5Q-8

CLOSの記述性について

○湯浦克彦((株)日立製作所 中央研究所)
船津隆(日立ソフトウエアエンジニアリング(株))
高橋久(日立超LSIエンジニアリング(株))

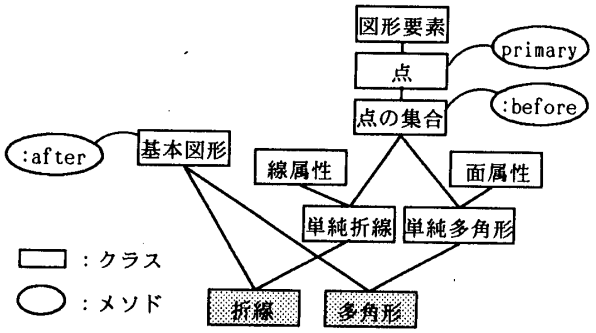
1．はじめに

オブジェクト指向は、Smalltalk以来いくつかの言語が提案されている。その中でCLOS(Common Lisp Object System)[1]は、多重継承、メソド結合に特徴があり、これらを用いた差分プログラミングにより効率良くプログラムを作成することを目指したものと言えよう。本稿では、CLOSによる差分プログラミングの実験、分析と従来のCommon Lispでの記述との比較を述べる。

2．ＣＬＯＳによる記述実験

HOBJ-2プロトタイプ[2]を用いてCLOSの記述実験を行った。例題は、多角形、円弧、文字列などの画面要素、編集システムであり、図１のような継承木を構成する結果となった。この継承木は、外部仕様として実験者に与えたものではない。下線を引いたクラスの画面要素の編集機能を実現するために、多くの部分は初期設計の段階で構成した。また、作成する過程でクラス間の共通属性を抽出して構成していった部分も少なくない。

この継承木の上でのメソド結合の利用例を図２に示す。この例では、移動の前置メソド(:before)、基本メソド(primary)、後置メソド(:after)の各メソドを折線と多角形でそれぞれの経路で継承し、移動の機能を実現（前置メソド、基本メソド、後置メソドの順に実行）している。



:before ･･･ 各頂点のｘ方向、ｙ方向の移動分を計算する
primary ･･･ 現在位置の座標を移動先の座標に置き換える。
:after ･･･ 現在、ウィンドウに表示されている対象図形を消去し、移動先位置に図形を表示する。

図２　メソド結合の利用例（移動機能）

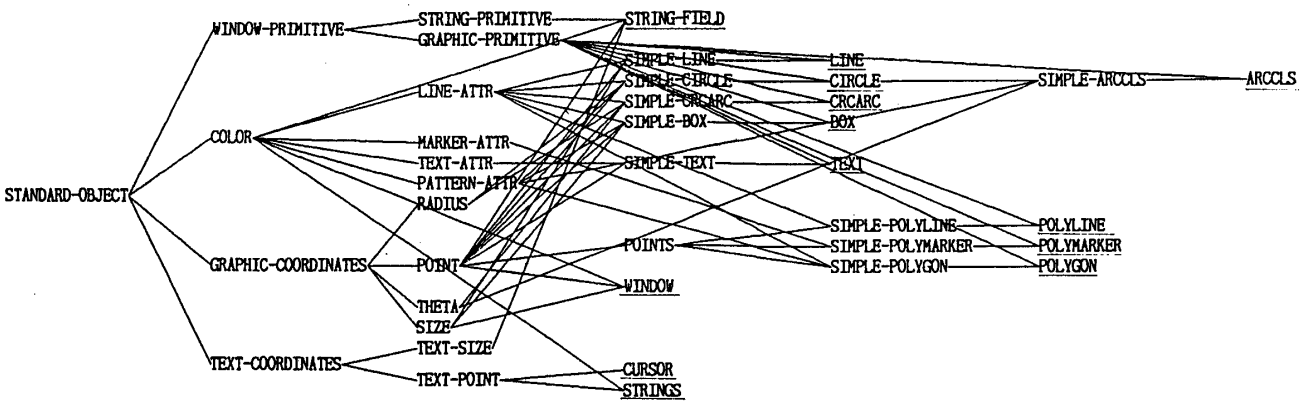


図１　例題の多重継承木（簡易グラフィックシステム）

Advantage of Code Reuse in CLOS Programming
Katsuhiko YUURA[1],Takashi FUNATSU[2],Hisashi TAKAHASHI[3]
[1]Central Research Laboratory, Hitachi, Ltd., [2]Hitachi Software Engineering Co. Ltd.,
[3]Hitachi VLSI Engineering Corp.

