QUIKSIM - A BLOCK STRUCTURED SIMULATION LANGUAGE WRITTEN IN SIMSCRIPT

DAVID G. WEAMER
National Cash Register Company
Hawthorne, California

I. Introduction

Users of simulation languages have historically had to choose between one of two types of language. On the one hand, they could choose a block type language such as GPSS. Languages of this type have the advantage that the learning and programming of the language is both quick and relatively easy. Also, since the number of debugging runs is usually small, turn around time is likely to be quite rapid. However, the block structuring of the language may also create some difficulties. The fixed nature of the blocks, both as to the types of blocks available in the language and the manner in which the blocks simulate a particular activity, may render the language unsuitable for the simulation of certain types of systems. In addition, the need to pre-allocate much of the available memory space of the computer may make the use of a block structured language impossible on a small computer.

The alternative to a block structured language is an algebraic language, such as SIMSCRIPT or FORTRAN. Languages of this type are extremely flexible and give the user a wide range of options in simulating a system. However, the programming time required to implement a model written in an algebraic language may be considerable due to the likelihood of a number of debugging and recompilation runs. In addition, making changes to a previously written program will also involve a certain amount of debugging and recompilation time.

Clearly, it would be desirable to have a language which combines the advantages of both types of language without some of the disadvantages. QUIKSIM represents an attempt to produce such a language.

II. Fundamentals of the QUIKSIM Language

QUIKSIM is a block structured language which operates

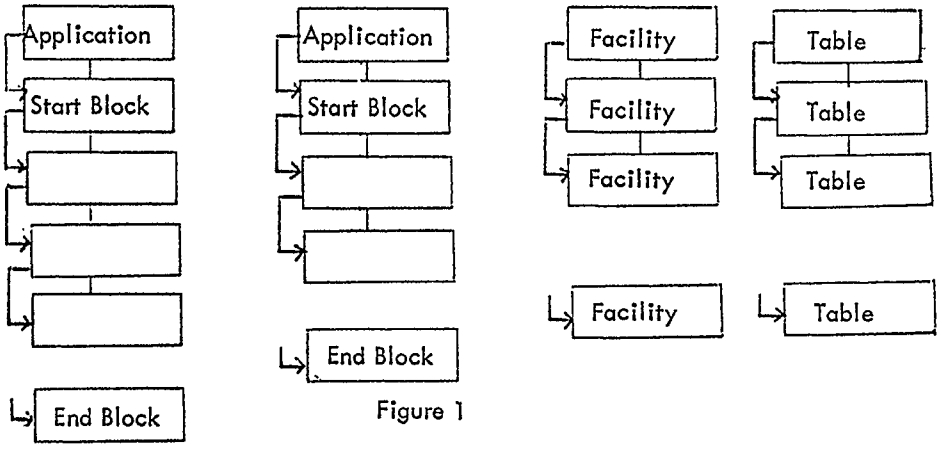


Figure 1

through an interpreter, written in NCR SIMSCRIPT. Each QUIKSIM program consists of one or more applications. An application is defined as a sequence of blocks, beginning with a START block and ending with an END block, which simulates some part of a system. A system may also contain various types of entities which are used to analyze the traffic in the system. Such entities might include processing entities such as facilities, storage and switches, data collection entities such as tables and computational entities such as functions. These features of QUIKSIM are similar to those found in other block type simulation languages.

Each block and entity in a QUIKSIM program is represented internally in the computer as a temporary entity. When an input record containing a block or entity description is read, the QUIKSIM interpreter creates an appropriate size record, containing all the information about the block or entity. The temporary entities are then linked together by filing them in an appropriate list. Each application consists of a list containing all the blocks in the application. Each type of QUIKSIM entity is also filed in its own list, i.e., there is a list of storages, a list of tables, etc. Internally, a QUIKSIM program might appear as in Figure 1.

The use of temporary entities instead of fixed sized arrays for storing data has both advantages and disadvantages. On the plus side, the available memory space is allocated dynamically, as needed. This feature results in an efficient allocation of memory space, maximizing the size of program that can be run. A second feature, which is potentially quite valuable, is the ability to insert or delete new blocks or entities on a list while a program is running. This ability would be an aid in on line debugging as well as enabling the user to run similar programs consecutively without reloading his basic program each time. (This feature is not currently implemented in the QUIKSIM interpreter.) On the minus side, the user may pay a penalty in run time if there is a need to perform list searches to find an entity or block whose address is unknown. In order to reduce this problem, the QUIKSIM interpreter contains an array of addresses so that items on a list may be found more quickly.

