# 対話型シミュレーション・システム

田畑孝一・大野　豊
京都大学工学部

## 1．はじめに

大規模なシステムのシミュレーションを行なう場合，バッチ処理的な手法による計画では不充分で，対話的な手法によるマン・マシン・システムが必要とされる．それにはオンライン対話型シミュレータが必要であり，このシミュレータにおけるモデルの形成・修正は容易でかつ即時でなければならない．我々はこの対話はグラフィック端末による図形も含むより高次で，より利用容易なものにすべきと考えている．

このようなねらいを持った対話型シミュレーション・システムを構成する作業の一環として，われわれは

(a) 対話によるモデル形成ならびにシミュレーション
(b) グラフィック端末による機械と人間との対話．

のあり方について研究を進めた結果，次の2点，すなわち

(1) トップダウン的モデル形成とシミュレーションの方法
(2) モザイク式手法によるグラフィックス

について一応の結果を得たのでこれらについて報告する．

## 2．グラフィックスを用いたトップダウン的モデル形成とそのシミュレーション

### 2－1．はじめに．

一般に，大規模で複雑なシステムのモデル形成およびその計算機シミュレーションの正確なプログラミングを短時間に行なうことは，困難とされている．このことは大規模な情報システムを開発する場合にもまたあてはまる．すなわち，そのために要するテストあるいはデバッグの労力は，全体の開発時間あるいは全コーディングの半分にも達するといわれている．そのためこのようなテストとデバッグの問題に関する種々の技法や手法が多くの研究者によって提案されてきている[1]．構造化プログラミング技法[2]に基づいた〝トップダウン〟的システムの設計とプログラミングは[3][4]，そのような提案の1つである．

この節では，最初にこのトップダウン的設計過程について整理する．というのは，われわれのモデル形成過程にも同様なトップダウン的手法を適用するからである．その次に，われわれのシミュレーション・システムのインタラクティブ性について述べる．

伝統的なプログラミング設計においては，設計はフローチャートを用いて紙の上でトップダウン的に進められる．フローチャートはシステムの構造の記述と，システムの各部分に対応するモジュールを定義するのに用いられている．その設計が完了すると，それらのモジュールがコーディングされ，各モジュール単位でデバッグが行なわれる．それから，これらのモジュールは，寄せ集められ，部分システムが構成される．部分システムもまた各々テストされデバッグされ，その後それらが寄せ集められ1つのシステムが構成され，最終的なテストとデバッグ

が行なわれる。すなわち，コーディングとデバッグの過程は"ボトムアップ(bottom-up)"的に行なわれている。このようなボトムアップ法の主な欠点は，システムや部分システムのテストに失敗した際に，どのモジュールにバグがあるかを容易には指摘できないことである。あれこれといろいろなモジュールを再びテストしてみなければならないはめにおちいる。

いっぽう，トップダウン的プログラミング法においては，システムの設計のみならずプログラミングあるいはコーディングの過程も，トップダウン的に進められる。トップダウンの各レベルで，コーディングされたプログラムは，少なくともそのシンタックスがチェックでき，また"program stub"を用いてテストのための実行も可能である。ここにprogram stubとは，いまだインプリメントされていないモジュールの存在をシミュレートするもので，ある機能的な部分仕様の代役をするものである。

トップダウン的過程で要求されるもう一つのことは，個々のプログラムモジュールはできるだけ短かいこと，できればコードあるいはstatementのテキストが1ページを越えないことが望ましい。さらにプログラムはGOTO statementを用いずに構成され，3つの型のstatementすなわち"concatenation"，"selection"あるいは"repetition" statementを用いて構成されることが仮定されている。プログラムの正当性および理解容易性はこの制約の下に維持されるといえる。

GPSS，SIMSCRIPT，SOLあるいはSIMULA 67 class SIMULATIONのような既知のシミュレーション言語は，必ずしも上の要求を満たしていない。

われわれのGMSS (Graphical Modeling and Simulation System)におけるモデル形成・シミュレーション言語は，上の要求を満足するように定義されている。したがって，プログラム・テキスト数ページをこえる飛越し制御は決して現われなく，またモデル・プログラムの各ステップをトップダウン法で直進的に書いてゆくことができる。

