並列オブジェクト指向言語A´UMによるプログラミング

和田久美子　　瀧和男　　古田かおる
（沖電気工業株式会社）（　　ICOT　　）

1．はじめに

現在、第５世代コンピュータの研究開発プロジェクトでは、並列推論マシンＰＩＭを開発中である。Ａ´ＵＭは、ＰＩＭのユーザ向け言語として設計された並列オブジェクト指向言語である。Ａ´ＵＭを用いて並列プログラムを記述することにより、従来、ＫＬ１で表現すると冗長とならざるを得ない記述を避けることができる。また、オブジェクトを用いて並列実行の単位やストリーム接続をより明確にすることによって、プログラムの見易さの向上を図ることができる。さらに、クラス継承機能等により、より大きなアプリケーション・プログラムや、オペレーティング・システムを含むシステム・プログラムを記述することも可能である。本稿では、最適経路問題に対するＡ´ＵＭによるプログラミング例を示すことにより、その実用性について論じる。

2．Ａ´ＵＭによるプログラム記述の例…最適経路問題

最適経路問題とは、与えられたネットワークに対して、各辺にそれぞれ非負の値（コスト）が割り当てられているとき、ネットワークのある２点間を結ぶ経路のうちで最もコストの低いものを求める問題である。ここでは特に、ある開始点を与えたとき、その他のすべての点への最適経路を求める問題を考える。

2．1　アルゴリズムの概略

まず、与えられたネットワークに対して、ネットワークに存在するだけの node オブジェクトを生成し、隣接する node に関する情報などを与える。

次に、開始点の node オブジェクトに対して、最初のメッセージ　:cp(0, [])　を送る。

各 node オブジェクトは常に、その時点までに求まった最適と思われるコストＣとそのときの経路Ｐをスロットに保持していて、 node ｎがメッセージ :cp(Cost, Path)を受け取ったとき、もし Cost ≧Ｃならば、 node は、このメッセージを無視する（∵自分の持っている値の方がより最適解に近い）。もし Cost ＜Ｃならば、スロットの値を Cost, Path で更新し、自分と辺で隣接している他のすべての node ｍへメッセージ　:cp(Cost＋MC, [n | Path])を送る（ただし、MCは node ｎから node ｍへのコスト）。

このようなメッセージ通信がすべてなくなったとき、各 node オブジェクトは、開始点からそれぞれの node に至る最適経路とその最小コストをスロットに保持していることになる。ネットワーク上にもうメッセージが流れていないかどうかの終了判定を行うために、実際に node 間でやりとりされるメッセージに short circuitのスイッチをつける。ひとつのメッセージを受け取った node が、他の node に複数のメッセージを送る場合には、受け取ったスイッチを分解して各メッセージにつけて送り、メッセージを消滅させる場合には、スイッチを閉じる。

2．2　Ａ´ＵＭのプログラム

最適経路問題のための主なクラスは以下にあげる２つで、その他、ネットワーク、リスト、ベクタを表現するクラスが用意されている。ここでは便宜上、リストとベクタについてマクロを定義した。

１）クラス bestpath

最適経路問題のメイン・クラスである。与えられたネットワークに対する各 node オブジェクトや結果の出力先ウインドウ・オブジェクトの生成、スロットの初期化（隣接する node オブジェクトやコスト等）を行う。次に、開始点の node オブジェクトに対して最初のメッセージを送る。

２）クラス node

各 node オブジェクトを表すクラスである。各nodeオブジェクトは、生成と同時に並列に実行可能となり、メッセージ待ち状態となる。

メッセージ :cp(^Cost, ^Path, ^T0, T)を受け取ったときのオブジェクトの動作は、プログラム中(a) で示される。ＫＬ１では、複数の候補節のガード部にそれぞれの条件を記述することにより条件分岐を表現していた。Ａ´ＵＭでは、条件分岐はボラタイル・オブジェクトを用いて容易に記述される。

また、プログラム中(c) は、隣接する各 node へ新たに :cpメッセージを送る動作を示している。各 node オブジェクトは、スロット neighbors, costs にそれぞれ隣接するオブジェクトとそこへのコストをリストで保持している。Nsは、(b) でスロットneighbors から渡されたリスト・オブジェクトである。:[^N | ^Ns1]は、リスト・オブジェクト Ns に対する、 car部と cdr部を得るメッセージのマクロ表現で、:get_car(^N) :get _cdr(^Ns1)という２メッセージを発行した場合と同一の意味を持つ。

3．Ａ´ＵＭの長所

以上のプログラム例からもわかるように、Ａ´ＵＭの長所をまとめてみると、次のようなものになる。

- ＫＬ１では、ストリーム・コミュニケーションを用いた perpetual process をオブジェクトに見たてることによってオブジェクト指向のプログラミング・スタイルを形成していた。Ａ´ＵＭによって、より直接的にオブジェクト指向が確立されたため、並列実行単位や各オブジェクトのコミュニケーションの対象等が明確に示されるようになった。
- ＫＬ１では、メッセージ通信のためのストリームを他の perpetual process （オブジェクト）に分け与える場合、マージャを挿入しなければならなかった。Ａ´ＵＭでは、システムにより自動的に挿入される。

