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

CONVERSIM – A teaching simulation language incorporating a conversational model builder

Paul F. Roth
Department of Computer Science and Engineering
Bi-County Center for Engineering
University of South Florida at Sarasota
Sarasota, Florida 34243-2197 U.S.A.

Robert Brown
Computer Science Corporation
4001 Oak Manor Office Park
King George, Virginia 22485 U.S.A.

ABSTRACT

This paper describes the CONVERSIM simulation language. CONVERSIM is a developmental general-purpose, discrete-event language which has been used in the classroom to introduce the use and operation of simulators prior to the introduction of languages such as GPSS and SIMSCRIPT.

The distinctive innovation of CONVERSIM is that model building is implemented by a conversational query language which interacts with the modeler in English questions with prompted, symbolic answers, thus imparting no language jargon or bias, other than such "neutral" terminology as would be used in describing queuing models. The CONVERSIM "world view" is transaction-oriented.

With a modest repertoire of modeling options, CONVERSIM has been found in the classroom to be an asset in illustrating the performance of queuing models and networks, prior to the immersion of the student in the intricacies of standard simulation languages and their various idiosyncrasies. The query system is of sufficient completeness to enable the student to completely describe basic models without the use of a reference manual or other written documentation.

CONVERSIM is written in Pascal to be hosted by a wide variety of DOS-based microcomputers.

INTRODUCTION AND SUMMARY

Since most simulation languages are used less frequently than many other programming languages, the number of proficient practitioners available at any one time is limited. Yet the number of potential users of simulation languages far exceeds the number of proficient practitioners.

This situation helped motivate the following questions:

Was it possible that that ideas such as conversational front-end programs and syntax-directed editors could make simulation available to persons not proficient in a specific simulation language?

Was it possible for a conversational program to front-end a simulation language similar to, say, GPSS?

Could this be implemented for a microcomputer application?

If it was deemed possible to implement these ideas then simulation could possibly be made more accessible to everyone. As it turned out, the answer to all questions was positive. A conversational model-building routine--called "CONVERSE"--for transactional discrete-event models was accomplished. This was mated to a GPSS-like simulation execution program - called "DSIM." The combined package, called "CONVERSIM," was implemented for a DOS machine environment. Although this tool has not been tested in the simulation applications context, it has found utility in the classroom, where, as a teaching tool, it provides the initial programming environment for modeling and simulation, prior to introduction of a standard language such as SIMSCRIPT II.5 or GPSS.

CONVERSIM was recently used in concurrent, graduate classes in simulation at the University of South Florida, its first exposure outside of the development environment. It was employed successfully in sequence after the use of a manual simulator, "Hyposim," and before the introduction of GPSS, and later SIMSCRIPT II.5. As a result of this use, certain adjustments, primarily in user-friendly augmentations to CONVERSE, were implemented. The execution module, DSIM, performed flawlessly. "Beta-testing" in other teaching environments is being considered. Appropriateness for use in an applications context has not been determined.

OVERVIEW OF CONVERSIM

CONVERSIM is composed of two program modules and an editor, implemented in Pascal for a DOS microcomputer. The module, CONVERSE, is a conversational model-builder tool which interacts with the programmer through a series of screens, translating the output of the query process into an input file for the simulation execution module, DSIM, and a text file to be used by the CONVERSIM editing routine. DSIM executes the simulation: it is a generic discrete-event simulator with a "transaction" world-view. Its input data are comprised primarily of integers, located in various line and columnar formatted records. DSIM produces an autonomous simulation data output which contains no optional reports. Figure 1 depicts the program modules and their data flow.

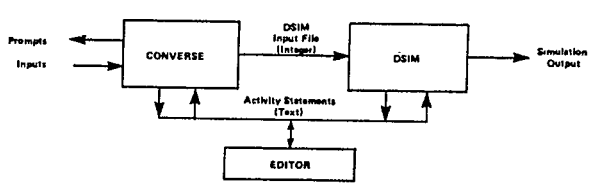


FIGURE 1CONVERSIM MODULES AND INFORMATION FLOW

Model Building Language--CONVERSE

Design goals

The primary design goal for the model-building routine was to implement a stand-alone program which, when executed, would produce as output, an input file

for a simulation execution program. The model builder was to provide a sequence of queries which would prompt responses which would be translated to the output file. It was not intended that the model-building capability be exhaustive, but rather usable in the synthesis of basic waiting-line systems of modest scale. This design goal did not, however, preclude the expansion of the options should such be desired in future incarnations.

The second design goal was to neutralize the terminology incorporated in the query language, so as to avoid imparting any application or special-use bias, which is sometimes encountered in general-purpose simulation languages. This goal has led to what might be considered a "generic" terminology, such as that used in queuing theory.

A third design goal was to incorporate modularity in both the query and execution packages so that options and improvements could be added with facility.

