3E−2　例題によるCLOSの利用実験と評価

○湯浦克彦((株)日立製作所)、高橋久(日立超LSIエンジニアリング(株))、筑田隆広((株)明電舎)、
菅野幹人(三菱電機(株))、六条範俊(富士通(株))、川辺治之(日本ユニシス(株))

1　はじめに

ユーザインタフェースをはじめとする幅広い分野でオブジェクト指向によるプログラミングが注目され、いくつかの言語が提案されている。そのなかでCLOS (Common Lisp Object System)[1]は、多重継承、メソド結合、総称関数などの特徴機能を有し、プログラムの記述性を一層向上させるものとして期待されている。本稿では、CLOSを用いて例題を実際に記述した結果と、そこでの上記特徴機能の効果について述べる。なお、本実験は(社)電子協Lisp応用技術WGの活動として実施されたものである[2]。

2　CLOSの利用実験

以下の5題について記述を実施した。それぞれ、100より数百行の規模となった。

(1) パソコン通信センタプログラム

回線とセンタがあり、センタでは各ユーザがログイン処理状態とインサービス処理状態、インサービス処理状態ではさらにメンバとゲストが識別される。そこで、回線よりの着信をそのユーザの状態に応じて処理していくものである(図1)。

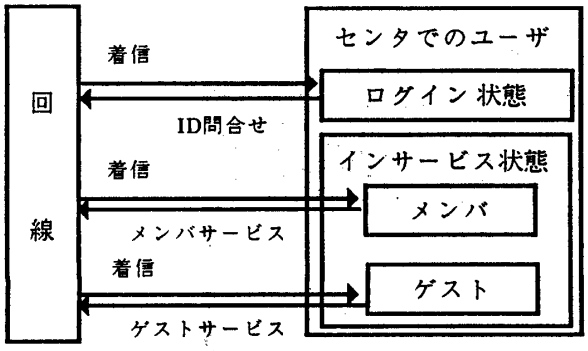

図1 例題(1)パソコン通信センタプログラムの概要

(2) アセンブラ

命令群(表1)のニーモニックよりバイナリコードを生成するもので、命令の識別、ラベルの処理をする第一パスと、バイナリを出力する第二パスよりなる。

表1 例題(2)アセンブラの処理する命令群

大分類	中分類	小分類
instruction	template-1	load
		store
		add
		subtract
		jump-on-condition
	template-2	add-register
		subtract-register
pseudo-code		define-constant
		end

(3) オブジェクト･インスペクタ

CLOSのオブジェクト(standard-object, standard-classなど)の専用インスペクタおよびトレーサである。

(4) 階層型ウィンドウ制御

階層関係にあるウィンドウ間でのウィンドウの移動を制御するものである。子の動きに合わせて親を変形させたり、親に合わせて子をシュリンクさせたりする。

(5) 簡易グラフィックシステム[3]

多角形、円弧、文字列などの図形要素を画面表示および編集するものである。

3　評価

3.1　CLOS利用のメリット

(1)「もの」中心の分かりやすいプログラム構成

問題の定義に用いる「もの」や概念をそのままクラスとすることができた。つまり、例題(1)では回線とユーザ、例題(2)では各命令、例題(4)ではウィンドウの種類、例題(5)では各図形要素などである。

CLOSでは、以下の点でさらにオブジェクト指向設計の幅を広げることができた。例題(1)においてユーザの状態の違いにより別のクラスを設けているが、これは状態の変化をCLOSのchange-class(オブジェクトのクラス変更)により実現することで可能になっている。

An Experimental Evaluation of CLOS Programming

K. YUURA[1], H. TAKAHASHI[2], T. CHIKUDA[3], M. KANNO[4], N. ROKUJOU[5] and H. KAWABE[6]
[1]Hitachi Ltd., [2]Hitachi VLSI Engineering Corp., [3]Meidensha Corp., [4]Mitsubishi Elec. Co. Ltd.,
[5]Fujitsu Ltd., [6]Nihon Unisys Ltd.

