並列オブジェクト指向言語 $A'UM$

吉田かおる，近山隆

(財) 新世代コンピュータ技術開発機構 (ICOT)

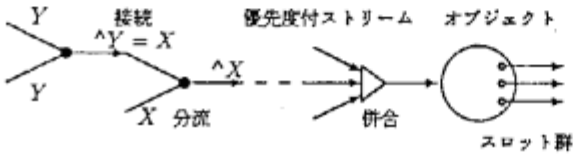
1 はじめに

$A'UM$[1] は，大規模な並列応用プログラムおよびシステムプログラムの開発を第一の目的に設計された並列オブジェクト指向言語である．現在，ICOT では第五世代コンピュータプロジェクトの一環として，並列推論マシン PIM を研究開発しており，$A'UM$ は PIM のオペレーティングシステム PIMOS の記述言語として，また PIM 上のユーザ言語として予定されている．

$A'UM$ を構成するのは，ストリームを外部インタフェースとする副作用を持たない永続的プロセスで実現される並列オブジェクトである．ストリームの自動併合および閉鎖，条件分岐および繰返しに関するオブジェクトの概念の徹底，マクロ展開手法に基づく文法，そしてクラス継承によるモジュール化支援が $A'UM$ の大きな特徴である[1]．本稿では，$A'UM$ のオブジェクト像を中心に述べる．

2 $A'UM$ オブジェクト

$A'UM$ オブジェクトの概念図を以下に示す．



ストリームによる外部インタフェース　オブジェクトとして外部から認識されるものは，ストリームである．“オブジェクトを知る”とは，そのオブジェクトへのストリームを得ることに他ならない．ストリームを得た後は，そのストリームをオブジェクト自身と思い，メッセージを送信すればよい．逆に，“オブジェクトを渡す”とは，そのオブジェクトへのストリームを分流し，片方の支流を与えることである．したがって，各オブジェクトへ木構造をなすストリーム群からメッセージが送られることになる．

急行通信のために，ストリームには優先度が付いている．オブジェクト生成時に異なる優先度のストリームを得ることができる．

純永続的プロセスによる実現　オブジェクトは，インタフェース・ストリームからのメッセージを一個受信しては新しい世代の内部状態を作り出し次のメッセージ受信へと再帰する，一つの永続的プロセスである．

オブジェクトを生成すると，まず起動メッセージが暗黙裏に送信され，以降プログラムで定義するメッセージがインタフェース・ストリームへ送信される．

オブジェクトは終了メッセージの受信により，全スロットを閉鎖した後，その生命を閉じる．メソッドの末尾に，再帰 (.) か終了 (..) かを明示することができる．終了処理とは，外部からのインタフェース・ストリームを切断し，自分自身へ終了メッセージを送信することである．終了時点が明示されないときは，インタフェース・ストリームの閉鎖により，終了処理が施される．

Parallel object-oriented language $A'UM$.
Kaoru Yoshida and Takashi Chikayama (ICOT)

[1] 「あ・うん」と発音する．

スロットの名前連想　オブジェクトは，内部状態としてのオブジェクト (へのストリーム) 群をそれぞれ名前で連想して保持している．このような連想セルをスロットと呼ぶ．スロット名はクラス内でのみ有効であり，同一のスロット名でもクラスが異なれば別物である．

スロットの参照および更新は，自分自身へのメッセージ送信として実現される．スロットの参照とは，現スロットが保持しているストリームを分流し，片方をスロット参照メッセージの引数に渡し，もう一方を新スロットとすることである．スロットの更新とは，現スロットを閉鎖し，スロット更新メッセージの引数として渡されたストリームを新スロットとすることである．

自分自身へのメッセージ送信　オブジェクトは，インタフェース・ストリームの受信口を保持している．スロット・アクセスを含め，自分自身へのメッセージはこの受信口へ挿入される．あるメッセージを受信し，それにより幾つかのメッセージを自分自身へ送信すると，これらのメッセージは受信メッセージの次に来ているメッセージより前に位置することになる．

クラス継承によるメソッド共有　$A'UM$ オブジェクトは，あるクラスで定義されるインスタンスである．クラスは，内部状態を持たない不変オブジェクトとして扱われるが，$A'UM$ にはクラスをインスタンスとするメタクラスの概念はない．

