Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Introduction to SIMNET v2.0

Hamdy A. Taha

Department of Industrial Engineering
EC4207
University of Arkansas
Fayetteville, Arkansas 72701

ABSTRACT:

SIMNET is a network-based general-purpose discrete simulation language developed in totally compatible versions for the micro, mini, and mainframe computers. The language utilizes a fresh design approach that limits the number of nodes to exactly four: a source, a queue, a facility, and an auxiliary. Traffic among the four nodes is controlled by using special assignments, a strategy that is particularly suited for use with the IF-THEN-ELSE-ENDIF constructs. PROCs are used in a "stand alone" fashion to simulate an entire system with repetitive elements. A unique feature of SIMNET is the use of interactive (characters) graphics to estimate the transient period, following which the steady state run length is specified by the user and independent or global statistics are collected, all within the interactive mode of execution. The paper compares SIMNET with GPSS, SIMAN, and SLAM.

INTRODUCTION

The first version of SIMNET was released early in 1988. Release 2.0, which offers important enhancements and extensions, will be available in the first quarter of 1989. The basic features of the language are summarized below.

1. It employs four nodes only in a manner that does not necessitate the use of external (FORTRAN) inserts.
2. It provides file manipulations operations in the form of assignments which, together with the IF-THEN-ELSE-ENDIF constructs, allow full control over transaction flow anywhere in the network.
3. It employs PROCs in a "stand alone" fashion for simulating entire systems with repetitive elements.
4. It utilizes interactive character graphics to estimate (and delete) the transient period, following which independent or global statistics (based on the subinterval or the replication method) are collected, all within the interactive mode of execution.
5. Release 2.0 allows the use of any user-defined subscripted and nonsubscripted variables. It also includes all FORTRAN (single-precision) intrinsic functions and permits their use with mathematical expressions (with grouping parentheses) anywhere in the model. These enhancements offer considerable additional modeling power and adds versatilty to the language.

A complete documentation of the language appears in (Taha 1988a). A teaching manual (Taha 1988b) that introduces SIMNET through "hands-on" experimentation has also been prepared for the language.

BASIC STRUCTURE OF SIMNET

The structure of SIMNET includes three basic components:

1. Four nodes representing a source, a queue, a facility, and an auxiliary
2. Seven types of branches that can be used to direct traffic selectively between successive nodes.
3. Twenty special assignments that can be employed to move trans-actions among disjointed nodes of the network. These assignments are executed as transactions traverse branches.

This design strategy differs significantly from those employed by available process languages (e.g., GPSS, SIMAN, and SLAM). Specifically, present languages utilize special blocks/nodes to model transaction flow. This approach increases model abstractness and reduces the language flexibility, as evident by the fact that most of these systems still necessitate the use of external subroutines and/or functions. SIMNET, on the other hand, does not make use of external inserts. Its four nodes together with the special branches and assignments can be used to model any situation. Future expansion of the language can be effected by introducing new assignments; an approach that does not alter the basic four-node structure of the language.

SIMNET NODES

Figure 1 depicts SIMNET's four nodes. The source node creates new transactions, the queue node houses waiting transactions, the facility node is where service is performed, and the auxiliary node represents an infinite-capacity facility designed to enhance the model flexibility of the language. The choice of the first three nodes (source, queue, and facility) stems from the fact that most discrete simulation systems can be viewed, in some form or another, as queueing models. By limiting the language to these three nodes (and an auxiliary), developed models will be less abstract in nature.

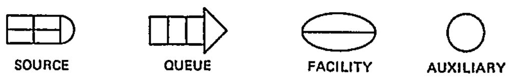


Fig. 1. SIMNET Nodes

Each node in SIMNET is assigned a user-defined name. This name can be used both as a node reference and as a label for orienting flow of transactions to the node (in all other process languages, such as GPSS, SIMAN and SLAM, a distinction is made between node/block reference name and label). Recognition of the node type is made by suffexing the node name by one of the symbols *S, *Q, *F, and *A to represent source, queue, facility, and auxiliary, respectively.

Nodes in SIMNET are defined to be self-contained in the sense that each node is equipped with information that defines the manner in which transactions enter, leave and reside in a node. Data about the node are expressed as statements consisting of a number of ordered fields separated by semicolons. The last field of the node terminates with a colon.

Figure 2 demonstrates a typical definition of a facility node named F1. The facility has two parallel servers and the service time per transaction is EXponential with mean 18 time units. When a service is completed, the associated transaction will go to the shorter of two "out" queues named Q3 and Q4. When the facility becomes idle, it will draw transactions from the longer of two "in" queues named Q1 and Q2.

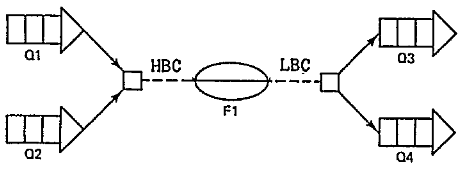


Fig. 2. A SIMNET Model Segment

The SIMNET statement of SRVR is expressed as follows:

```
    F1 *F;HBC(Q1,Q2);EX(18);2;LBC(Q3,Q4)

   node name (1)       (2) (3)     (4)
```

Field (1) specifies that F1 will draw an incoming transaction from Q1 or Q2, whichever has the Highest Busy Capacity. Similarly, field (4) indicates that a transaction representing a completed service will enter either Q1 or Q2, whichever has the Lowest Busy Capacity. Fields (2) and (3) are used to specify the service time and the number of parallel servers. Any or all the fields may be defaulted. Thus, a default in either (1) or (4) would signify fixed route specification (normally dictated by the physical sequence of the nodes). On the other hand, a default in field (2) signifies zero service time, whereas a default in (3) will represent a single server facility.