Each application creates its own transactions, called jobs. These jobs flow through an application, executing each block in sequence, unless otherwise diverted. A temporary entity is created for each job in the START block of an application and destroyed in the END block of the application.

III. The QUIKSIM Interpreter

The QUIKSIM Interpreter is written in NCR SIMSCRIPT for an NCR 315 RMC with an 80K memory (20K 48 bit words). It consists of several routines. The main routine is a driver which simulates the flow of a job through an application. This driver consists of a sequence of calls to routines which simulate the various blocks in a system, each block being simulated by its own routine. In addition, there is an input and setup routine which reads the input cards, sets up a temporary entity for each input record in the user's program and files the entities in the appropriate lists. The QUIKSIM Interpreter also makes use of the SIMSCRIPT driver, a simulated time clock which drives the system from event to event.

With respect to the block routines, the QUIKSIM Interpreter is modular in design. A block routine may be added, deleted or altered without changing any of the other block routines. This modularity feature is a major objective of the QUIKSIM language, because it allows the user to expand QUIKSIM by writing his own block routines in SIMSCRIPT or FORTRAN. The user causes his routine to be executed by including an input record with an appropriate block name in an application. As a simple example, consider a three block application with the blocks:

START
USER 1
END

After execution of the START block, a job flows to the block USER 1. This block causes the execution of a user-written SIMSCRIPT or FORTRAN subroutine called USER 1. Following the subroutine execution, the job terminates in the END block.

The use of user-written routines may be as extensive as the

user desires. Although the QUIKSIM blocks are designed to be sufficient for most simulation models, the user may re-write any or all of the block routines to suit his purposes. In an extreme case, the user might have a QUIKSIM program in which only the input and driver routines are retained with the user writing all the block routines. When used in this manner, QUIKSIM becomes similar to the simulation language GASP.

IV. Features of the QUIKSIM Language

The preceding sections have indicated some of the considerations which led to the development of the QUIKSIM language. In this section, some of the details of the language are discussed. Many of the features of QUIKSIM are similar to those found in other block structured languages. Hence, this section will concentrate on the differences between QUIKSIM and the other languages.

A. Blocks

QUIKSIM contains 27 blocks, not including any that the user might write. They are:

1. START
2. END
3. PRINT
4. REPORT
5. TRACE
6. ENDTRACE
7. RECLASS
8. CALL
9. RETURN
10. ACTIVATE
11. REPEAT
12. LOOP
13. GO TO
14. ADVANCE
15. SEIZE
16. RELEASE
17. PRIORITY
18. PREEMPT
19. ASSIGN
20. VARIABLE
21. EXECUTE
22. ENTER
23. LEAVE
24. TABULATE
25. WAIT FOR
26. SEND TO
27. COMPARE

The first six blocks are fairly standard. QUIKSIM has a complete range of reports including block count reports, a queue report, a facility and storage utilization report and any table reports that the user may desire. The tracing feature allows the user to monitor the flow of jobs through the system. One feature of QUIKSIM is that reporting and tracing can be controlled on line, through the use of console sense switches.

The seventh block, RECLASS, is used to group the user's processing entities (facilities, storages, and switches) into classes. This feature enables the user to search partial lists of these entities.

Blocks 8 and 9, CALL and RETURN, are used with QUIKSIM routines. A routine is the QUIKSIM equivalent of the subroutine found in other languages. It is a sequence of blocks which is executed when called by a CALL block. The RETURN block causes a job to return to the calling application or routine. Routines have been widely used in QUIKSIM models and this feature is a highly desirable part of the language.

Block 10, ACTIVATE, causes a job to begin flowing through an application. Certain types of applications, called dependent applications, do not generate their own jobs. A job is generated for these applications only when an ACTIVATE block is executed. This block is frequently used in systems where parallel transactions are performing activities in different parts of the system at the same time.

Blocks 11, 12, and 13, REPEAT, LOOP and GO TO, are similar to their FORTRAN counterparts - the DO loop and GO TO statement.

Block 14, ADVANCE, is a standard delaying block. A job enters an ADVANCE block and is delayed until its block departure time. At that time, it continues its flow through its application.

Blocks 15 and 16, SEIZE and RELEASE, are used in connection with facilities. Their use in QUIKSIM is slightly different than their use in other languages. QUIKSIM assumes that each processing entity in a system has its own queue. Thus, the QUIKSIM SEIZE block combines the functions of queuing and seizing in one block. Because of this feature, the SEIZE block does not deny entry to any job which attempts to enter it. Instead, if the facility is busy, the job is queued at the facility and program control is passed to the SIMSCRIPT driver for execution of the next event. The execution of a RELEASE block is then required before a queued transaction can seize a facility.