Programming in Parallel Object-Oriented Language A'UM
K.WADA(OKI ELECTRIC Ind.), K.TAKI, K.YOSHIDA(ICOT)

・A´UMでは、オブジェクトは、スロットに値を持つことができるので、ＫＬ１のようにすべてを引数で持ちまわる必要がない。
・マクロ展開による文法により、プログラムがより見易く、またコンパクトになる。
・変数に、入力／出力のいずれかのモードを示す^をつけるため、メッセージの流れる方向（ストリームの方向）、オブジェクトの存在する方向を理解しやすい。
・クラス継承に基づく高度なモジュール化支援を行うことが可能である。

（以下、プログラム抜粋）

```
% Program is invoked as follows:
%     #bestpath :new(^BP), BP :bestpath(a,G).
% where, G is a graph object whose slots are
%                                 defined as follows:
%     !nodes = [a,b,c,d,e,f,nil],
%     !edges = [[a,b,10],[a,f,5], ... ].

class  bestpath.

:bestpath(^Start,^Graph) ->
   #window :new(^Window),
   :bp(Start,Graph,Window,^ObjectList) ..

:bp(^Start,^Graph,^Window,OL) ->
   Graph :get_nodes(^Nodes) :get_edges(^Edges),
   :create_nodes(Nodes,Window,^OL),
   :set_neighbors(Edges,OL,S0,^S),
   S :end,
   ^S0 [
      :end ->
         :retrieve(OL,Start,^StartNode),
         StartNode :cp(0,[nil|nil],T0,^T),
         T :end,
         ^T0 [
            :end -> :send_end(OL).
         ].
   ].

:create_nodes(^Nodes,^Window,OL) ->
   Nodes :[^H|^Rest],
   ( H == nil ) [
      :true -> ^OL = [nil|nil].
      :false ->
         #node :new(^Node),
         Node :set_name(H) :set_window(Window),
         ^N = [H,Node],     % <- macro for vector
         ^OL = [N|OL1],                  % creation
         :create_nodes(Rest,^OL1).
   ].

:set_neighbors(^Edges,^OL,^T0,T) ->
   Edges :[^E|^Rest],
   ( E == nil ) [
      :true -> ^T = T0.
      :false ->
         E :[^N1,^N2,^C], % <- macro for getting
         :retrieve(OL,N1,^O1),  % vector element
         :retrieve(OL,N2,^O2),
         O1 :set_neighbors(O2,C),
         O2 :set_neighbors(O1,C),
         :set_neighbors(Rest,OL,T0,^T).
   ].

       ......
end.
```

```
class  node.

slot name,      % node identifier
     neighbors,% list of neighboring node_objects
     costs,     % costs of outgoing edges
     cost,      % minimum cost so far
     path,      % path with !cost
     window.    % result will be written here

:initiate -> !neighbors = [nil|nil],
             !costs = [nil|nil],
             !cost = 99999.

:set_name(^Name) -> !name = Name.
:set_window(^Window) -> !window = Window.
:set_neighbors(^Node,^Cost) ->
   !neighbors = [Node|!neighbors],
   !costs = [Cost|!costs].

:cp(^Cost,^Path,^T0,T) ->                    ―――― (a)
   ( Cost >= !cost ) [
      :true -> ^T = T0.
      :false ->
         :send_cp(!neighbors,!costs,Cost, ―― (b)
                                [!name|Path],T0,^T),
         !cost = Cost,
         !path = Path.
   ].

:send_cp(^Ns,^Costs,^Cost,^Path,^T0,T) -> ―― (c)
   Ns :[^N|^Ns1],
   ( N == nil ) [
      :true -> ^T = T0.
      :false ->
         Costs :[^C|^Cs],
         N :cp(Cost+C,Path,T0,^T1),
         :send_cp(Ns1,Cs,Cost,Path,T1,^T).
   ].

:end ->
   !window :result(!name,!cost,!path),
   :close_neighbors(!neighbors).

       ......
end.
```

4．おわりに

今回、いくつかの並列問題についてA´UMによる記述を行ってみた結果、ＫＬ１に比べて、放送機能がない、オブジェクトの分身を作ることができないなど、いくつかの問題点もあげられた。現在、これらに対する効率の良いプログラミング・テクニックの開発を行っている。また、オペレーティング・システムなどのより大きなプログラムや、より様々な並列問題に対する記述によって、A´UMの特性をさらにひき出し、A´UM独自のプログラミング・スタイルについて考察して行く予定である。

<参考文献>

[1] 古田，近山：A'UM－ KL1上の並列オブジェクト指向言語－，情報処理学会プログラミング言語研究会 14-4, 1987-12

[2] 高木 他：A Collection of KL1 Programs－Part 1－, TH-311, ICOT, 1987

[3] 和田 他：並列オブジェクト指向言語 A'UM によるプログラミング，TH-435, ICOT, 1988 (to apper)