ソフトウェア工学 73－2
（1990．7．19）

意味モデルによる並行システムの表現形式

間野 暢興
電子技術総合研究所

実時間システムやオペレーテングシステムなどの並行システムの、要求、設計、仕様、プログラム、データ、知識などほとんど全ての情報を、対象物－関係指向の意味モデルによりモデル化して表現する形式を提案する．並行プロセス間の全ての交信は、それらの間の入出力事象としてとらえられる．各プロセスの各手続きについてそのデータ列を単一の入力木および出力木表現に統合した後で、それらの両方に対応する形のプログラムモデルを組み立てる．この二次元グラフ表現のモデルを用いることにより、抽象構文木、状態遷移図、およびプロセス交信図を統合した表現が可能となる．例題として単一ソースソート・エコーアルゴリズムおよびエレベータ問題を取り上げる．

REPRESENTATION FORM OF CONCURRENT SYSTEMS BY THE SEMANTIC MODEL

Nobuoki Mano
ElectroTechnical Laboratory

1-1-4 Umezono, Tsukuba-city, Ibaraki-ken 305, Japan

I propose the representation formalism of S-model (Semantic model) for representing almost all information --- requirements, design, specifications, program, data and knowledge --- of concurrent systems such as real-time systems and operating systems. S-model is represented with object-relationship formalism. Each of the communication between concurrent processes is taken as the input-output data-sequence between them. After unifying these into a single input-tree and a single output-tree representation for each procedures of each process, we construct a graph-like program model of the procedure corresponding with both of them. The model is the unified form of abstract syntax, state-transition-diagram, and communication-diagram between processes. The single-source-sort-echo-algorithm for process networks and the elevator problem are used as the example problems.

1．はじめに

筆者は、モデルを用いてファイル処理およびデータ構造処理プログラムの生成を行なうために、要求、仕様、設計、プログラム、データ、知識、のほとんど全ての情報を対象物－関係の一様な表現形式により表現する意味モデル、および意味モデルを操作することによりプログラム生成を行なう意味モデル操作システムの研究を行なってきた[1][2][3]．

この意味モデルを、実時間システム[4]、オペレーティングシステム、分散システム[5]などの並行システムの記述ができるように拡張し、プログラムの自動生成を含むソフトウェアの知的開発環境を作り出すことを目標とする．本論文では、そのための、並行システムのモデル化と仕様－プログラムの表現形式を提案する．この並行事象のモデル化には、ジャクソン・システム開発法（ＪＳＤ）[6]などの従来からある並行システムの設計法やＡＤＡなど各種プログラミング言語を参考にした．しかし、並行システムに対する意味モデルの表現形式は、これらの特定のものに依存してはおらず、従来の意味モデルの持つ機能を素直に延長する形式で表わせることが分かってきた．

本論文の内容は次ぎの通りである．第２章では、これまでの意味モデルの概要を紹介する．第３章では、並行処理のために意味モデルの拡張される部分、すなわち、知識構造、プロセス間通信の統一的表現、抽象構文木、状態遷移、および交信図を統合したプログラム構造、などについて述べる．第４章と第５章では意味モデルの記述応用例を用いて、その仕様表現からのプログラム表現導出の過程を示すことにより、意味モデルの表現形式の、ソフトウエア開発自動化への有効性を示す．第４章ではエコーアルゴリズム[7]の一つであるsingle-source-sortアルゴリズム、第５章ではエレベータ問題[5][8][9]、を例題として用いる．

2．　従来の意味モデルの概要[1][2][3]

意味モデルは、述語論理、集合論、抽象構文、知識構造、抽象データ型、フロー構造などの記法を、対象物（部分空間を含む）－関係の、部分空間に分割された意味ネット的表現に統一したものである．対象物と関係には、それぞれクラスとインスタンスがあり、実際のデータやプログラムは、対象物と関係のインスタンスの集まりにより記述される．なお、クラス名は英小文字のラベルで、インスタンス名は英大文字の固有名で表わされる．