一つのクラスは，複数個のクラスを継承することができる．オブジェクトは，クラス継承によりそれぞれのクラスに対するプロセスを有するわけではなく，単一のプロセスである．クラス継承はメソッドの空間を広げるためのものである．

受信メッセージに対するメソッドの探索は，委託 (delegation) の機構を用いて実現している．オブジェクトはスーパークラスの継承木情報を保持しており，この継承木を左優先深さ優先の規則で登りながら，“このメッセージを受信せよ”という委託メッセージを自分自身に送信する．

原始オブジェクト　整数，アトム，ストリングなどは原始オブジェクトと呼ばれ，一般のオブジェクトと全く同様に扱われる．すなわち，原始オブジェクトもストリームを外部インタフェースとする永続的プロセスと解釈してよい．例えば，整数オブジェクト (へのストリーム) へ減算メッセージを送信すると，差に相当する整数オブジェクト (へのストリーム) が渡される．このように，$A'UM$ の世界にはストリーム以外の何物も存在しない．

3 ストリームの接続，併合および閉鎖

ストリームの方向指定と自動併合　電池に＋－があるように，レゴに凸凹があるように，ストリームには出力端と入力端がある．出力端と入力端を互い違いに繋げていくと，最後にオブジェクトへ辿り着く長いストリームができあがる．$A'UM$ における変数は，いつかどこかのオブジェクトへ繋がるであろうストリームの端点を表し，次のような意味を持つ．

(1) 同一変数に対して，'^' の付いた最大一個の出力端と，'^' の付かない任意個の入力端が出現する．(例えば ^X 一個と X 二個)
(2) オブジェクトは出力端の先のどこかにいる．
(3) 二本のストリームは，一方の出力端を他方の入力端に接続して一本となる．
(4) メッセージは，入力端に送信される．

(5) 入力端に送信されたメッセージは併合されて，出力端の先のオブジェクトに送られる．

メッセージの識別　各メッセージは，そのメッセージ名と引数列のストリーム方向により識別される．すなわち，同じメッセージ名と引数個数でも引数の方向が異なれば，別のメッセージと解釈される．例えば，メッセージ送信で引数の方向を誤ると，“未定義メソッド”のエラーとなる．

ストリームの自動閉鎖　オブジェクトの終了が明示されない場合，そのオブジェクトはインタフェース・ストリームの最終閉鎖により終了する．どこかの支流ストリームが開放されていると，オブジェクトはそこからのメッセージを待ち続けることになる．したがって，ストリームを確実に閉鎖し開放端を残さないようにしなければならない．

A'UM では，可視領域内でストリームへの参照がなくなる時，そのストリームは自動閉鎖される．具体的には，
(1) 出力端のみ出現しているストリーム，
(2) メッセージ送信後どこにも接続されないストリーム，
(3) スロット更新時に現スロットが保持するストリーム，
(4) オブジェクト終了時にスロットとして保持している全ストリームが自動閉鎖される．

4　揮発オブジェクトによる条件分岐と繰返し

オブジェクトは，受信メッセージに応じてある振る舞いをし，そして次のメッセージ受信へ再帰する．前者は条件分岐を，後者は繰返しを意味する．いわゆるプログラムの制御構造と呼ばれるものは，オブジェクトの枠組で自然に表現できる．ところが，一つの条件分岐のためにクラスを定義しているのでは，小さなクラスを数多く定義することになり，プログラムの文脈がわかりにくくなる．そこで *A'UM* では，ある文脈の中だけに現れるその場限りのオブジェクトを揮発オブジェクトと呼び，メソッド本体内に

<インタフェース> '[' <揮発不変オブジェクト定義> ']'
<インタフェース> '{' <揮発可変オブジェクト定義> '}'

のように一文として定義できるようにしている．これを用いて，条件分岐および繰返しが容易かつ簡潔に記述できる．

基本: 出力端をインタフェースとして　出力端とはその先にオブジェクトがあることを示す．出力端をインタフェース・ストリームとして指定し，それからの受信メッセージに対するメソッドを定義する．