The model-structuring reflected in CONVERSIM is closely related to that of "BOSS," a Burroughs discrete-event language of the 1970's, well-known to the authors, which was felt to possess both a neutral terminology and high modularity. As a result, the simulation objects are also "BOSS-like:" that is, strongly typed.

Generic Queuing Model and Terminology

The following statements characterize the basic structural aspects of a CONVERSE model:

1. The system owns server object types, called "Resources," which are structured as multiservers. Each resource type is user-named and represents a fixed pool of from zero to n interchangeable units. Units may be transferred pool-to-pool. Each pool owns a queue, with service rule selectable, default FIFO. Resource allocation is referenced by type; individual units are not addressable.

2. Dynamic object types, called "Users," are transaction-oriented entities. Each type is programmer-named. There may be multiple types in a model. Each type carries with it a specification for generation of its replicative instances (arrivals) and, optionally, its maximum number of instances.

3. User types comprise sequences of activities; each activity is some operational event such as resource allocation, time advance, etc.

4. Queuing is implicitly implemented with each resource request, and queue-servicing with each release.

5. The system owns reference objects, called "Parameters," global system variables to which Users have access. Users may own Generation Parameters, or locally-defined variables, to which each user instance has access alone.

CONVERSE Query Sequence

The sequential flow of a CONVERSIM model-building session, imparted by the CONVERSE program, reveals a highly-modular, top-down approach. The top-down sequence of model description can be expressed as a flow-chart, Figure 2. Note that the double-lined boxes represent complex, repetitive query routines, several of which will be illustrated in detail.

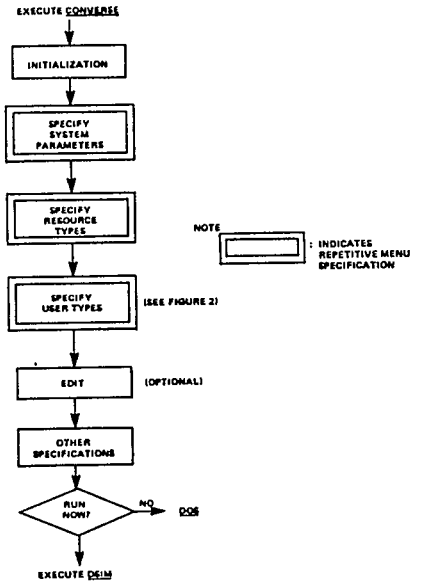


FIGURE 2 CONVERSIM MAIN MODEL-BUILDING ROUTINE

Initialization comprises such preliminary operations as naming the model.

A query routine successively identifies system-level parameters. Memory space is reserved and parameters are given programmer-assigned names.

Resources are initialized via a repetitive query loop which prompts for identification of each resource type by type name, then specifies the initial number of units assigned to a type.

The user-type-specification query sequence is quite complicated and is described under its independent heading.

At this point in CONVERSE, all the model objects have been defined. The option is then given to review the user activity declarations, which if accepted, brings the process into edit mode. The declarations are one-line text word-descriptions of each specified user activity, automatically screen-printed by CONVERSE to echo the model. The editor displays the declarations, initiates the editing operation, and when acceptable, returns to the CONVERSE query stream. What has happened internally at this point is that the translation to DSIM input code is completed and the code saved; also the text declarations are saved for future editing.

The last group of specifications in a CONVERSE query routine include requesting optional snapshot data outputs and setting an optional maximum run time.

Finally, CONVERSE prompts for "Run Now?" If accepted, the DSIM program execution is initiated, otherwise control is returned to DOS.

User Type Specification Query Sequence

A detailed user specification flow chart is shown in Figure 3.

The routine is entered immediately following specification of system resources. The flow shown in Figure 3 is repetitively entered from Point A until no more user types are to be described.