意味モデル操作システムには、対象物と関係のクラスの上位下位階層からなる（上位概念のスロットやメソッドを下位概念が継承する機能を持つ）知識構造があり、これらのクラスを定義する操作群が意味モデル管理ルーチン群となっている．これらの意味モデル管理ルーチン群の上には、それらを駆使した、直接実行機、記号実行機、書き換え規則機、問題解決機、などのツール群が存在する．上記の知識構造には、その一部としてデータ型に関する知識構造が含まれている．

対象物と関係の知識構造は、次に示すような対象物のクラスを含んでいる．

原子対象物－－－論理演算子、データ集合、
　　場所集合、算譜construct、（変数などの）
　　識別子．
構造体対象物－－データ型、操作、規則、など．
部分空間－－－－整式、状態、など．

構造体対象物は、次に述べるようなスロットを持つ．

抽象データ型－－操作群、データ型不変量、公理、
　　その他．
操作（関数あるいは手続き）－－－事前条件、事後
　　条件、入力部分、プログラム部分、出力部分．
規則－－－－－－ＬＨＳ、ＲＨＳ．

よく用いられる関係のラベルの定義と意味を以下に記す．

データ集合どうしを結ぶ関係：
　'p','p' --- 部分を指す関係、
　'o','o' --- 選択、
　'memb' ---- 繰り返し（集合）、
　'e' ------ 繰り返し（列）、
　'int-e' --- 入り交じり．

算譜構成要素（算譜construct と操作を合わせたもの）どうしを結ぶ関係：
　'1','2' --- 文リスト型の算譜構成要素からその成分の文を指す関係、
　'body' ---- 繰り返し型の算譜構成要素からその本体の算譜構成要素を指す関係、
　'then','else' --- 判定文型の算譜構成要素から条件の真偽に応じて制御が移行する算譜構成要素を指す関係、
　'bf','af' ----- ある算譜構成要素からそれぞれその前始末および後始末の算譜構成要素を指す関係．

仕様表現は、データ集合どうしを結ぶ関係（クラスあるいはインスタンス）の属性として持たせた対応情報により記述される．対応情報は、矢印の方向に従って見た一方のデータ集合の一つの要素に対応する他方の集合の要素数の最小値と最大値（ただし、'∞'は不定あるいは無限を表わす）のペアを、関係の正方向と逆方向の各々について表わしたものである．この対応情報を用いることにより、１対１、多対１、の上への写像、の中への写像、全域関数、部分関数、集合の基数、超集合、等価関係、(c)鍵、関数従属、などが表

現できる．

データおよびプログラムの構造は抽象構文木により表現される．操作の入力部分、プログラム部分、出力部分の主要部は、それぞれ入力データ木、プログラム木、出力データ木により表現され、入力データ木とプログラム木に関しては関係ipc 、入力データ木と出力データ木に関しては関係ioc でそれぞれの対応する要素どうしが結ばれている．さらに、プログラム抽象構文木中の各算譜構成要素の前後の状態記述を併用できる表現形式を取る．状態の内容は、その状態部分空間内の、変数識別子とデータ集合（の構造）とを結ぶ関係rng (range) により示される．

意味モデル（操作システム）の特徴として、対象物－関係の構文論的には単純な表現形式、構造表現機能、柔軟な可塑性、マルチパラダイム言語（構造を辿る手続き型言語と、規則や操作の事前条件事後条件とのパターン整合問題解決言語）、コンテキスト機構、分割部分空間による閉領域仮説の利用、連想データ構造に基づく逆引き機能などが挙げられる．

意味モデルは、ファイル処理におけるジャクソン法の自動化、集合・関係データ構造処理プログラムの生成、抽象プログラムの配列・ポインタを用いる物理プログラムへの実現変換などを目的に研究されて来た．また、意味モデル管理ルーチンや各種ツールのプログラムも意味モデルを用いて記述される．