さて，次にわれわれの言語のインタラクティブ性について述べる。

ここしばらく，離散的計算機シミュレーションの分野でもっとも発展の見込のあるのは

(1) インタラクティブ・モデル形成
(2) シミュレーション向きグラフィックス

であるとされている[6]。もっともよく知られているインタラクティブ・シミュレーション言語は，MITで開発されたOPS-3である。OPS-3によって，ユーザはタイムシェアリング環境でオンラインのインタラクティブ・モデル形成ができる。シミュレーション向きグラフィックスの例はGPSSのオンライン手書きフローチャート入力とその絵的な形式の出力である。この種のインタラクティブなモデル形成と出力の解析にはCRTがより適している。

われわれのモデル形成・シミュレーション・システムは上の2点両方とも重視している。OPS-3やSIMSCRIPTによるプログラムでは，シミュレートされるべき対象すなわちentityはいずれもユーザによって書かれたステートメントのテキストによって定義される。しかし，ユーザによって任意に定義されたentityのような抽象的な概念の各々に対して具象的な絵の形態を与えて表示することは困難である。

絵的な表現を行なうために，われわれは"facility"，"storage"あるいは"queue"

のような "elementary" な entity を用意し、それらに対して各々個有な絵の形態を与えた。したがって entity に関していえば、われわれの言語は SIMSCRIPT よりもむしろ GPSS に似ている。われわれによって導入された抽象的な entity である "actentity" を表示するためには、ラベル付のボックスを用いる。

2-2. トップダウン的モデル形成

いかなるシステムでもそれをシミュレートするためには、そのシステムのモデルをつくる必要がある。システムのモデルとは、関連した entity の集合を意味する。各々の entity はまた互いに関連した属性によって性格づけられている。ここに1つの entity はシミュレーションされるべき1つの対象である。1つの event は1つの entity の状態の1つの変化を表わす。システムのシミュレーションは時間的に並んだ event の順序にしたがって進んでゆく。1つの activity は1つの entity の状態を変化させる operation の集合である。

われわれのモデル形成・シミュレーション言語では、"elementary" entity として、"transaction", "facility", "storage", "queue" および "semaphore" などを用意している。それら elementary entity に加えて、われわれは特別な抽象的な entity である "actentity" をまた用意した。transaction と呼ばれる entity は facility, storage, actentity などを含めた他の型の entity に作用する。transaction はシステムのモデルに入り、他の entity に作用しあるいは種々の operation に影響され、それからそのシステムから出てゆく。

この言語には3つの型の statement (operation) がある。すなわち primitive, structure および activity statement である。structure statement は単1の入口と出口をもつ "selection" および "repetition" statement であり、それらは transaction のフローを制御する。

ここで actentity および activity statement の意味を説明する。1つの actentity は、そのシステムのある1部のモデルを表わす1つの抽象的な entity である。activity statement は transaction に作用し、上の actentity に対応するシステムのその部分の振舞をシミュレートする。actentity の作用 (機能) —— すなわち、何をするか —— は完全に定義されている必要があるが、それをどのようにインプリメントするかは、この時点では必ずしも定義されている必要がない。その作用すなわち activity は、モデル化段階の次の下位レベルの [ACTIVITY] 定義でインプリメントされる。

さて、与えられたシステムのモデルを構成しよう。

(1) そのシステムの入力/出力は transaction に対応するであろう。

(2) システムの機能はいくつかの部分に分割されよう。これらの各部は elementary entity あるいは actentity に対応するであろう。分割の総数は全プログラムのテキスト (あるいは図) が1ページ (あるいはディスプレイ装置の高々1～2面) に納まるように制限されるべきである。

(3) 入力 transaction のフロー制御は、primitive, activity, あるいは structure statement の直列順序を用いてプログラム・テキストに記述されよう。これらの statement は transaction やその他の型の entity に作用する。structure statement の節にはいかなる他の structure statement も用いてはならないことに注目しよう。すなわち structure statement の nesting は許されない。機能の部分的仕様 —— activity statement に対応する actentity の activity (機能) —— は完全に定義され、その仕様は文書化されモデル化段階の次の低位のレベルに送られる。

(4)上の(1)～(3)において"システム"という用語を"actentity"と置き換えて、すべてのactentityが最終的にelementary entityによって表わされるまで、これらの手続きを繰返せ。