SIMNET treats queues and facilities as user-defined files with ordered entries. A queue may have a finite or infinite capacity. The capacity of a facility is necessarily finite, and it represents the number of parallel servers.

ROUTING TRANSACTIONS IN SIMNET

Transactions routing in SIMNET can take place in one of three ways:

1. Direct sequencing of nodes.
2. Transfers.
3. Branches.

In the first case, the physical ordering of nodes decides the route of the transaction. We can override the physical ordering of nodes by using either transfers or branches. Both transfers and branches perform equal functions in so far as transaction flow is concerned. The main difference occurs in that branches must be used only when it is desired to check conditions or execute assignments (among other functions). Otherwise, in the absence of these functions, transfer should be used since it provides more compact coding and also utilizes less storage.

In SIMNET, any number of branches (transfers) may emanate from the same node. Traditionally, available process languages, such as SIMAN and SLAM, utilize three types of branches: deterministic, probabilistic, and conditional. SIMNET takes advantage of the fact that branches (transfers) are scanned in their strict order in the list by proposing an additional three types of branches:

1. Dependent (D)
2. Exclusive (E)
3. Last choice (L).

A dependent branch (or D-branch) is designed to be taken only if at least one of the (deterministic, probabilistic and/or conditional) branches preceding it is taken. This requires the D-branch to be placed following the deterministic, probabilistic and/or conditional branches.

An exclusive branch (or E-branch) performs a function similar to that of the D-branch in the sense that it will be taken only if at least one of the preceding branches is taken. However, it differs from the D-branch in that it will block all the preceding routes if the E-branch itself cannot be taken.

The last choice branch (or L-branch) always appears at the very botton of the list of branches emanating from the node. The L-branch is designed to be taken only if none of its preceding branches are taken. Example:

To illustrate the power of branching in SIMNET, consider the following situation. In the final stage of automobile manufacturing, a car moving on a transporter is situated between two parallel workstations to allow work to be done on both the left and right sides of the car simultaneously. The service time on the left station follows a UNiform distribution between 18 and 28 minutes. That on the right station is also uniform between 20 and 30 minutes. When both operations are completed, the transporter moves the finished car outside the stations area. Transporters arrive at the parallel workstations every 25 to 35 minutes, uniformly distributed.

Figure 3 provides the network for the model. Transporters are created at source FEED every UN(25,35) minutes and wait in queue LINE until the workstations LEFT and RITE are both available. When one of the stations finishes its operation, it will send a transaction to auxiliary OUT. A job is said to be finished when the workstation with the longer service time is completed. (For the sake of simplicity, we will not detail how the model senses the termination of the longer of the two service times.)

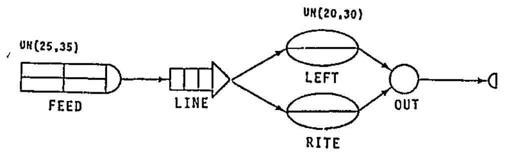


Fig. 3. Synchronized Parallel Stations

The important aspect of the present model is that work may not begin in either station unless both are free simultaneously. There are several ways for accomplishing this result in SIMNET using appropriate branching. Figure 3 shows how two conditional branches out of LINE are used to effect the desired result. These two branches, recognized by the symbols *B, are defined as follows:

```
LINE *Q:
     *B;LEFT/2;LEN(LEFT)+LEN(RITE)=0?:
     *B;RITE/2;LEN(LEFT)+LEN(RITE)=0?:
```

The combination LEN(LEFDT)+LEN(RITE)=0? (located in field 2 of each branch) tests the sum of the LENgths of LEFT and RITE. If the sum is zero, both stations are empty. The code /2 in field 1 of each branch indicates that (at most) two conditional branches will be taken out of LINE when their (respective) conditions are satisfied.

We can effect the same result by using a dependent D-branch in the following manner:

```
LINE *Q:
     *B;LEFT/1;LEN(LEFT)+LEN(RITE)=0?:
     *B;RITE/D:
```

In this case, the branch to LEFT is conditional and carries the code /1 to indicate that at most one conditional branch will be taken. The second branch to RITE now has the type /D, indicating that it is dependent and that it will be taken only if the preceding conditional branch is taken. Since the condition LEN(LEFT)+LEN(RITE)=0?, when satisfied, guarantees that both stations are empty, the combined use of conditional dependent branch will produce the same result as when two conditional branches are used.

A third way for effecting the same result is to use exclusive branching as follows:

```
LINE *Q:
     *B;LEFT/A:
     *B;RITE/E:
```

In this arrangement, the branch to LEFT will always (A-branch) be taken provided LEFT is empty. Simultaneously, since the (second) branch to RITE is an E-branch, it will be taken only if the A-branch to LEFT is taken, but will also block the A-branch if RITE happened to be busy.