3． 意味モデルによる並行システム記述の概要

3．1 知識構造

従来の意味モデル操作システムの知識構造を拡張する．その一部を図１に示す．

対象物
- 原子対象物
  - データ列
  - 並行プログラム用construct
- 構造体対象物
  - 操作
    - 事象
    - プロセス手続き
      - 不可分手続き
    - プロセス
      - 外部プロセス
        - センサ、アクチュエータ
      - 内部プロセス
        - リソース
        - 計算プロセス
- 部分空間対象物
  - 状態

図１ 上位下位構造

3．1．1 プロセス集合

プロセスは次のように抽象データ型と類似した構造を持つ．

内部プロセス、
データ型インスタンス、
プロセス手続き群、
内部手続き群、
内部変数、輸出変数、
プログラム本体．

3．1．2 データ列の分類

プロセス集合間の交信が１対１、１対多、多対１、多対多のどれであるかは、プロセス集合間の関係の対応情報、あるいは、プロセス集合の基数により示される．交信には、非同期と同期とがある（図２参照）．非同期送信の場合、送信プロセス集合からデータ列を関係in、受信プロセス集合から同じデータ列を関係out で結ぶことにより送信の方向性を示す．同期交信の場合、送信主体プロセス集合とデータ列を関係io、受信主体プロセス集合と同じデータ列を関係oiで結ぶ．この場合データ列は送信について記述し、返信についてはデータ記述が必要な場合には別にデータ集合対象物をもうけて送信データ対象物から関係rvで結ぶことにより表わす．

データ列には、次のような種類がある．

純データ列(x-seq) 、
メッセージ列－－ (op X:x,---)
輸出入操作およびそれらのパラメータを含む、
状態ベクトル列、 (Y:y,---)
同期信号列．

これらのどれであるかは個々のデータ列の構造の定義により示される．

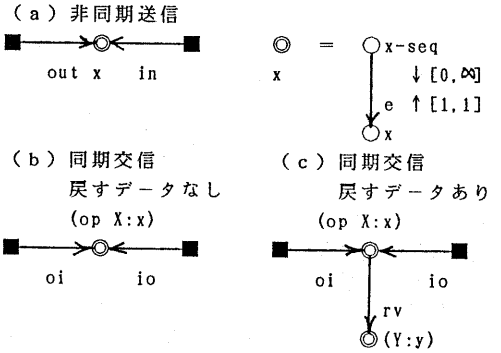


図２ プロセス間交信

3.1.3　並行用プログラム用construct

loop－－－無限繰り返し文

exit－－－loopからの脱出文

when－－－ガーデッド節の条件部分

select－－非決定性選択文

3.1.4　事象

事象には事前条件と事後条件がある、通信事象には、同期送信callとその終了callend 、同期受信acceptとその終了acceptend 、非同期送信send、非同期受信receive などがある．事象インスタンスは、それと関係ins および関係outsでそれぞれ結ばれる直前および直後の状態を持つ．

3.1.5　プロセス手続き

プロセス手続は、事前条件、事後条件、入力部分、プログラム部分、出力部分の他に、プロセスネット部分図、および、プロセス交信部分図を持つ．

通常、プロセスのエントリはデータ型queue である．ゆえに、エントリ名をデータ型に関する知識構造における抽象データ型queue の下位概念とすれば、クラスqueue の操作dequeue 、enqueue およびisinqueue を継承して使用することができる．acceptはエントリキューについてこれらの操作を内部で用いた不可分手続きである．

3.1.6　事象と時区間（図３）

時区間は、それと関係fromおよび関係toで結ばれる二つの事象間の時間的区間をいう．これは、遅延時間の設定や、二つの事象列の占める時間区間の相互関係を論じる場合に用いる．後者の場合、プロセス集合からプロセス集合へのデータ列の送信開始事象から送信終了事象までの時間幅を表わす．時区間どうしは、overlap,disjoint, next-to ,includeなどの関係がある．