The PRIORITY block assigns a priority to a job. This priority is used in ranked queues and in the preemption of a facility. All jobs which do not enter a PRIORITY block are assumed to have the same priority.

Block number 18, the PREEMPT block, simulates the preemption of a facility. All preemption is based on priority. If a job currently using a facility has a lower priority than a job attempting to preempt a facility, and the facility is not in preempt status, the current user will be preempted and filed in the queue at the facility.

Blocks 19 and 20, ASSIGN and VARIABLE, are used to store data in temporary storage locations. The ASSIGN block stores data in parameters, which are attached to the job record and may be referenced only by a specific job; the VARIABLE block stores data in variables which can be referenced by any job in the system.

Block 21, the EXECUTE block, is used for off line execution of a block or blocks. These blocks are stored in a table which is referenced by the EXECUTE block. This feature allows the user to simulate activities without diverting the flow of a job through an application.

Blocks 22 and 23, ENTER and LEAVE, perform the same operations on storages that SEIZE and RELEASE perform on facilities. They are responsible for the operations of the queues at the storages.

The TABULATE block allows the user to file data in a table which is later printed out in a report. Tables give the user the ability to obtain specific statistics of interest on the behavior of any part of the system.

The blocks 25 and 26, WAIT FOR and SEND TO, are fairly general blocks which can be used for a variety of purposes. Their inclusion in QUIKSIM was originally inspired by the need to simulate the transmission of signals in computer simulation models. These blocks operate through an entity called a switch which is a multi-level indicator containing a value. A job which arrives at a switch also contains a value. If the two values match, the job flows past the switch; if not, the job is queued. The WAIT FOR block assigns a value to the job and causes it to be queued, if the job values does not equal the switch value. The SEND TO block sends a value to a switch and examines the queue at the switch to determine if anyone is waiting for the value.

A switch may be used for a number of purposes. It may be used as a logic switch, taking on the values 0 or 1 exclusively. It may be used as a queue, in which case the WAIT FOR and SEND TO blocks operate in a manner similar to the QUEUE and DEPART blocks in other languages. It may also be used as a filter, allowing certain jobs to pass through but blocking others. Finally, it may be used as an indefinite time delay block in which jobs are delayed until the execution of some activity in another part of the system which causes their release.

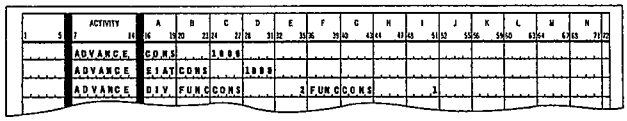
The final QUIKSIM block is a COMPARE block. This block works like the three way IF statement in FORTRAN or SIMSCRIPT, transferring control to one of three blocks depending on the relative magnitude of two values.

The purpose of this brief discussion of the blocks in QUIKSIM is to give the reader some idea of the types of programs which can be run without the user having to write his own SIMSCRIPT or FORTRAN routines. For most simulations, the QUIKSIM blocks should be adequate. However, all the blocks involve certain assumptions about the manner in which activities are simulated and, to the extent that these assumptions don't correspond to the user's system, the ability to reprogram the QUIKSIM blocks becomes valuable.

B. Block Arguments

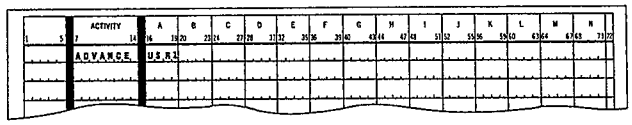
Each block simulates the performance of some activity. For most activities, it is not sufficient to merely specify that the activity is taking place - it is also necessary to specify the manner in which it is taking place. In a SEIZE or RELEASE block, for example, the user must specify which facility is being seized or released. Or, as another example, in an ADVANCE block, the user must specify the amount of time a job is to be delayed at the block. The user performs this specification by using an argument or arguments. A QUIK-

SIM block argument has a fairly well defined form. It consists of one or more operations, specified in four column alphabetic fields, and one or more integer values related to the operations, specified in eight column fields. Some examples below, illustrated in connection with an ADVANCE block, should clarify the nature of arguments.



In the first ADVANCE block, the argument takes the integer 1000 and uses it as a constant. In the second example, the value of the argument is given from the QUIKSIM function EIAT which itself has an argument, the integer constant 1000. EIAT generates values from an exponential distribution with a specified mean. In this case, the mean is 1000 and so the value of the argument will be a random variable from an exponential population with mean 1000. The final example gives an illustration of the use of an arithmetic operation in a block argument. The argument gets values from user functions 1 and 2 (FUNC specifies a user function) and then divides the value of function 2 by the value of function 1.

