わる研究開発」事業の一環として行われた [4]．ここに関係諸氏に感謝いたします．

参考文献

[1] 各務寛之他: ISO 規格 ISLISP 処理系の実装方式, 情報処理学会第 56 回全国大会 5E-07, 1998 年 3 月

[2] 新谷義弘他: ISO 規格 ISLISP 処理系の開発, 情報処理学会第 56 回全国大会 5E-06, 1998 年 3 月

[3] 伊藤貴康: LISP 言語国際標準化と日本の貢献, 情報処理 38 巻 10 号, 1997 年 10 月

[4] Lisp の ISO 標準規格 ISLisp の処理系の研究開発, 情報処理振興事業協会 第 16 回技術発表会論文集, pp.339-343, 1997 年 10 月

[5] ISO/IEC 13186:1997, "Information technology — Programming languages, their environments and system software interfaces — Programming language ISLISP", 1997

[6] Guy L. Steele Jr., "COMMON LISP THE LANGUAGE Second Edition", Digital Press, 1990