知識工学と人工知能　39－9
（１９８４．９．１９）

手続き指向型プロダクションシステムＰｏ－ＰＳとその応用

小野典彦　小林重信
（東京工業大学　総合理工　システム科学専攻）

１．はじめに

複数の同時進行的な対象があって、各対象は事象を発生すると共に、他の対象からの事象に反応して事象を発生するというように、事象の発生がシステム全体を制御するシステムのことを事象駆動型システムという。オフィス業務、ジョブショップ型生産システム、シーケンス制御、オペレーティングシステム人間の問題解決行動などはその例である。

近年、分散協調型プロダクションシステムを用いて事象駆動型システムの表現と制御を行なう試みがいくつかみられるようになってきた。分散協調型プロダクションシステムでモデル化することの利点にはつぎのようなものがある。

１）対象指向性：対象およびそのサブタスクを１つの独立したプロダクションシステムに対応させることによってこの性質をうる。

２）事象駆動性：プロダクションシステムによる対象のモデル化はこの性質を自然に実現する。

３）柔軟性、拡張性：プロダクションシステムを分散化させることにより、システムのモジュラリティを促進し、拡張性や柔軟性をもたらし、プロダクションシステムの欠点を極小化することができる。

従来の分散協調型モデルでは、各プロダクションシステムは標準的な枠組みを保持し、プロダクションシステム間の相互作用を表現する比較的簡単な機構を導入するものであった。

本研究では、事象駆動型システムをモデル化する際、従来の手続き的言語における並列処理やタスク間通信のための機構を導入することにより、表現と制御が自然に行なえるとの考えから、これらの機構を取り入れた分散協調型プロダクションシステムの表現言語であるＰｏ－ＰＳを設計、開発したので、その概要を報告する。

２．Ｐｏ－ＰＳの基本設計

２．１　Ｐｏ－ＰＳの目的

Ｐｏ－ＰＳはつぎの分野を応用領域とする。

１）事象駆動型システムのモデリング
ＦＡシステムの設計、拡張段階におけるシステムのモデリングやシミュレーション。

２）事象駆動型システムの管理と制御
オフィス業務、ＦＡシステムのモニタリングや実時間制御。

３）分散協調型問題解決システムの記述
分散協調型問題解決システムにおける知識表現および推論のための枠組の提供。

２．２　Ｐｏ－ＰＳの要求仕様

Ｐｏ－ＰＳが提供する分散協調型プロダクションシステムは以下の機能をもつこと。

１）対象の同時進行性
非同期、同時進行的な複数の対象がそれぞれ自律的タスクとして表現し得ること。

２）対象間の競合解消
対象間で資源の共有などによる競合が生じた場合、これを解消し得る機構をもつこと。

３）対象間の協調
対象間での協調による問題解決が必要とされる場合、同期をとり、調整するための機構をもつこと。

４）対象間の通信
発生した事象を対象間で通知し合えること。

５）対象の階層性
複数のサブタスクを階層的に展開して実行させる対象はこれらに相当する子供の対象を生成、管理していると考えてよい。対象をこのように階層的に生成し得ること。

６）対象の抽象化
対象は自律的なタスクであり、一般に内部状態をもつ。これを他の対象から隠蔽したほうが対象のモジュラリティが高くなる。内部状態を隠すために、対象を抽象データ型として表現し得ること。

7）事象の重要度の理解

対象を記述するプロダクションシステムは表現力に富み、事象の重要度を把握し、現在のタスクの引き金となった事象よりも重要な事象が発生した場合には、現在のタスクを中断し、それに反応し得ること。

２．３　Ｐｏ－ＰＳの基本アーキテクチャ

われわれは分散協調型プロダクションシステムとの整合性に富み、要求仕様を満たし得る計算モデルの一つとしてＨｏａｒのＣＳＰ[2]を選んだ。即ち、システムはメッセージを介して通信し合うタスクの集まりとして、タスクは通信機構を付加したプロダクションシステムとして表現する。記述力に富み、構文的にも洗練されたＯＰＳ５[3]をプロダクションシステムの基本的枠組とした。

