論　文

シミュレーションプログラムの構造化設計法*

金　田　悠紀夫**

Abstract

The new design technique of big discrete event type simulation programs is is proposed. We introduce a new world view——we call it "actor"——and make it easy to build well structured simulation programs. We regard a complex system is organized by a set of actors which play active part in the system respectively.

By associating each actor to one procedure, the structure of the simulation program can be directly reflected by the structure of the simulated system.

The difficulties of making coding, verification, debugging and modification of simulation programs can be extensively reduced.

1. はじめに

コンピュータを用いたシステムシミュレーション技法は，様々なシミュレーション言語やシミュレーションパッケージの開発により発達をとげてきた．特に離散事象（イベント）形のシミュレーション技法は，GPSS, SIMSCRIPT, GASP, SIMULA, SIMPL などの言語の開発により広く用いられるようになってきている．

しかし，われわれがシミュレーションの対象として取り扱うシステムは複雑で巨大化したものが多くなってきている．したがってシミュレーションプログラムも細部にわたる情報まで必要な場合は必然的に巨大化しプログラムの作成，デバッグが困難となり，そのことがシミュレーション技法のより一層の普及に対するネックとなっている．

このプログラムの巨大化によって生じる問題はシミュレーションプログラムに限られた問題ではなく，ソフトウェア技術の根幹にかかわる問題として注目され，大規模プログラム作成にともなう困難さを打開するための有力な手段として，明解な構造を持ったプログラムの作成を指向する構造化プログラミング技法というものが最近特に注目されるようになってきている．

システムシミュレーションを行う場合，対象としているシステムはある構造を持っているのが普通であり，しかも時間の経過にともなってダイナミックな動きをする．したがってこのようなシステムを対象としたシミュレーションプログラムはプログラム作成の面からも，またシミュレーションモデルの正しさの確認という点からしてもできるかぎり対象のシステムの持つ構造を explicit に反映していることが望ましい．

しかるに，前述した GASP などのイベントルーチンを基本としたシミュレーション言語では，全イベントルーチンが主コントロールルーチンに対して一様レベルにならんでいて対象としているシステムの構造を explicit な形では反映しにくいという欠点がある (**Fig. 1**)．

そこでわれわれは actor と呼ぶ新しい概念を導入し

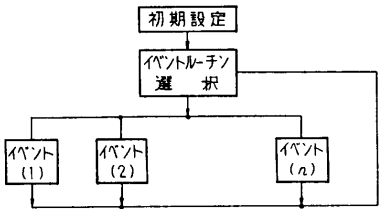

Fig. 1 Control structure of a discrete event simulation program.

* Structuael Design of Simulation Programs by Yukio KANEDA (Fauculty of Engineering, Kobe University)

** 神戸大学工学部システム工学科

対象システムの持つ構造をシミュレーションプログラムの構造に反映させることを可能とし，構造化プログラミングを促進させることができたので報告する．

2. 従来のモデル作成技法の問題点

離散イベント形のシミュレーションプログラムにおいては，対象のシステムの任意の時点での状態は，その時点でのシステム変数によってあらわされている．またシステムのダイナミックな動きは，次々とコールされていくイベントルーチンがこのシステム変数にアクセスし，操作を加えていくことによってシミュレートされている．

したがってシミュレーションプログラムは，システム変数とイベントルーチンが中心となって構成されており，プログラム作成に際しては，

全システム変数の決定
全イベントの決定
イベント相互間の関係の確立

を行ってからプログラム作成を行うという bottom up のアプローチをとることになる．

このことから問題点として

1. 全イベントルーチンは対等で一様に取り扱われるためシステムの持つ構造（特に階層構造）がプログラムの構造に反映しにくい．
2. システムを大きなモジュールに分割し，さらに各モジュールを細分化していくという top down のモデル作成を行うのに不向きで，多人数でプログラム作成を行うのに不向きで，多人数でプログラム作成を行う場合に特に大きな欠点となる．
3. あるイベントルーチンで発生したエラーが伝播し他のイベントルーチン中でそのエラーが発見される場合が多いが，イベントルーチン間にexplicitな形で存在する壁が少ないので非常に多くの伝播経路が想定でき，原因の解明が困難で，デバッグを難しくする．
4. モデルの変更を行う場合，システムの構造をプログラムが反映していないので，予期し得ない影響を他に及ぼす恐れがある．

