Vol. 27 No. 6 情報処理学会論文誌 June 1986

通信/同期機構をもつ分散型プロダクションシステム Po-PS†

小野典彦†† 小林重信††

事象駆動型システムの表現と制御を目的とした分散型プロダクションシステム（分散型 PS）の記述言語 Po-PS の概要が示される．従来の分散型 PS の記述言語は事象駆動型システムの論理構造をそのまま表現できるという利点をもつ反面，PS 間の相互作用が陽に表現できないため，相互作用の間の依存関係が不明確となり，それが行われる順序やタイミングを適切に表現，制御するのが難しい．Po-PS の特徴は PS 間の相互作用を陽に表現する点にある．Po-PS では，事象駆動型システムは，非同期，同時進行的な逐次型タスクの集まりとして認識することにより，その論理構造を失うことなくモデル化される．各タスクは，メッセージを介した通信/同期機構が付加された拡張 PS として表現されるために，対象の事象駆動的な挙動が自然にモデル化される．タスクはその原型を表すタスク型により記述され，タスク型の複製として生成される．タスク型およびタスクは対象指向言語におけるクラスおよびインスタンスに相当する概念であり，タスク型間での定義記述の静的継承機構も実現されている．本論文では哲学者の食事問題の記述例が示される．Po-PS の有効性は，種々の同期/スケジューリング問題，オフィス業務のモデル化，線画のラベル付け問題などへの応用経験から確認されている．

1. はじめに

複数の同時進行的な対象から構成され，各対象は事象を発生するとともに，他の対象が発生する事象に反応して別の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．

近年，分散型プロダクションシステム（以下分散型 PS という）を用いて事象駆動型システムの表現と制御を行う試みがいくつかみられる．集中医療モニタ VM[9]，オフィス業務自動化システム SCOOP[8]，音声理解システム Hearsay-II[13]，IBM MVS オペレーティングシステムのオペレータ支援システム YES/MVS[10]などはこの例である．また事象駆動型システムの記述に向いた分散型 PS 記述用言語として APN[7]，Age[14]，Co-PSs[1),2)] などが提案されている．

分散型 PS によるモデル化の利点は，その対象指向性，事象駆動性，柔軟性および拡張性の四つに要約される．

従来の分散型モデルでは，PS として表現された対象間の相互作用を陽に記述する機構が組み込まれていない．例えば，VM，SCOOP および Age の各モデルでは相互作用は対象間の共有メモリを介して間接的に行われる．YES/MVS や Co-PSs では，対象間のデータ送信および共有メモリによって相互作用が表現される．

著者らは，事象駆動型システムにおける対象間の相互作用を最近の手続き型言語における並列処理や通信/同期のための機構で表現することに基礎をおく分散型 PS 記述言語 Po-PS (Procedure Oriented Production Systems) を設計し，MELCOM-COSMO 700 IIIの Lisp 1.9[6] 上に実現したので報告する．

2. 基本設計

2.1 応用領域

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ル化：FA および OA システムの設計，拡張段階におけるシステムのモデル化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制御．

3) 分散型問題解決システムの記述：分散型問題解決システムにおける知識表現や推論のための枠組みの提供．

2.2 設計方針

以下の機能をもつことを Po-PS の設計方針とした．

1) 対象の同時進行性：システムを互いに非同期，同時進行的な対象の集まりとして表現し得ること．

2) 対象の階層的生成：対象はその活動中に新しい対象を動的に生成，管理し得ること．

3) 対象間の通信/同期機構：発生した事象を対象間で通知し合えること．また対象間の同期が行えること．

† Distributed Production Systems with Communication and Synchronization Mechanism; Po-PS by NORIHIKO ONO and SHIGENOBU KOBAYASHI (Department of Systems Science, Graduate School of Science and Engineering at Nagatsuta, Tokyo Institute of Technology).

†† 東京工業大学大学院総合理工学研究科システム科学専攻

4) 事象に対する反応の制御：他の対象の発生する事象の順序およびタイミングはあらかじめ予測し尽くすことはできない．そこで，対象が現在反応可能な事象の範囲ならびにそれらの間の優先順序を柔軟で明確な形式で表現，制御し得ること．

5) 対象の経済的表現：対象間の性質の継承機構を導入することにより対象の経済的な表現が可能であること．

2.3 基本モデル

上記の応用領域ならびに設計方針を基に設計した分散型 PS のモデルの概要を示す．

2.3.1 システムの構成要素

Po-PS では，事象駆動型システムは互いに非同期，同時進行的な逐次型**タスク**の集まりとして表現される．タスクは，その**サブタスク**として，新たにタスクを生成できる．