３．Ｐｏ－ＰＳの概要

本節では簡単な例題を用いながらＰｏ－ＰＳの概要を解説する。例題としてつぎの２つを取り上げる。

［例題１］哲学者の食事

通常の　哲学者の食事　問題につぎの変形を施す。１）哲学者は食事の時間、思索の時間および平然と食事を待てる許容時間に個人差がある。

２）彼らは食事は笑顔で、思索は真剣な表情で、許容時間を超えて食事を待たされると顔をしかめる。

３）彼らは争いを嫌い、しかめ顔をしているもの、つぎにより長い時間食事を待たされているものにフォークの使用権を譲る。

哲学者とフォークを図３．１のように表わす。この例をＰｏ－ＰＳで記述したのが図３．２である。

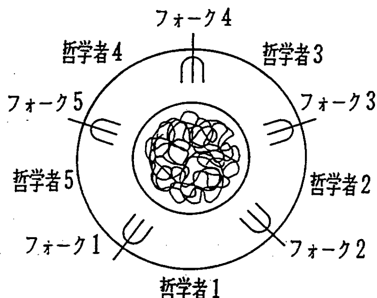

図３．１　哲学者の食事問題

```
(tasktype  scheduler
  hit:     exhaustive
  message: use     ^no ; to call for using the adjacent forks
           release ^no ; to release the adjacent forks
  element: fork ^no ^status
  scheduling:
    p1 (:from ?phil release ^no ?n)
       (fork ^no ?n              ^status up)
       (fork ^no (++ (mod ?n 5)) ^status up)
     =>(modify 2 ^status down) (modify 3 ^status down)
       (accept 1)
    p2 (:from ?phil use ^no ?n)
       (fork ^no ?n              ^status down)
       (fork ^no (++ (mod ?n 5)) ^status down)
       (:task ?phil Appearance ^look frowning)
     =>(modify 2 ^status up) (modify 3 ^status up)
       (accept 1)
    p3 (:from ?phil use ^no ?n)
       (fork ^no ?n              ^status down)
       (fork ^no (++ (mod ?n 5)) ^status down)
     =>(modify 2 ^status up) (modify 3 ^status up)
       (accept 1)
  initial:
  (make fork ^no 1 ^status down)
  (make fork ^no 2 ^status down)
  (make fork ^no 3 ^status down)
  (make fork ^no 4 ^status down)
  (make fork ^no 5 ^status down)
  (create ph-1 philosopher ^no 1 ^dine 1 ^think 2 ^endure 3)
  (create ph-2 philosopher ^no 2 ^dine 1 ^think 3 ^endure 2)
  (create ph-3 philosopher ^no 3 ^dine 2 ^think 1 ^endure 3)
  (create ph-4 philosopher ^no 4 ^dine 2 ^think 3 ^endure 1)
  (create ph-5 philosopher ^no 5 ^dine 3 ^think 1 ^endure 2) )
(tasktype  philosopher ^no     ; the identification number
                       ^dine   ; the time for dining
                       ^think  ; the time for thinking
                       ^endure ; the time for endurring
  hit:     single
  element: Appearance ^look
           self       ^activity
  routine:
    p1 (self ^activity nothing)
       (Appearance ^look ?look)
     =>(request scheduler use ^no *no
          :within *endure
          :else  (modify 2 ^look frowning) )
       (modify 1 ^activity dining)
    p2 (self ^activity dining)
       (Appearance ^look ?look)
     =>(modify 2 ^look smiling) (delay *dine)
       (send scheduler release ^no *no)
       (modify 1 ^activity thinking)
    p3 (self ^activity thinking)
       (Appearance ^look ?look)
     =>(modify 2 ^look serious) (delay *think)
       (modify 1 ^activity nothing)
  initial:(make self ^activity nothing)
          (make Appearance ^look serious) )
; -- root task creation --
(create scheduler scheduler)
```

図３．２　例題１のＰｏ－ＰＳによる記述

［例題２］分割征服法による階乗の並列計算

図３．３の２式に基づいて非負整数ｎの階乗を並列に計算する過程をＰｏ－ＰＳで記述したのが図３．４である。

```
factorial(n) = fact(0, n).
fact(i, j)   = if      i = j then 1
               elseif  i+1=j then j
               else    fact(i,mid)*fact(mid,j)
                       where mid=round((i+j)/2)
               endif.
```

図３．３　階乗の計算式

```
(tasktype factorial
  message: factorial ^n ^factorial ?
  request:
    p1 (:from ?f factorial ^n ?n)
     =>(request (create * fact)
                fact ^i 0 ^j ?n ^fact ?fact)
       (accept 1 ^factorial ?fact) )
(tasktype fact
  message: f ^i ^j ^fact ?
  request:
    p1 (:from ?f fact ^i ?i ^j ?i)
     =>(accept 1 ^fact 1)
    p2 (:from ?f fact ^i ?i ^j (++ ?i))
     =>(accept 1 ^fact (++ ?i))
    p3 (:from ?f fact ^i ?i ^j ?j)
     =>(request
         (create * fact)
          fact ^i ?i ^j (round (% (+ ?i ?j) 2)) ^fact ?f1
       :and
         (create * fact)
          fact ^j ?j ^i (round (% (+ ?i ?j) 2)) ^fact ?f2)
       (accept 1 ^fact (* ?f1 ?f2)) )
```

図３．４　例題２のＰｏ－ＰＳによる記述

３．１　システム構成

Ｐｏ－ＰＳではシステムを非同期、同時進行的な自律的なタスクとして表現する。事象駆動型システムにおける対象、対象が生成する副タスクはすべてタスクとして表現される。

各タスクはＨｏａｒのメッセージによる通信／同期機構[7]をもつ。事象駆動型システムにおける対象の間の通信、協調および競合解消はこの機構で実現される。タスクはサブタスクを生成し、それをメッセージを介して制御することができる。例題１の場合、図３．５のように、システムはフォークの割り当てをスケジュールするタスクｓｃｈｅｄｕｌｅｒ、このタスクが生成する哲学者タスクｐｈ－１からｐｈ－５として表現される。