などを上げることができる．

3. 構造化されたプログラムの作成法

前述したように，シミュレーションの対象となっているシステムは homogeneous な形態をしている場合はほとんどなく一般にある種の構造を持っている．したがってイベントルーチン中心のシミュレーションプログラムを作成しても，各イベントを1つ1つチェックしていくと，相互関係が強く1グループとして考えた方が適当と考えられるイベントのグループに分割でき，システムは全体としていくつかのグループとそれらグループ間の比較的ソフトな相互関係によって成り立っていることが判る．例えばそれぞれが特徴を持った複数台のコンピュータシステムによって構成されているネットワークを考えた場合，各コンピュータシステムの機能を記述しているイベント群それぞれは一つのグループとして考えるのが自然であろう．

また，いくつかの機械グループから構成されるジョブショップモデルにおいても各機械グループの機能を記述してるイベント群はそれぞれ1グループと見なせる．このような1グループ内の相互関係の強いイベントルーチン群間の制御の移動とグループ間にまたがるイベントルーチン間の制御の移動とを分離して考えるとそのプログラムの制御構造は **Fig. 2** のようになる．

このことはシステムのモデル化を行う場合，全イベントの洗い出しを行う前にグループ分けを行いその相互関係を決め，次に各グループ内に属するイベント——サブイベントと呼ぶ——の洗い出しとそれらの相互関係の確立を行うような top down のモデル化の手法が採用し易くなることを示している．

3.1 actor の概念

システムのモデル化を進めていく場合の world view として actor という考えを導入する．ここで提案している手法では，複雑なシステムの動きを「相互に連絡し合いながら半ば独立に働く actor の集合」としてとらえる．

actor とはシステムの基本構成要素で対象のシステムの持つあるまとまった機能をシミュレートしておりシミュレーション実行中ずっとシステムに存在している．したがって actor はその機能を記述しているプロセデュアとは一対一の対応があり，各 actor はシステ

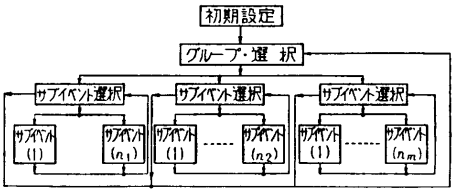

Fig. 2 Control structure of a simulation program which is composed of several subevent routine groups.

ム内に1つずつ存在することになる.

各 actor は「他の actor からの信号により動作を開始し，自分の立場を認識し，ある判断を行ってその判断にもとづいてアクションを起し，システム環境を変えたり，他の actor に信号を送る」機能を持っておりここでの actor にとっての環境とは観察可能なシステム変数を指すことになる.

特に離散形シミュレーションプログラムにおいてはシステムのダイナミックな動きを記述するため，多くの待ち行列を用いるが，本方式では将来サブイベント表以外の待ち行列は原則として actor に所属させ別の actor からの不用意な操作を禁じている．このように actor 間の待ち行列の共有を禁じると若干の自由度は失われるが，ある待ち行列に対して操作を加える actor が唯一に決まるので，発見困難なバグとなり易い予想困難なルーチンによる待ち行列操作の可能性が減り，しかも actor 間の機能の分担が明確になる——例えばある actor が別の actor に所属すべき待ち行列に item を挿入したり，取り出したりする操作がなくなる——という利点が生じるのである.

このように actor* を中心としたモデル化では，まず最初にシステム中のあるまとまった機能を actor としてとらえるというアプローチをとるので，モデル化では top down のモデル作成方式が極めて取り易く，シミュレーションモデル作成の常とう手段である包括的なモデルから詳細モデルに改変していく過程が自然にもり込めるという利点が生じる．また各 actor はある程度の自主判断機能を持つことになるが，これによって実際のシステムにおいてしばしば見られる各セクションごとに分散して存在するローカルな判断機能をすなおな形で表現できる.

このようにして作成されたプログラムは対象のシステムの持つ機能的な構造を直接反映しており，システムのモデル化およびモデルの正しさの確認などが容易になるという特徴があらわれる.