The structure of the QUIKSIM block arguments is designed to be straightforward. The rules for their use are few and mainly specify which operations are followed by an integer and which are followed by another operation. From the point of view of the interpreter, input is fairly efficient in terms of both run time and memory requirements. All alpha fields are directly converted to an integer code and the code is stored in the appropriate temporary entity; all integer fields, being right adjusted, can be stored in an entity as read. Each type of argument format gives rise to a specific execution code. (Given the QUIKSIM rules and the 72 column limit on an input card, there are about 40 different argument formats available to the user.) The execution code allows rapid execution of fairly complex arguments. The major defect of the argument structure is that it is inconvenient to use in performing complicated arithmetic operations. The user is, in effect, restricted to no more than two arithmetic operations per block. This restriction does not mean that a user can't perform complicated arithmetic; it simply means that the user must store intermediate calculations and use several blocks to calculate a desired value, much as he would do if he were writing a program in assembly language. At times, it may be more convenient for the user to write a FORTRAN or SIMSCRIPT subroutine to generate an argument value and QUIKSIM gives this capability. In a block such as



the argument to the ADVANCE block will be the value generated from the user written subroutine USR1.

V. Example

The preceding three sections have illustrated some of the features of the QUIKSIM language.

In this section, these features are illustrated in greater detail in an example. This problem is a simplified model of a communications system. The configuration of the system is as diagrammed below

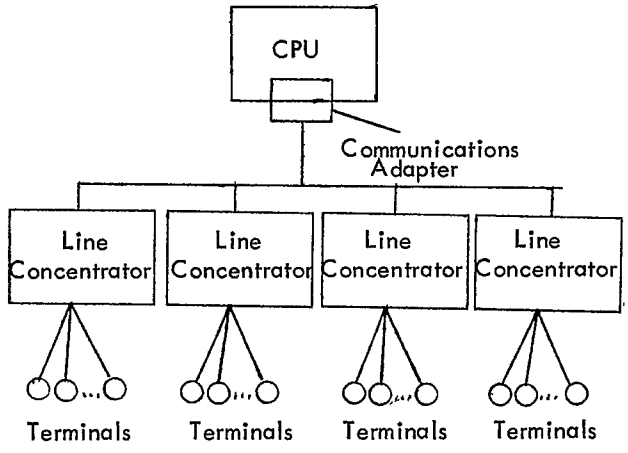


Figure 2

Each line concentrator is attached to a certain number of remote terminals. Each concentrator, in turn, is connected by a high order line to the CPU. To determine if there is a request for service, the CPU polls each concentrator in sequence. When it finds a concentrator with a request, the CPU sets up the line and allows the request to begin processing. When the request has finished processing and an output message has been sent, the CPU resumes polling. The following pages present flowcharts and QUIKSIM coding. (Figures 3-6). Figure 3 is a flowchart of the operation of a concentrator. Figure 4 is a flowchart of the poll mechanism of the CPU. Figure 5 is a flowchart of the processing activities connected with servicing a request and polling.

The QUIKSIM coding is designed to be reasonably self-explanatory. The operation of each concentrator has been modeled separately. This modular technique has two advantages - first, it allows the user to add or remove concentrators from the configuration of the system without any major reprogramming and second, it allows the user to alter the arrival rate to one or more of the concentrators without making any major changes. Application RPT initializes two variables which relate to the operation of the poll table and schedules a report. Application POLL simulates the processing of a request and polling. The three routines - LINE, EX-1 and APL 1 - simulate the seizure and utilization of the high order line and the CPU. The final two applications, ENV 1 and ENV 2, simulate the utilization of the CPU by requests which are generated from outside the system.

The results which can be obtained from a program of this type may be useful in aiding in the design of systems. Specifically, the user can determine the effect of adding or removing concentrators from the high order line on the utilization of the line and the CPU. He can determine the effect of changing the arrival streams to the concentrators (i.e., changing the number of terminals per concentrator) on device utilization. He can also determine the effects of changing the assumptions about device utilization times. These assumptions are imbedded in the three user defined functions, which specify utilization times and the probabilities associated with each time. In all these cases, the QUIKSIM language is designed to allow a user to alter his program quickly and easily.

VI. Required Resources

The development of the final version of the QUIKSIM interpreter required approximately one man year. This time included the production of a preliminary version of a user's manual.