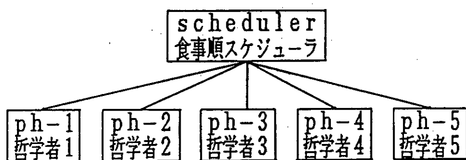

図３．５　例題１のタスク階層

例題２の場合、ｆａｃｔｏｒｉａｌ（ｎ）の計算をＰｏ－ＰＳで行なうためには図３．６のようなタスク階層が生成される。

Ｐｏ－ＰＳはタスクを１つ生成することにより起動され、このタスクの終了と共に実行を終了する。このタスクのことをルートタスクと呼ぶ。

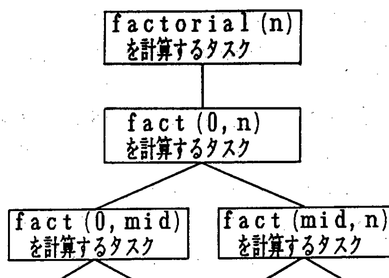

図３．６　例題２のタスク階層

３．２　タスク

タスクはメッセージ通信機構が付加された拡張プロダクションシステムとして表現され、つぎの基本要素よりなる。

１）ワーキングメモリ（ＷＭ）：タスクの内部状態を表わし、初期状態はタスクの生成時に与えられ、プロダクションの実行により更新される。

２）メッセージメモリ（ＭＭ）：他のタスクから送信されてくるメッセージのキューである。これはプロダクションの実行により参照、除去される。

３）プロダクションメモリ（ＰＭ）：ＷＭとＭＭの変化に事象駆動的に反応して、適用されるプロダクションの集まりである。

４）ｉｎｉｔｉａｌ節：タスクの生成時に実行される文の列である。

５）ｆｉｎａｌ節：タスクが強制終了される直前に実行される文の列である。

６）インタプリタ：通常のプロダクションシステムにおける認識－行動サイクルを管理すると共に、ｉｎｉｔｉａｌ節やｆｉｎａｌ節を実行する。

タスクはタスク型のインスタンスとして生成される。タスク型はタスクのプロトタイプを表わし、属性名でパラメータ化されている。タスク型定義の形式はつぎの通りである。

（ｔａｓｋｔｙｐｅ　タスク型名
　　属性名の並び　タスク要素定義の並び）

タスク要素定義は、１）ワーキングメモリ要素定義、２）メッセージ定義、３）プロダクション定義、４）ｉｎｉｔｉａｌ節定義、５）ｆｉｎａｌ節定義、および６）ヒット戦略定義からなる。６）については後述する。

３．３　拡張プロダクションシステム

これはＯＰＳ５を基本的枠組とする。

そのＷＭ要素、ＭＭ要素（メッセージ）はクラス名、属性名／値の並び、整数値タグよりなる。ＷＭ要素定義、メッセージ定義はそれぞれＷＭ、ＭＭ要素に関するｌｉｔｅｒａｌｉｚｅ宣言に相当する。英大文字で始まるクラス名のＷＭ要素は他のタスクからも参照できる。ＷＭ要素のタグは最近作られたものほど大きく、ＭＭ要素のタグは最近到着したものほど小さい。

プロダクションはつぎの形式をしている。

名前　左辺　＝＞　右辺

左辺でＭＭ要素や他タスクのＷＭ要素を参照するにはそれぞれつぎの条件要素を用いる。

(:from　送信タスク名　ＭＭ要素パターン)
(:task　他のタスク名　ＷＭ要素パターン)

右辺等で実行される文としては、ＷＭ要素に関するｍａｋｅ、ｒｅｍｏｖｅ、ｍｏｄｉｆｙ文、ＭＭ要素に関するｒｅｍｏｖｅ文に相当するａｃｃｅｐｔ文がある。この文によるＭＭ要素の削除をこの要素の受理と言う。

また、タスク間の同期／通信、タスクの階層的生成のため、以下の文が用意されている。

１）並列ａｎｄ型ｒｅｑｕｅｓｔ文：複数のタスクの各々にメッセージを送り、そのすべてが受理されるまで待つ。但し、指定した時間を越えて待たされると：ｅｌｓｅ以下が実行される。：ｗｉｔｈｉｎ以下はなくともよい。

(request　タスク名１　メッセージ１
 :and　　タスク名２　メッセージ２
 :and　　　　　・・・
 :within　時間　　:else　文の並び)

２）非同期ｓｅｎｄ文：タスクにメッセージを送るだけで相手との同期はとらない。

(send　　タスク名　　メッセージ)

３）ｄｅｌａｙ文：指定した時間だけタスクを中断する。

(delay　時間)

４）ｃｒｅａｔｅ文：任意の型のタスクを生成し、その名前を値とする関数でもある。

(create タスク名　型　属性名／値の並び)