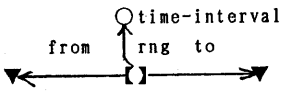


図３　時区間　　（【】：時区間、▼：事象）

３．２　仕様の表現形式

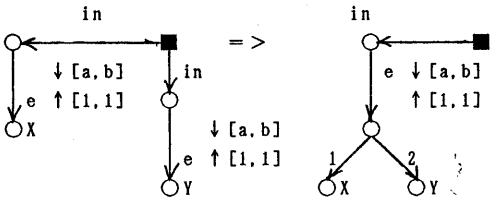
3.2.1　問題処理構造

問題処理構造は、外部プロセスと内部プロセスのプロセス集合のインスタンスの集まりを表わすプロセスネット（全体／部分）図、およびそれらの間の非同期／同期指定のデータ列の集まりを表わすプロセス交信（全体／部分）図により示される．並行システムの設計において、プロセス間インターフェースの仕様記述は、システムをそれを構成するプロセスに分断して個々のプロセスを定義できるようにするために、非常に重要な役割を演じる．

3.2.2　データ列の併合

あるプロセス集合に対して、複数のデータ列インターフェースがある場合、それらデータ列どうしおよびそれら構成要素どうしの対応と（時間的）前後関係が問題となる．そこで、それらの時区間がoverlap の関係にあるとき、そのプロセス集合の入力データ列および出力データ列をそれぞれ図４に示すように併合を計る．（ａ）は固定併合であり、（ｂ）は粗併合で複数のデータ列の入り交じりを示す関係int-e と関係 oを用いている．

（ａ）　固定併合



（ｂ）　粗併合

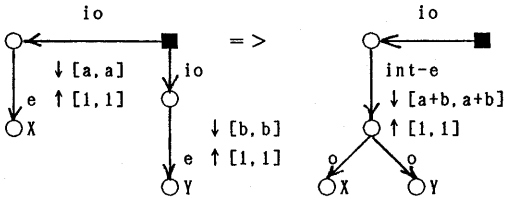


図４　プロセスの入出力データ列とその併合

（■：プロセス集合）

3.2.3　リソースとプロセス間のacquire とrelease

一つのリソースプロセスに対し関連するプロセスが複数個の場合に、状態検知関数と取得操作とは一体とした不可分手続きでなければならない．

制約条件は、全ての状態におけるプロセス間のアクセス関係についての論理式で表現される．

3.2.4　複合データ構造

複数のデータ構造が、共通するデータを含むときには、データ構造の内容の更新に拘らず、それらの内容が常に同一であるように保たなければならない．これは（分散）データベースにおける意味保全性の拘束条件と同じ性質の事柄である．このためには、複数データ構造に関する、登録、除去あるいは更新の同時操作において、コミットアクションの機構が必要となる[10].

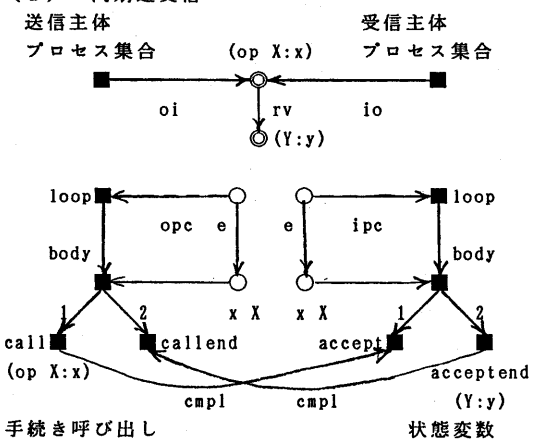
3.2.5　時間に関連する記述

外部の物体の移動を待つ必要があるような場合、ふたつの事象間の時区間にそのことに関する仕様を記述する．決まった時間の遅延をもたらす操作delay は時計プロセスとの交信として実現される．時計プロセスからの時間標識データ列を用いることも多い．