(2)総称関数による手続の統合化

各例題の中心となる手続群を総称関数で統合化することができた。つまり、例題(1)での各ユーザ状態での着信処理、例題(2)での各命令毎の第一パス、第二パス、例題(3)での各データ型毎のインスペクタの対話処理および検索処理、例題(4)でのウィンドウ移動の判定処理、例題(5)での各図形の移動、表示などである。

CLOS特徴機能に関する効果としては、Lispの組込み型のメソドが定義できたこと(例題(3))、複数の引数を型指定するメソド(例題(4)の親ウィンドウと子ウィンドウを引数とするメソド)が利用できたことが挙げられる。また、メソド結合の付加メソドでエラー処理などの関連処理を記述し、これらも含めて総称関数を定義することができた(例題(2),(3),(5))。

(3)多重継承、メソド結合による記述量削減

問題の定義に現れる「もの」の分類系統そのままに継承木を構成することで、クラスやメソドの記述の共通化を図ることができた(例題(1),(2))。問題の定義に現れる「もの」の共通属性を分析して得た抽象的なクラスを設定すると、継承の効果はさらに大きい(例題(4),(5))。抽象的なクラスの継承では多重継承が頻繁に利用された(例題(5)、色クラス、角度クラス等の各図形クラスへの継承)。

メソドの継承では、上位クラスのメソドを単純に継承するケースよりも、継承メソドと自己クラスのメソドを結合して利用するケースが多かった(例題(2),(4),(5))。

3.2 定量化

各例題でのクラス再利用率およびメソド再利用率3)を図2に示す。これらは、それぞれ一度定義したデータ構造および手続が継承によって何倍に活用されているかを示すことになる。

クラス再利用率
=(実クラス‡数+クラス継承数)/クラス数

メソド再利用率
=(実クラス‡のメソド数+メソド継承数)
/メソド数

(‡;実際にオブジェクトを持つクラス)

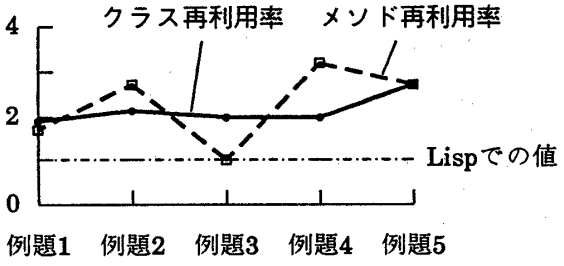

図2 再利用率

さらに、プログラムの簡潔さを示すものとして以下の指標を導入する。

プログラム要素表現効率
=プログラム要素数/ユーザ記号数

実効手続き表現効率
=実効手続き数/ユーザ記号数

ただし、

ユーザ記号数=クラス数+総称関数の数†
+関数の数+大域変数の数
(†;自動生成されるsetf用総称関数をのぞく)

プログラム要素数=クラス数+メソド数
+関数の数+大域変数の数

実効手続き数=実効メソド数§+関数の数
(§;実クラスでの継承を含んだ結合メソド数)

例題におけるプログラム要素表現効率および実効手続き表現効率の値を図3に示す。総称関数、メソド結合および継承のない言語(例えば、オブジェクト指向なしのただのLisp)では、これらの値または上限値は1である。CLOSでは、少ないユーザ記号で多くのプログラム要素および実効手続きを表現する効果を上げていることがわかる。

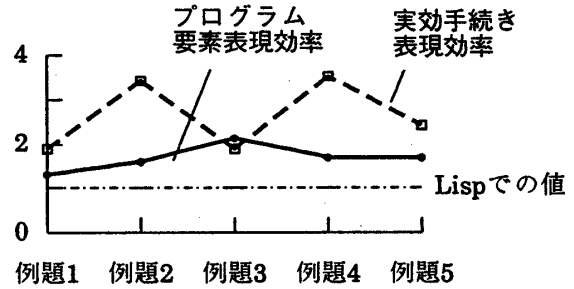

図3 記号による表現効率

4 おわりに

オブジェクト指向の適合する範囲は広い。さらに、CLOSの特徴機能も多くの場面で有効に利用可能であり、プログラムの記述性や簡潔さを一層向上させると考えられる。

参考文献

1) D., G., Bobrow, et., al., *Common Lisp Object System Specification,* ANSI X3J13 Document 88-002R, 1988.
2) 湯浦外、CLOSの利用法について、日本電子振興協会編、マイコン技術フォーラムproceeding、pp. 128-145, 1990.
3) 湯浦外、CLOSの記述性について、情報処理学会第39回全国大会論文集、pp. 1352-1353, 1989.