オブジェクト指向言語ＥＳＰにおける クラス名空間多重化

2H－6

藤本誠司[1]，萩尾弦一郎[2]，近藤誠一[3]，近山隆[4]
[1]三菱電機東部コンピュータシステム㈱，[2]ビーコンシステム㈱，[3]三菱電機㈱，[4]（財）ＩＣＯＴ

１．背景

ＥＳＰ[1][2]は、クラスをモジュール管理の基本要素とするオブジェクト指向言語である。Smalltalk [3]などのオブジェクト指向言語では、クラスを生成するメタクラスとクラスの登録・修正・削除・参照などの管理やサービスを行うブラウザを持つが、ＥＳＰではＳＩＭＰＯＳ[2]のライブラリとライブラリアンがこれらの機能を果たしている。

ライブラリはクラス管理の基本機能を提供し、ライブラリアンがこれを使って快適なプログラミング環境を提供している。つまりライブラリは、ＳＩＭＰＯＳの中でＥＳＰプログラムの管理を行うサブシステムであるといえる。ＳＩＭＰＯＳとは、逐次推論マシンＰＳＩ上のオペレーティング・システムである。

他のオブジェクト指向言語ではクラス名を一元管理しているため、重複するクラス名は許されない。例えば、Smalltalk ではシステム・ブラウザによって、既存のクラスを修正するか、新しいクラスのクラス名を変えるしか方法がない。2.0 版までのＥＳＰでもそうであった。そこで、クラス名の衝突を避けるため、プログラマは以下の配慮が必要であった。

・　ユーザ・プログラムがクラスを作る場合、システム・クラスのクラス名と衝突していないか注意する必要があった。

・　システム・プログラマが内部クラスを作る際、ユーザ・プログラマが使わないような冗長なクラス名を使わなければならなかった。特に改版の際、既存のユーザ・プログラムの互換性保証の為、注意が必要だった。

・　システム・プログラマは新版クラスを旧版クラスと共存させてデバッグする場合、新版クラスのクラス名を変更しなければならなかった。

クラス名空間を多重化すれば、この様な不便は無くなる。さらに、新版クラスと旧版クラスを共存させることができるので、ＳＩＭＰＯＳ上でのシステム生成も可能となる。

ＥＳＰでは、多重クラス名空間をライブラリのパッケージ機能によって実現した。パッケージとは、Common Lisp [4]のパッケージと同様の概念である。Common Lisp と異なる点は、Common Lisp ではシンボル全ての多重化によって関数のモジュール化を実現しているのに対して、ＥＳＰではオブジェクト指向によってすでに述語がクラス単位にモジュール化されているのでクラス名のみを多重化していることである。

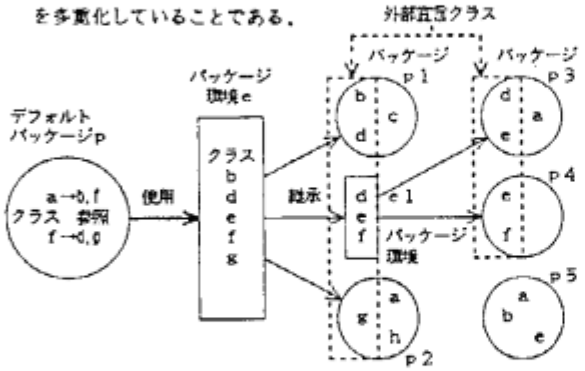

図1　パッケージとパッケージ環境の概念図

２．ＳＩＭＰＯＳライブラリのパッケージ機能

パッケージとは、クラスの集合である。ライブラリは、パッケージ名とクラス名の対によってクラスを一意に決定する。ただしクラス指定において、常にパッケージ名までを指定するのでは不便である。また、既存ソフトウェアの流用性も問題となる。そこで、パッケージ名を省略してクラス名のみでクラスを特定できるように、ユーザ毎、あるいはプロセス毎に設定されたデフォルト・パッケージを用意した。さらに、パッケージ環境と呼ばれる概念を導入した。

パッケージ環境は、デフォルト・パッケージ以外のパッケージのクラスをクラス名のみで特定するための環境である。パッケージ環境は、クラスの特定のため、パッケージやパッケージ環境を継承する。パッケージ環境はパッケージに設定して使用される。