3.2 シミュレーションプログラムの構造

actor を中心としたプログラムの構造は従来のイベントルーチンを中心としたプログラムと構造が異なっていることはすでに述べた．すなわち actor を記述しているプログラムはいくつかのサブイベントルーチンとそれらをローカルに制御するサブコントロールルーチンから成り立っており，シミュレーションプログラムはこれら actor ルーチンとそれら全体を制御する主コントロールルーチンから成り立つことになる.

ここでは構造上の主な相違点として(1)コントロールの移動と，(2)サブイベントのキャンセルや再スケジューリングの問題を取り上げ若干の説明をする.

3.2.1 コントロールの移動

シミュレーションはコントロールが各 actor や actor に属するサブイベントルーチン間を移動して，それぞれのルーチンが実行されることによって進められるが，このようなコントロールの移動は次の3つの場合

(i) 同一 actor 内のサブイベントルーチン間のコントロールの移動

(ii) 時間の経過にともなって生じるコントロールの移動

(iii) actor 間通信にともなうコントロールの移動

に分けて考えられる.

(i)は同一 actor 内のサブイベント間の時間経過をともなわないコントロールの移動で，サブイベントコントロール部が制御する.

すなわち，サブイベントルーチンは同一 actor 内の別のサブイベントルーチンにコントロールを渡す場合は相手のサブイベント番号を指定してリターンするとサブイベントコントロールは指定されたサブイベントルーチンをコールする.

(ii)のコントロールの移動は主コントロール部が行う.

これは主コントロール部にのみ時間の流れを制御する機能があるからで，このように時間の管理機能が主ルーチンにのみ存在するのは，各 actor とも同一の時間系で動作しているため必然として集中管理を行わねばならないからである.

全将来サブイベント (future subevent) は共通の待ち行列に発生予定時刻順に格納されており，主ルーチンは(ii)のコントロールの移動に際しては先頭のサブイベントの発生時刻にシステム時刻を進めて対応する

---

* actor の概念はプロセスの概念と共通する点も多いが以下の点が異なっている.

すなわちプロセスとは時間経過をともなって行われる一連の関連の強い活動 (アクティビティ) を呼び，このアクティビティの開始，中断，再開，終了等の一連のイベントの発生の軌跡を示すことになる.

world view としてプロセスの概念を採用するとモデル作成者はシステム中を流れていく item (GPSS では transaction であり SIMSCRIPT では temporary entity に当る) に着目し，各 item が流れていく途中でうける処理過程を追って記述していくという方式をとる．またプロセスはそのアクティビティを記述しているプロセデュアとは実行時には遊離してコルーチンの関係になり，多くのプロセスが生成されたり消滅したりしながらダイナミックに動き，システムはこれらプロセス間の相互制御作用を利用することによりシミュレートされている．以上がプロセスが actor と異なっている主な点である.

サブイベントルーチンをそのサブイベントルーチンが所属している actor を介してコールすることになる.

(iii)の actor 間の通信によるコントロールの移動も主コントロールを介して行う. 別の actor にコントロールを移動する際には, 主ルーチンに相手の actor 番号を指定してリターンすることになる. 主ルーチンは指定された actor をコールする.

(i), (ii), (iii)いずれの場合もルーチン間のパラメータの授受を行う必要があるが, FORTRAN のラベル付き COMMON 領域のような機能を用いて, 各ルーチンは必要最少限度のデータにしかアクセスできないようにすることが望ましい.

3.2.2 サブイベントのキャンセルと再スケジューリング

シミュレーションにおいては, 将来イベント表に登録されたイベントのキャンセルや再スケジューリングがしばしば行われる. 任意のイベントルーチンが任意の将来イベントのキャンセルとか再スケジューリングが可能であると, 自由度はあるが, 発見困難なバグのもとになり易い. 本方式では, 将来サブイベントのキャンセルや再スケジューリングは, そのサブイベントが属している actor ルーチンを介してのみ実行できるようにして不注意により誤ったキャンセルや再スケジューリングが発生するのを防ぐことができる.