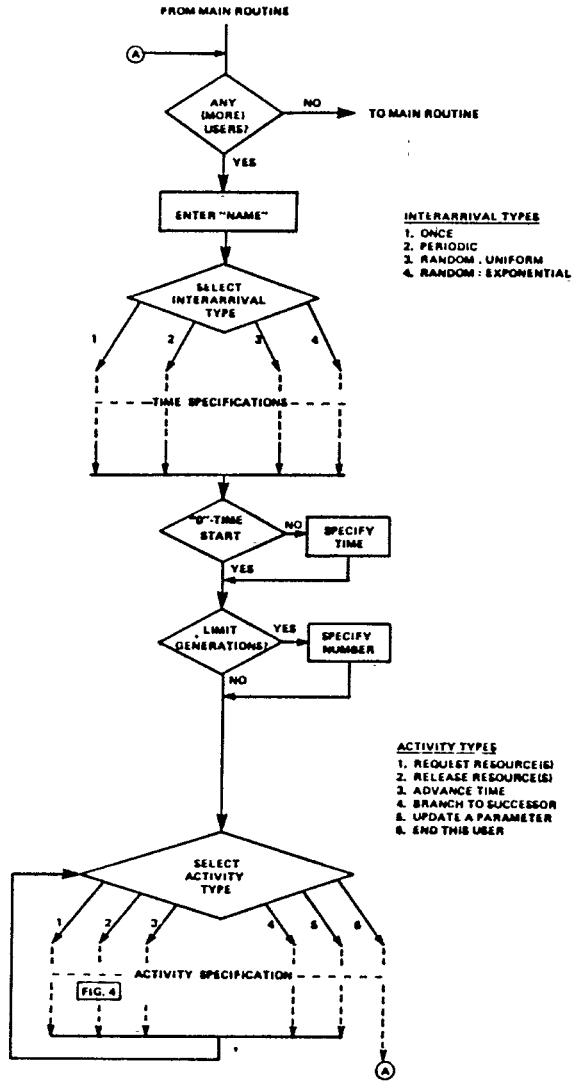


FIGURE 3 CONVERSIM USER TYPE SPECIFICATION ROUTINE

Initially, the user type is given a name, say "Customer," which is referenced during the building phase and in the data output.

"Customer" is then provided a generation, or interarrival type, selected from four choices:

1. One instance
2. Periodic arrivals
3. Uniform random interarrival
4. Negative exponential random interarrivals

Note that the complete set of specifying details is not shown here; note also that this modular form of program construction can accommodate future selections.

Two type-specifications follow interarrival selection:

1. Optional delayed generation starting time (default is "0")
2. Optional limit to the number of instances of this user (default is a maximum quantity).

The routine which structures each user type's activities can be approximated by the following pseudocode:

```
Procedure Usertype
     Begin
     Until MODELEND
          Begin
          Until USEREND
               Begin
                    Specify activity
                    If last activity then USEREND
               End
          If last user then MODELEND
          End
     End.
```

To restate, a User or transaction is a system object which is composed of a series, or network, of activities which are executed in some sequence. Following is a menu of CONVERSE activity types:

A. Request resource(s)
B. Release resource(s)
C. Advance time
D. Branching specification
E. Update a parameter
F. Terminate the User.

For instance, a primitive queuing user would require, as a minimum, Activity Types A,C,B.

Activity Type A is defined through a sequence of user prompts which serve, in succession, to request and allocate quantities of resource units from the available resource types. This process can be represented by the pseudocode:

```
Process resreq
     Begin
          Until ENDRESOURCE
          Begin
               Specify type name
               Specify quantity of units
               If last type then ENDRESOURCE
          End
     End.
```

Thus each request from a different pool is made with a subsequent series of prompts. This results in a sequential, piecemeal acquisition during execution.

Activity Type B, "Release Resource(s)," can be represented by the pseudocode segment:

```
Procedure Resrel
     Begin
          Until ENDRELEASE
          Begin
               Name type
               Specify quantity of units
               If transfer release then give new
                 type name
               Select discipline
                    Case: FIFO:
                          LIFO;
                          etc.;
               If last release then ENDRELEASE
          End
     End.
```

Activity Type C, "Advance Time," is approximated by:

```
Procedure timadv
     Begin
     Case:  If service time deterministic,
              enter value;
            If service time uniform, enter
              minimum, maximum values;
            If service time negative
              exponential, enter mean
     End.
```

Note that this modular structure provides for the adaptation to other selection rules.

Activity Type D, "Branch specification," involves selecting a successor activity. Its use is optional since the default succession connects activities in succession in the order they are specified in the program. It, as is Type E, "Update Parameter," very complex, having many choices and formats, and thus will not be elaborated upon here.

Activity Type F, "Terminate User," is used to signify that specification of a particular user is complete,and returns control to the query routine.

Of particular interest is Type B, "Release Resource(s)" (Figure 4), which incorporates two innovations not usually seen in other simulation languages:

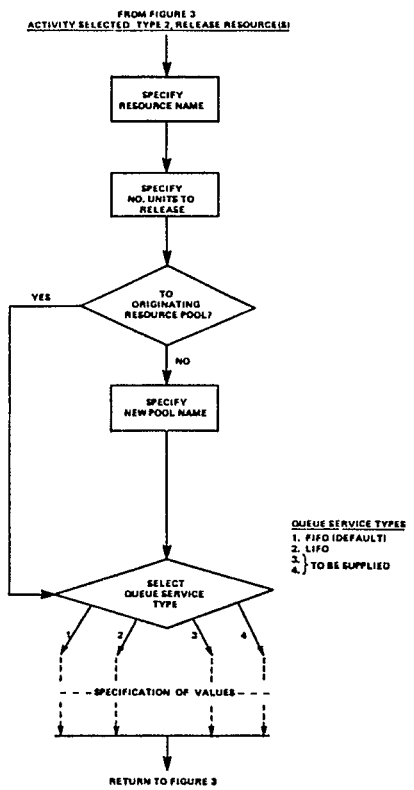


FIGURE 4 CONVERSIM RESOURCE(S) RELEASE ACTIVITY ROUTINE

Resource pool transfer
User-determined queue service discipline

The ability to transfer resource units between pools is implemented in CONVERSIM to enhance programming facility when confronted with the requirement to move resources to a pool other than from which it was acquired. Typically, this can occur where there is some spatial distance between pools, for instance in a transportation simulation where locomotives or tugboats belong to different pools at the beginning and end of a transaction or operation. Although this can be effected with some difficulty in many general-purpose simulation languages, it is explicitly and easily accommodated in CONVERSIM.