システムにタスクを命名させるにはタスク名をダミー（＊）にしておけばよい。最初に生成されるタスクをルートタスクと呼ぶ。

４．Ｐｏ－ＰＳの実行

Ｐｏ－ＰＳはデータ駆動性と要求駆動性の両側面をもつ。ここではＭＥＬＣＯＭ－ＣＯＳＭＯ　７００ⅢのＬＩＳＰ１．９上で実現中のＰｏ－ＰＳインタプリタについて述べる。

４．１　タスクのヒット戦略

Ｐｏ－ＰＳにおけるタスクは、各々独立したタスクインタプリタをもつ拡張プロダクションシステムであり、各インタプリタを逐次マシン上で擬似並列実行させるための戦略を必要とする。

各タスクに割り当てる処理時間の単位として、“１度に実行されるタスクのサイクル数”を選択した。これはＰｏ－ＰＳの使用者からみたシステムの挙動の透析性や制御性を考慮したことによる。タスクのヒット戦略とは、１度に実行するタスクのサイクル数の設定法のことをいう。２つの戦略が用意されている。

１）単数（ｓｉｎｇｌｅ）戦略：この型のタスクは１度に高々１つのインスタンシエイションしか実行することが許されない。

２）全数（ｅｘｈａｕｓｔｉｖｅ）戦略：この型のタスクは競合集合が空になるまで実行を続けることが許される。

タスク型にヒット戦略の指定が省略された場合は、デフォルト値として単数戦略が選択される。複数のタスクのスケジューラ的タスクには全数戦略を指定する方が望ましい。

タスクのヒット戦略としては、その他にもいくつかのバリエーションが考えられよう。しかしヒット戦略はＰｏ－ＰＳを逐次マシン上で実行させるための便宜上の手段であることからここではこれ以上言及しない。

４．２　タスクの状態

Ｐｏ－ＰＳでは、タスクの状態をつぎのように分類して管理している。

１）［ｄｅｌａｙ　Ｕ］：　時刻Ｕまで、ｄｅｌａｙ文を実行中。

２）［ｒｅｑ　Ｔ　Ｍ　Ｕ　Ｅ］：期限付きｒｅｑｕｅｓｔ文を実行中。Ｔ，Ｍ，Ｕ，Ｅはそれぞれ送信先、メッセージ、期限時刻、ｅｌｓｅ節を表わす。

３）［ｓｕｓｐｅｎｄｅｄ］：中断中。

４）［ｔｅｒｍｉｎａｔｅｄ］：終了済み。

5）［ａｂｏｒｔｅｄ］：強制終了済み。
6）［ａｃｔｉｖｅ］：１）～５）以外。

Ｐｏ－ＰＳインタプリタは、各タスクのインタプリタに対し、そのヒット戦略に応じた処理時間を割り当てる。しかし１）～５）の状態にあるタスクについては処理時間を割り当てる必要はない。

４．３　インスタンスツリー

Ｐｏ－ＰＳのインタプリタは、ルートタスクを根とするタスクの親子関係を表わすツリーを維持する。これをインスタンスツリーと呼ぶ。このツリー上では兄弟タスクに相当するノードは年長のものをより左に配置する。

インスタンスツリーの各ノードは、それが表わすタスクを実行するための情報をまとめた活動記録と呼ぶ表である。この表は、タスク生成時につくられたタスクの名前、型名、属性値リスト、ＷＭ／ＭＭの内容、ＷＭ／ＭＭ要素に付けるタグ値の管理表、実行されたインスタンシエーションの履歴および前節で述べた状態などに関する情報を含んでいる。

４．４　Ｐｏ－ＰＳの実行

Ｐｏ－ＰＳインタプリタは、インスタンスツリーとタスク定義表および定数定義表に基づいて各タスクを擬似並列に実行する。

この２つの表は、それぞれタスク型定義および定数定義を効率良く参照するための内部表現である。

図４．１は、Ｐｏ－ＰＳインタプリタを、Ｐａｓｃａｌ風に記述したものである。以下これを簡単に説明する。

このインタプリタは、ルートタスクが生成されると、直ちに呼び出され、つぎの順で各タスクを実行する。

1）ルートタスクの活動記録（変数ｒｏｏｔ）を根とするインスタンスツリーをつくる。
2）現在時刻（変数ｃｌｏｃｋ）の初期設定。
3）ルートタスクの状態が［ｔｅｒｍｉｎａｔｅｄ］になるまでつぎの処理を繰返す。
  - 現在時刻の更新。
  - 手続きｔｒａｖｅｒｓｅにより、インスタンスツリーの各ノードを横型探索で訪れ、このノードが表わすタスクをその状態、ヒット戦略に応じて手続きｅｘｅｃｕｔｅで実行。

```
program Po-PS-Interpreter;
var root:  activation_record;
    clock: integer; { current time }
    ...
function children(t:activation_record):activation_record_queue;
  begin children := the queue of children of the task t end;
function head(q:activation_record_queue):activation_record;
  begin head := the first element of q end;
function tail(q:activation_record_queue):activation_record_queue;
  begin tail := q with the first element removed end;
function append(x,y:activation_record_queue):activation_record_queue;
  begin append := concatenation of x, y end;
    ...
function now():integer;
  begin now := clock end;
procedure execute(t:activation_record);
  begin
    case t.state of
      [active]:
         execute t according to its hit strategy;
      [delay Until]:
         if Until > now() then resume(t) endif;
      [req To Message Until Else]:
         if Message has been accepted then resume(t)
         else if Until > now() then execute Else
         endif;
      [terminated],[aborted],[suspended]: ;
    endcase
  end;
procedure resume(t:activation_record);
  begin
    t.state := [active];
    resume the execution of t according to its hit strategy
  end;
procedure traverse(succ:activation_record_queue);
  begin
    if succ <> nil then
      execute(head(succ));
      traverse(append(tail(succ), children(head(succ))))
    endif
  end;
begin
  root  := the root task's activation record;
  clock := 0; { initialize the current time }
  repeat clock := clock + 1;
         traverse(the queue comprising the value of root only)
  until  root.state = [terminated]
end.
```