システムにおける対象およびそれが生成し，同時進行的に行う活動はすべてタスクとして表現される．

2.3.2 タスク間の相互作用

タスクは，つぎの 1)～6) に示すメッセージを介した通信/同期機構をもつ．

1) **非同期型送信**：他タスクにメッセージを送信する．

2) **同期型送信**：他タスクにメッセージを送信し，その受信タスクがそれに対して**受理**と呼ぶ操作を施すまで待つ．

3) **並列 and 型の同期型送信**：複数のタスクにメッセージを送信し，それらがすべて受理されるまで待つ．

4) **並列 or 型の同期型送信**：複数のタスクにメッセージを送信し，それらの一つが受理されるまで待つ．

5) **ブロードキャスト**：タスクは**チャネル**と呼ばれる通信路を設定し，それに対して**接続**および**接続解除**と呼ぶ操作を施すことができる．タスクはあるチャネルに接続中のすべてのタスクに対してメッセージを送信できる（これをメッセージのブロードキャストという）．

6) **遅延**：一定時間だけ実行を中断する．これにより，特定の時間だけ，事象の発生を遅らせたり，他タスクによる事象の発生を待つことができる．

メッセージの同期型送信/受理は CSP[20] における出力/入力に相当し，これにより CSP で提案された待ち合わせ (rendezvous) 方式のタスク間の同期が行われる．ただし，タスクは 2.3.3 節で述べるような PS であるので，その基本サイクル (recognize-act cycle) の中途で受信したメッセージは次回以降のサイクルでないと受理できないなどの制約がある．

2.3.3 タスクの枠組み

タスクは通常の PS の枠組みに上記の通信/同期機構を付加した**拡張 PS** として表現される．**図 1** に拡張 PS の基本的枠組みを示す．以下にその概略を示す．

1) **ワーキングメモリ** (Working Memory—WM)：タスクの内部状態の記憶場所であり，通常の PS の WM に相当する．WM はタスクの生成時に実行される**イニシャル節**（後述）により初期化され，プロダクションの実行により参照，更新される．

2) **メッセージメモリ** (Message Memory—MM)：他のタスクから送信されたメッセージの記憶場所である．MM に記憶されたメッセージは，プロダクションの実行により参照，受理される．

3) **プロダクションメモリ** (Production Memory—PM)：WM と MM の変化に事象駆動型に反応して実行されるプロダクションの集まりである．

プロダクションは，構文的には，OPS 5[11] におけるプロダクションをつぎのように拡張したものである．すなわち，左辺では，WM だけでなく，MM および時間を参照でき，右辺では，WM の更新操作に加えて，2.3.2 節で示したタスク間の通信/同期機構，メッセージの受理，チャネルの設定，チャネルに対する接続/接続解除およびタスクの生成など

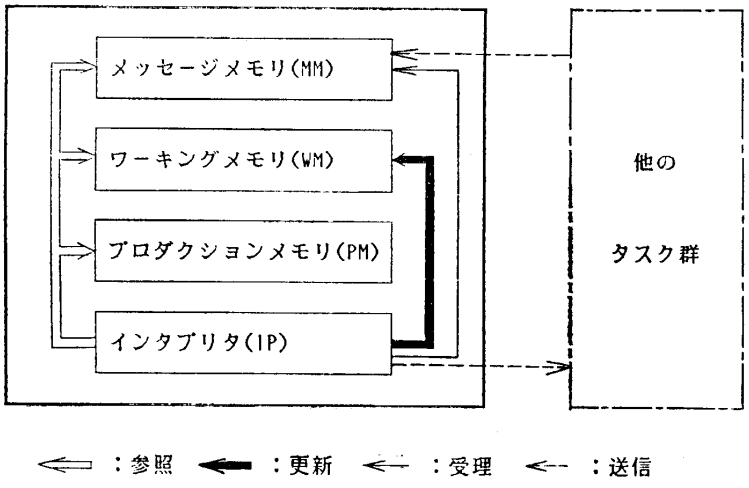

図 1 拡張プロダクションシステムの枠組み
Fig. 1 Framework of an augmented production system.