Allowing each user type, at each resource release activity, to select the queue service discipline to be employed, constitutes another innovative feature of CONVERSIM. Noting that each resource queue is treated with FIFO discipline as a default condition, a particular user type imparts its copies with the facility to modify that default at a given point in the transaction. The situational justification for this innovation is not as direct as the previous one, however.

Input and editing

CONVERSE input is accomplished as a sequence of prompts, conveyed by screen to the model builder. All of the screens have a response field which can vary from a single character to a string. Many responses are pre-initialized to a default, requiring a response only when exception is taken.

During the user-structuring phase of CONVERSE, the text description of each activity is displayed when specification is complete. A prompt to verify correctness is then given. If affirmative, CONVERSE cycles to the next activity; if negative, CONVERSE repeats the prompts for the current activity.

To illustrate the query process, a set of screens for a "Request Resource(s)" activity is shown.

```
SYSTEM USER: TRANS                    MODEL: MM1

                    CONVERSIM

          What does TRANS        do next?

              1. Request a resource
              2. Release a resource
              3. Advance time
              4. Select a branch
              5. Update a parameter
              6. End this system user

                          1
```

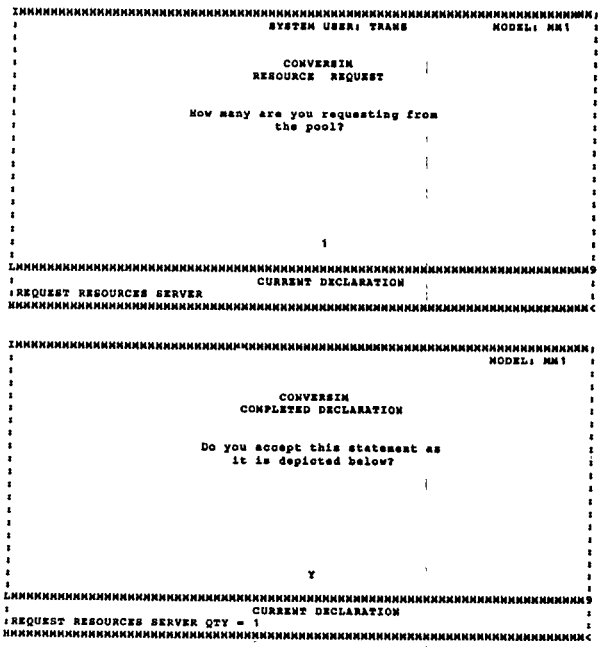
```
SYSTEM USER: TRANS                    MODEL: MM1

                    CONVERSIM
                RESOURCE  REQUEST

          What is the name of the resource
             that you wish to request?

                    SERVER

                CURRENT DECLARATION
REQUEST RESOURCES
```



When the model specification cycle is done, CONVERSE prompts to determine if the entire model is to be reviewed for correctness. If affirmative, the program transitions to edit mode. At the end of edit mode, the modeler is again prompted for a return to editing, or to go on.

Edit mode may be entered from two points: from the end of the model query routine, or when starting a CONVERSE session and accessing a saved CONVERSE output file. In edit mode, activity statements are displayed 10-at-a-time, and the modeler is prompted to select a line to edit and then presented with a menu of line-edit command types (delete, move, modify, etc.). Upon completion of editing in this manner, a prompt is issued to repeat the cycle, or to go on.

The CONVERSE activity-statements for a basic M/M/1 model are shown in Figure 7.

SIMULATION LANGUAGE - DSIM

Design Background

DSIM, a database record SIMulation language, was conceived and constructed to be an easy language for building small and medium simulations that could be hosted by a DOS microcomputer. It is conceptually a descendant of BOSS, in that its input database consists of a series of fixed-format, integer files which can be structured, via a strongly-typed syntax, to represent a wide range of model constructs. At this point, DSIM's connection with BOSS ends. This is because the translation of this type of database, left entirely to the user, while efficient in the same sense as assembly language, is not a viable option in today's software practice.

However, the concept of a database-driven language using a series of numbers as the internal representation can be effective if the user is always presented the translated representation. Integer numbers are easier to store than text, and a language incorporating such a database is efficient for storage and manipulation as a simulation language.