図４．１　Ｐｏ－ＰＳインタプリタ

手続きｔｒａｖｅｒｓｅの引数は、つぎに訪問すべきノードのキューである。手続きｅｘｅｃｕｔｅの内容は以下の通りである。

1）タスクの状態が［ａｃｔｉｖｅ］のとき、ヒット戦略に従ってタスクを実行する。
2）状態が［ｄｅｌａｙ　Ｕ］のとき、時刻が期限Ｕを過ぎていれば再開する。
3）状態が［ｒｅｑ　Ｔ　Ｍ　Ｕ　Ｅ］のとき、メッセージＭがタスクＴにより受理済みであれば、再開する。期限Ｕが過ぎていればｅｌｓｅ節Ｅを実行する。
4）その他の状態のとき、タスクに処理時間を与えない。

４．５　Ｐｏ－ＰＳにおける時間の概念

タスク型定義で使用され、現在時刻を参照する関数ｎｏｗは図４．１のプログラムにおける変数ｃｌｏｃｋの値を返す。この値は、Ｐｏ－ＰＳインタプリタがインスタンスツリーの根を訪れる度に１づつ加えられる初期値が０の整数である。

５．雑誌編集プロセスへの応用

Ｐｏ－ＰＳの事象駆動型システムへの応用として、学術雑誌の編集プロセスのモデリングを取り上げる。ここで扱う問題はＺｉｓｍａｎ[6]、小林ら[1,2]によって分散型プロダクションシステムによるモデル化が試みられたものと同一である。

図５．１に雑誌編集プロセスの処理階層の一例を示す。編集プロセスはつぎのような手順で進められる。

１）始めは、編集者と論文の著者だけがいるものとする。

２）論文が投稿される毎に、これを処理するための独立な編集プロセスが起動される。

３）編集プロセスは編集者に論文の査読者を２名指定することを要求して、２つの査読者タスクを生成し、査読者名の指定を待つ。

４）査読者が指定されると、その各々に対して査読者監視プロセスをつくり、これを管理する。

図５．２は雑誌編集プロセスの核となる編集プロセスおよび査読監視プロセスの作業の流れをペトリネットによって表現したものである。この図の各トランジションに付随するプロダクションルールを示したのが図５．３である。

図５．４は雑誌編集プロセスをＰｏ－ＰＳによって実際に記述したものである。以下この図を簡単に説明する。

１）編集者、著者、査読者の各タスクは、実行時に査読者の名前、論文題目、判定結果などを非決定的に指定できるよう端末との対話型タスクとした。

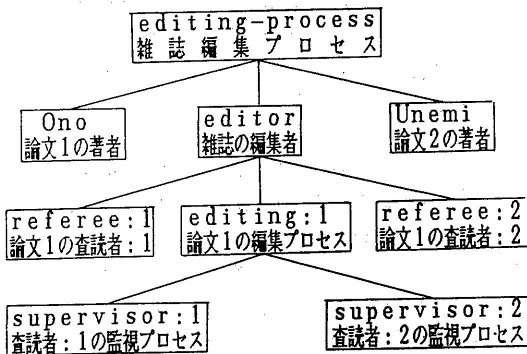

図５．１　雑誌編集プロセスの階層

２）編集および査読者監視タスクはできるだけ図５．２－３を忠実に再現することをこころがけた。非同期のメッセージ通信だけを用いたが、期限付きｒｅｑｕｅｓｔ文を用いるとより簡潔に表現できる。