An early version of the interpreter and manual was completed in about six months; however, this version contained a number of undesirable features and was abandoned. In developing a language of this type, it is necessary to estimate what features a potential user is likely to find desirable and to include those features in the language. Balanced against the desire to include a large number of user oriented features are the dual considerations of run times and memory requirements for users' programs. The interpreter which was ultimately produced attempted to balance all of these factors.

The QUIKSIM interpreter is written for an NCR 315 RMC with 80K of memory. The memory requirements for the FORTRAN system, QUIKSIM interpreter object program, and all other FORTRAN and SIMSCRIPT system requirements is approximately 45K. The user has the remaining 35K to use for his jobs, blocks and entities. While the exact size of program which a user may run depends on the structure of the system being modeled, a program with in excess of 500 blocks would probably fit the available memory.

Experience with the language has indicated that programming and turn around time is fairly rapid. Large models (in excess of 200 blocks) have been written, debugged and run in about a week. This turn around is undoubtedly more rapid than would have been obtained using SIMSCRIPT; however, run times have been quite lengthy. The example given in Section V required approximately one hour of computer time to simulate 10 minutes of real time. During the 10 simulated minutes, approximately 33,500 jobs were processed in all applications. Part of the run time problem is due to the relatively slow speed of the computer being

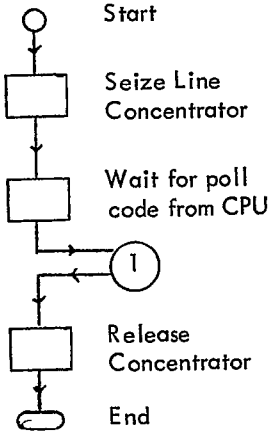


Figure 3

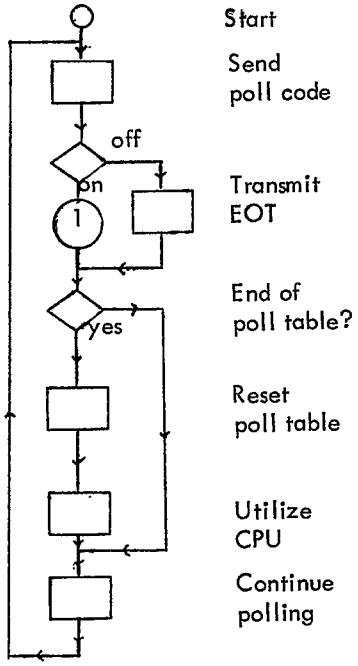


Figure 4

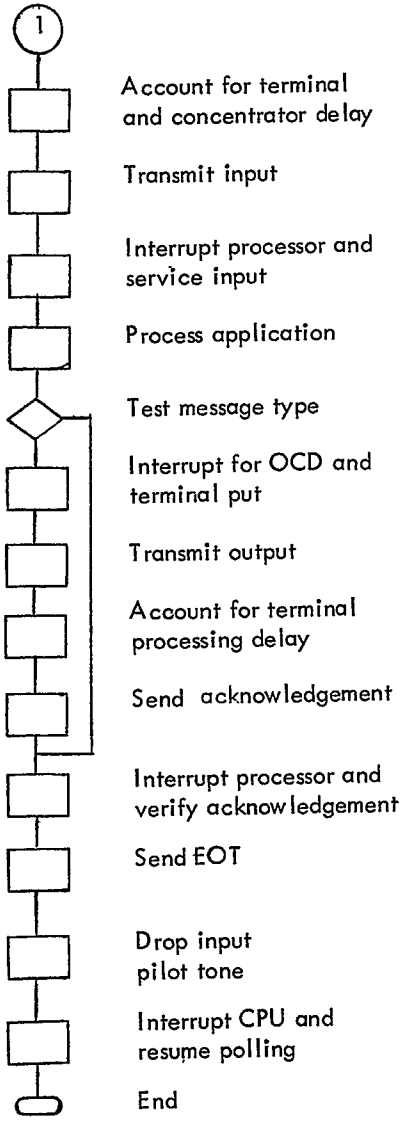


Figure 5

```
C-----SET UP OF FACILITIES,SWITCH, AND RUN LABEL                    COMMENT
    1 FACILITY CPU FIFO                                            FACILITY
    2 FACILITY HOL FIFO                                            FACILITY
    3 FACILITY CON1FIFO                                            FACILITY
    4 FACILITY CON2FIFO                                            FACILITY
    5 FACILITY CON3FIFO                                            FACILITY
    6 FACILITY CON4FIFO                                            FACILITY
    1 SWITCH   QUE              0                                  SWITCH
      LABEL    CONCENTRATOR POLLING                                LABEL
      END                                                          FACS


C-----SET UP OF TABLES                                             COMMENT
    1 TABLE     100000   50000                                     TABLE
 TOTAL SERVICE TIME
    2 TABLE     100000  100000                                     TABLE
 WAIT TIMES FOR HOL
      END                                                          TABLE


C-----SET UP OF FUNCTIONS                                          COMMENT
    1 FUNCTION DISC      3RNI CONS       0CONS      999           FUNCTION
                  442  301200     934   53666      999  32000     FUNCTION
    2 FUNCTION DISC      3RNI CONS       0CONS      999           FUNCTION
                  442   10000     934    8000      999   17000    FUNCTION
    3 FUNCTION DISC      3RNI CONS       0CONS      999           FUNCTION
                  818  209900     936  245000      999  145200    FUNCTION
      END                                                          FUNCTION


C-----APPLICATION TO SIMULATE ARRIVALS AT CON1                     CON1
    1 START      10000       1EIAT 3333333              CON1     0 CON1