ここで、structure statementの役割に注意しよう。通常のシミュレーション言語では、GOTOを含めたprimitive operationの任意の順序および任意の順序制御が許されている。一見、そのような順序と制御の自由度はモデル化過程に大なる柔軟性を与えているようにみえるが、結果として出来上るモデルの構造は非常に複雑となりがちで、他のだれかのみならずモデル作成者自身でさえも容易にはそれを読めなくなってしまう。いっぽう、われわれの場合はフロー制御のoperationは通常の直列順序を除けばたった2つの型のstructure statementに制御されている。実際、そのような制約はモデル化の過程にある種の概念的な規範——一歩一歩どのようにモデルを形成してゆくかという規範を与えてくれる。

## 2-3. シミュレーション

われわれのモデル形成・シミュレーション言語においては、システムのモデル化のみならずそのシミュレーションもtop-down的に進めることができる。top-down過程の各レベルでモデル化プログラムは容易にそのシンタックスについてチェックされうる。さらに"simulation stub"を用いてそのモデル化プログラムをテスト実行することができる。ここにsimulation stubとはまだインプリメントされていないactentityの存在をシミュレートするもので、ある機能の部分的仕様(actentityのactivityに相当する)の代役をするものである。simulation stubは、システム・アナリストの望みのままに簡単にも技巧を凝らしたものにもでき、[ACTIVITY]定義を用いて定義される。

われわれのシミュレーション言語のホスト言語はGPSSである。われわれの言語はGPSSソース言語に変換される。モデル形成とGPSSソース言語への変換は、グラフィックス・ミニコンを用いてなされる。GPSSソース言語は中型の計算機に送られシミュレーションが実行される。実行結果は再びグラフィック・ミニコンに送り返えされ表示される。画素のディスプレイ・ファイル、変換結果、実行結果などはディスクに格納される。

## 2-4. 例題とグラフィックス

この言語をformalな形で表現することは可能であるが[6]、ここでは例題を用いてその大要を説明する。W.R.Franta等が用いたシミュレーションの例題[7]をここではわれわれのTop-down的な手法で取り扱う。　図1のようなマルチプログラミング・システムを考える。これは単一プロセサ(CPU)、主メモリ(MEMORY)および2つの異なる型の入出力装置(IO1, IO2)から成っている。ジョブの到着時間間隔は既知の分布(interval)に従っていると仮定する。プロセサのサービスはタイムスライスされている。1量子時間(quantum)の処理すなわちサービスを受けるごとに、1つのジョブがI/Oサービ

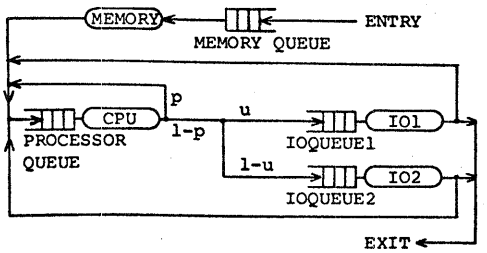



Fig.1 Multiprogramming system
(After FRANTA & MALY, IFIP74).

スを要求するか，さらにプロセサ時間を要求するかは，確率 $p, 1-p$ の Bernoulli 試行によって決まる。I/O の時間は装置に依存し（iotime 1, iotime 2），それら装置の選択は確率 $u, 1-u$ の Bernoulli 試行で決まる。ジョブの完了はジョブ・パラメタ（IOOPES），すなわちそのジョブによって要求される I/O の総数に基づいて決まる。到着ジョブは，それらのメモリ要求量（SIZE）が満足されるまで，待ち行列に残っている。メモリの割付は smallest first policy に基づくと仮定されている。