３）この例題では時間の概念が重要であるが、時間は現在時刻を参照する関数ｎｏｗによって表現されている。

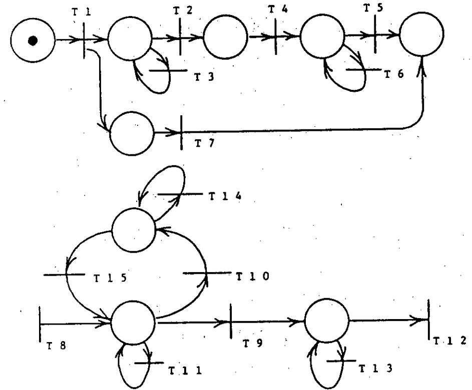

図５．２　編集プロセスのペトリネット表現

（Ｔ１）無条件
＝＞著者に謝状を送り、編集者に査読者（２人）の指定を依頼する。

（Ｔ２）編集者が査読者のリストを提出する。
＝＞査読者の各々に対して査読者プロセスを発生し、Ｔ８を発火可能にする。

（Ｔ３）Ｔ３が発火可能になってから２週間経過しても、編集者からの返事が到着しない。
＝＞編集者に催促状を送る。

（Ｔ４）すべての査読活動が終了する。
＝＞編集者に論文に対する最終判定を求める。
編集者が論文に対する最終判定を下す。
＝＞著者に対して最終報告書を提出する。

（Ｔ５）編集者が論文に対する最終判定を下す。
＝＞著者に最終報告書を提出する。

（Ｔ６）Ｔ６が発火可能になってから２週間経過しても、編集者が最終判定を下さない。
＝＞編集者に催促状を送る。

（Ｔ７）著者が論文を撤回する。
＝＞編集プロセスを打ち切る手続きをとる。

（Ｔ８）無条件
＝＞査読者に査読依頼状を送る。

（Ｔ９）査読者から査読を引き受けるとの返事が到着する。
＝＞査読のために１ケ月の期限を与える。

（Ｔ１０）査読者から査読を断るとの返事が到着する。
＝＞編集者に別の査読者を指定するよう依頼する。

（Ｔ１１）Ｔ１１が発火可能になってから２週間経過しても、査読者が査読依頼に対する返事をしない。
＝＞査読者に催促状を送る。

（Ｔ１２）査読者から報告書が到着する。
＝＞査読者に謝状を送る。

（Ｔ１３）Ｔ１３が発火可能になってから１ケ月経過しても、査読者からの報告者が到着しない。
＝＞査読者に催促状を送る。

（Ｔ１４）Ｔ１４が発火可能になってから２ケ月経過しても、編集者が別の査読者を指定しない。
＝＞編集者に催促状を送る。

（Ｔ１５）編集者が別の査読者を指定する。
＝＞査読依頼状を送る。

図５．３　編集プロセスのプロダクション

```
;... Journal Editing Processes ...
#aWeek     = 7
#twoWeeks = 14
#aMonth   = 31

;... Root Task Type ...
(tasktype   editing-process
  initial: (create editor editor ^name "S.Kobayashi")
           (create Unemi  author ^name "T.Unemi"    )
           (create Ono    author ^name "N.Ono"      ) )


;... Interface for Interaction with the Editor ...
(tasktype editor ^name
  hit:     exhaustive
  element: editing           ^author ^title ^from ^task
  message: submit            ^author ^title
           remind-submit     ^author ^title
           withdraw          ^author ^title
           remind-request    ^author ^title
           request-referees  ^author ^title
           judge             ^author ^title ^result1 ^result2
           remind-judge      ^author ^title
           renew             ^author ^title
           remind-renew      ^author ^title
  routine:
    p1 (:from ?f submit ^author ?a ^title ?t)
     =>(make editing ^author ?a ^title ?t ^from ?f
            ^task (create * editing ^author ?a ^title ?t ^from ?f))
       (accept 1)
    p2 (:from ?f withdraw ^author ?a ^title ?t)
       (editing ^author ?a ^title ?t ^from ?f ^task ?task)
     =>(send ?task withdraw ^author ?a ^title ?t)
       (remove 2)
       (accept 1)
    p3 (:from ?task remind-request  ^author ?a ^title ?t)
       (:from ?task request-referees ^author ?a ^title ?t)
       (editing ^author ?a ^title ?t ^task ?task)
     =>(call REQUEST-REFEREE-LIST (@ *name)(@ ?a)(@ ?t))
       (send ?task referee-list ^author ?a ^title ?t
             ^ref1 (create * referee ^name (call READLINE))
             ^ref2 (create * referee ^name (call READLINE)))
       (accept 1 2)
    p4 (:from ?task judge ^author ?a ^title ?t ^result1 ?r1 ^result2 ?r2)
       (:from ?task remind-judge ^author ?a ^title ?t)
       (editing ^author ?a ^title ?t ^task ?task)
     =>(call REQUEST-FINAL-DECISION (@ *name)(@ ?a)(@ ?t)(@ ?r1)(@ ?r2))
       (send ?task judgement ^author ?a ^title ?t ^result (call READ))
       (accept 1 2)
    p5 (:from ?ref renew ^author ?a ^title ?t)
       (:from ?ref remind-renew ^author ?a ^title ?t)
     =>(call REQUEST-REFEREE-RENEWAL (@ *name)(@ ?a)(@ ?t))
       (send ?ref renew ^author ?a ^title ?t
             ^ref (create * referee ^name (call READLINE)))
       (accept 1 2) )


;... Editing Process ...
(tasktype  editing    ^author ^title ^from
  hit:     exhaustive
  element: paper          ^status ^since ^ref1 ^ref2
  message: referee-list ^author ^title ^ref1 ^ref2
           reply          ^result
           judgement      ^author ^title ^result
           withdraw       ^author ^title
  routine:
  t2 (paper ^status request)
     (:from editor
       referee-list ^author *author ^title *title ^ref1 ?r1 ^ref2 ?r2)
   =>(modify 1 ^status review ^since (now)
       ^ref1 (create * supervisor ^author *author ^title *title ^ref ?r1)
       ^ref2 (create * supervisor ^author *author ^title *title ^ref ?r2))
     (accept 2)
  t3 (paper ^status request ^since<(- (now) #twoWeeks))
   - (:from editor referee-list ^author *author ^title *title)
   =>(send editor remind-request ^author *author ^title *title)
  t4 (paper ^status review ^ref1 ?r1 ^ref2 ?r2)
     (:from ?r1 reply ^result ?result1)
     (:from ?r2 reply ^result ?result2)
   =>(send editor
           judge ^author *author ^title *title ^result1 ?a1 ^result2 ?a2)
     (modify 1 ^status judge ^since (now))
     (accept 2 3)
  t5 (paper ^status judge)
     (:from editor
       judgement ^author *author ^title *title ^result ?result)