It should be noted that, after careful design consideration and planning, it was determined that a conversational model building language could not be achieved until after a simulation execution module was developed. Therefore, DSIM was created first.

It should also be noted that, while the main purpose of DSIM is to execute, as input, the CONVERSE output file, complete accessibility to the DSIM input file is available to the user for editing, modification, and perhaps even generation of input code independent of the CONVERSE process.

The DSIM Input Data Base

The following sections describe the main record formats for inputting model-specification records to DSIM, the simulator. These formats, many corresponding to activity types, are automatically created by CONVERSIM.

All records contain multiple fields, some of which are optional. All numeric fields are left-justified and all alphanumeric fields are free format within the prescribed lengths.

Comment Record. A comment record may be placed into the input source file wherever it is appropriate.

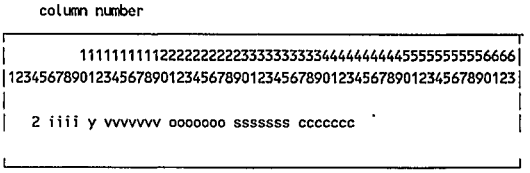
```
column number
        1111111111222222222233333333344444444455555555556666
12345678901234567890123456789012345678901234567890123456789012345
. description of text used as a comment (75 chars)
```

Time Advance Activity Type Records. Time advance activities advance the system clock. The amount of time advanced depends on the distribution type and value.

```
column number
        1111111111222222222233333333344444444455555555556666
12345678901234567890123456789012345678901234567890123456789012345
1 iiii y vvvvvvv ooooooo
```

WHERE: iiii = label number (optional)
y = distribution types
1 = use vvvvvvv immediately to advance time
2 = use random uniform distribution between vvvvvvv (min) and ooooooo (max) for clock advancement
3 = use random exponential distribution with mean of vvvvvvv for clock advancement
vvvvvvv = value for distribution types
ooooooo = max value for uniform distribution (optional)

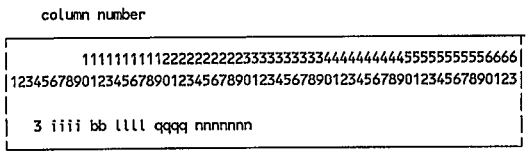
Generation Record. The generation record causes the system to generate a new instance into the system periodically based upon the distribution types and the limits found on the generation record. Each new generation contains 10 generation parameters, numbered 0 through 9, which are local to the individual generation.



WHERE: iiii = label number (optional)
y = generation interval types
1 = use vvvvvvv as the amount of time between generations
2 = use random uniform distribution between vvvvvvv (min) and ooooooo (max) for the time between generations
3 = use random exponential distribution with mean of vvvvvvv for the time between generations

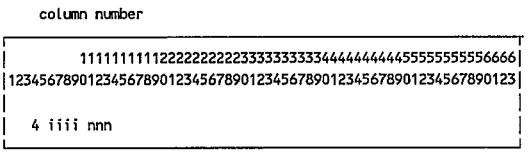
vvvvvvv = value for time interval types
ooooooo = max value for uniform distribution (optional)
sssssss = start time for the first generation (optional)
ccccccc = maximum number of generations that may begin (optional)

Transfer Activity Type Record. Transfer activities are used to branch, based upon some condition, to the prescribed label. This ability to branch means the simulation model will not always take a single path through the system.



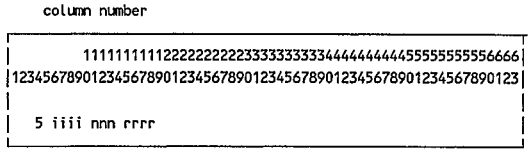
WHERE: iiii = label number (optional)
bb = branch type
1 = unconditionally taken
2 = branching occurs qqqq percentage of the time.
3-8 = branching occurs on a comparison of a system parameter qqqq and the value in nnnnnnn.
9-14 = branching occurs on a comparison of a resource qqqq queue's length and the value in nnnnnnn.
15-20 = branching occurs on a comparison of resource qqqq and the number of units nnnnnnn available for immediate resource request.
21-26 = branching occurs on a comparison of resource qqqq and the total number of units already in use and immediately available.
27-32 = branching occurs on a comparison with the system clock and the value of nnnnnnn.
33-38 = branching occurs n a comparison with the s. stem clock and a system parameter number nnnnnnn.
llll = label where a successful comparison will be sent to. (Note: unsuccessful comparison will result in passing through to the next simulation record.)
qqqq = quantity used for one side of the comparison. (optional)
nnnnnnn = second comparison quantity. (optional)

Resource Allocation Record. Resource allocation records define resources and their initial quantity in the resource pool. It is important to note, that the quantities in the resource pool may fluctuate during the simulation, since a quantity of the resource's pool may be given to another resource's pool.