われわれの GMSS 言語では，次のように記述される。

```
Line
 1   GMSS MULTI(MULTI1);
 2   ACTIVITY MULTI1;
 3        STORAGE MEMORY(k);
 4        QUEUE QTOP(SIZE);
 5        PRIVATE IOOPES,SIZE;
 6        ACTENTITY JOBCPU,JOBIO;
 7     ENTRY;
 8        IOOPES=UNIFORM(l1,l2);
 9        SIZE=UNIFORM(s1,s2);
10        IN QTOP;
11        PUT SIZE TO MEMORY;
12        REPEAT ACT JOBCPU;
13               ACT JOBIO;
14               IOOPES=.-1;
15        UNTIL FINISH:(IOOPES=0);
16        GET SIZE FROM MEMORY;
17     EXIT;
18   END MULTI1;
           /* JOBCPU */
19   ACTIVITY JOBCPU;
20        FACILITY CPU(1);
21        QUEUE QCPU(FIFO);
22        PRIVATE SELVAL1;
23     ENTRY;
24        REPEAT IN QCPU;
25               USE CPU FOR quantum;
26               SELVAL1=BERNOU(p);
27        UNTIL IOREQ:(SELVAL1=1);
28     EXIT;
29   END JOBCPU;
           /* JOBIO */
30   ACTIVITY JOBIO;
31        FACILITY IO1(1),IO2(1);
32        QUEUE QIO1(FIFO),QIO2(FIFO);
33        PRIVATE SELVAL2;
34     ENTRY;
35        SELVAL2=BERNOU(u);
36        CASE SELVAL2 OF 2;
37           JOBIO1:BEGIN;
38              IN QIO1;
39              USE IO1 FOR iotime1;
40              END JOBIO1;
41           JOBIO2:BEGIN;
42              IN QIO2;
43              USE IO2 FOR iotime2;
44              END JOBIO2;
45     EXIT;
46   END JOBIO;
           /* BERNOU */
47   FUNCTION BERNOU(A);
48     IF DEC:(UNIFORM(0,100)<=A)
49              THEN BERNOU=0;
50              ELSE BERNOU=1;
51   END BERNOU;
           /* EXECUTION */
52   GENERATE(initial, interval, count);
53   EXEC MULTI1;
54   TERMINATE;
55   STOP n;
56   END GMSS;
```

## 註釈

1行　MULTIはプログラム名．この例においては．モデル化の最初の段階ではMULTI 1 と名付けられた actentity（抽象的な entity）はただ1つである。

2～18行．MULTI 1 の［ACTIVITY］定義；マルチプログラミングシステムの全体が記述される。

3～ 6行．MULTI 1 の declaration 部分。メモリ・サイズは k．メモリ待ち行列QTOPはSIZEパラメタを持つ"minimum mode"と宣言されている；すなわちもっとも小さいメモリ量を要求しているジョブがその待ち行列の先頭に置かれる．JOBCPU および JOBIOはそれぞれジョブに対するプロセサおよびI/O サービスを表わしている actentity．

7～17行．MULTI 1 の activity の body．

8～9行．ジョブに要求される I/O 回数とメモリ量をここで与える。UNIFORMは2

フのパラメタによって規定された値の間の一様乱数を与えるシステム関数である。

10行. ジョブは smallest first policy でメモリ待ち行列へ入る。

11行. ジョブは SIZE で与えられるユニット数のメモリを確保する。

12～15行. ジョブによって要求された I/O 回数が0になるまで、ジョブはプロセサ・サービスおよび I/O サービスを繰返して受ける。ACT は activity statement を示す。

14行. I/Oの回数が1減少される。"." は左辺の変数 IOOPES を表わす。

16行. ジョブは11行目で確保したメモリ・ユニットを解放する。

19-29行. JOBCPU の[ACTIVITY]定義；ジョブに対するプロセサ・サービスの振舞の記述。

20-22行. プロセサの容量は1。プロセサ待ち行列は first-in-first-out policy である。

24-27行. ジョブは、I/O サービスの要求があるまで、繰返しプロセサ・サービスを受ける。

25行. ジョブはスライス時間だけプロセサの処理を受ける。

26行. SELVAL1 はサービスの選択に使われる。BERNOU はモデル作成者によって定義された関数。

30～46行. JOBIOの[ACTIVITY]定義；ジョブに対する I/O の振舞の記述

36行. SELVAL2 は装置の選択に使われる。それが 0, 1 のとき、ジョブはそれぞれ IO1, IO2 を使う。

37-40行. この複合 statement は IO1 による I/O サービスを記述している。

41～44行. この複合 statement は IO2 による I/O サービスを記述している。

47-51行. 関数 BERNOU の定義。BERNOU は0か1である。0となる確率(%)がパラメタで与えられる。

52-55行. このプログラムの実行部分。

52行. この statement は transaction を創成する。"initial" は最初の transaction が創成される初期時間である。"interval" は既知の分布関数で、到着間隔時間を規定する。
"count" は創成される transaction の数

54行. transaction が消滅する。

55行. 規定数の transaction が消滅すると、シミュレーションの実行は停止される。

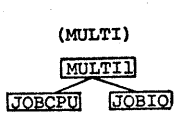
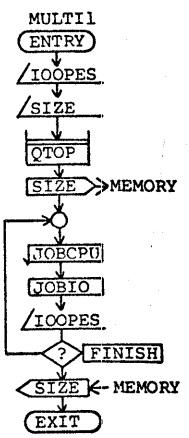



Fig.3 Model structure.