The above illustrations demonstrate the power of using branches in SIMNET where complex routing logic can be simulated by using the proper combination of branches. This should prove helpful in modeling certain situations in a convenient manner.

Actually, branches play important roles in SIMNET modeling that go beyond transactions routing in the network. Specifically, branches perform the following additional functions:

1. Execution of assignments.
2. Collection of data for statistical variables.
3. Return model's resources back to their base stock.

These points will be discussed later in this paper.

SIMNET ASSIGNMENTS

SIMNET utilizes two distinct types of assignments:

1. Arithmetic assignments
2. Special assignments

The arithmetic assignments are the familiar ones that allow carrying out numeric calculations in the model. The special assignments, on the other hand, provide means for manipulating (swapping, deleting, replacing, and reordering) the entries of the systems files (queues and facilities) as well as cessation, movement and destruction of transactions anywhere in the network. They may also be used to collect data on statistical variables and adjust the level of a resource.

An important feature in SIMNET is that both arithmetic and special assignments can be used within the conditional statement IF-THEN-ELSE-ENDIF. The use of the conditional statement provides powerful modeling opportunities, particularly with regard to the use of the special assignments.

Arithmetic Assignment/Expressions:

The original release of SIMNET limits arithmetic expressions and assignments to the reserved variables I,J,K,L,M,N and the subscripted variables A(.), V(.) and W(.,.). Release 2.0 relaxes these restrictions and allow the utilization of any user-defined subscripted and nonsubcripted variables. The subscripted variables are limited to one and two dimensions only and their size must be declared by the $DIMENSION statement. For example,

```
$DIMENSION; ENTITY(200),TIME(10),A(3),TOTAL(3,5):
```

defines two single-dimensional arrays named TIME and A whose sizes are 10 and 3. Array TOTAL is two-dimensional with size 3X5. We remark that ENTITY does not represent a single-dimensional array. Rather, it is a reserved word that allocates 200 entries for the operation of SIMNET files. Also, the array A(.) is always reserved to represent the attributes of transactions.

In addition to SIMNET's reserved arithmetic function (e.g. LENgth of a file or LEVel of a resource), release 2.0 now incorporates all known single-precision functions that are available in FORTRAN, including all trigonometric functions. Additionally, release 2.0 admits the use of grouping parentheses in expressions and sets no restrictions on using (complex) expressions as subscripts functions arguments, or special assignments data. The following is a typical example of a SIMNET arithmetic assignment:

```
TOTAL(NV+(J-1)*MM,KK)=ABS((A(I)+MM)/LEN(QQ))
```

Notice that NV, J, MM, KK, and I are all user-defined variables.

Special Assignments:

All SIMNET's special assignments assume the format

A=B

where A and B represent special codes. Among the most useful of the special assignments are those dealing with file (queue and facility) manipulations. For example, the execution of the special assignments

2(QQ)=K(WW)

will move the Kth entry of file WW and place it second in file QQ. Such an operation, when implemented in available process language such as GPSS, SIMAN, and SLAM, will require considerable programming effort. The reason is that these languages are capable of routing transactions to labels only. The node/block associated with the label then decides how the arriving transaction will be processed.

Actually, the file manipulation assignments have the following general format:
(expression)(file name) = (expression)(file name)
This means that we can use the following assignment:

(A(I)+J**2)(QQ)=(SQRT(K))(WW)

In this case, we remove entry SQRT(K) from WW and place in position A(I)+J**2 in QQ.

Another important special assignment deals with switch control. A switch in SIMNET is defined by using $SWITCHES statement. For example,

$SWITCHES: GATE,ON,QQ:

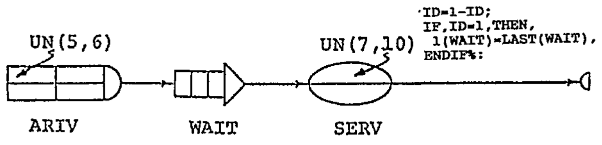
indicates that the switch named GATE is initially ON and that it controls a queue named QQ. The special assignment for changing the status of a switch assumes the format:

Switch name = ON or OFF

A declared status can then be used as a condition to either block or unblock a branch leading to a node. A notable use of the switch occurs when its status is changed to ON. In this case, an attempt will be made to release transactions from queues controlled by the switch.

SIMNET provides a total of 20 special assignments which, in addition to file manipulations and switch control, allow resuming or suspending creations from a source, stopping the simulation altogether, controlling the level of a resource, and collecting data on statistical variables. The use of SIMNET assignments is enhanced by the use of the conditional statement IF-THEN-ELSE-ENDIF. To illustrate the modeling power of conditional statement, consider the situation of a simple single-server model in which the selection of customers for service in the facility alternates between the head and the tail of the queue on a strict rotational basis: Figure 4 provides the network and its associated SIMNET model. The assignments

```
ID=1-ID,
IF,ID=1,
   THEN, 1(QQ)=LAST(QQ),
ENDIF:
```



```
$PROJECT;ROTATE,4/27/88,TAHA:
$DIMENSION;ENTITY(30):
$BEGIN:
 ARIV   *S;UN(5,6):
 WAIT   *Q:
 SRVR   *F;;UN(7,10)

        *B;TERM;;ID=1-ID;
                 IF,ID=1,THEN,
                  1(WAIT)=LAST(WAIT),
                 ENDIF%:
$END:
```

Fig. 4. SIMNET Single Server Model

are executed as transactions complete service in facility FF. Here, the (user defined) variable ID initially has a zero value (by default) and will alternate rotationally between 0 and 1 following each service completion. For the cases where ID=1, the IF-statement will move LAST(QQ) and place it as 1(QQ), immediately before facility FF attempts to draw transactions from QQ.