    2 SEIZE    CON1                                               CON1
C-----IF CONCENTRATOR HAS BEEN SEIZED, ENTER QUEUE TO WAIT FOR HOL CON1
    3 WAIT FOR QUE         CONS       3                           CON1
C-----TABULATE WAITING TIME FOR HOL                                CON1
    4 TABULATE CONS       2TRAN                                    CON1
C-----WAIT FOR END OF PROCESSING BEFORE RELEASING CONCENTRATOR     CON1
    5 WAIT FOR QUE         CONS       1                           CON1
    6 RELEASE  CON1                                               CON1
    7 END                                                         CON1


C-----APPLICATION TO SIMULATE ARRIVALS AT CON2                     CON2
    1 START      10000       1EIAT 3333333              CON2     10 CON2
    2 SEIZE    CON2                                               CON2
C-----IF CONCENTRATOR HAS BEEN SEIZED, ENTER QUEUE TO WAIT FOR HOL CON2
    3 WAIT FOR QUE         CONS                                   CON2
C-----TABULATE WAITING TIME FOR HOL                                CON2
    4 TABULATE CONS       2TRAN                                    CON2
C-----WAIT FOR END OF PROCESSING BEFORE RELEASING CONCENTRATOR     CON2
    5 WAIT FOR QUE         CONS       1                           CON2
    6 RELEASE  CON2                                               CON2
    7 END                                                         CON2


C-----APPLICATION TO SIMULATE ARRIVALS AT CON3                     CON3
    1 START      10000       1EIAT 3333333              CON3     20 CON3
    2 SEIZE    CON3                                               CON3
C-----IF CONCENTRATOR HAS BEEN SEIZED, ENTER QUEUE TO WAIT FOR HOL CON3
    3 WAIT FOR QUE         CONS       5                           CON3
C-----TABULATE WAITING TIME FOR HOL                                CON3
    4 TABULATE CONS       2TRAN                                    CON3
C-----WAIT FOR END OF PROCESSING BEFORE RELEASING CONCENTRATOR     CON3
    5 WAIT FOR QUE         CONS       1                           CON3
    6 RELEASE  CON3                                               CON3
    7 END                                                         CON3


C-----APPLICATION TO SIMULATE ARRIVALS AT CON4                     CON4
    1 START      10000       1EIAT 3333333              CON4     30 CON4
    2 SEIZE    CON4                                               CON4
C-----IF CONCENTRATOR HAS BEEN SEIZED, ENTER QUEUE TO WAIT FOR HOL CON4
    3 WAIT FOR QUE         CONS       6                           CON4
C-----TABULATE WAITING TIME FOR HOL                                CON4
    4 TABULATE CONS       2TRAN                                    CON4
C-----WAIT FOR END OF PROCESSING BEFORE RELEASING CONCENTRATOR     CON4
    5 WAIT FOR QUE         CONS       1                           CON4
    6 RELEASE  CON4                                               CON4
    7 END                                                         CON4


C-----APPLICATION TO CONTROL REPORTING AND INITIALIZE VARIABLES    RPT
    1 START         1       1EIAT     0                RPT       RPT
C-----VARIABLE 1 EQUALS NUMBER OF FIRST CONCENTRATOR               RPT
    2 VARIABLE CONS       1CONS       3                           RPT
C-----VARIABLE 2 EQUALS NUMBER OF LAST CONCENTRATOR,PLUS 1         RPT
    3 VARIABLE CONS       2CONS       7                           RPT
C-----SCHEDULE REPORT AT 600 SECONDS                               RPT
    4 ADVANCE  MULTCONS60000000CONS       10                       RPT
    5 REPORT          3                                           RPT
    6 END                                                         RPT
```