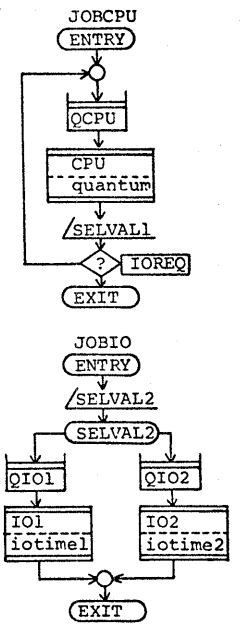




Fig.2 Graphical forms of the model of MULTI.

```
MULTI1      ASSIGN      5,FN1
            ASSIGN      4,FN2
            QUEUE       1
LINKOO      LINK        1,P4,LINKOO+1
            TEST GE     R1,P4,LINKOO
            ENTER       1,P4
            DEPART      1
            UNLINK      1,LINKOO+1,1
LABOO       TRANSFER    SBR,JOBCPU,3
            TRANSFER    SBR,JOBIO,3
            ASSIGN      5-,1
            TEST E      P5,0,LABOO
            LEAVE       1,P4
            UNLINK      1,LINKOO+1,1
            TRANSFER    P,2,1
1           STORAGE     k
1           FUNCTION    RN1,C2
0, l1/1, l2+1
2           FUNCTION    RN1,C2
0, s1/1, s2+1

JOBCPU      #UNDEF
            TRANSFER    P,3,1
JOBIO       #UNDEF
            TRANSFER    P,3,1
```

Fig.4 GPSS source program translated from the program "ACTIVITY MULTI1" written by GMSS language.

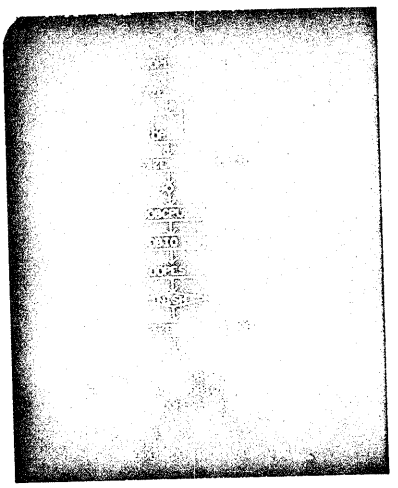


Fig.5 Display of a part of textual form and a graphical form of the model.

図2はこの例のモデルのグラフィックスによる表現である．プログラムのテキスト表現とグラフィック表現がよく対応していることが判る．図3は，この例のモデルの構造を示しており，モデルのactentityはtree構造を成している．

図4はわれわれの言語によって書かれたプログラムからGPSSソース言語への変換の1例で，[ACTIVITY MULTI 1]の部分に対応するものである．

図5は，グラフィック・ディスプレイ上の表示の一例である．プログラムのテキストと共に図形も表示され，現在処理の対象となっている行および画素がそれぞれ三角印(▲)および矢印(→)のポインタで指し示されている．これらのポインタは連動している．テキストおよび図形表示ともに巻紙型で取り扱われていて，ディスプレイ・ウィンドウはこれらのポインタで指し示された行および画素が実際に見えるように自動的に調整される．すなわち画面の上あるいは下にかくれた部分も，これらのポインタを下あるいは上に移動すれば，実際に見えるようになる．また，これらのポインタを用いて，置換，挿入，消去が出来る(グラフィック・ディスプレイ上の図形の取り扱いには後述のモザイク式手法の考えが取り入れられている)．

なお，われわれの言語からGPSSソース言語への変換の際にactentityが未定義であると，メッセージがディスプレイ上に表示されるので，モデル作成者はその時点で望みの代役のstatement ── すなわちsimulation stub ── をキイ入力することができる．

## 3. モザイク的手法によるグラフィックス

### 3-1. はじめに.

グラフィックスを用いて，絵(平面図)を画面から入力する場合，問題になるのは，

(1) 画面上にどのように絵を描くか
(2) 一画面に入りきらない絵をどのように取り扱うか
の2点である.