The power of SIMNET's file manipulation assignment in conjunction with the conditional IF-statement is realized by comparing the SIMNET model with those of GPSS, SIMAN and SLAM. In GPSS the logic of the (otherwise simple) single-server model will be significantly changed through the use of the advanced LINK/UNLINK blocks. Figure 5 gives the GPSS model of the problem.

```
     GENERATE   55,5
     LINK       1,FIFO,GO
GO   SEIZE      1
     ADVANCE    85,15
     RELEASE    1
     TEST E     X1,1,HEAD
     SAVEVALUE  1,0
     UNLINK     1,GO,1,BACK
     TERMINATE
HEAD SAVEVALUE  1,1
     UNLINK     1,GO,1
     TERMINATE
```

Fig. 5. GPSS Single Server Model

The SIMAN model follows a logic similar to (though simpler than) that of the GPSS model through the use of the REMOVE block. The SLAM model, on the other hand, is considerably more involved, both in logic development and in the use of file manipulations which can be effected only through the use of external FORTRAN subroutines. Figure 6 gives the SLAM model.

```
GEN,TAHA,ROTATE,4/27/88,1;
LIMITS,2,2,100:
NETWORK;
        RESOURCE,RS(1),1;
        CREATE,UNFRM(5,6);
        AWAIT(1),RS/1;
 AX1    GOON,1;
        ACT/1,,XX(1).EQ.0.0,AX2
        ACT/2,,XX(1).EQ.1.0;
        ASSIGN,XX(1)=0.0;
        ACT/3,UNFRM(7,10);
        FREE,RS/1;
        TERM;
 AX2    ASSIGN,XX(1)=1.0;
        ACT/4,UNFRM(7,10);
        EVENT,1;
        TERM;

        QUEUE(2);
        ACT,,,AX1;
        END
INIT,0,300;
FIN;

SUBROUTINE EVENT(IX)
COMMON/SCOM1/ ATRIB(100),DD(100),DDL(100),DTNOW,II,
1             MFA,MSTOP,NCLNR,NCRDR,NPRNT,NNRUN,NNSET,
2             NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
DIMENSION A(3)
IF (NNQ(1).LE.1) RETURN
CALL REMOVE(NNQ(1),1,A)
CALL FILEM(2,A)
RETURN
END
```

Fig. 6. SLAM Single Server Model

REMOTE CONTROL IN SIMNET

Remote control in SIMNET allows the modeler to initiate the movement of transactions out of queues and facilities from a disjointed segment of the network. To illustate this point, consider the network segments in Figure 7. Here, the transaction coming out of source SS executes the special assignment SW=ON, where SW is the name of the switch that is defined to control a queue named QQ. The second special assignment on the branch from source SS is 1(LL)=TRANS, which instructs the system to take a copy of the current transaction leaving SS and place it first in the queue named LL. The dashed routes show the impact of executing the asignments. First, SW=ON, will release a transaction from QQ and then sends it as far as it will go in the network (in this example, to facility FF provided it is free), before returning back to the branch on which SW=ON was executed. Next, the assignment 1(LL)=TRANS is executed. In this case, an attempt will also be made to send a copy of TRANS out of LL, if possible. This can only happen if the node following LL is free and the conditions, if any, on the branch from LL to WW are satisfied. Otherwise TRANS will simply occupy the first position in LL.

The preceding example illustrates two points. (1) File manipulation and switch assignments can be used to remotely effect movement of transactions in a disjointed segment of the network. (2) The assignments are truly dynamic in the sense that they will attempt to move "dislodged" transactions as far as they will go in the network.

SIMNET RESOURCES

Resources in SIMNET represent scarce items that can be used by one or more facilities in the model. The facilities in turn may have exclusive rights to given resources or may compete for them with or without preemption privileges. In this regard, we can think of facilities as "workplaces" whereas

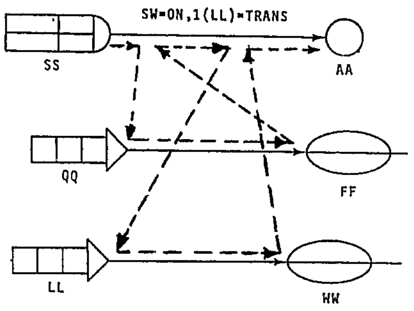


Fig. 7. Remote Control in SIMNET

resources represent the "tools" needed to perform services at these workplaces. A workplace may thus become (or made) idle because of the lack (or forced absence) of necessary tools. Distinction between facilities and resources in SIMNET provides opportunities for modeling certain logics that otherwise may be difficult to accomplish.

Resources acquisition and release are specified by using special codes in a specific field of each of the four nodes. In this case, SIMNET stipulates that resources be acquired only at a facility, but may be released anywhere in the network, including the exit end of a branch. We can override the restriction requiring a resource to be acquired only at a facility by using a special resource level assignment, which assumes the format

LEV(resource name)=expression

In this case, the level of a resource may be adjusted up or down anywhere in the network.