の操作が施せる．左辺での時間参照は実時間制御への応用に使われる．

4) **インタプリタ** (Interpreter—IP)：イニシャル節を実行した後，OPS 5 と同様の基本サイクルによりプロダクションを実行する．

対象をこのような拡張 PS で表現することにより，それが反応し得る事象の範囲およびそれらの優先順位を柔軟で明確な形で表現できる．

3. 言語仕様

3.1 タスク型

システムは，それを構成する各タスクの原型を表す**タスク型**の集まりとして記述され，タスクは，システムの実行時にタスク型の複製として生成される．

タスク型は**図2**(a)に示す**タスク型定義**で記述される．

タスクはそのタスク型で指定された**パラメータ**（記号'^'で始まる名前）をもつ．その値は，タスクの生成時に設定され，タスクから参照可能である．

タスクは図2(b)の形式の create 文により生成される．

タスク型とタスクの関係が，対象指向言語[15)~17)]におけるクラスとインスタンスの関係に相当するのに対して，**上位タスク型**は，上位クラスに相当する概念であり，各タスク型は，システムの実行前に，その上位タスク型のパラメータおよび諸宣言を深さ優先で多重継承する．

宣言の各々に関しては以下の節で述べる

3.2 拡張プロダクションシステム

タスクの枠組みとなる拡張 PS の構成要素の仕様ならびにその宣言方法を示す．

3.2.1 ワーキングメモリ

WM はタスク自身が記録したデータの記録順および内容を表すつぎの2項組の集合である．

〈タイムタグ，ワーキングメモリ要素〉

タイムタグは WM 要素の記録順を示す整数値であり，最近記録された WM 要素のタイムタグほど大きい．

WM 要素は，つぎの例のようにクラス名および属性/値の並びよりなるリストである．

(resource ^type LBP ^no 9 ^status busy) (1)

タスクの WM 要素の形式は，そのタスク型で宣言される．例として WM 要素(1)の宣言(2)を示しておく．

element : resource ^type ^no ^status (2)

```
(tasktype 〈タスク型名〉〈パラメータの並び〉
  supers : 〈上位タスク型名の並び〉
           〈ヒット戦略の宣言〉
           〈メッセージの形式の宣言〉
           〈WM 要素の形式の宣言〉
           〈カテゴリの宣言の並び〉
           〈イニシャル節の宣言〉)
```

(a) Format of task type definition.

(create 〈タスク名〉〈タスク型名〉〈パラメータ/値の並び〉)

(b) Format of task creation statement.

図2 タスクの定義と生成
Fig. 2 Definition and creation of task.

3.2.2 メッセージメモリ

MM は他のタスクより送信されたメッセージの到着順，送信タスクおよび内容を表すつぎの3項組の集合である．

〈タイムタグ，送信タスク名，メッセージ〉

タイムタグはメッセージの到着順を示す整数値であり，先に到着したメッセージのタイムタグほど大きい．

メッセージは WM 要素と同形式であり，タスクが受信し得るメッセージの形式は，そのタスク型でつぎのように宣言される（この例は二つを同時に宣言する）．

```
message : callfor ^type ^no ?
          release ^type ^no          (3)
```

ここで，記号'?'の後置された属性は**返答型属性**と呼ぶ特別な属性であり，それを含むメッセージはつぎの性質1)～3)をもつ．

1) 同期型の送信（並列 and/or 型も含む）だけが可能．

2) 返答型属性の値は，値が未結合の変数（**返答変数**という）でなければならない．

3) 返答変数は，受信タスクが，メッセージの受理時に指定する値に結合される．

例えば，宣言(3)を含むタスク型のタスクに対して，つぎのメッセージ(4)を同期型送信することにより，返答変数'?no'の値をその受信タスクによって決定させることができる．

(callfor ^type LBP ^no ? no) (4)

3.2.3 プロダクションメモリ

PM はプロダクションの格納場所であり，**カテゴリ**と呼ぶプロダクションの集合に分割されて記憶され

```
p1(:from ?user  callfor ^type ?t)          (5)
- (:from ?other release ^type ?t)          (6)
  (resource ^type ?t ^no ?n ^status free)(7)
=>(accept 1 ^no ?n)                       (8)
  (modify 2 ^status busy)                 (9)
```

(a) Example 1.

```
demon(self ^status waiting ^since ?t)     (10)
    (:where < (+ ?t 24) (:clock))         (11)
  =>(modify 1 ^status reminding)          (12)
```

(b) Example 2.

図3 プロダクションの例
Fig. 3 Examples of productions.

