2M−7

CLOSの実現

景山 辰郎* 河津 祐子* 高橋 陽徳** 佐治 信之*

* 日本電気(株) C&C共通ソフトウェア開発本部

** 日本電気マイコンテクノロジー(株)

1 はじめに

筆者らは, Common Lisp上のオブジェクト指向言語拡張[1]であるCLOS[2]を実現した.

本稿では, はじめにCLOSの基本的なオブジェクトの構造について述べる. 次に, 処理系の実行効率の向上を主眼にメソッドの適用について述べる. 次に, CLOSの振舞いを決定するメタオブジェクトプロトコルについて述べる.

2 オブジェクトの構造

2.1 標準オブジェクト

インスタンスの構造は, そのクラスが再定義されることによって変更される. このために, インスタンスは自分が生成(変更)された時のクラスの状態を保持しており, インスタンスをアクセスする時に現在のクラスの状態と比較して, クラスが再定義されているかどうかを判定し, 再定義されている場合にはインスタンスの構造を変更することによって, クラスが全インスタンスを保持しなくても良いようにする. 図1に標準オブジェクトの構造を示す. [3]

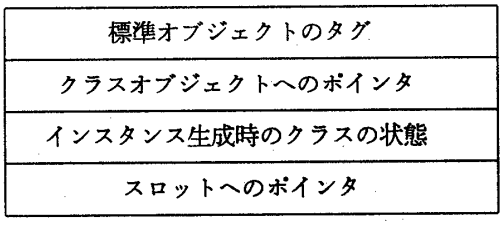

図1: 標準オブジェクトの構造

2.2 関数呼出し可能な標準オブジェクト

CLOSでは, 総称関数と呼ばれる関数が追加されている. 総称関数は, 関数であると同時に標準オブジェクトとしてアクセス可能でもある. 総称関数を含む関数呼出し可能な標準オブジェクトは, その環境に標準オブジェクトを持って関数呼出し可能な標準オブジェクトの情報を保持しているクロージャで実現する. 図2に関数呼出し可能な標準オブジェクトの構造を示す.

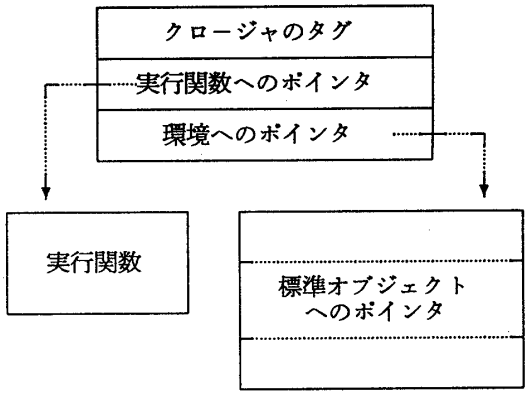

図2: 関数呼出し可能な標準オブジェクトの構造

3 メソッドの適用

CLOSでは, 総称関数に渡された引数によって, 総称関数に登録されたメソッドの中から適当なものを見つけて実行する.

3.1 実効メソッド

CLOSでは, 適用可能なメソッドの特定化された順にソートされたリストから実効メソッドを得る. 実効メソッドはcompute-effective-methodを実行することにより得られるが, compute-effective-methodの返却値は定義されていないため, 筆者らの処理系では実効メソッドをクロージャとして実行効率の向上を図る. 標準メソッド結合に対する実効メソッドは, テンプレートを用意することで更に実行効率が向上する.

"An Implementation of CLOS"

Tatsuro KAGEYAMA*, Yuko KAWATSU*, Youtoku TAKAHASHI**, Nobuyuki SAJI*

* C&C Common Software Development Laboratory, NEC Corporation,

** NEC Microcomputer Technology, Ltd.

[1] X3J11委員会によって"Common Lisp Object System Specification"[1]の第1章および第2章がCommon Lisp言語に含められた[2].

[2] Common Lisp Object System

[3] 筆者らの処理系では各Common Lisp処理系に依存しないデータ構造を実現するために, 標準オブジェクトを既存の型, つまりvectorで代用する. vectorの第0要素に標準オブジェクトであることを表すタグを入れる.