クラス名からクラスの特定は、以下のような手順で行われる（図１参照）。

(1)　デフォルト・パッケージ内のクラスを探索し、あればそのクラスとする。なければ(2) を実行する。

(2)　パッケージ環境に継承されたパッケージを継承順に探索し、最初にみつかったクラスを指定されたクラスとみなす。

このパッケージ環境の導入により、パッケージの包含関係と継承関係が分離され、すっきりしたものとなった。

大規模ソフトウェアを構築する場合、外部ユーザに見せたくない内部クラスが存在するものである。このような可視性の制御を行なえるようにするため、パッケージに外部宣言クラスという概念を導入した。パッケージの中の外部宣言クラスは、パッケージからも、パッケージ環境を通しても見ることができる。しかし、外部宣言がされていないクラスは、クラス名のみで指定した場合、パッケージから直接は見えるが、パッケージ環境を通して見ることは出来ない。

ＥＳＰソース・プログラムにおけるクラスの指定方法には、次の２つがある。

①クラス名
②パッケージ名＃＃クラス名

①のように指定した場合は、コンパイル時にライブラリが自動的にパッケージを求め、クラスを特定する。また、定義するクラスをどのパッケージに入れるかをソース中に指定することができる。図２にクラスの継承や参照の例を示す。

ライブラリアンのコマンドやライブラリのメソッド・コールにおけるクラスの指定方法には、次の３つがある。

①クラス名
②パッケージ名＃＃クラス名
③クラスＩＤ

①のように指定した場合は、その場で自動的にクラスが特定される。③のクラスＩＤは、ライブラリがクラスを管理する際クラスを一意に識別できるように付けた内部的名称であり、一般ユーザは使用しない。図３にライブラリアンのコマンドにおけるクラスの指定例を示す。

３．Common Lisp のパッケージとの機能比較

前にも述べたとおり、Common Lisp のパッケージはシンボルの多重化であるのに対し、ＥＳＰのパッケージはクラス名の多重化

Multi-Class-Name-Space in Object Oriented Language　ＥＳＰ
Seiji Fujimoto[1], Genichiroh Hagio[2], Seiichi Kondoh[3], Takashi Chikayama[4]
[1] Mitsubishi Electric Computer Systems(TOKYO) Co.,[2] Bcon Systems Co.,[3] Mitsubishi Electric Co., [4] ICOT

であるので、単純な比較は難しいが、どちらも名称から実体へのマッピング機能であるので、こういう見方で比較を試みる。

Common Lisp では、パッケージにシンボル単位で取込む(import)か、パッケージが他のパッケージを使用(use-package)することによって、他のパッケージのシンボルを参照(intern)することができる。パッケージの使用しているパッケージのリスト(package-use-list)が、パッケージ環境にあたるものである。これからわかるように、マッピングに使われる構造体が一階層のみであり、拡張は（もしやろうとすれば）展開して作り直さなければならない。

Common Lisp とＥＳＰで多少異なる機能として、shadow機能がある。これは、使用パッケージに定義されたシンボルと同じシンボルをカレント・パッケージに登録することによって、元のシンボルを隠す(shadow)ものである。ＥＳＰでは多階層のパッケージ環境を提供しているので、デフォルト・パッケージにクラス名を登録するregister機能と、パッケージ環境から見えるクラス名を隠すshadow機能を分けて考えている。register機能は既に実現され、shadow機能は実現に向けて検討中である。

また、これは多重化するものの違いによることであるが、Common Lisp ではシンボルを多重化しているので、大域的シンボルであるキーワードはキーワード・パッケージと呼ばれる特殊なパッケージに入れなければならない。この点、ＥＳＰではシンボルは多重化せず、クラス名のみを多重化しているので、キーワードのような大域的シンボルは誰でも共通に使用できる。