There are two advantages to specifying the use of a resource by a node field in place of a resource level assignment. First, the field code is designed to allow positive transit time between the location of the resource and its requested destination (and vice versa); and, second, preemption of resources from lower priority facilities is only possible when the use of a resource is specified by a node field.

Priority and preemption of resources are instituted in the definition statements, as opposed to the use of special blocks in GPSS, SIMAN, and SLAM. For example, consider the definitions:

```
$RESOURCES: R1;4(F1/F2,F3):
            R2;V(1)(F4,F5(NPR)/F6):
```

Resource R1, which has an initial level of 4, may be assigned to facilities named F1, F2, and F3, with F1 having a higher preemptive priority over F2 and F3. Resource R2, on the other hand, has an initial level of V(1) (a SIMNET arithmetic variable) and can be used with facilities F4, F5, and F6. In this case, F4 and F5, though of higher priority than F6, do not have preemptive privileges because of the use of the NPR code.

STATISTICAL VARIABLES IN SIMNET

There are two types of statistics collection in SIMNET:

(1) those dealing with queues, facilities, and resources, and (2) those requested by the user. The first type is totally automated as part of the standard output of SIMNET. User-defined variables on the other hand, are of three types:

1. Time based.
2. Observation based.
3. Run end.

The first two types are familiar elements in all simulation languages. The run-end type is added to allow computing certain variables only once at the end of the simulation run.

All variables are given names and defined in SIMNET by using the $VARIABLES statement. Collection of data within the model is achieved either by using the special assignment:

COLLECT = Variable name

or by listing them in the fourth field of a branch. In either case, the output of the model will provide a summary of the statistics of all user-defined variables.

INTERACTIVE DEBUGGING

SIMNET interactive debugger is designed to isolate model logic errors. The idea of the debugger, as in other languages, is to suspend execution at selected MARKpoints along the simulation time scale. At these points, the user can modify and/or display pertinent input and output data as well as view the model source statements. Snapshots of execution may also be stored and retrieved as desired.

The interactive TRACE command is particularly useful because of the level of automatic details it provides. The user may employ the command in a single step or interval format by using TRACE or TRACE T1 T2, respectively. Figure 8 provides a single step TRACE for the single server model introduced in Figure 4, immediately after a transaction leaves facility SRVR.

```
MARK *F/

-------------------------------------

Debugger MARKpoint at node SRVR  **
-------------------------------------

***Enter debugger command (or ?): CUR.TIME =   178.331400

TRACE

.1783E+03    Exit 'SRVR **'
               K =  .1000E+01
               Unlink entry  12 from 'WAIT **' --CUR.LEN =  11
               Link as entry   1 to 'WAIT **' -- CUR.LEN =  12
               Terminate transaction
               Make server in 'SRVR **' idle -- CUR.UTILIZ =   0
               Leave 'WAIT **' -- CUR.LEN =  11
               Enter 'SRVR  **' -- CUR.UTILIZ =   1
               File departure from 'SRVR **' at T =  .1868E+03

------------------------

Debugger TRACE endpoint
------------------------
```

Fig. 8. SIMNET Single Step Trace

INTERACTIVE EXECUTION

One of the unique features of SIMNET is the use of interactive (character) graphics to estimate the length of the transient period during execution. The estimated transient period is then truncated and independent or global statistics (based on either the subinterval or the replication method, or both) are collected, all without leaving the interactive mode of execution. Such procedures are implemented in most available process languages by using either manual or automated post-run analysis.

The estimation of the transient period in SIMNET is based on a heuristic that has been proposed in a variety of forms by a number of researchers (Conway 1963, Fishman 1972, Schriber 1974, Gordon 1975, Gafarian 1977). Essentially, we say that a simulated system has reached steady state if its output measures become stationary over time. A necessary (but not sufficient) condition for reaching steady state is that the mean and variance of desired output measures become stationary with time. This idea is the crux of SIMNET's procedure for estimating the length of the transient period.

The implementation of the procedure is based on the use of character graphics to represent the (cumulative) mean and standard deviation of each of the desired output variables. Plots of these variables are then displayed interactively for various (user-specified) incremental simulation periods. At the point in time where the user can visually detect "stability" in the plots, commands are issued to delete the "warm-up" period, following which steady state statistics can be gathered.

Desired graphs of the means and standard deviations are generated by using SIMNET's $PLOT statement as part of the original model. This statement assumes the following format:

```
$PLOT=NAME or STD(NAME)/
        symbol/lower limit/upper limit, repeats...
```

where NAME may represent a file, a statistical variable, or a resource. There is no limit on the number of plots to be generated by the statement.

The interactive session starts by issuing the following command at time T=0:

PUT PLOT=(n,t)

This command proposes (a maximum of) n plot points spaced t time units apart. For example, PLOT=(100000,20) allows plotting as many as 100000 points spaced 20 time units apart.

The next step is to execute the simulation for a time period which we "suspect" may subsume the transient period. This step is achieved by issuing the following command:

ADVANCE T

where T is a time increment specified by the user.

Now, we can view the $PLOT of the variables for the period (0,T) by issuing the following command:

DISPLAY PLOT

The ADVANCE/DISPLAY commands can be repeated as many times as needed until the user is satisfied that the obtained $PLOT reflects steady state conditions. At this point, the simulation period up to the end of the last ADVANCE increment is taken to represent the transient period, meaning that all the statistical arrays must be cleared in preparation for collecting steady state statistics.