3.2 メソッドキャッシュ

総称関数は，受け取った引数から適用可能メソッドのリストを計算する．更に，適用可能メソッドのリストと，総称関数のメソッド結合から実効メソッドを計算する．

ここで，総称関数のメソッドには，eql 引数特定子を持つものと持たないものがある．eql 引数特定子を持たないメソッドについて，引数の型のリストをキーとし，適用可能メソッドのリストを値とするキャッシュを持てば，総称関数が呼び出されるたびに，適用可能であるかどうか計算する必要はなくなる．しかし，eql 引数特定子を持つメソッドについては，毎回適用可能であるかどうか計算しなければならない．

また，実効メソッドは，適用可能メソッドのリストより一意に決定することが可能である．そこで，適用可能メソッドのリストをキーとし，実効メソッドを値とするキャッシュを持てば，この部分での実行効率の向上を図ることが可能となる．

3.3 call-next-method

call-next-method は，仕様では大域関数となっているがその意味するところは，各メソッドがそれぞれ call-next-method を局所関数として持っていて，メソッドの実行時にその定義が決定される．筆者らの処理系では他の処理系[4]と同様に，メソッドの中を解析して，そのメソッドの中で call-next-method が使用されていれば，call-next-method の定義を与えるようにメソッドを変更することで，無駄に call-next-method を定義することを回避する．

next-method-p についても同様である．

3.4 キーワード引数の妥当性

CLOS の仕様では，各メソッドのラムダリストで独自のキーワード引数を受け付けて良いことになっており，総称関数は実際に呼び出された引数によって適用可能メソッドを計算し，すべての適用可能メソッドのラムダリストのキーワード引数から，引数の妥当性のチェックを行わなければならない．しかしこの仕様を受け付けることによって情報の隠蔽を損なう可能性がある．つまり各メソッドのラムダリストにあるキーワード引数を知っていなければ総称関数を正しく呼び出せないことが起こり得る．また，キーワード引数の妥当性のチェックを行うことによる実行効率の低下を考え，筆者らの処理系では実現していない．

4 メタオブジェクトプロトコル

メタオブジェクトプロトコルには，いまだ仕様上の不備が残っており，また “Common Lisp Object System Specification”[1] の第 1 章および第 2 章と矛盾する部分もある．

[4]PCL (Portable CommonLoops)

筆者らの処理系では，仕様の不明瞭な部分を除いて，ほぼメタオブジェクトプロトコルをサポートしている．このことにより，処理系の振舞いをかなり自由に変更することが可能になっている．

4.1 初期化

筆者らの処理系では，メタオブジェクトプロトコル[5]の定義にしたがって，クラスオブジェクト，総称関数オブジェクト，メソッドオブジェクト，メソッド結合オブジェクト，スロット定義オブジェクトの生成，初期化，再初期化を行っている．

4.2 スロットアクセス

メタオブジェクトプロトコルによれば，関数 slot-value は総称関数 slot-value-using-class を呼び出すことになっており，筆者らの処理系でもこの仕様を実現している．実際に総称関数が実行された時には，メソッドを呼び出すために，総称関数オブジェクトのスロットをアクセスする必要がある．しかし，対象となっているオブジェクトのクラスが明らかである場合には slot-value を用いる必要はないので，総称関数 slot-value-using-class を実行する際に，スロットをアクセスするために関数 slot-value を呼び出さなくとも良くなり，slot-value-using-class の実現が可能となる．

5 おわりに

Common Lisp 上のオブジェクト指向言語拡張である CLOS に関して，その仕様に関して様々な不明瞭な点，および実現に伴って現実的でない点を洗い出し，問題点を踏まえた上で実現をした．メタオブジェクトプロトコルに関しては，仕様も含めて今後の研究課題と考えている．

筆者らの処理系の性能は PCL の Victoria Day Version と比較して，20% 程度高速である．また，NEC の Common Lisp (NX-LISP) に実装される予定である．

参考文献

[1] X3J13 Document 88-002R “Common Lisp Object System Specification”, Daniel G. Bobrow, Linda G. DeMichiel Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales, and David A. Moon, 1988.

[2] “Common Lisp the Language, second edition”, Guy L. Steele Jr., Digital Press, 1990.

[5]89-003 MOP Draft number 10