４．今後の展望

ＥＳＰのパッケージ機能は、昨年度リリースされたＳＩＭＰＯＳ３．０版で既にサポートされている。リリースから既に１年を経たので、次第に普及してきている。最近の使用例を見ていると、自分のパッケージにデフォルトのパッケージ環境を使うという初期の頃の使用方法ではなく、大規模な開発でパッケージ環境をプロジェクトで共用するような高度な使用方法が一般的に行われるようになってきている（図４参照）。それに応じて、パッケージ機能はさらに高速化、省メモリ化、機能拡張され改版を続け、本年度リリースのＳＩＭＰＯＳ４．０版ではパッケージのポータビリティを高めるパッケージ・テンプレート・ファイル機能や前述のregister機能等がサポートされるに至った。

今後の課題としては、

（１）パッケージ環境のshadow機能

（２）パッケージ環境変更のより使いやすいユーザ・インターフェイス

（３）エラー・モードの導入

（４）ネットワーク・ワイドなパッケージ、パッケージ環境の利用

などを考えている。（４）は、現在でもパッケージ・テンプレート・ファイルで転送することはできるが、他ノードのパッケージを常に使用するように宣言することはできない。これを実現するためにはネットワークの転送速度がネックとなり、なるべくネットワーク転送を行わなくて済むような実現方式を取らなければならない。今日の分散開発環境の普及から考えれば、このような機能は必要になると思われるので、今後この実現に向けてさらに検討したい。

参考文献：

1)近山,"ESP Referrence Manual", ICOT TR-044, (1984)

2)近山,"Unique Features of ESP", Proc.FGCS'84, (1984)

3)高木, 横井, 内田, 黒川, 服部, 近山, 坂井, 辻,"Overall Design of SIMPOS-(Sequencial Inference Machine Programming and Operating System)", Proc.the 2nd ILPC, (1984)

4)Steele,G., "Common Lisp,The Language", Digital Press, (1984)

5)Goldberg,A.,Robson,D.,"Smalltalk-80,the Language and Its Implementation", Addison-Wesley, (1983)


```
emacs_47
use_package(puzzle).

class eight_package has
  nature nqueen;
% caution: "nature puzzle##nqueen;" is O.K.

:create(Class,Obj):-
    :new(Class,Obj),
    :create(#game##chess_board_window,CW),
    :create(#pmacs_window,[size(500,500)],W),
              :
              :

instance
  attribute
    board is game##chess_board;

              :
              :

end.

PMACS(esp)[45,22) #47/1# puzzle)8queen.esp.1 --
Read: >sys>user>puzzle>8queen.esp.1
```

図2　ESPソース例

```
librarian_42 ( default: puzzle )
  Check      Class Name>nqueen,game##chess_bo
 Catalogue   ard
   Save
   Load      ---- puzzle##nqueen ----
  Delete     source    15-Jan-88 17:13:07
 Execute     template  15-Jan-88 17:13:07
 Package       file    none
 Utility     object    15-Jan-88 17:13:08
             ---- game##chess_board ----
             source    15-Jan-88 17:12:49
             template  15-Jan-88 17:12:49
               file    none
             object    15-Jan-88 17:12:53
             - - - - - - - - - - - - - - - -
```

図3　ライブラリアン使用例

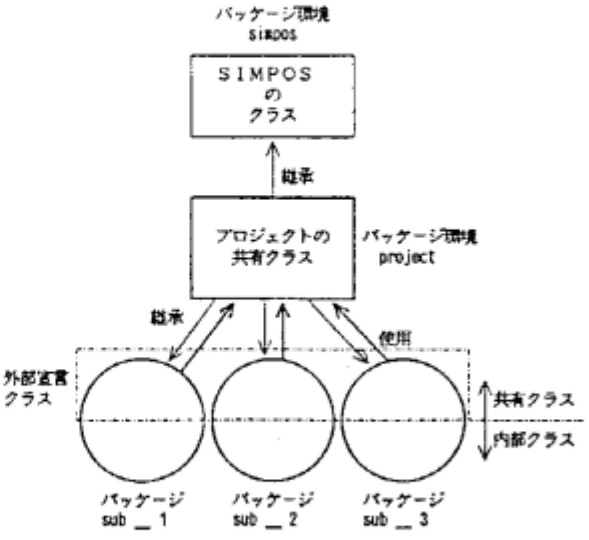

図4　大規模ソフトウエア開発プロジェクト