(1)については,代表的なものにIBM2250ディスプレイ装置を用いた方法がある[8]. これによるとまずGRAF(Graphics Addition to FORTRAN)言語によって絵の構成要素(画素片)を用意する。これら用意された画素片からその一つがファンクション・キーボードにより選択されて画面に表示されるがその位置については,別のDot-Gridルーチンでライトペンを用いて指示する.画面上にはラスターモード式に$\frac{1}{2}$インチ間隔の(正方)格子がドットで表示されていて,これらの一つをライトペンでヒットすることにより画素片の表示位置が決定される。この方法は有力な絵の入力方法の一つであるが,(2)の点については必ずしも充分な考慮がはらわれていないようである。

(2)については,画面のページングあるいは,ウィンドウイングが主として利用されている。しかし前者は2ページ以上(上下または左右)にまたがる絵の取扱いに難点がある。後者は望ましい方法であるが,それを真正面に取り入れると非常に手の込んだウィンドウイング・ソフトウエア手法(または高価なハードウェア手法)が要求される。

モザイク手法は,このような2点を満足するグラフィック入力の有効な一手法として提案されたものである。

## 3-2. モザイク的手法を用いたグラフィックス

無限の大きさの平面を仮定し,その平面全体にわたって一定の大きさの長方形のます目を碁盤目状に仮定しておく(ます目の大きさは1画面5×8=40のます目に相当する大きさ)。このます目に丁度あてはまる大きさで,予め種々の絵の構成要素を描いた画素片(これをモザイクという)を用意しておく。画面には,この平面の一部が表われていると考える。

別に,画面上を自由に動き,任意のます目を指示できるワーキング・ボックスが用意されている。これは,画面上ではます目の大きさに一致する,破線を辺とする長方形である(図6参照)。注目したます目が,このワーキング・ボックスの内部,上,下,左,右のどれになるかをライトペンで指示しながら,上述の種々の模様のモザイクをあてはめ,寄せ集めて一つの絵を構成する。もちろん,置換,消去,挿入もこのワーキング・ボックスを用いて可能である。

画面上にあらわれている部分は,無限の大きさの平面の一部で,ウインドウイングされていると考えるが,ウィンドウイングの枠は上下左右ともいずれかのます目の境界に一致させているので,一般のクリッピングに必要な複雑な手続きはいっさい必要ない。ワーキング・ボックスを画面の上,下,左,右の枠外へ移動させようとすると,平面が逆に下,上,右,左に1ます動いて等価的にワーキング・ボックスが1ます動いたことになる。この操作は,したがって,ウインドウイングの操作をも兼ねている。

構成した絵は,それに実際用いたモザイクの種類と相対位置関係のみを記憶しておけば,表現できるので,全体としての記憶容量は少なくてすむ(使用したモザイクの数のみに依存する)。空白のます目に関する情報は何も記憶する必要がないので,平面は実質的に無限に広いと考えることができる。

2-4でのべたグラフィックス・システム(図5)は現在のところここで述べた

モザイク的手法を主として上下方向に取り入れたものを採用している。しかし、GMSS 言語の case statement などで分岐の多い場合の取扱いを考慮するとそれでは不充分なのでここで紹介したような上下のみならず左右も考慮にいれたモザイク的手法を取り入れたシステムを採用することにした。これは現在インプリメント中である。

3-3. GPSS プログラミングへの応用.

モザイク式手法は2章でのべた GMSS システムのモデル表現に応用されているが、いっぽうこの手法を GPSS (GMSS のホスト言語である) のブロック図を CRT 画面上から入力することに応用し、それによって GPSS のプログラミングを柔軟に行なえるシステムを開発した。

グラフィックスを用いて GPSS のプログラミングを行なうシステムとしては金田、島田によるシステム[9]があるが、3-1で述べた(1)についてはグラフィック画面の片端にあらかじめ用意された GPSS のブロックの1つをライトペンでピックして任意の位置に移動することによってブロック図を指定した場所に表示する。(2)については画面が一杯になる前に接続表示ボックス・ブロックを作って、変数としてボックス名をつけて新しい画面を指定しなければならない。