Figure 6

```
C*****APPLICATION TO SIMULATE POLLING AND UTILIZATION OF CPU                    POLL
 1000 START      10000        2CONS       0                                    POLL
 1005 ASSIGN   CONS       3VAR CONS       2                                    POLL
C*****SEND POLL CODE                                                           POLL
 1010 ASSIGN   CONS       2CONS   28666                                        POLL
 1020 CALL     LINE                                                            POLL
C*****CHECK IF POLLED CONCENTRATOR HAS A REQUEST FOR SERVICE                   POLL
 1040 COMPARE  1390105013902CONS       1UFLGVAR CONS       1                  POLL
C*****POLLED CONCENTRATOR HAS A REQUEST                                        POLL
 1050 SEND TO  QUE          VAR CONS       1                                   POLL
C*****SERVICE REQUEST                                                          POLL
C*****ACCOUNT FOR TERMINAL AND CONCENTRATOR DELAY                              POLL
 1120 ADVANCE  CONS   5000                                                     POLL
C*****SELECT NUMBER OF CHARACTERS AND TRANSMIT INPUT                           POLL
 1130 ASSIGN   CONS       2FUNCCONS       1                                    POLL
 1140 CALL     LINE                                                            POLL
C*****INTERRUPT PROCESSOR AND SERVICE                                          POLL
 1150 ASSIGN   CONS       2CONS    8000                                       POLL
 1200 CALL     EX=1                                                            POLL
C*****SELECT APPLICATION COMPUTE TIME AND REQUEST CPU USAGE                    POLL
 1210 ASSIGN   CONS       2FUNCCONS       2                                    POLL
 1220 CALL     APL1                                                            POLL
C*****DETERMINE TYPE OF MESSAGE                                                POLL
 1230 EXECUTE  CONS       1CONS       1                                       POLL
 1235 GO TO    PARM       10                                                  POLL
C*****OCD AND TERMINAL PUT                                                     POLL
 1240 ASSIGN   CONS       2CONS    6000                                       POLL
 1250 CALL     EX=1                                                            POLL
C*****SELECT NUMBER OF CHARACTERS AND TRANSMIT OUTPUT                          POLL
 1260 ASSIGN   CONS       2FUNCCONS       3                                    POLL
 1270 CALL     LINE                                                            POLL
C*****TERMINAL PROCESSING DELAY                                                POLL
 1280 ADVANCE  CONS   3220                                                     POLL
C*****SEND ACKNOWLEDGEMENT                                                     POLL
 1290 ASSIGN   CONS       2CONS   20333                                        POLL
 1300 CALL     LINE                                                            POLL
C*****INTERRUPT PROCESSOR,VERIFY ACK                                           POLL
 1310 ASSIGN   CONS       2CONS    7000                                       POLL
 1320 CALL     EX=1                                                            POLL
C*****SEND EOT                                                                 POLL
 1330 ASSIGN   CONS       2CONS   20333                                        POLL
 1340 CALL     LINE                                                            POLL
C*****DROP INPUT PILOT TONE                                                    POLL
 1350 ADVANCE  CONS   39222                                                    POLL
C*****INTERRUPT PROCESSOR,RESET INTERVAL TIMER AND RESUME POLLING              POLL
 1360 ASSIGN   CONS       2CONS   1000                                         POLL
 1370 CALL     EX=1                                                            POLL
C*****SEND SIGNAL TO CONCENTRATOR TO INDICATE END OF PROCESSING                POLL
 1375 SEND TO  QUE          CONS       1                                       POLL
 1380 GO TO    CONS    1430                                                    POLL
C*****NO REQUEST FOR SERVICE                                                   POLL
C*****TURN IPT AND IC ON                                                       POLL
 1390 ADVANCE  CONS   25000                                                    POLL
C*****SEND EOT TO CENTRAL                                                      POLL
 1400 ASSIGN   CONS       2CONS   20333                                        POLL
 1410 CALL     LINE                                                            POLL
 1420 ADVANCE  CONS   18000                                                    POLL
C*****CONTINUE POLLING                                                         POLL
C*****TABULATE SERVICE TIME                                                    POLL
 1430 TABULATE CONS       1TRAN                                               POLL
C*****INCREMENT POLL TABLE POINTER                                             POLL
 1440 VARIABLE CONS       1ADD VAR CONS       1CONS       1                  POLL
C*****TEST IF END OF POLL TABLE IS REACHED                                     POLL
 1450 COMPARE  1490146014902PARM       3VAR CONS       1                     POLL
C*****RECYCLE POLL TABLE                                                       POLL
 1460 VARIABLE CONS       1CONS       3                                       POLL
 1470 ASSIGN   CONS       2CONS    6000                                       POLL
 1480 CALL     EX=1                                                            POLL
 1490 END                                                                      POLL


C*****ETABLE TO DETERMINE MESSAGE TYPE                                         ETABLE
    1 ETABLE                                                                   ETABLE
  441 ASSIGN   CONS       10CONS     1310                                      ETABLE
  999 ASSIGN   CONS       10CONS     1240                                      ETABLE
      END                                                                      ETABLE


C*****COMMUNICATION LINE USAGE ROUTINE                                         LINE
    1 ROUTINE  LINE                                                            LINE
    2 SEIZE    HOL                                                             LINE
    3 ADVANCE  PARM       2                                                    LINE
    4 RELEASE  HOL                                                             LINE
    5 RETURN                                                                   LINE
    6 END                                                                      LINE


C*****EXECUTIVE PROCESSING ROUTINE---NON-INTERRUPTABLE CPU UTILIZATION         EX=1
   11 ROUTINE  EX=1                                                            EX=1
   12 PRIORITY CONS     300                                                    EX=1
   13 PREEMPT  CPU                                                             EX=1
   14 ADVANCE  PARM       2                                                    EX=1
   15 RELEASE  CPU                                                             EX=1
   16 RETURN                                                                   EX=1
   17 END                                                                      EX=1


C*****APPLICATION PROCESSING ROUTINE---INTERRUPTABLE CPU UTILIZATION           APL1
   21 ROUTINE  APL1                                                            APL1
   22 SEIZE    CPU                                                             APL1
   23 ADVANCE  PARM       2                                                    APL1
   24 RELEASE  CPU                                                             APL1
```

```
   25 RETURN                                                        APL1
   26 END                                                           APL1


C------EXOGENOUS EXECUTIVE PROCESSING APPLICATION                   ENV1
   31 START      100000         1ELAT   20850          ENV1         ENV1
   32 ASSIGN     CONS           2CONS    8000                       ENV1
   33 CALL       EX=1                                               ENV1
   34 END                                                           ENV1


C------EXOGENOUS APPLICATION PROCESSING APPLICATION                 ENV2
   41 START      100000         1ELAT  833000          ENV2         ENV2
   42 ASSIGN     CONS           2CONS   15000                       ENV2
   43 CALL       APL1                                               ENV2
   44 END                                                           ENV2


   ENDFILE
```