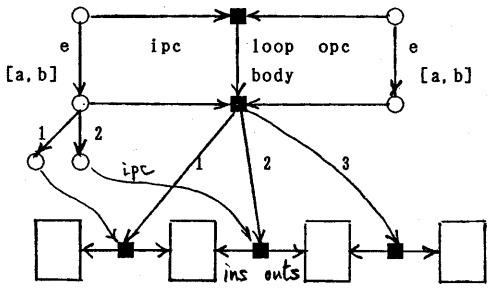
3．3　プログラムの表現形式

意味モデルにおけるプログラムモデルの表現は、

（a）　同期送受信



（b）　固定併合とその処理プログラム



（c）粗併合とその非決定性処理プログラム

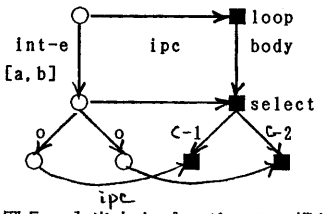


図5　入出力とプログラムの構造の対応

（■：算譜構成要素）

抽象構文木、状態遷移図、およびプロセス交信における相補的な事象どうしの結合図、を融合したグラフ表現である．図4に示したプロセスの併合された入出力データ列表現を基に、図5に示す入出力データ列表現とプログラムモデルとの対応を用いて、プログラムモデルの構造を作り出す．

抽象構文木は、レベルの深さの方向に、連接、判定、繰り返し、並行用construct 、および操作呼び出しの算譜構成要素を結合した表現形式である．繰り返しの集合の基数とループの中の事象の生起回数とが1対1であることを基礎としている．データ列の対応情報↓[a,b] が [0,∞] の場合 exit 文が存在する．繰り返しを表わす入力木の葉の対象物がプログラム繰り返し構文本体内の状態においてアノニマスな対象物として用いられる．

状態遷移図は、木の横（水平）方向の各レベルにおいて事象列および手続き列の生起に伴う状態遷移を表わしたものである．ループ本体部分のプログラムは、その内部にゴールと考えられる状態述語を持つことが多い．そのゴールを満たす事象列の後は、再びループ不変量が満たされるように残りの事象列が決められる．これらの部分事象列は、事象手続きの事前事後条件や上記のアノニマス対象物に関する述語を用いたゴール指向的な問題解決の成功の軌跡から導出される．

プロセス交信における相補的な事象どうしの結合は、送信から受信へ、および受信終了から送信終了へと、関係cmplで結ぶことにより示される．

4．　Single-source-sortエコー・アルゴリズム[7]

意味モデルにおける入力木とプログラム木の要素どうしの対応付けは、木をpreorderの順に辿る処理アルゴリズムに対して有効である．並行プログラムにおいては、エコーアルゴリズムや、8-queen 問題の状態空間探索アルゴリズム[5] などの、縦型並列アルゴリズムがこれに属する．

下に示すように、入力木の節の型に何を選ぶかは、問題分野により異なる．しかし、知識構造の継承およびコンテキスト機構を利用することにより、柔軟なあてはめと運用とが可能となる．

ファイル処理やデータ構造操作プログラム
　－－－　データ集合、
直接実行機や記号実行機
　－－－　算譜構成要素（をデータ集合と解釈）、
後戻り形状態空間探索機
　－－－　状態集合（をデータ集合と解釈）、
エコーアルゴリズム
　－－－　プロセス集合（をデータ集合と解釈）．

エコーアルゴリズムとは、あるプロセスからその場

における残りの全プロセス集合にexplorerを放送して、そのechoとして戻って来るデータ（プロセス- idなど）をパス上のプロセスで処理することにより、場のネットワークに関する情報を得るアルゴリズムを指す．Single-source-sortは、１回の往復で全プロセス-idのソートされたものが返されるアルゴリズムである．

図６において、（ａ）の関係'cmd' は、グラフとそれを構成する全ノード集合との関係を表わしている．その関係の再帰的定義を与えるものが（ｂ）である．これは、traversal execution graph (teg) （あるいはedge-spanning-treeの持つノードを表わすグラフ）の再帰表現を示している．○記号のノードをプロセス集合を表わすものととると、図の左半分はプロセスteg の再帰表現を表わしている．関係P1は単一のプロセスを表わし、関係P2より下の部分はそのプロセスとチャンネルで結ばれたプロセス集合から親プロセスだけを除いたプロセス集合（およびその子孫）を表わしていると解釈できる．この構造表現から、交信に必要なデータ構造とその操作プログラムの導入がなされる．プロセスteg の再帰表現に対応した右側の図は、そのプロセスの（■記号のノードが算譜構成要素を表わす）プログラム木である．再帰表現されたプロセスteg 上