これに対して、我々の場合は実質的に無限の平面の上にブロック図を書くことになっているので、ブロック図は上下左右どんなに大きくてもよく上のシステムの不便さが改善されている。すなわち、上述のモザイクとして、GPSS のブロック図用の名要素を描いたもの、およびフローを示す直線、屈曲線を描いたものを用意し、GPSS のブロック図をモザイク的手法で構成するのである。もちろん、ブロック図のどの位置での消去・挿入・置換も自由である。

ブロック図名要素にはパラメタ、ラベルなどを必要とするものがあるが、その場合には前述のワーキング・ボックスおよびキーボードを用いて入力し画面上に表示する。

出来上がったブロック図から GPSS ソース言語プログラムのテキスト表現への変

(a) (b)

Fig.6 Display of GPSS block diagram.

換を行なう。画面上で2つ以上にフローが分岐している場合，テキスト表現への変換アルゴリズムには少々の工夫を要した。図6a，6bは，このシステムで描かれたGPSSブロック図の例で，1つのブロック図をそれぞれウィンドウの位置を変えてみたものである。

## 4. むすび

われわれは対話型シミュレーション・システムのあり方について検討しているが，(1)トップダウン的モデル形成とシミュレーション・システム，(2)モザイク式手法によるグラフィックス，の2点について一応の成果を得た。(2)はグラフィック端末による図形をも含むマンマシン対話に有効な方法で，これは(1)のシステムのモデル形成の際の対話に応用されている。

(1)のシステムによると，モデル形成の過程のみならずシミュレーションの過程もトップダウン的に進めてゆくことができる。このために2つの概念，すなわちactentity および simulation stub の概念を導入した。このシステムの言語はGPSSをホスト言語としており，シミュレーションの実行は，この言語から変換されてでて来るGPSSソース言語を用いて，中型計算機で行なわれる。

この研究は今後，次の2つの方面に発展させてゆきたい。

(i) トップダウン的モデル形成・シミュレーション・システムを，SIMULA 67 のsubclassあるいはたとえばPL/1のようなよりポピュラーな言語を用いて開発する。

(ii) シミュレーションの実行途中でtransaction やその他のentityを逐次に追跡できるようなグラフィック的モニター能力を持ったトップダウン的モデル形成・シミュレーション・システムを，バルク・メモリを有する自立型のグラフィック・ミニコンを用いて開発する。

終りに，この研究を手伝って下さった昨年度の京都大学大学院学生・和田豊君および京都大学学生・酒井隆司君に感謝いたします。

なお，この研究は文部省科学研究費の援助を受けている。

## 文　献

[1] W.C.HETZEL, "Program Test Methods", Prentice-Hall, Englewood Cliffs, 1973.

[2] O.J.DAHL, E.W.DIJKSTRA and C.A.R.HOARE, "Structured Programming", Academic Press, London, 1972.

[3] H.MILLS, "Top Down Programming in Large System", DEBUGGING TECNIQUES IN LARGE SYSTEMS by R.Rustin, Prentice-Hall, Englewood Cliffs, 1971, 41-45.

[4] E.F.MILLER,JR. and G.E.LINDAMOOD, "Structured Programming; Top-down Approach", DATAMATION, December, 1973, 55-57

[5] J.N.MAGUIRE, "Discrete Computer Simulation---Technology and Applications--- The Next Ten Years", AFIPS Proceeding--Spring Joint Computer Conference, Vol.39, 1972, 815-826.

[6] K.TABATA, Y.WADA and Y.OHNO, "Top-down Modeling and Simulation with Graphics", 2nd USA-JAPAN Computer Conference, 1975.

[7] W.R.FRANTA and K.MALY, "Simulation Structured and SETL", INFORMATION PROCESSING 74, North-Holland PUblishing Company, 1974, 208-212.

[8] E.D. BERHOLD, M.P.BERHOLD and L.P.MCNAMEE,"Structured Operational Data Sets from Arbitrarily Arranged Computer Graphic Symbols", Advanced Computer Graphics--- Economics Techniques and Applications, by R.D.PARSLOW and R.ELLIOT GREEN, Plenum Press, London and New York, 1971, 161-178.

[9] 金田，島田，"対話型グラフィック・シミュレーション・システム" 情報処理，Vol.13，No.10，1972.