拡張: 入力端への誰何メッセージ　四則あるいは比較演算式の評価結果は整数オブジェクトや真理値オブジェクトである．すなわち，オブジェクトへの入力端が渡されることになる．入力端はメッセージを送信するための口であり，メッセージを送信し続けても条件分岐は行えない．条件分岐は出力端を握る受信者の役割である．そこで入力端から出力端へ切り替えるために“誰何の原理”を用いる．インタフェースの評価結果が入力端である場合，それに対して所定通りに誰何メッセージ who_are_you(Who) が送信される．この引数 Who にはオブジェクトの値がメッセージとして送信され，対応する出力端 ^Who がインタフェース・ストリームと見なされる．オブジェクトの値は，基底を示す原始オブジェクトに対しては言語で定められている．

5　マクロ展開に基づく文法

A'UM の文法的特徴は，メソッドがマクロ展開手法に基づいて解釈されることである．このマクロ展開手法に加えて，ストリーム自動併合および自動閉鎖の支援により簡潔で安全なプログラムを表現することができる．

[プログラム例]　次のようなスタックを考えよう．
スタックはトップのエレメントを保持し，最初これはスタックのボトムを指している．各エレメントは，データと次のエレメントを保持している．スタックは，メッセージ push を受け取ると新しいエレメントを生成し，そのエレメントに，受信メッセージ push の引数で渡されたデータを保持させ現在のスタック・トップを次のエレメントとして指させる．そしてスタックはこのエレメントを新たにトップとする．またメッセージ pop を受け取ると，現在トップであるエレメントに対して，そのデータと次のエレメントを尋ね，前者を受信メッセージ pop の引数として渡し，後者を新たに自らのトップとする．メッセージ read の場合，メッセージ pop と同様に現在のエレメントに対してそのデータと次のエレメントを尋ねるが，その後のトップの更新は行わない．*A'UM* によるプログラムを以下に示す．

```
class  stack.
     slot  top.
  :initiate     -> #bottom:new(^Bottom), !top = Bottom.
  :push(^Data) -> #element:new(^Element),
                  !top = Element:set(Data,!top).
  :pop(Data) ->   !top:get(^Data,^Next), !top = Next.
  :read(Data) ->  !top:get(^Data, ^Next).
end.

class  element.
     slot  data, next.
  :set(^Data,^Next) -> !data = Data, !next = Next.
  :get(!data,!next) -> .
end.

class  bottom.
  :get('$end_of_stack',Next) -> .
end.

class  test.
  :test -> #stack:new(^Stack0),
           Stack0:push(1):read(^A):pop(^B):pop(^C) ..
end.
```

メッセージ送信表現式　　<オブジェクト> ':' <メッセージ>
A'UM の基本操作は，ストリームへのメッセージ送信であり，メッセージ送信後のストリームをその評価結果とする表現式を用いて表わす．この評価を繰り返すことにより，複数個のメッセージ送信も同様に記述でき，またメッセージの引数にメッセージ送信表現式を指定することもできる．さらに，スロットへのメッセージ送信表現式は，現スロットを参照し，メッセージ送信し，送信後のストリームを新スロットとすることを意味している．このため，スロット操作に関してはほとんどストリームを意識する必要はない．

自分自身へのメッセージ送信の順序　各メソッドは左から右へ，各メッセージの引数は中から外への順に評価される．スロット・アクセスを含め，自分自身へのメッセージ送信順序は，この評価順序に従う．この自分自身へのメッセージ送信が受信メッセージに指定される場合は受信直後に，また送信メッセージに現れる場合は送信直前に送られる．独立に実行しうるのは，受信メッセージの引数あるいはメソッド内部に変数として出現する，オブジェクトとは独立のストリームへのメッセージ送信である．

6　おわりに

現在，*A'UM* から PIM の核言語 KL1 へのコンパイラを試験実装しており，この上で，上記の例題プログラムを含む幾つかのプログラムが試験されている．今後，*A'UM* によるプログラミングを進め，*A'UM* の記述力を確かめるとともに強化を図る予定である．また原始オブジェクトの効率的な実行を支援する本格版処理系とデバッガの開発を中心に *A'UM* の環境づくりを行っていきたい．

参考文献

[1] 吉田，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 年 12 月