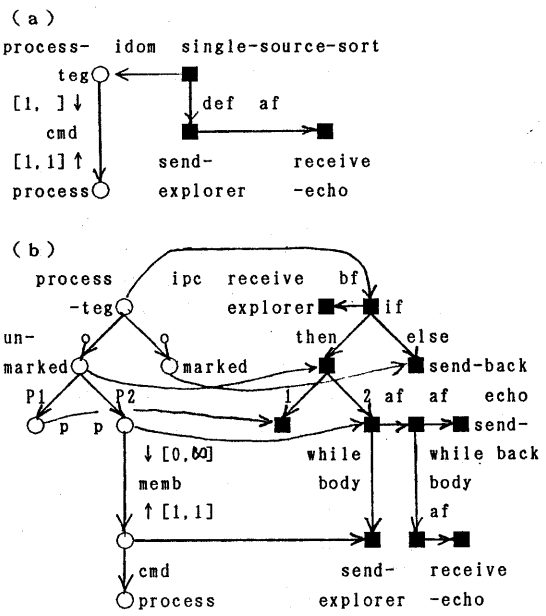


図６　Single-source-sortエコー・アルゴリズムにおけるグラフのtraversal execution graph とプログラム木の対応（○：プロセス、　■：算譜構成要素）

を、根－＞葉の方向にはexplorerが流れ、辿る際に出会うプロセスがマークされていない場合はさらにexplorerが非同期送信され、マーク済みのときコマンドの伝播は終了しechoが親プロセスに戻される．葉－＞根の方向には、子孫のプロセス集合から戻されたソートされたプロセス-id 列の集合と自身のプロセス-id とを併せてソートしたデータが、echoとして親プロセスに非同期的に戻される．

５．　エレベータ問題[5]

ここでは、トップダウン・アプローチにより、エレベータ問題の設計を次の手順に従って行なう．なお、この問題の要求仕様は[5] を基にしており、[8][9]とは少し異なっている．

５．１　問題の仕様

建物の各フロアにそれを押すことにより上昇と下降の要求をエレベータ・コントローラに伝えるボタンがある．また、各階の標識を知らせるセンサがある．各エレベータには降りる希望の階を指すリフトボタンがある．各エレベータにはアクチュエータとして、扉、およびモータが付属している．これらのプロセス集合どうしの関係を示すプロセスネット全体図はここでは省略する．これらのプロセス集合の基数は次の通りである（ｅとｈは、エレベータの個数と建物の階数）．

```
  # (lift-process) = # (door)
  = # (motor ) = e
  # (up-floor-button-process )
  = # (up-floor-button )
  = # (down-floor-button-process )
  = # (down-floor-button ) = h
  # (lift-button )
  = # (floor-mark-reader ) = e * h
lift-process－door、motor : →[1,1] ←[1,1] .
lift-process－lift-button 、floor-mark-reader :
                          →[h,h] ←[h,h] .
lift-process－up-floor-button-process 、
            down-floor-button-process :
                          →[h,h] ←[e,e] .
```

motor へはstopおよびdir (up/down) 命令があり、stopのときは完全に停止してから戻り、up/down は直ちに戻る．doorへはopenおよび close命令がある．floor-mark-reader は、エレベータがフロアに近づくたびにfloor-markを知らせる．

エレベータの状態は、変数current-floor, current-directionおよび配列goingto に保持される．floor-buttonの要求にエントリcomingを呼び出してての行くことの約束、およびlift-button の要求の統合したも