4. モデル作成の実例

actor の概念によるモデル化の一例としてタイムシェアリングシステムのモデル化をとり上げシミュレーションプログラム作成技法の概略を述べる.

4.1 システム構成

システムは **Fig. 3** に示すようにターミナル群, CPU, 主記憶装置, ディスク装置から構成されている.

各ターミナルにユーザはランダムに到着し, ターミナルが空いていれば, ターミナルから次々とジョブを実行しすべてのジョブの実行が終るとシステムから去っていく. 一方ターミナルが空いていなければユーザは何もせずにシステムから去っていくとする.

ここでジョブとは, ターミナルからの一回の指令によりシステムが実行する計算で, 必要な主記憶容量, CPU 使用時間, ディスクへの入出力回数が与えられている.

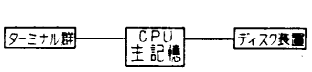

Fig. 3 Modele of Time Sharing System.

4.1 actor によるモデル化

システムのモデルを作成するため3つの actor

actor (1): CPU 部および主記憶部の機能を記述

actor (2): ディスク部の機能を記述

actor (3): ターミナル群, ユーザの機能記述

を考える.

1) actor (1): CPU 部および主記憶部におけるジョブの処理を担当する部分で以下の4つのサブイベントが考えられる.

サブイベント(1):

ジョブが主記憶の割当を要求するサブイベントで, 十分な領域が空いていれば割当を行い引き続いて CPU を要求するサブイベント(2)をスケジュールする. 十分な領域がなければ主記憶待ちの状態となる.

サブイベント(2):

CPU を要求するサブイベントで, 空いていれば割当, 空いていない場合は CPU 待ちとなる. CPU の割当が行われたジョブに対しては, 残り入出力回数および残り CPU 時間から次の入出力要求発生時刻を推定して CPU 開放サブイベント(3)をスケジュールする.

残り入出力回数が0のときは, 残り CPU 時間後にジョブは終了するとしてサブイベント(4)を終了時刻に発生するようスケジュールする.

サブイベント(3):

入出力要求により CPU を開放するサブイベントで CPU は開放され, 処理装置待ちのジョブがあれば, 先頭のジョブを取り出し, サブイベント(2)が現時刻で発生するようにスケジュールし, actor (2) にコントロールを移す.

サブイベント(4):

ジョブ終了を示すサブイベントで, 主記憶および CPU が開放される. 主記憶, CPU の待ちが調べられ, 待たされているジョブがある場合には, 先頭のジョブが取り除かれ, それぞれサブイベント(1), (2)が現時刻に発生するようにスケジュールされ, コントロールは actor (3) に移される.

2) actor (2): ディスク装置に対する入出力要求に対する処理を行う actor で2つのサブイベントを持つ.

サブイベント(1):

入出力開始の処理をするサブイベントでディスクが空いている場合には入出力要求は受付けられ, 入

出力時間が計算され，終了時刻に入出力終了サブイベント(2)のスケジュールをする．ディスクが空いていない場合はディスク待ちとなる．

サブイベント(2)：

入出力終了を示すサブイベントで，ディスクは開放され，ディスク待ちのジョブがあれば同時刻にサブイベント(1)が発生するようにスケジュールをし，actor (1) にコントロールが移される．

3) actor (3)：ターミナル部におけるユーザの動作をシミュレートする actor でサブイベントを3つ持つ．

サブイベント(1)：

ユーザのターミナルへの到着を示すサブイベントで，まず最初に次のユーザのターミナル到着のサブイベント(1)をスケジュールする．指定されたターミナルが空いていれば一連のジョブシークェンス表を発生してサブイベント(2)をスケジュールする．一方ターミナルが空いていない場合はシステムからそのユーザは消える．

サブイベント(2)：

ジョブ開始のサブイベントで，そのジョブの処理に必要な主記憶容量，CPU 使用時間，ディスクへの入出力回数をパラメータにしてジョブ実行のため actor (1) にコントロールを移す．

サブイベント(3)：

ジョブ終了の処理をするサブイベントで，そのユーザにとって最後のジョブならターミナルを開放する．そうでなければ次のジョブの開始のため思考時間後にサブイベント(2)が発生するようにスケジュールする．

