6P−7

オブジェクト指向言語におけるシンボルオブジェクトをもちいた名前空間の階層化の実装と利用

吉田 実, 小池 汎平, 田中 英彦
{minoru,koike,tanaka}@mtl.t.u-tokyo.ac.jp

東京大学工学部*

1 はじめに

LISP, Prologなどの記号処理言語では, シンボルは基本的なデータタイプの一つで, そのはたす役割は大きい. 本論文では, シンボルの名前空間の階層化法について論じ, また, 階層化によって可能になるシンボルの新たな利用法を探る.

2 名前空間の階層化

シンボルは1) 印字文字列と対応付ける, 2) 関数や述語と対応付ける, 3) 二つのシンボルの同一性を調べることなどに用いられる. オブジェクト指向言語においては, メッセージと呼び出されるメソッドを対応付けるためにも用いられる.

リーダなどにおいては, 文字列をシンボルに変換する必要が生じる. この対応を保持するテーブルをシンボルの名前空間と呼ぶ. プログラム中で用いるシンボルの名前空間が複数あると同じ印字文字列を持つ異なるシンボルを作ることができる (本論文では, 説明を簡単にするために印字名とリーダがシンボルに変換する時のテーブルに表れる文字列を同じものとして扱う). このようにシンボル空間を階層化した言語はいくつかある. 例えば, Common LISP[2] のパケージシステムは名前空間を分割することを可能にしている. オブジェクト指向言語では, クラス毎にメッセージシンボルによって呼び出されるメソッドを定義できるので, ある程度名前空間を分割したものと言えよう.

上記の名前空間の分割においては, パッケージ名, クラス名が一つの名前空間上にしかないため高々2段の階層化である. この名前空間の階層化は任意の深さの木, 更には, 任意の有向グラフに拡張できる. このような階層化は以下のように利用できる.

利用の制限 あるシンボルを利用するためには, そのシンボルを知っていなければならない. それには, そのシンボルを知っているところから教えてもらうか, そのシンボルの属するパッケージを知らなければならない. 知り得るシンボルの集合が限られることを保護に利用できる. 例えば, 特殊なシンボルを知らないと呼び出せないメソッドを作ることができる.

シンボル名の衝突の回避 マルチユーザ環境や大きな環境においては, クラス名やパッケージ名が衝突することがある. この衝突を避けることができる.

ディバッグなどへの利用 メソッドの実行の結果生じる変数への束縛 (代入) に特別なシンボルをあてると, マーカとして利用できる.

別名, 仮名 複数のシンボルが実は一つのシンボルであるようにできる. また, システム述語に自分専用の別名を付けたり, 逆に開発中の述語を他のユーザには安定したバージョンを見せて, 自分は開発中のバージョンを利用するなどの利用法もある.

階層化は, 有向グラフ > 任意の深さの木 >2段の階層化の順に自由度が大きいが, 無闇に利用すると混乱を招く可能性がある.

3 シンボルの名前空間の階層化方式の具体例

この章では, シンボルの名前空間の階層化の具体例として, 現在我々が開発中の並列オブジェクト指向言語 Fleng++[1] 上のシンボルについて述べる. Fleng++ については詳しくは説明しないが, 並列論理型言語 (Comitted-choice 型言語) にオブジェクト指向パラダイムを採り入れたものである.

Fleng++ のシンボルは, 普通のインスタンスオブジェクトとして実装されていて, シンボルオブジェクトとも呼ばれる. このように実装ても実行効率への影響が少なく, 一方, シンボル操作に対して柔軟な対応ができるということが理由である. シンボルオブジェクトは, 内部属性として様々な属性を持つが, もともとのシンボルとしての機能のために, "印字シンボル", そのシンボルに "対応する述語" という2つの属性を持つ. "対応する述語" は, Prologの call 相当のメタな呼び出しなどに用いられる.

シンボルオブジェクトは, 通常のシンボルとしての機能の他にパッケージとしての機能, クラス / コンストラクタ / ローダとしての機能を持つ.

パッケージとしての機能のための属性として, "印字名からサブシンボルへの変換テーブル" を持つ. サブシンボルには原理的には任意のシンボルが属することができる. よって, 一般にはシンボル (=パッケージ) からサブシンボルへの対応は全体で有向グラフを成す.

シンボルオブジェクトは, クラス / コンストラクタ / ローダとしての機能も持つ. そのために必要な属性は, "ロードするファイル名", メモリ上の "ロードイメージ", "ロード時に用いるシンボル", "ロードステータス" などである. シンボルオブジェクトはメッセージ new を受けとると, 自分に対応する

* "How to implement and to use the structured naming space on object oriented languages", YOSHIDA Minoru, KOIKE Hanpei, TANAKA Hidehiko, Univ. of Tokyo, Faculty of Engeneering