る．タスクのカテゴリは，そのタスク型定義中につぎの形式でそれぞれ独立に宣言される．

〈カテゴリ名〉〈プロダクションの並び〉

プロダクションの形式はつぎのとおりである．

〈プロダクション名〉〈左辺〉=>〈右辺〉

左辺は，WM および MM に関する記述を目的とする**条件要素**および時間記述を主たる目的とする**制約**の並びであり，その各々が真となるときに限り，真となる．条件要素には，WM 要素あるいはメッセージのパターンである**正の条件要素**およびその前に記号'-'を付加した**負の条件要素**がある．正の条件要素は，そのパターンに照合する WM 要素やメッセージがあれば真となり，負の条件要素は照合する WM 要素やメッセージが存在しないと真となる．制約については後述する．

右辺には，WM の更新に関する文のほか，付録に示すタスク間の相互作用のための文の列が記述される．

以下では，**図3**に示す二つの例を用いてプロダクションの形式，機能を示す．

宣言(1)(2)を含むタスク型定義では，図3(a)のプロダクションを要素とするカテゴリを宣言できる．

このプロダクションは，正の条件要素(5)(7)および負の条件要素(6)を左辺とする．この例のように，条件要素を表す WM 要素およびメッセージのパターンはそれぞれつぎのように記述される．

(〈クラス名〉〈属性/値のパターンの並び〉)

(:from 〈送信タスク名のパターン〉

〈クラス名〉〈属性/値のパターンの並び〉)

送信タスク名および属性値のパターンとしては OPS 5 におけるアトム，述語，選言，連言，変数('?'で始まる名前）のほか，**リストのパターン，パラメータ値および関数値**などが記述できる．

この例では右辺は二つの文(8)(9)よりなる．

(8)はメッセージを受理するとともにその返答型属性'^no'で指定された返答変数に値を結合するための文 (accept 文) であり，左辺の1番目の正の条件要素(5)に照合するメッセージを受理するとともにこの返答変数を変数'?n'の値に結合する．この文で受理されたメッセージは自動的に MM より削除される．

(9)は WM 要素を書き換えるための文 (modify 文) であり，左辺の2番目の正の条件要素(7)に照合する WM 要素の属性'^status'の値を'busy'に書き換える．

図3(b)に示すプロダクションは左辺に制約を含む．

制約は，つぎの形式であり，指定された関数をその引数の値の並びに作用させた結果を値とする．

(: where 〈論理型の関数名〉〈引数の並び〉)

条件要素(10)がつぎの表明を表すものとしよう．

「状態'waiting'が時刻'?t'から継続中である」

すると，制約(11)における'(:clock)'は，現在時刻を返す組み込み関数であるので，このプロダクションはつぎのような意味になる．

「状態'waiting'が24単位時間を越えて続いたら，
状態'reminding'に移りなさい」

この例のように，制約は，時間記述などの副作用的な条件記述に使われる．

3.2.4 イニシャル節

タスクが生成されたときに実行すべき文の列であり，そのタスク型においてつぎのように宣言される．

initial:〈イニシャル節となる文の列〉

3.2.5 インタプリタ

タスクのインタプリタは，そのイニシャル節を実行した後，つぎの3ステップよりなる基本サイクルを繰り返し実行する．

〔照合〕 左辺の条件要素がすべて真となるプロダクションおよびその左辺に照合する WM要素/メッセージの組よりなる対の集合を求める．この対を**インスタンシエイション** (instantiation)，その集合を**競合集合**と呼ぶ．

〔競合解消〕 OPS5 の LEX 戦略[11]に準拠した競合解消戦略により，競合集合中のインスタンシエイションの優先順位を決定する（メッセージのタイムタグは，WM 要素のタグよりも大きくなるように設定さ

```
loop
  階層木上の節を広さ優先に訪れながら,
  節に対応するタスクのインタプリタを実行
end loop
```

図 4 システムの実行過程
Fig. 4 Execution processes of systems.

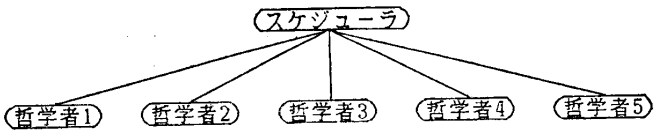

図 5 例題システムの階層木
Fig. 5 The hierarchy tree of example system.