To clear the arrays and collect steady state statistics, we issue the following command:

```
PUT  OBS=(N,T)  RUNS=M
```

where T represent the time base per observation. The specific choices of N and M automatically define the method to be used, as the following table shows:

Method	Number of Observations/Run N	Number of Runs M
Independent runs	0	≥ 1
Global statistics:		
Subinterval	>1	1
Replication	1	> 1

For example,

```
PUT   OBS=(5,500)  RUNS=1
```

will collect global statistics based on the subinterval method by subdividing a single run (past the transient period) into five subintervals, each with a time base of 500 time units. On the other hand,

```
PUT   OBS=(1,500)  RUNS=5
```

will specify the replication method consisting of five (independent) runs. Finally, the command

```
PUT   OBS=(0,500)   RUNS=5
```

will produce five independent runs. In this case, each run is expected to have its own initial data (see the section titled Data Initilization).

Figure 9 gives a typical interactive session applied to the single server model of Figure 4. The associated $PLOT statement is defined as follows: $PLOT= WAIT/W/.4/1.5;SYS TIME/S/3/12;STD(WAIT/*/0/4; STD(SYS TIME)/+/2/10:

The figure shows that steady state is estimated to start approximately at T= 300.

The global statistical summary may now obtained by using the following command (which implements the subinterval method):

```
PUT   OBS =(5,500)  RUNS=1
```

Figure 10 summarizes the results of the command. Observe that the output automatically provides 95% confidence intervals for all the output variables of the model.

SIMNET PROCS

In the model in Figure 11, three parts manufactured in three different departments are assembled

```
***Enter debugger command (or ?): CUR.TIME =   .000000000

PUT PLOT=(100000,10)

***Enter debugger command (or ?): CUR.TIME =   .000000000

ADVANCE 300

***Enter debugger command (or ?): CUR.TIME =   300.000000

DISPLAY PLOT

                         *** S I M N E T  PLOTS ***

SCALES:
W=WAIT          .4000E+00              .9500E+00              .1500E+01
S=SYS TIME      .3000E+01              .7500E+01              .1200E+02
*=STD-WAIT      .0000E+00              .2000E+01              .4000E+01
+=STD-SYS TI    .2000E+01              .6000E+01              .1000E+02

            0%        20%       40%       60%       80%       100%
  TIME      +....+....+....+....+....+....+....+....+....+....+
  .1000E+02 +                        .                        .
  .2000E+02 +  *                     .                        .
  .3000E+02 W S *    +               .                        .
  .4000E+02 W    *  S+               .                        .
  .5000E+02 W     +S*                .                        .
  .6000E+02 .     +S      *   W      .                        .
  .7000E+02 .             *     S  W .+                       .
  .8000E+02 .            *        W  .          S      +      .
  .9000E+02 .             *          .   W     S +            .
  .1000E+03 .             *          .          +             .
  .1100E+03 .             *          .      +       S         .
  .1200E+03 .             *          .       +S W             .
  .1300E+03 .             *          .      +S                .
  .1400E+03 .               *        .    +   S     W         .
  .1500E+03 .               *        .  +     S       W       .
  .1600E+03 .               *        .  +      S         W    .
  .1700E+03 .               *        .   +        S        W  .
  .1800E+03 .               *        .   +     S        W     .
  .1900E+03 .               *        .   +    S      W        .
  .2000E+03 .               *        .  +     S   W           .
  .2100E+03 .               *        .  +   S   W             .
  .2200E+03 .               *        .  +  S  W               .
  .2300E+03 .               *        .  + S W                 .
  .2400E+03 .              *         . +  S W                 .
  .2500E+03 .              *         . +S  W                  .
  .2600E+03 .              *         . +SW                    .
  .2700E+03 .              *         .+S                      .
  .2800E+03 .              *         .W+                      .
  .2900E+03 .              *         .+S                      .
  .3000E+03 .              *         .+WS                     .
  TIME      +....+....+....+....+....+....+....+....+....+....+
            0%        20%       40%       60%       80%       100%
```

Fig. 9. Interactive Estimation of Transient Period

to form one component. Assembled units are then sent to one of two departments to be painted.