   =>(send *from report ^result ?result ^author *author ^title *title)
     (accept 2)
  t6 (paper ^status judge ^since<(- (now) #twoWeeks))
   - (:from editor judgement ^author *author ^title *title)
   =>(send editor remind-judge ^author *author ^title *title)
  t7 (:from editor withdraw ^author *author ^title *title)
   =>(accept 1)
     (terminate)
  initial:
     (send *from acknowledge ^author *author ^title *title)
     (send editor request-referees ^author *author ^title *title)
     (make paper ^status request ^since (now))   )
```

```
;... Referee Supervisor ...
(tasktype  supervisor ^author ^title ^ref
  hit:     single
  element: paper            ^status ^since
  message: reply-to-request ^result
           report           ^result
           renew            ^ref ^name
  routine:
  t9  (paper ^status request)
      (:from *ref reply-to-request ^result yes)
    =>(modify 1 ^status review ^since (now))
      (accept 2)
  t10 (paper ^status request)
      (:from *ref reply-to-request ^result no)
    =>(send editor request-renew ^author *author ^title *title)
      (modify 1 ^status renew ^since (now))
      (accept 2)
  t11 (paper ^status request ^since < (- (now) #twoWeeks))
    - (:from *ref reply-to-request)
    =>(send *ref remind-request ^author *author ^title *title)
  t12 (paper ^status review)
      (:from *ref report ^result ?result)
    =>(send *ref thank-you ^author *author ^title *title)
      (send editing reply ^result ?result)
      (accept 2)
  t13 (paper ^status review ^since < (- (now) #aMonth))
    - (:from *ref report)
    =>(send *ref remind-report ^author *author ^title *title)
  t14 (paper ^status renew ^since < (- (now) #twoWeeks))
    - (:from editor renew)
    =>(send editor remind-renew ^author *author ^title *title)
  t15 (paper ^status renew)
      (:from editor renew ^ref ?ref)
    =>(update ^ref ?ref)
      (modify 1 ^status request ^since (now))
      (send ?ref request-service ^author *author ^title *title)
  initial:
    (make paper ^status request ^since (now))
    (send *ref request-service ^author *author ^title *title)
  final:
    (send *ref abort-by-withdrawal ^author *author ^title *title)  )


;... Interface for Interaction with the Author ...
(tasktype  author ^name
  hit:     single
  element: paper        ^status ^since ^title
  message: acknowledge ^author ^title
           report       ^result ^author ^title
  routine:
    p1 (paper ^status writing ^title nil)
     =>(call REQUEST-PAPER-TITLE)
       (modify 1 ^title (call READLINE))
    p2 (paper ^status writing ^title ?t <> nil)
     =>(modify 1 ^status  submitting ^since (now))
       (send editor submit ^author *name ^title ?t)
    p3 (paper ^status submitting ^since < (- (now) #twoWeeks) ^title ?t)
     - (:from ?editing acknowlege ^author *name ^title ?t)
     =>(send editor remind-submit ^author *name ^title ?t)
    p4 (paper ^status submitting ^title ?t)
       (:from ?editing acknowledge ^author *name ^title ?t)
     =>(modify 1 ^status waiting ^since (now))
       (accept 2)
    p5 (paper ^status waiting ^since < (- (now) #aMonth) ^title ?t)
     - (:from editing report ^result ?r ^author *name ^title ?t)
     =>(send editor remind-report ^author *name ^title ?t)
    p6 (paper ^status waiting ^title ?t)
       (:from ?editing report ^result ?r ^author *name ^title ?t)
     =>(modify 1 ^status writing ^since (now) ^title nil)
       (accept 2)
  initial:(make paper ^status writing ^since (now) ^title nil) )


;... Interface for Interaction with the Referee ...
(tasktype referee ^name
  hit:     single
  message: request-service      ^author ^title
           remind-request       ^author ^title
           remind-report        ^author ^title
           thank-you            ^author ^title
           abort-by-withdrawal ^author ^title
  routine:
    p1 (:from ?ref request-service ^author ?a ^title ?t)
       (:from ?ref remind-request  ^author ?a ^title ?t)
     =>(call REQUEST-SERVICE (@ ?a)(@ ?t))
       (send ?ref reply-to-request ^result (call READ))
       (accept 1 2)
    p2 (:from ?ref remind-report ^author ?a ^title ?t)
     =>(call REQUEST-DECISION (@ ?a)(@ ?t))
       (send ?ref report ^result (call READ))
       (accept 1)
    p3 (:from ?ref thank-you ^author ?a ^title ?t)
     =>(terminate)
    p4 (:from ?ref abort-by-withdrawal ^author ?a ^title ?t)
     =>(call ABORT-BY-WITHDRAWAL (@ ?a) (@ ?t))
       (terminate) )
```