れるため，メッセージに照合するプロダクションのインスタンシエイションが優先される)．

〔実行〕 実行可能なインスタンシエイションがあれば，その中で優先順位の最も高いものを選び，それを構成するプロダクションの右辺の各文を順に実行する．

3.3 システムの実行

現在の Po-PS の処理系では，タスクは，並列に実行されるのではなく，そのタスク型で宣言された**ヒット戦略**に従って数サイクルずつ交互に実行される．この戦略には，1サイクルの実行を許す 'single' および実行可能なインスタンシエイションが尽きるまで実行を許す 'exhaustive' の二つがあり（デフォルトは前者)，つぎのように宣言される．

hit : 〈戦略名〉

システムはタスクを一つ生成することにより起動され，このタスクの終了とともに終了する．このタスクを根として，各タスクを節，それらの間の親子関係を枝とする木を**階層木**と呼ぶことにすると，システムの実行は，**図 4** のループの実行に相当する．

現在時刻を返す関数 '(: clock)' のデフォルト値は，階層木の探索回数に設定されている．

4. 例　　題

通常の「哲学者の食事」問題につぎの変形を施したものを Po-PS で記述した例を示す．

1) 哲学者が食事および思索に要す時間ならびに平然と食事を待てる時間（許容時間）には個人差がある．
2) 哲学者は左右のフォークを同時に取る．
3) 食事順はつぎの規則を順に適用して決定する．
   ①食事の待ち時間が許容時間を越えている者の食事を優先する．
   ②食事の待ち時間の長い者の食事を優先する．

哲学者およびフォークには時計回りに1～5の識別番号が付けられ，哲学者とその右隣のフォークの識別番号は一致するものとする．

4.1 システムの概要

この例題は，哲学者およびそれらの共有資源となるフォークのスケジューラの各々に相当するタスクの集まりとしてモデル化できる（簡単のため，これらをそれぞれ哲学者およびスケジューラという)．ここでは哲学者をスケジューラのサブタスクとして生成する．**図 5** にシステムの階層木を示す．

哲学者は，食事前に，フォークを要求するメッセージ（要求メッセージ）をスケジューラに同期型送信し，それが受理されてから食事を開始する．

ただし，哲学者は，許容時間を越えて食事を待たされると，フォークを催促するメッセージ（催促メッセージ）をスケジューラに非同期型送信する．

一方，食事を終えた哲学者は，フォークの解放を通知するメッセージ（解放メッセージ）をスケジューラに非同期型送信した後，思索を開始する．

4.2 システムの記述

哲学者およびスケジューラの各々を記述したタスク型 'philosopher', 'scheduler' の定義を**図 6** に示す．

以下，これらのタスク型について解説する．

〔タスク型 scheduler〕

スケジューラは，まずイニシャル節において，フォークの状態を初期化し，哲学者 1～5 ('ph-1'～'ph-5') を 'philosopher' 型のタスクとして生成する．各哲学者の識別番号，食事時間，思索時間，許容時間はその生成時にパラメータ ^no, ^dine, ^think, ^endure の値として指定される．

空き状態および使用状態にあるフォークはそれぞれつぎの WM 要素で表される．

(fork ^no 〈フォークの識別番号〉 ^status down)
(fork ^no 〈フォークの識別番号〉 ^status up)

哲学者から送信される要求，解放および催促メッセージはそれぞれつぎの形式である．

(use ^no 〈哲学者の識別番号〉)
(release ^no 〈哲学者の識別番号〉)
(remind)

哲学者の PM はプロダクション 's1'～'s3' からなる唯一のカテゴリ 'scheduling :' で構成される．以下に各プロダクションの意味の概略を示す．

s1 :「哲学者から解放メッセージが到着しているなら，

```
(tasktype  scheduler
 hit:      exhaustive
 message:  use      ^no
           release  ^no
           remind
 element:  fork     ^no ^status
 scheduling:
 s1(:from ?phil release ^no ?n)
   (fork ^no          ?n        ^status up)
   (fork ^no (+ (:mod ?n 5) 1) ^status up)
 =>(accept 1)
   (modify 2 ^status down)
   (modify 3 ^status down)
 s2(:from ?phil use ^no ?n)
   (:from ?phil remind)
   (fork ^no          ?n        ^status down)
   (fork ^no (+ (:mod ?n 5) 1) ^status down)
 - (:from ?any release)
 =>(accept 1)
   (accept 2)
   (modify 3 ^status up)
   (modify 4 ^status up)
 s3(:from ?phil use ^no ?n)
   (fork ^no          ?n        ^status down)
   (fork ^no (+ (:mod ?n 5) 1) ^status down)
 - (:from ?any release)
 - (:from ?any remind)
 =>(accept 1)
   (modify 2 ^status up)
   (modify 3 ^status up)
 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 50 ^think 40 ^endure 30)
   (create ph-2 philosopher
          ^no 2 ^dine 10 ^think 50 ^endure 40)
   (create ph-3 philosopher
          ^no 3 ^dine 20 ^think 10 ^endure 50)
   (create ph-4 philosopher
          ^no 4 ^dine 30 ^think 20 ^endure 10)
   (create ph-5 philosopher
          ^no 5 ^dine 40 ^think 30 ^endure 20))
```