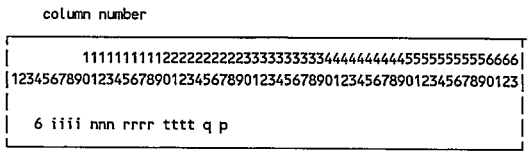
WHERE: iiii = Label number (optional)
rrrr = resource pool number
nnn = initial quantity in the resource pool (Note: may be "0")

Resource Request Activity Type Record. Resource request activities request a quantity of a particular resource from the available pool. If the system is unable to satisfy the request, the request will be suspended and placed into a queue.



WHERE: iiii = label number (optional)
nnn = quantity of the resource being requested
rrrr = resource post number

Resource Release Activity Type Record. The resource release activities release a quantity of one resource to either the same or another resource type. If there are items in queue waiting for this resource, the resource release record also specifies how this particular release wishes to take items from the queue. Each particular resource release request may handle queued items in a different manner.



WHERE: iiii = label number (optional)
nnn = quantity of the resource being released
rrrr = resource pool number which is releasing quantity nnn.
tttt = resource number which will receive the released quantity

to be added to its resource pool. (optional)

q = release from queue, how:
1 = FIFO
2 = LIFO (not implemented)
others

p = generation parameter used in releasing from the queue. (optional)

The remaining records include formats for accessing parameters, indicating end of a transaction or block of records, indicating end of the input file, specifying maximum run time, and exercising data output options.

Figure 5 depicts the DSIM input file for a basic M/M/1 model. Appropriate comments are shown.

```
.  CONVERSIM TIME_UNITS = MINUTES
.
.  THIS NEXT RECORD DECLARES A RESOURCE NAMED SERVER
.  THERE IS ONE SERVER
.
.  CONVERSIM RESOURCE#  11 SERVER

4   11   1
.  CONVERSIM SYSTEM_USER = TRANS
.
.  THE NEXT RECORD IS A GENERATION RECORD
.  THERE IS A SINGLE SYSTEM USER NAMED TRANS IN THIS MODEL
.  IT WILL GENERATE A NEW TRANS EXPONENTIALLY EVERY 10 TIME
.   UNITS.  THE FIRST GENERATION WILL START AT 0 TIME UNITS.
.   THE SYSTEM WILL ATTEMPT TO GENERATE 9999999 NEW TRANS
.
2     3      10                0 9999999
.
.   THE NEXT RECORD IS A RESOURCE REQUEST RECORD
.   A REQUEST IS MADE FOR THE ONE UNIT OF THE SERVER
.
5       1   11
.
.   THE NEXT RECORD IS A TIME ADVANCE RECORD
.   TIME WILL BE ADVANCED EXPONENTIALLY WITH A MEAN OF 5.
.
1     3       5
.
.   THE NEXT RECORD IS A RESOURCE RELEASE RECORD
.   ONE UNIT OF THE SERVER WILL BE RELEASED.
.   IT WILL BE RETURNED TO THE SERVER AND ANY ITEMS
.    RELEASED FROM THE QUEUE WILL BE IN A FIFO MANNER
.
6       1   11    0
.
.   THE NEXT RECORD IS THE END OF THIS SYSTEM USER
.
990
.
.   THE NEXT RECORD WILL PERIODICALLY DISPLAY STATISTICS
.    EVERY 1000 SIMULATED MINUTES
.
996        1000
.
.   THE NEXT RECORD WILL CAUSE PROBE OUTPUT TO BE PLACED
.    INTO THE OUTPUT FILE
.
995
.
.   THE NEXT RECORD IS THE MAXIMUM TIME LIMIT RECORD
.   THIS MODEL WILL RUN FOR 10000 SIMULATED MINUTES
.
998        10000
```

Figure 5. DSIM Input Data File for M/M/1 Model

Internal Data Structure and Processing

Input Source Records. The simulation system stores the simulation model as a linked list of source records in memory. The source records are kept in memory to decrease processing time by the simulation system.

Each transaction in the simulation contains a pointer to one or more of the input source records. Each record contains a pointer to the next input record and a pointer to the last record. These records also contain statistical information including the probe counters.

Transaction Records. The simulation system relies on the fact that all processing will be performed in simulated time. This means that all operations in the simulator must be stored in numerically ascending time, based upon the requested time. All transactions are stored in a linked list. When a transaction has completed an operation in the source input records, it will create a new transaction, copy any information that is necessary for the entire run, point to the next source input record, and establish a new time for this transaction. The new time may be the present time as determined by the system clock or it may be an advanced time. If the time is the present time, it is inserted into the linked list of transaction records at the head of the list. If it is not the present time, it is inserted in at its proper position. The time sort routine will then pass control back to the simulation system.

Sort Routine for Transactions List. Transactions are stored in a linked list. Each transaction contains the time that it will be processed. When the transaction makes its way to the head of the list, the system clock is updated to the time of the transaction that is being processed. The transaction list contains all of the transactions that are in the system, including the termination transaction, all present generations, and the next generation time for each generation statement. The transactions list is sorted numerically ascending based upon the requested time to process.