The repetitive nature of the elements of the model suggests the need for a modeling facility that allows representing the system in a compact and nonredundant fashion. SIMNET offers this facility by using the concept of a PROC. Figure 12 gives the model statements associated with the stated example. The PROC is defined for the range (1-3) to represent the three parallel departments of the model. Each node is given a base name and blind index ( ). The blind index assumes one of the values specified by the PROC range. Thus, the "generic" node ARIV( ) *S represents sources ARIV(1), ARIV(2), and ARIV(3).

```
***Enter debugger command (or ?): CUR.TIME =    300.000000

PUT OBS=(5,500) RUNS=1

*** Revised run specifications:
    Run length        =     2800.000000
    Number of runs    =           1
    Number of obs/run =           5
    Time base per obs =      500.000000
    Transient period  =      300.000000

                       ***  G L O B A L   S T A T I S T I C A L   S U M M A R Y  ***
                                 (SUBINTERVAL METHOD - NBR OF OBS =  5)

                                    ---------------------
                                          Q U E U E S
                                    ---------------------

        CAPA-   IN/OUT   MEAN/S.D.     MIN/MAX/LAST  MEAN/S.D.    MEAN/S.D.    PERCENT NO-WAIT
        CITY    RATIO    LENGTH        LENGTH        DELAY(ALL)   DELAY(+VE)   TRANSACTIONS
WAIT    ****     1/ 1     .7423E+00     0/ 11/  0     .3466E+01    .6282E+01    .4500E+02
                          .5605E+00                   .2415E+01    .3043E+01
          95% LOWER CL =  .4646E-01                   .4683E+00    .2505E+01
          95%-UPPER CL =  .1438E+01                   .6464E+01    .1006E+02
                                    ---------------------
                                      F A C I L I T I E S
                                    ---------------------

        NBR     MIN/MAX/LAST   MEAN/S.D.     MEAN/S.D.    MEAN/S.D.     MEAN/S.D.     MEAN/S.D.
        SRVRS   UTILIZATION    UTILIZATION   BLOCKAGE     BLKGE TIME    IDLE TIME     BUSY TIME
SRVR     1        0/  1/  1     .5648E+00     .0000E+00    .0000E+00     .4622E+01     .6180E+01
                                .7199E-01     .0000E+00    .0000E+00     .3071E+00     .1557E+01
                95% LOWER CL =  .4755E+00     .0000E+00   ..0000E+00     .4241E+01     .4247E+01
                95% UPPER CL =  .6542E+00     .0000E+00    .0000E+00     .5003E+01     .8113E+01
                                    ---------------------
                                       V A R I A B L E S
                                    ---------------------

          GLOBAL MEAN    GLOBAL S.D.    GLOBAL MIN    GLOBAL MAX    95% LOWER CL  95% UPPER CL
SYS TIME   .6390E+01      .2427E+01      .1617E-01     .4746E+02     .3376E+01     .9403E+01


*** TRANSACTIONS COUNT AT T = .2800E+04 OF RUN  1:
NODE        IN       OUT    RESIDING    SKIPPING    UNLINKED/LINKED    TERMINATED
                                        (BLOCKED)   (DESTROYED)

*S:
 ARIV                566                             (     0)               0
*Q:
 WAIT       311      311       0           255           0/     0           0
*F:
 SRVR       566      565       1         (     0)    (     0)             565
```

Fig. 10. SIMNET Global Statistical Summary

The assemble facility MACH, on the other hand, is not blind-indexed because the model requires one assemble facility only.

The code ASM( ) appearing in Field 4 (/4/) of queues LINE( ) signifies that LINE(1), LINE(2), and LINE(3) are ASseMbled into one transaction before entering MACH. A transaction leaving MACH will then choose one of two paint departments on a ROTational basis by using the select code ROT(QPAINT(1-2)) in field 4 of MACH. Notice carefully that the range (1-2) overrides the PROC range (1-3) since the system has two paint departments only. Thus, within the same PROC, we are able to represent one, two, and three parallel elements. The example demonstrates the flexibility of PROCs in SIMNET.

We now compare SIMNET's PROCs with similar modeling facilities in other languages. GPSS allows the modeling of members of a repetitive set by using indirect addressing. SIMAN accomplishes the same task by employing STATION blocks and indexed names. SLAM offers this option by allowing the user to define resources, queues and service activities over indexed ranges. In general, the implementations in the three languages are based on similar ideas that require visitation of one member at a time, primarily through an "external" index change.

SIMNET PROCs provide special modeling power not available in GPSS, SIMAN, or SLAM. Specifically, a PROC can represent a "stand alone" model from the moment of creation until termination. Moreover, simultaneous actions on parallel members of a (repetitive) set are permissible as demonstrated by the application of ASM( ) operation to LINE( ). Indeed, implementation of simultaneous actions is possible in SIMNET mainly because each (self-contained) node carries information that describe the manner in which transactions enter and leave the node.

In GPSS, SIMAN, and SLAM, a "stand alone" representation is not feasible since, unlike SIMNET, the GENERATE/CREATE block/node (among others) cannot be indexed. More importantly, simultaneous actions on parallel members of the set are impossible, at least in a direct sense, since these members are reached only one at a time.

We should point out that the design of PROCs in SIMNET does allow modeling implementations in which the members of a set can be visited one at a time. In this regard, an "external" routing segment of the model is used to decide on the specific member to be visited next, exactly as in GPSS, SIMAN, and SLAM.

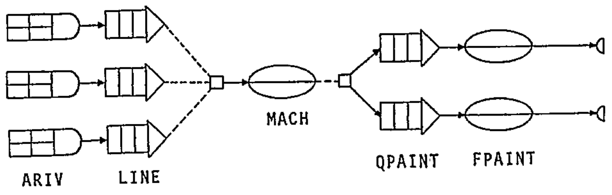


Fig. 11. A SIMNET Parallel Network Segment

INDEXING IN SIMNET

The design of PROCs in SIMNET dictates the need for indexed referencing of nodes, variables, resources, and switches. For example, in Figure 11, ARIV(1) is recognized as a legitimate (source) name by the SIMNET processor.