(a) Task type scheduler.

```
(tasktype philosopher ^no ^dine ^think ^endure
 hit:      single
 element:  self ^activity
 duty:
 d1(self ^activity waiting)
 =>(request (:parent) use ^no *no
     :within *endure
     :else (send (:parent) remind)  )
   (modify 1 ^activity dining)
 d2(self ^activity dining)
 =>(delay *dine)
   (send (:parent) release ^no *no)
   (modify 1 ^activity thinking)
 d3(self ^activity thinking)
 =>(delay *think)
   (modify 1 ^activity waiting)
 initial:(make self ^activity waiting)  )
```

(b) Task type philosopher.

図 6 スケジューラおよび哲学者のタスク型定義
Fig. 6 Task types of scheduler and philosophers.

それを受理し，その哲学者の両側のフォークを空き状態にする」

s2:「解放メッセージがすべて受理されており，かつ，要求/催促メッセージを両方とも送信している哲学者の中に食事可能な者がいるなら，それが送信した要求/催促メッセージを受理し，その哲学者の両側のフォークを使用状態にする」

s3:「解放/催促メッセージがすべて受理されており，かつ，要求メッセージを送信している哲学者の中に食事可能な哲学者がいるなら，その要求メッセージを受理し，その哲学者の両側のフォークを使用状態にする」

スケジューラの挙動がこの例題の仕様に従うことは，つぎの2点から確認できる．

●各プロダクションの左辺は，それより先に記述されているプロダクションの左辺が一つでも真のときは偽となる．

●競合解消戦略により，同一のプロダクションのインスタンシエイションは，左辺がより先に到着したメッセージに照合するものから順に実行される．

スケジューラのヒット戦略が 'exhaustive' に設定されているのは，スケジューラが一度に受理し得るメッセージの数をできるだけ多くして，哲学者が必要以上に食事を待たされないようにするためである．

〔タスク型 philosopher〕

哲学者は，イニシャル節においてその初期状態を設定した後，プロダクション d1〜d3 のインスタンシエイションを繰り返し実行する．

このタスクはつぎの WM 要素によりその状態（食事待ち中，食事中および思索中）を表現している．

(self ^activity 〈状態名〉)

各プロダクションの意味はつぎのとおりである．

d1:「食事待ち状態に移ったなら，要求メッセージをスケジューラに同期型送信し，その受理を待つ．受理されたら，食事状態に移る．ただし，許容時間を越えて待たされたなら，催促メッセージを非同期型送信する」

d2:「食事状態に移ったなら，食事時間だけ食事した後，解放メッセージをスケジューラに非同期型送信し，思索状態に移る」

d3:「思索状態に移ったなら，

思索時間だけ思索した後，食事待ち状態に移る」

図6の定義中，'*' で始まる名前はパラメータの値(例えば '*no' は '^no' の値) であり，': parent', ': mod' および '+' は，それぞれ，親タスク，剰余値および和を返す関数である．

4.3 システムの実行効率

例題の実行には Po-PS の処理系の核部分としてプログラム領域 60 kB (キロバイト)，データ領域 95 kB を要するほか，タスク型およびタスクの内部表現としてそれぞれ 5.8, 3.2 kB のデータ領域を要し，各タスクの1サイクルの実行には平均 0.17 秒を要す (使用計算機の処理速度は 0.7 MIPS)．

なお，Po-PS のインプリメンテーションについてはプロダクションの内部表現およびパターン照合アルゴリズムにおいて改善すべき点が残されている．

5. 考　　察

プログラミング言語や方法論の観点から Po-PS について考察する．

1) 対象指向言語との類似性：Po-PS は対象指向言語と多くの共通点をもつ．Po-PS および対象指向言語における諸概念の対応関係は**表1**のように要約される．

Po-PS と逐次型処理を基本とする従来の対象指向言語との違いは，対象の挙動の記述法および対象の性質の継承法にある．すなわち，①タスクは，その挙動がプロダクションの集合で記述され，非決定的に送信されるメッセージに柔軟に反応できること，②タスク型間での継承が静的に行われることである．

2) 並列処理言語との関係：Po-PS は，occam[18] や Ada[19] などの並列処理言語のモデルである CSP をその出発点としている．システムをメッセージを介して通信し合うタスクの集まりとして表現するという点ではどちらも同じであり，Po-PS におけるメッセージの受理/同期型送信は，CSP における入力/出力に相当する．拡張 PS の PM およびプロダクションは，それぞれ CSP における繰り返し命令 (repetitive command) およびガード付き命令 (guarded command) に対応付けられる．

表1 対象指向言語および Po-PS における諸概念の対応
Table 1 Concepts in object oriented languages and their counterparts in Po-PS.