System Control. The system control routine is DSIM's time-flow mechanism. Whenever a transaction is processed, the system clock is updated, and the transactions record list is re-sorted by system control. Initialization, an important function is also scheduled by system control. The control sequence is depicted by the flow chart, Figure 6.

Random Number Generation. Random numbers serve an important function in any simulation system. DSIM is no exception. There are presently 10 random number streams which are initialized at the start of the simulation execution. Every location in the source input records which requires a random number is assigned a new random number stream up to the maximum number allowed in the system. Each stream serves independently to generate uniformly distributed numbers over the range $0.0 < \text{rnd\_num} < 1.0$. This type of random number generation lends credibility to any results derived from a model which is run using DSIM.

Generation Processing. The system initializes all generations statements at system start up. When the transaction processor recognizes that a generation statement is now being processed, it will generate the next future generation transaction and have it inserted into the sorted transaction lists. A check is made to insure that the next generation will not be

more than the requested number of generations on the source input record.

Queue Storage. If a request for a resource can not be completely satisfied, the request is placed into a single queue. This request is tagged with the resource number it requires, the quantity required, and other important statistics. All items added to the queue are added to the tail of the queue so that they may be removed later in the proper order.

Resource Release and Queue Removal. When a resource releases a quantity back to the resource pool, a check is made to see if there are any transactions in the queue which could now be satisfied. If so, the first-fit algorithm is employed in conjunction with the other queuing disciplines that the resource release has requested. For example, assume that a resource is returning 3 units to the resource pool. The release request has requested a FIFO discipline. In the queue for that particular resource (in order) are requests for 5 units, 2 units, and 1 unit. The 5-units request would now remain unsatisfied. The 2-units request would be added to the present transactions list. And lastly, the 1-unit request would then be also added to the transactions list.

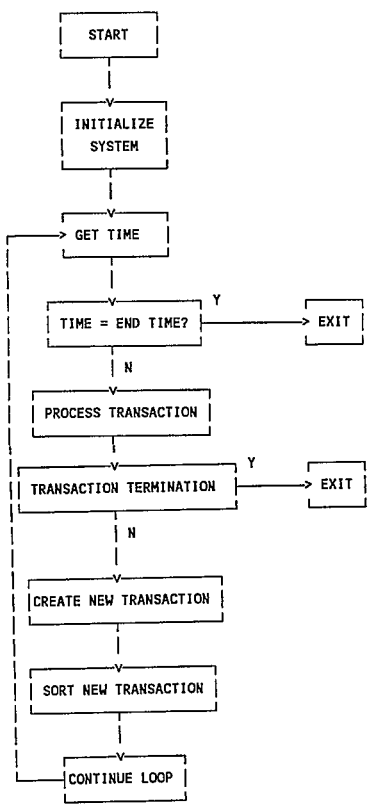


Figure 6. DSIM Control Routine

Output Report

An M/M/1 queuing model was implemented with the following description:

Single-server resource called SERVER;
User transaction called TRANS;
TRANS interarrival dist'n: neg. exp., mean 10 min.;
Mean service time: 5 min.;
Length of run: 1000 min.

The basic, autonomous output report is shown as Figure 7.

Note that the CONVERSE activity statement file is reproduced. Also shown are the basic data outputs including resource utilization and transaction performance. These are highly self-explanatory and are not discussed further.

```
                                DSIM
                            INPUT LINES

 stmnt  rcrd unit
  #     type  #

   1     .  CONVERSIM TIME_UNITS = MINUTES
   2     .  CONVERSIM RESOURCE#  11 SERVER
   3 DECLARE RESOURCES SERVER QTY = 1
   4     .  DATA INITIALIZATION SECTION
   5     .  CONVERSIM SYSTEM_USER = TRANS
   6 GENERATE A USER MEAN = 10 OFFSET = 0 # GENS = 9999999
   7 REQUEST RESOURCES SERVER QTY = 1
   8 ADVANCE TIME MEAN = 5
   9 RELEASE RESOURCES SERVER QTY = 1 OUT Q BY FIFO
  10 END OF BLOCK
  11
  12
  13
*********************************************************************

                                DSIM
                        UTILIZATION OUTPUT

STATEMENT  NUMBER OF COMPLETE        AVERAGE TIME
   #          TRANSACTIONS           TO COMPLETE

   6              103                   4.6311


THE SYSTEM CLOCK IS                   1000


  STATEMENT        AVERAGE TIME
   NUMBER      BETWEEN GENERATIONS

     6               9.7282

   RESOURCE     # COMPLETED       AVG. TIME        AVERAGE %
    NAME                            USED          UTILIZED

     SERVER          103           4.6311           0.48


   RESOURCE    PRESENT QUEUE     TOTAL IN     MAXIMUM IN        AVG. QUEUE
    NAME         QUANTITY         QUEUE        QUEUE              TIME

     SERVER          0              7            1              0.0000

*********************************************************************
```

Figure 7. DSIM Output Report for M/M/1 Model

CLASSROOM EXPERIENCE AND RESULTS

Description of Course

As of the date of this document, CONVERSIM has been used in two courses, both run concurrently at the

University of South Florida. The course was a Computer-Science-oriented survey of simulation and simulation languages taught at the graduate level. CONVERSIM was employed as a tool in the transition between a manual simulator, called "Hyposim," and the introduction of GPSS. It was introduced at about the fifth week in a fifteen-week semester.