図５．４　雑誌編集プロセスのＰｏ－ＰＳによる記述

６．おわりに

６．１　Ｐｏ－ＰＳの言語としての位置付け

プログラミング言語や方法論の観点からＰｏ－ＰＳについて簡単に触れておく。

１）並列型対象指向言語としての性格：Ｐｏ－ＰＳは対象指向性をもった並列処理言語とみなせる。Ｐｏ－ＰＳにおけるタスクは対象指向言語[11]における対象よりもずっと自律的である。即ち、メッセージが到着しなくとも実行すべきことがあればこれを遂行し、メッセージが到着してもそれを受理するに足る状況に至らなければ受理しない。またその受理順を単に到着順とすることも、実行時にある程度制御することもできる。

２）ストリーム入出力の実現：タスクのメッセージメモリは、メッセージパターンにより振るい分けられたメッセージの複数個のキューとみなせる。したがって関数型言語[9]や論理型言語[5]における基本的機構であるストリーム入出力を実現できる。

６．２　事象駆動型システムの表現と制御

ＰＳの記述能力とその限界を考察する。

１）事象駆動型システムのモデリング：このレベルの記述に関してはＰｏ－ＰＳと同様の目的で開発されたＳＣＯＯＰ[6]やＣｏ－ＰＳｓ[2]で扱われている問題はより自然に表現できる。もっと複雑な競合の解消や協調が要求されるシステムにも適用できよう。

２）事象駆動型システムの管理と制御：この領域では実時間の扱いが不可欠となる。現在のＰｏ－ＰＳの適用はオフィス業務のようにタイミングに関する制約が比較的緩いシステムのモニタリング、例えばＳＣＯＯＰで記述された雑誌編集管理システムのようなものに限られよう。

Ｐｏ－ＰＳでは一般にメッセージの送信先のタスク型は実行時に決まる。したがって送信するメッセージの形式の妥当性は相手によって実行時に確認される。Ｓｍａｌｌｔａｌｋ－８０[11]でも同様のことが起こるが、このような実行時のオーバヘッドも場合によっては問題になるであろう。

３）分散協調型問題解決システムの記述：Ｐｏ－ＰＳは、このための機構を陽には提供していない。黒板モデル[10]や並列ＡＮＤ／ＯＲ型推論[5]も本質的にはタスクの同期問題であり、並列ｒｅｑｕｅｓｔ文により実現できる（Ｐｏ－ＰＳには並列ｏｒ型ｒｅｑｕｅｓｔ文もある）が、この分野における言語としては機械語レベルのものとしか言えない。現在のＰｏ－ＰＳインタプリタは第１の分野における使用を意図しており、問題解決プロセスのように多くのタスクが推論の連鎖をつくって待ち状態にあり、少数のタスクだけが実行可能状態にあるようなシステムには向かない。

以上の問題点を段階的に解決し、Ｐｏ－ＰＳの応用領域を拡大していくことは、Ｐｏ－ＰＳのためのプログラミング環境の構築と共に今後の課題である。

参考文献

[1] 市川惇信・小林重信：事象駆動型システムの表現と制御 22,10(1982)

[2] 小林重信・畝見達夫：分散型プロダクションシステムＣｏ－ＰＳｓによる事象駆動型システムのモデリング，情報処理学会第28回全国大会論文集(1984)

[3] 小林重信・畝見達夫・望月浩二：参考文献解析エキスパートシステム，情報処理学会第34回知識工学と人工知能研究会資料(1984)

[4] 田代勤・春名公一・薦田憲久：プロダクションシステムを用いた離散型システムの運用自動決定機構，第１回ＳＩＣＥ知識工学シンポジウム資料(1983)

[5] 古川康一・国藤進・竹内彰一・上田和紀：核言語第１版概念仕様書(1984)

[6] M.D.Zisman:Use of Production Systems for Modeling Asynchronous, Concurrent Processes, in Pattern-Directed Inference Systems (1978)

[7] C.A.R.Hoar: Communicating Sequential Processes, CACM, Vol.21, No.8 (1978)

[8] C.L.Forgy: OPS5 User's Manual, Dept. of Computer Science, CMU (1981)

[9] D.A.Turner: Recursion Equations as a Programming Language, in Functional Programming and its Applications (1982)

[10]H.P.Nii et al: Age Reference Manual, Computer Science Department, Stanford University (1981)

[11]A.J.Goldberg et al: Smalltalk-80:the language and its implementation (1983)