対象指向言語	Po-PS
上位クラス	上位タスク型
クラス	タスク型
インスタンス	タスク
インスタンス変数	ワーキングメモリおよびパラメータ
メッセージ	メッセージ
メソッドの動的継承	カテゴリの静的継承

3) 他の分散型 PS モデルとの関係：Po-PS の特徴は PS 間の通信/同期が陽に記述される点にある．YES/MVS や Co-PSs では通信だけが陽に記述され，VM, APN, Age では通信/同期は共有メモリを介して暗に行われる．

Po-PS は，種々の同期問題[5]，雑誌編集過程のモデル化[3]，サービス業務のスケジューリング[4]，線画のラベル付け[4] などに適用されている．これらの使用経験および Po-PS と同様の目的で開発されたシステムとの比較を交えながら，事象駆動型システムの記述言語としての Po-PS の能力および限界について考察する．

1) 事象駆動型システムのモデル化：OPS5 のような均一的 PS よりはずっと容易である．均一的 PS では対象の挙動だけでなく，対象間の同時進行性や相互作用もプロダクションで表現する必要があるからである．また，APN や Co-PSs で扱われている問題をより自然にモデル化できる．もっと複雑な競合解消や協調が要求される問題にも適用できよう．

2) 事象駆動型システムの管理と制御：現在の Po-PS は，実行速度上の制約から，高度の実時間処理の記述には不向きであり，適用領域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管理や制御，例えば APN で記述された雑誌編集管理システム[8] のようなものに限られる．処理系の高速化には，Rete[12] のような高速照合アルゴリズムの導入およびタスクのスケジューリング方法の改善などが必要である．

3) 分散型問題解決システムの記述：Po-PS では，黒板モデル[14]や Smith による協調機構[21] などを容易に記述できる．しかし，分散型問題解決のための機構やパラダイムを陽には提供していないから，使用者は，問題ごとにその解決過程をモデル化する必要がある．

6. む　す　び

事象駆動型システムの表現と制御を目的とした分散型プロダクションシステムの記述言語 Po-PS の概要を示した．

Po-PS では，事象駆動型システムは，非同期，同時

進行的な逐次型タスクの集まりとして認識することにより，自律的なサブシステムの疎結合としてモデル化される．各タスクは，通信/同期機構が付加された拡張プロダクションシステムとしてモデル化されるために，対象の事象駆動的な挙動が自然に表現される．

タスクはその原型を表すタスク型により記述され，タスク型の複製として生成される．タスク型およびタスクは対象指向言語におけるクラスおよびインスタンスに相当する概念であり，タスク型間での定義記述の静的継承機構も実現されている．

いくつかの例題的な事象駆動型システムへの応用経験から Po-PS の有用性が確認されているが，今後は，実行速度の向上，分散型問題解決機構の導入およびプログラミング環境の構築などの課題を段階的に解決し，Po-PS の応用領域を拡大していく予定である．