used. A typical SIMSCRIPT job will take about three times as long to run on an NCR 315 RMC as it would on a 7094.

VII. Future Developments

There are a number of potential improvements to QUIKSIM which may be desirable when the interpreter is reprogrammed for a more sophisticated computer system, such as the NCR Century series. At present, all input in a QUIKSIM program is on cards and this feature makes it somewhat inconvenient to make on line changes to a program. If input were from a remote terminal, on line changes could be made very easily since the user could easily insert, delete or change an item on any of the lists which comprise his program. Another desirable improvement would be an expanded ability to display and analyze output. QUIKSIM currently produces a fairly broad range of reports. However, if the user wants to analyze or display his data, he must use some other program. It would be desirable to hook up the QUIKSIM interpreter with a statistical package to allow the user to perform regression analysis, hypothesis testing, etc. without the need to use a separate program. Finally, there may be some desirable changes to SIMSCRIPT which would improve the interface between SIMSCRIPT and QUIKSIM. (In writing the QUIKSIM interpreter, the SIMSCRIPT translator has not been altered in any way.)

VIII. Summary

This paper has described an approach toward the task of producing a general purpose simulation language. The approach has been to utilize a high level simulation language, SIMSCRIPT, to produce an interpreter for an even higher level language, QUIKSIM. A user may use as much or as little of the QUIKSIM system as he desires. At one extreme, he may program entirely in QUIKSIM; at the other extreme, he may program entirely in SIMSCRIPT or FORTRAN, using the QUIKSIM interpreter as a foundation upon which to build his own SIMSCRIPT or FORTRAN program. This approach to a simulation language is designed to provide a maximum of flexibility and programming options to a wide variety of potential users.

Acknowledgment

The author wishes to gratefully acknowledge the many helpful suggestions and criticisms of Mr. K. N. Hayward, Mr. G. W. Dangel, Mr. R. J. Ceci and Mr. S. Wong.