のを、配列going-toに作る．

図７にプロセス間交信全体図を示す．

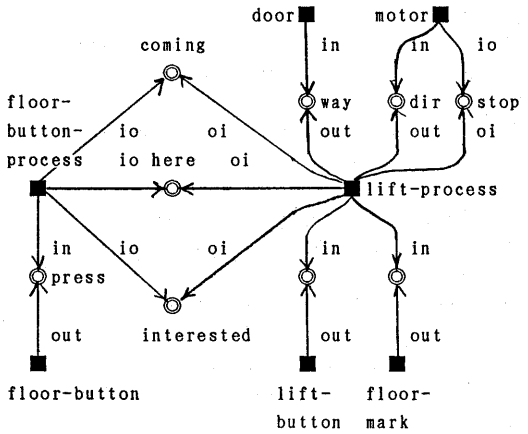
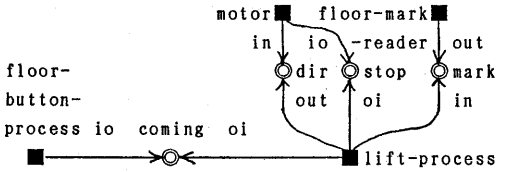


図７　エレベータ問題におけるプロセス間交信全体図

（プロセス集合：■、データ列：◎）

（ａ）　プロセス間交信部分図



（ｂ）入力部分－プログラム部分－出力部分

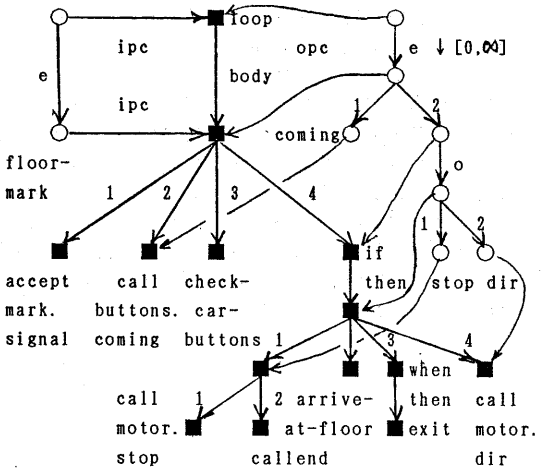


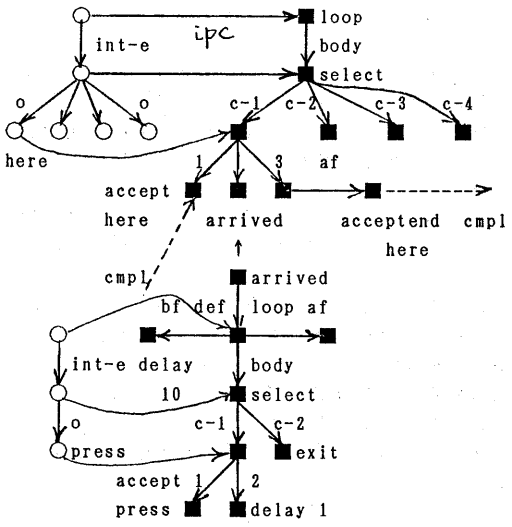
図８　エレベータ問題におけるリフトプロセス内の手続きmove-til-stop

５．２　各プロセスのモジュール設計

lift-process内の手続きmove-til-stop

図８（ａ）は図７からlift-processの手続きmove-til-stopに関する部分だけ取り出したプロセス間交信部分図、（ｂ）は手続き本体の、入力部分、プログラム部分、出力部分の各々およびそれらの対応を示したものである．エレベータは双方向に移動するが、現在の移動の方向についてその現在の階より先にある階での停止要求がある限りその方向への移動を続ける．このことはフロアに関する繰り返しの形の入力木とそれに対する繰り返しのプログラム木となる．ボタンが押されている階、すなわちエレベータの止まる階（これがゴールと考えることができるが）は、全ての階の部分集合となるので、判定文により場合分けして検討する必要が出て来る．