Use in Course

Its employment was primarily to introduce a programming tool which encompassed "generic" terminology, not very different from that of basic queuing theory which was a preceding unit. It allowed the student to experiment with a simulator and structure a model without having to understand a particular terminological or programmatic structure. The idea was to verify some of the results of queuing theory without "language stress." In this aspect it succeeded. The models programmed with CONVERSIM were the identical introductory queuing models used for queuing theory analysis and later for GPSS and Simscript II.5 application. These were the M/M/1 queue and a simple 2-server feedback network.

Results

Student reaction to CONVERSIM was positive in that it really provided a quick-access vehicle for computer implementation of a simulation. Student reaction to the CONVERSIM "process" was informative from a tool design standpoint, especially within the context that only classroom instruction in CONVERSIM, without access to a user's manual, was provided. From this experience it was concluded that CONVERSIM is self-instructing enough so that with minimal orientation construction of modest models is possible without written documentation. Also, student reactions to various features (or non-features) of the tool produced a "wish-list" of modifications and additions to both the tool and the process of using the tool.

CONCLUSIONS

That the CONVERSIM project successfully fulfilled the original goals there can be no question. Specifically:

A conversational program for building of discrete-event simulation models was accomplished.

It was successfully employed as a front end for a GPSS-like simulator.

It was implemented for microcomputer application.

That CONVERSIM has a greater, or even a continued utility beyond its rather limited trial employment, is an issue subject to a great deal of questioning.

Affecting its use as a general-purpose simulator in an applications environment is, for example, a structural limitation: its world-view, strictly confined to process interaction, exemplified by Fishman's "Process Interaction Approach, Concept 1." As a result it lacks the ability to represent servers as processes, thereby implementing true "producer-consumer" interaction, defined by Fishman as "Process Interaction Approach, Concept 2." To enable this would require the implementation of such constructs as "suspend" and "resume," or "passivate" and "activate," as available in SIMSCRIPT and Simula, respectively.

Other limitations are more in the area of "user-friendliness" and relate to editor features and certain options in outputting.

To dwell on such shortcomings at this point is argumentative. The plain result is that this enterprise has been successful, and that any undertaking of future development is a complex issue.

REFERENCES

Cox, S. (1984). The User Interface of GPSS/PC. In: Proceedings of the 1984 Winter Simulation Conference Institute of Electrical and Electronics Engineers, Dallas, Texas.

Ferrari, D. (1978). Computer Systems Performance Evaluation. Prentice-Hall, New York.

Fishman, G. S. (1973). Concepts and Methods in Discrete Event Digital Simulation. Wiley, New York.

Franta, W. R. (1977). The Process View of Simulation. Elsevier North-Holland, New York.

Meyerhoff, A. J., Roth, P. F., and Shafer, P. E. (1971). "BOSS Mark II Reference Manual." Technical Report 66099A, Burroughs Corporation, Paoli, Pennsylvania.

Nance, R. E. (1981). "Model Representation in Discrete Event Simulation: The Conical Methodology." Technical Report CS81003-4, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

Russell, E. C. (1983). Building Simulation Models with SIMSCRIPT II.5. CACI, Los Angeles, California.

Wirth, N. (1976). Algorithms + Data Structures = Programs. Prentice-Hall, New York.

AUTHORS' BIOGRAPHIES

Paul F. Roth is currently Distinguished Lecturer in Computer Science and Engineering at the University of South Florida. He has been involved in simulation, both continuous and discrete, for over 30 years, with service in industry, government, and academe, and as a consultant. He is currently doing research in simulation software for application to computer networks and manufacturing processes. He has twice served as Chairman of the ACM Special Interest Group on Simulation (SIGSIM), and has rendered nearly continuous service in various capacities to the Winter Simulation Conference since the early 1970's. His degrees are from Pittsburgh and Pennsylvania.

Robert Brown has been involved with computer software and applications since he received his degree in mathematics from Rutgers University in 1978. He has worked in various capacities for Computer Science Corporation since 1979, primarily doing scientific programming under the auspices of the AEGIS project for the Defense Department. He recently was awarded his Master's degree in Computer Science from Virginia Tech, where he contributed to the initial development of the techniques used in CONVERSIM.