3．再利用率

多重継承とメソド結合によるプログラムの再利用の効率を定量化する。

（1）クラス再利用率

＝ クラス定義参照数／クラス定義数

＝ １０７／３９ ＝ ２．７

ここで、クラス定義数は、プログラムに含まれるクラス定義の総数である。クラス定義参照数は「直接、間接に継承するクラス数＋１（自分自身）」の総計である。ただし、継承木の中には、自分自身はインスタンスを持たない継承の中継ノード的なクラス（例題では、図１の下線のないクラス）がある。これらのクラスでのクラス定義参照は、記述性向上に寄与しないので、クラス定義参照数０とする。例題のクラス再利用率は2.7であった。

（2）スロット再利用率

＝ スロット定義参照数／スロット定義数

＝ １１１／４３ ＝ ２．６

スロット定義に焦点を絞った再利用率であり、算出法はクラス再利用率と同様である。例題では、2.6であり、クラス再利用率と近い値となった。

（3）メソド再利用率

＝メソド定義参照数／メソド定義数

＝２８８／９９＝２．９

メソド定義に焦点を絞った再利用率であり、算出法はクラス再利用率とほぼ同様である。ただし、メソド結合に関しては、補助メソドおよびcall-next-method形式で呼ばれるものも参照数１としてカウントした。

（4）メソド結合利用率

＝ メソド結合数／メソド定義参照数

＝ １８３／２８８ ＝ ６４％

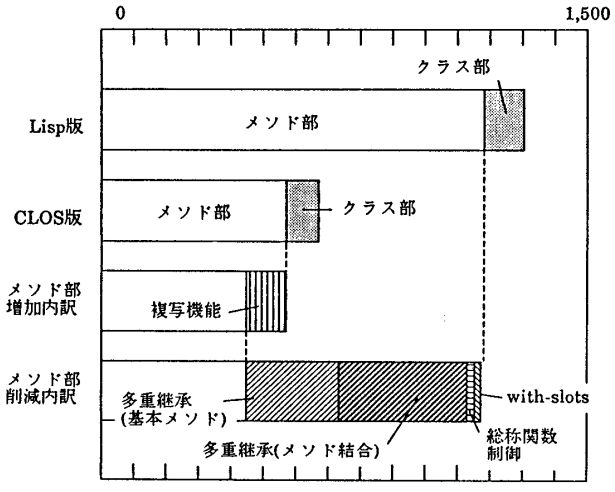


図3 例題におけるLispとの記述量比較(単位 ステップ)

メソド定義参照数のうち、基本メソドの定義参照数と、補助メソドまたは非標準メソド結合タイプのメソドの定義参照数（メソド結合数）の比率を求めたものである。この例題では、半数以上がメソド結合での再利用となっている。

4．Ｌｉｓｐによる記述と比較

例題をLispのみで記述してみた。コーディングは以下のような基準に従って行った。

（1）クラスのスロット定義は構造体で行う。構造体では:includeを用いずに同名スロットもそれぞれに定義する。そこで、CLOSで中継ノード的に設定したクラスのスロットについては記述を省くことになる。

（2）メソドはクラスごとに関数として定義する。これも中継ノード的なクラスへのメソドは除くことにする。メソド結合機能を利用している場合には、結合した総体を一つの関数として定義する。

（3）総称関数制御は、呼出し側で型判定し型毎の関数を呼び分けるようにして同等の機能を実現する。

CLOS版とLisp版の記述量の差を図３に示す。記述量総数では、CLOS版は、Lisp版の約２分の１で済んでいる。CLOS版およびLisp版は、クラスのスロット定義の部分(クラス部)とメソド定義の部分(メソド部)に分けられる。クラス部では、CLOS版がやや少ない程度となっている。これは、CLOS版ではスロット定義の再利用はされているが、継承利用のためにクラス数が増えているからである。各defclass形式の一定量の記述の分でスロット再利用効果が相殺されている。

メソド部は全体に占める割合が高く、ここでの２倍の差が全体の記述量低減に効いている。メソド部で差の原因を分析すると、多重継承とりわけメソド結合による効果が過半数を占めていた。その他では、Lispで総称関数制御のための型判定記述による差、with-slots形式によるスロット参照簡略化による差が少々あった。逆に、Lisp構造体ではコピア（構造体の複写関数）が自動生成されるのに対し、CLOSではこれがないため複写のメソドを用意することになり、記述量が増えている部分もあった。

5．おわりに

今回の実験では、メソド結合による手続きの木目細かい再利用が効果を上げた。この例題の拡張または大型のプログラムでは、再利用性はさらに向上するはずである。

参考文献

1)D.G.Bobrow et al.;Common Lisp Object System Specification,Draft X3J13 Document 88-002R,1988.

2)湯浦、外:CLOS準拠のHiOBJ-2の概要、情報処理学会第39回全国大会、1989.