（ａ）　入力部分とプログラム部分



（ｂ）　時系列に関する表現と修飾

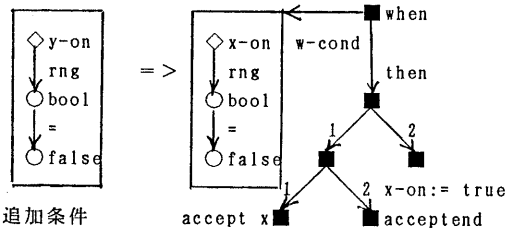


図９　エレベータ問題のフロアボタン・プロセス

floor-button-process

図９（ａ）はfloor-button-processのプログラムモデルを示す．floor-button-processで、accept hereが行なわれた状態において、手続きarrived はfloor-buttonのエントリpress の待ち行列を空にして、ボタンが押されてから１秒間エレベータが出るのを遅らせる．（ｂ）は、「最初のｘだけを受け付ける」という仕様をプログラムモデル化したもので、これにいろいろな条件を付加することで、優先順位を指定したプログラムを作り出すことができる．

６．　おわりに

対象物－関係からなる意味モデルによる並行システムの仕様とプログラムの表現形式を提案した．並行プロセス間のすべての交信をそれらの間のデータ列の入出力事象として捉え、各プロセスの各手続きについてそのデータ列を単一の入出力木表現に統合し、それに対応する形でプログラムのモデルを表現する．この二次元グラフ表現のモデルを用いることにより抽象構文木、状態遷移図、さらにプロセス交信における相補的な事象どうしの結合図、を融合した表現が可能となる．また同時にループ不変量と事象の事前事後条件およびゴール指向問題解決を結びつけることが可能となる．例題として、縦型並列のアルゴリズムとしてsingle-source-sort エコー・アルゴリズム、および実時間システム・プログラムとしてエレベータ問題への適用を示した．

今後の課題としては、上記の線に沿った問題解決による並行プログラム合成法の研究、eventuality などについての検証法の研究、シミュレーションを行なう方式の研究、半自動的にプロセスのモジュール化を行なう方法の研究などがある．

最後に、文献[8] を教えて頂いた大蒋和仁氏に感謝致します．

参考文献

[1] Mano, N.:"Semantic Model of File-processing Software and its Application to Automate Program Generation", Proc. of 11th Annual International Computer Software and Applications Conference (COMPSAC '87), pp.195- 204 (1987).

[2] Mano, N.:"Modeling of Data-processing Software for Generating and Reusingtheir Programs", Proc. of 10th International Conference on Software Engineering, pp.231-240 (1988).

[3] 間野暢興：”対象物－関係指向知的プログラミング環境における自己記述からのシステム生成”、情報処理学会ソフトウェア基礎論研究会資料SF34-1 (1990).

[4] Bustard,D., Elder,J., and Welsh,J.: "Concurrent Program Structures", Prentice Hall (1988).

[5] Filman,R.E., and Friedman,D. : "Coordinated Computing: Tools and Techniques for Distributed Software", McGraw-Hill Inc. (1984) ［雨宮真人、尾内理紀夫、高橋直久共訳：”協調型計算システム：分散型ソフトウェアの技法と道具立て”，マグロウヒル (1986)］.

[6] 有沢誠、山崎利治：”マイケルジャクソンのシステム開発法”、bit , Vol.22, No.1-No.6 (1990).

[7] Chang, E.J.H.: "Echo Algorithms: Depth Parallel Operations on General Graphs, IEEE Trans. on Soft. Eng. Vol.SE-8, NO.4, pp.391-401 (1982).

[8] Proc. of the Fourth International Workshop on Software Specification and Design, IEEE Computer Society (1987).

[9] 大蒋和仁、二木厚吉：”図書舘の問題とエレベータの問題のＬＯＴＯＳによる仕様記述”、情報処理学会ソフトウェア工学研究会資料64-12 (1989).

[10] Liskov,B.: "The Argus Langauge and System", in "Distributed Systems: Methods and Tools for Specification", Alford,M.W., and et al, Lecture Notes in Computer Science, Vol.190, Springer-Verlag (1985).