SIMNET allows considerable freedom in using indexing, not only in names, but also in arithmetic operations. Thus, A(LL(I)+NN**2) refers to the attribute of a transaction whose index is given as LL(I)+NN**2, where LL, I and NN are user-defined variables. In the model in Figure 11, we can use ARIV(A(2)+I) to refer to the source whose index is given by the current value of attribute A(2)+I.

One of the advantages of indexing in SIMNET is that its use is not restricted to the PROC domain only. Rather, indexing may be used anywhere in the model. In particular, the use of indexing with SIMNET's special assignments offers powerful modeling opportunities.

DATA INITIALIZATION IN SIMNET

The fact that SIMNET does not make use of external (FORTRAN) inserts necessitates that the language be self-contained, particularly in terms of data initialization. The following is a list of the types of initial data supported by SIMNET.

1. $INITIAL-ENTIRES define the initial transactions (by attributes) in SIMNET's queues and facilities at the start of the simulation.

2. $TABLE-LOOKUPS functions define pairs of independent/dependent variables using tabular format.

3. $DISCRETE-PDFS define empirical or discrete distributions used in the model (SIMNET also supports all common probability distributions).

4. $SUBSCRIPTED-VALUES provide initial values for SIMNET's (user-defined) subscripted variables.

5. $CONSTANTS provide initial values for SIMNET's (user-defined) non-subscripted variables.

Both $TABLE-LOOKUPS and $DISCRETE-PDFS can be accessed in the model in either discrete values or by using piecewise linear interpolation. For example, DI(1) will yield a discrete sample of DIscrete pdf 1, whereas DI(-1) will apply linear interpolation to the same function. A similar idea applies to $TABLE-LOOKUPS.

An important feature of SIMNET is that all initial data above can be presented in a run-specific format. In this manner, the user can execute as many runs as desired in the same session, each with different initial data as the following example illustrates:

```
$SUBSCRIPTED-VALUES:
          V: 1-2/NS/11,22,33:
             3-3/NS/-11,-22,-33:
```

These data specify the following values for the user-defined single subcripted variable V(.):

```
Runs 1 and 2: V(1)=11, V(2)=22, V(3)=33
Run 3       : V(1)=-11,V(2)=-22,V(3)=-33
```

There is no limit on the number of run levels to be used in a model.

GPSS and SLAM allow the use of run-specific initial values for certain arithmetic variables. Additionally, in GPSS a user may redefine functions (among other statements) prior to the start of each run. In SLAM, different initial entries in queues can be associated with different runs. SIMAN, on the other hand, does not have provisions for utilizing run-specific initial data in the same execution session. Instead, an experimental segment representing a single set of initial data is linked to the execution model. A different simulation session is thus needed with each new experiment.

CONCLUSIONS

SIMNET utilizes only four nodes, which, together with a number of special asignments, has proven effective in handling complex simulations.

SIMNET is the first language to use interactive graphics directly during execution to estimate the transient period, and then interactively choose the subinterval method, replication method, or independent runs to collect steady state statistics.

Release 2.0 now allows the use of complex mathematical expressions anywhere in the network and permits the user to specify any variables for use within these expressions.

PROCS offer the advantag of allowing the simulation of an entire system with repetitive segments by a "stand-alone" model. This differs from all available languages in that the equivalent of a PROC must be driven by an external segment of the model.

REFERENCES

Conway, R. 1963. "Some Tactical Problems in Digital Simulation." Management Science 10. no. 1 (Feb.): 47-61.

Fishman, G. 1972. "A Study of Bias Considerations in Simulation Experiments." Operations Research 20. no.5 (Aug.):785-790.

Gordon, G. 1975. The Application of GPSS V to Discrete System Simulation. Prentice-Hall, Englewood Cliffs, N.J.

Gafarian, A.; C. Ancher; and T. Morisaku. 1977. "The Problem of the Initial Transient with Respect to Mean Value in Digital Computer Simulation and the Evaluation of Some Proposed Solutions." Technical Report 77-1. University of Southern California.

Henriksen, J. and R.C. Crain. 1983. GPSS/H User's Manual. 2nd ed. Wolverine Software Corporation, Annandale, VA.

Pedgen, C.D. 1986. Introduction to SIMAN. Systems Modeling Corporation, College Station, PA.

Pritsker, A.A.B. 1986. Introduction to Simualtion and SLAM II. 3rd ed. Wiley (Halsted Press), New York.

Schriber, T. 1974. Simulation Using GPSS, Wiley, New York.

Taha, H. 1988a. Simulation Modeling and SIMNET. Prentice-Hall, Englewood Cliffs, N.J.

Taha, H. 1988b. SIMNET Teaching Manual, SimTec Inc., P.O.Box 3492, Fayetteville, AR 72702.

HAMDY A. TAHA is Professor of Industrial Engineering at the University of Arkansas and President of SimTec Inc. He is the developer of SIMNET and the author of Simulation Modeling and SIMNET, Prentice-Hall, 1988, SIMNET Teaching Manual, SimTec Inc., 1988, Operations Research, 4th Ed., Macmillan, 1987, and Integer Programming: Theory, Applications and Computations, Academic Press, 1975. He has extensive consulting experience in industry and government in the United States, Latin America, and the Middle East.

Hamdy A. Taha
Industrial Engineering Department
EC 4207
Bell Engineering Center
University of Arkansas
Fayetteville, AR 72701

(501) 575-6031

SIMNET is a trademark of SimTec, Inc.