4.3 プログラムの骨格

プログラムは初期設定部，主コントロール部，actor (1)，actor (2)，actor (3) から構成されていて全体の流れ図は **Fig. 4** (次頁参照)のようになる．

初期設定部はパラメータの読み込みと最初のユーザのシステム到着を示すサブイベントをスケジュールしている．

主コントロール部は actor 間のコントロールの移動および時間経過にともなうサブイベントの発生をトレースするシークェンスコントロールの働きをしている．actor (1)，actor (2)，actor (3) はそれぞれ前述した働きをする actor ルーチンで，それぞれサブイベントルーチンとサブイベントコントロールルーチンから成り立っている．

このプログラムではグローバル変数 INDCTR, TIME, A, E, T, J を用い，グローバルなリストとして TLIST を用いている．

INDCTR はコントロールの移動を制御するための標示子で INDCTR=1 のときは，**2.3** で示した(ⅱ)によるコントロールの移動を示し，INDCTR=2 は(ⅲ)によるコントロールの移動を示し，INDCTR=3 のときは(ⅰ)によるコントロールの移動を示している．

TIME はその時点でのシミュレーション時刻を示すグローバル変数である．

A, E, T, J はそれぞれコントロールの移動や待ち行列の操作のとき用いる変数で A は actor 番号，E はサブイベント番号，T は時刻，J はユーザパラメータを指すポインタである．

TLIST は将来サブイベント表で，一つのエントリに A, E, T, J が格納されており，リストは T の値の小さい順にソートされている．

時刻の更新の操作は TLIST の先頭のエントリを取り出し A, E, T, J にセットし合わせて TIME=T とすることによって行われる．逆にスケジュール(T)の操作は，A, E, T, J の値を TLIST に挿入することになる．

主記憶待ち，CPU 待ち，ディスク待ちにおいて実行される操作は，その時点における A, E, T, J の値をそれぞれ actor (1) に属する MLIST, PLIST および actor (2) に属する DLIST に挿入する操作である．

5. 結 論

前章までで actor の考えを中心としたシミュレーションプログラムの作成法について述べた．

本方式では対象のシステムの持つ構造を直接反映させることに力点を置いてモデル化するので

1. プログラムの正しさを確めるのが容易になる．
2. 1人のプログラマが構造およびダイナミックな動きを十分に理解できる程度に一つの actor のプログラムサイズをおさえることができる．
3. 各 actor は対象システム内の比較的独立した機能の働きをシミュレートしているので相互関連が比較的少く多人数のプログラマが並行してプログラムを作成し部分的にデバッグを行うことが可能である．
4. モデルの一部を変更する場合，actor 内の変更