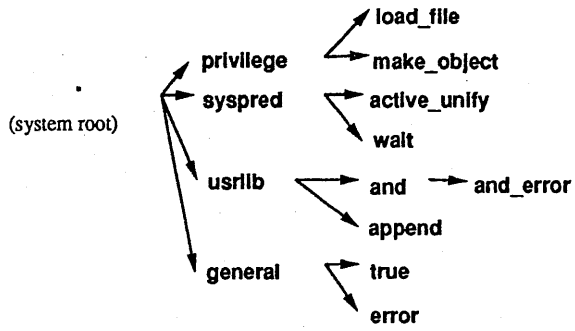

図 1: シンボル階層例

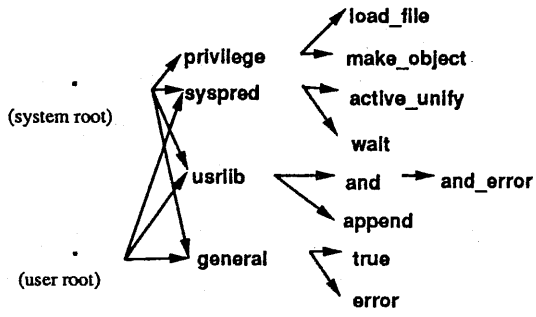

図 2: シンボルによる保護の例

クラスのインスタンスオブジェクトを作ろうとする．“ロードイメージ”がなければ，まず，ファイルからロードする．その時，シンボルの参照を解決する．つまり，ロードファイルの中に表れるシンボルへ変換されるべき文字列をシンボルに変換する．その時にパッケージとして用いられるシンボルが“ロード時に用いるシンボル”である．ローダは全てのシンボルの解決をこの“ロード時に用いるシンボル”に依頼する．“ロードイメージ”ができたら，それをもとにインスタンスオブジェクトを生成し，必要な初期化をすました後に呼び出し側に渡す．“ロードステータス”はロードを(再コンパイルした場合などを除いて)一度しかしないようにするためにローディング状況を保持するためのものである．

このようにシンボルオブジェクトに多くの機能を持たせた．シンボルの名前空間の階層化が便利に使えるためには，上記諸機能との一体化が不可欠であると考えたからである．

4 名前空間の階層化の利用

前章の述べたようにFleng++のシンボル空間の階層化は非常にフレキシブルなものである．これは種々の新しい使い方を可能にするが，同時に，ポリシーのない使い方が混乱を招くことになるであろうことは容易に想像される．そこで利用法が重要になる．ここでは，シンボルオブジェクトの利用法について具体例を用いて説明する．

Fleng++のプログラム中に表れるシンボルは`usrlib:and`のように“:”が間に1つまたは複数はさまった形をしているか，`true`のように“:”を用いない形をしている．“ロード時に用いるシンボル”が図1の“.”(sysmem root)だった場合`usrlib:and`というシンボルが表れた場合，まず，“.”(system root)に“usrlib”という文字列に対応するサブシンボルを教えるように要求する．次にその結果得られたサブシンボルに対して“and”という文字列に対応するサブシンボルを教えるように要求する．この結果得られるシンボルが目的のサブシンボルである．もう一段深い`usrlib:and:and_error`というシンボルが表れた場合には，上記の処理をした後に得られたシンボルに，更に“and_error”という文字列に対応するサブシンボルを教えるように要求しその結果得られるシンボルが目的のシンボルになる．“true”のように単体で表れたシンボルは，“general:true”と解釈される．

シンボルはクラス / コンストラクタ / ローダでもあるので，クラスの名前空間も階層化されたことになる．同じクラス名(クラス名を表す文字列)でも衝突しないように作ることができる．例えば，テスト用のクラスtestが複数存在しても構わない．ウィンドウインタフェースで，同じ機能を持ちlook and feelの異なる複数のウィンドウクラスを用意した場合，同じクラス名でアクセスできるので後で利用するスーパシンボル(親シンボル)を変更するだけで，look and feelを変えることができる．

次に，不当なアクセスなどからの保護にどのように用いることができるか見てみよう．図1においてユーザには，シンボル“privilege”およびそれに属するサブシンボルをアクセスさせたくないものとしよう．この要求は図2のようにユーザ用の“ロード時に用いるシンボル”を用意することで満たせる．図のように，このユーザ用シンボル“.”(user root)からは，“privilege”をアクセスすることができない．

この保護は，メッセージシンボルにも用いることができる．例えば，初期化ルーチンはそのインスタンスオブジェクトを作るシンボルオブジェクトにしか呼び出せないようにしたり，もっと，一般的には，あるオブジェクトが呼び出せるメソッドの集合の制限することに利用できる．

5 おわりに

オブジェクト指向言語におけるシンボルの名前空間の階層化について考え，並列オブジェクト指向言語Fleng++上に実装し，その利用法について考えてみた．ここれで触れなかったが，Fleng++は並列解釈をおこなうオブジェクト指向言語で，一つのシンボルオブジェクトに多くの要求が集中してもボトルネックにならない．このような特長をもたない場合には実装はやや凝ったものにする必要があろう．今後は，利用法についてもっと経験を積む必要がある．特に，便利なデフォルトの規則，ディバッガ，資源保護などへの応用への具体的な適用による実績が必要であると考える．

参考文献

[1] 吉田，田中，“並列オブジェクト指向言語におけるオブジェクト内並列性”，JSPP91, 1991.

[2] Guy Steele Jr., “Common LISP The Language”(2nd Edition), Digital Press, 1990.