謝辞 日頃，議論やプログラミング上の相談に応じていただく東京工業大学大学院総合理工学研究科システム科学専攻畝見達夫氏に感謝する．

参考文献

1) 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，pp. 1025-1026 (1984).
2) 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，34-6 (1984).
3) 小野典彦，小林重信：手続き指向型プロダクションシステム Po-PS とその応用，情報処理学会第 36 回知識工学と人工知能研究会資料，39-9 (1984).
4) 小野典彦，小林重信：分散協調型プロダクションシステム Po-PS とその応用，計測自動制御学会第 3 回知識工学シンポジウム資料，pp. 63-68 (1985).
5) 小野典彦，小林重信：Po-PS 言語仕様書第 1 版，Systems Science Research Report, No. 13, 東京工業大学大学院総合理工学研究科システム科学専攻 (1985).
6) 畝見達夫：Lisp 1.9 on MELCOM COSMO User's Manual, 東京工業大学大学院総合理工学研究科システム科学専攻 (1982).
7) Zisman, M. D.: *Use of Production Systems for Modeling Asynchronous, Concurrent Processes, Pattern-Directed Inference Systems*, pp. 53-68, Academic Press, New York (1978).
8) Zisman, M. D.: Representation, Specification and Automation of Office Procedures, Ph. D. Dissertation, Working Paper 77-09-04, Univ. of Pennsylvania (1977).
9) Fagan, L. M. et al.: Representation of Dynamic Clinical Knowledge: Measurement Interpretation in the Intensive Care Unit, *IJCAI-79*, pp. 260-262 (1979).
10) Griesmer, J. H. et al.: YES/MVS: A Continuous Real Time Expert System, *AAAI-84*, pp. 130-136 (1984).
11) Forgy, C. L.: OPS5 User's Manual, Technical Report CS-79-132, Dept. of Computer Science, Carnegie Mellon University (1979).
12) Forgy, C. L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artif. Intell.*, Vol. 19, pp. 17-37 (1982).
13) Erman, L. D. et al.: The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty, *ACM Comput. Surv.*, Vol. 12, No. 2, pp. 213-254 (1980).
14) Nii, H. P. et al.: Age Reference Manual, Computer Science Department, Stanford University (1981).
15) Goldberg, A. J. et al.: *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, Reading (1983).
16) Cannon, H.: Flavor—A Non-Hierarchical Approach to Object-Oriented Programming, MIT AI memo (1980).
17) Bobrow, D. and Stefik, M.: The LOOPS Manual (Preliminary Version), Xerox Corp. (1983).
18) INMOS Limited: *occam Programming Manual*, Prentice-Hall International, London (1984).
19) United States Department of Defense: Reference Manual for the Ada Programming Language (1980).
20) Hoare, C. A. R.: Communicating Sequential Processes, *Comm. ACM*, Vol. 21, No. 8, pp. 666-677 (1978).
21) Smith, R. G.: Frameworks for Cooperation in Distributed Problem Solving, *IEEE Trans. Syst. Man Cybern.*, Vol. SMC-11, pp. 61-70 (1981).

付録 タスク間の相互作用の記法

タスク間の通信/同期ならびにタスクの生成/終了に関する主たる文を例を用いて示す．

A. 通信および同期のための文

1) send 文：メッセージを非同期型送信する．例は，メッセージ '(release ^no 3)' をタスク 'scheduler' に非同期型送信するためのものである．

〔例〕 (send scheduler release ^no 3)

2) request 文：メッセージを同期型送信する．こ

の文では，送信後，一定時間を越えてその受理が待たされた場合に実行すべき文の列を指定できる．つぎの文はタスク 'scheduler' にメッセージ '(use ^no 3)' を同期型送信した後，'10' 単位時間を越えて受理が待たされたなら，タスク 'scheduler' に対してメッセージ '(remind)' を非同期型送信することを指定する．

〔例〕 (request scheduler use ^no 3
: within 10
: else (send scheduler remind))

3) channel 文および connect/disconnect 文：チャネルの設定およびチャネルに対する接続/接続解除を行う．チャネル名を 'IPSJ' とした場合の例を示す．

〔例〕 (channel IPSJ)
(connect IPSJ)
(disconnect IPSJ)

4) announce 文：チャネルを介してメッセージをブロードキャストする．チャネル名を 'IPSJ'，メッセージを '(CallforPapers ^theme AI)' とした例を示す．

〔例〕 (announce IPSJ CallforPapers ^theme AI)

5) delay 文：指定された時間だけ実行を中断する．

〔例〕 (delay 21)

B．タスクの生成および終了のための文

1) create 文：タスクを生成する．生成されるタスクの名前を値とする関数としても使用できる．つぎの文はタスク型 'philosopher' のタスク 'ph-1' をパラメータ '^no' の値を '1' として生成する．

〔例〕 (create ph-1 philosopher ^no 1)

2) terminate 文：自分自身を終了させる．

〔例〕 (terminate)

3) kill 文：指定されたタスクを消去する．

〔例〕 (kill backtracking)

(昭和 60 年 5 月 16 日受付)
(昭和 61 年 3 月 20 日採録)

小野 典彦（正会員）

1954 年生．1979 年東京工業大学理学部情報科学科卒業．1981 年同大学院総合理工学研究科システム科学専攻修士課程修了．現在，同博士課程在学中．プログラミング言語，分散型問題解決などに興味をもつ．ソフトウェア科学会会員．

小林 重信（正会員）

昭和 20 年 6 月 19 日生．昭和 49 年東京工業大学大学院経営工学専攻博士課程修了．工学博士．同年東京工業大学工学部制御工学科助手．昭和 56 年東京工業大学大学院総合理工学研究科システム科学専攻助教授，現在に至る．知識処理，システム制御，問題解決などの研究に従事．計測自動制御学会，ソフトウェア科学会などの会員．