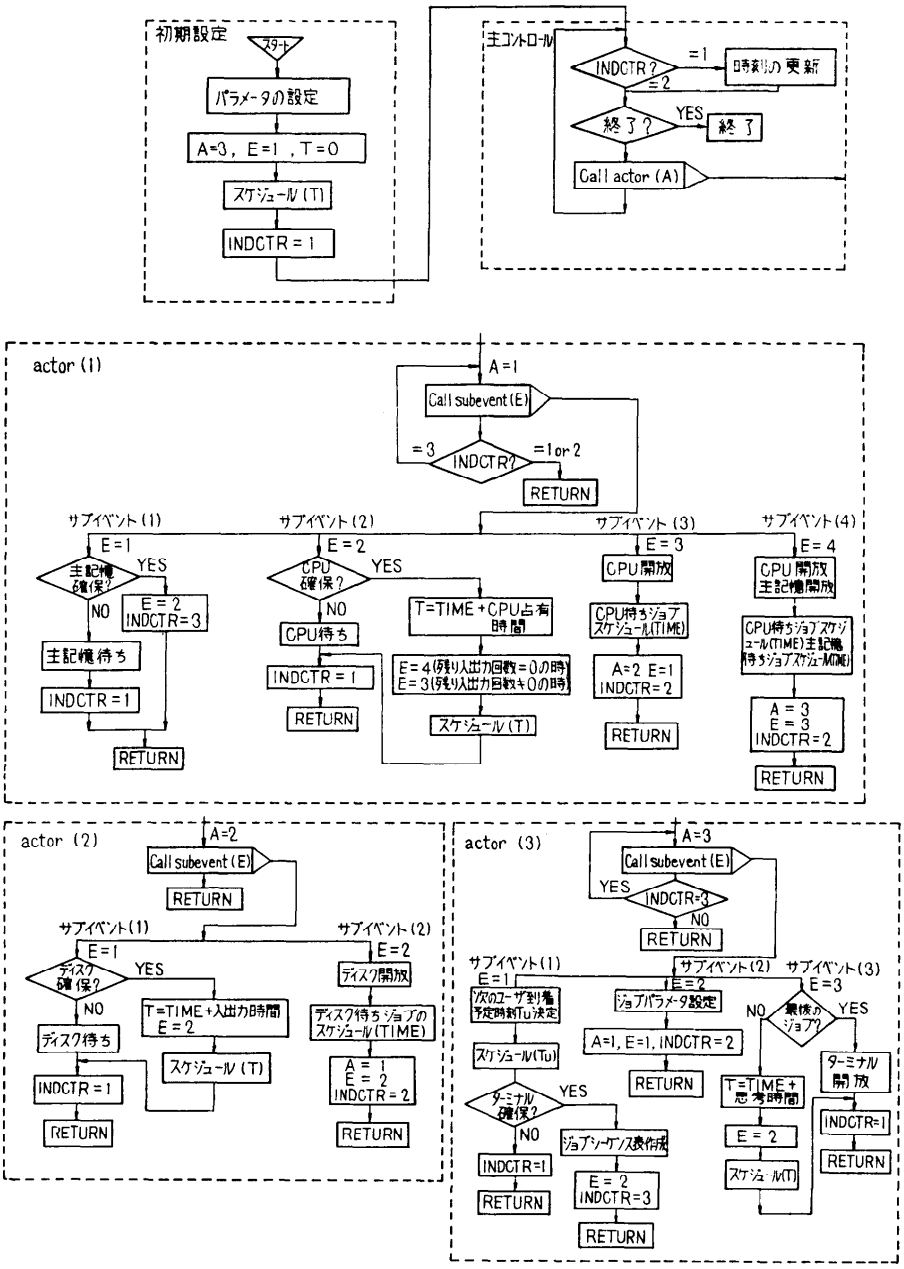

Fig. 4 Control structure of the simulation program.

のみで十分な場合が多く，他の actor に影響を与えずに変更が可能で拡張性が高いといえる．タイムシェアリングシステムの例をとると，多重プロセッサシステムを想定した場合とか CPU のスケジュールとして Time slice を導入したスケジュール方式を採用した場合も，actor (1) 内の変更を行うだけで十分の対応ができる．

5. 3に述べたように actor は他の actor とは比較的独立した動きをするように構成されていて，actor 間のコントロールの移動は比較的限られた

経路しか存在しないこと，また各種の待ち行列やリストはほとんどが actor に属しており，他の actor が不用意にアクセスしたり，操作を加えたりすることが不可能となっている．したがって大形のシミュレーションでは極めて発見が困難なリップルエラー (ripple error) が発生しにくい．

などを特徴として上げることができる．

われわれはまだ actor を採用したモデル化技法を本格的には採用していないが，対象システムの構造を反映した構造化プログラム作成技法として有効なものであると考えている．

謝辞 本研究は電子技術総合研究所で進めている自動プログラミングの研究の一環として行ったものである．日頃から熱心に御討論いただいている計算機方式研究室，言語処理研究室，推論機構研究室の諸氏に感謝いたします．

参 考 文 献

1) E. W. Dijkstra: "structured programming", Academic Press (1972).
2) W. P. Stevens 他: "structured design" IBM System J No. 2 (1974).
3) H. S. Krasnow 他: "The Past, Present and Future of General Simulation Languages", management science vol. 11, No. 2 (November 1964).
4) A. A. B. Pritsker 他: "Simulation with GASP II" Prentice-Hall (1969).
5) 淵一博，金田悠紀夫:「タイムシェアリングシステム」産業図書.

(昭和50年5月7日受付)
(昭和51年2月6日再受付)