Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

USING MOGUL™ 2.0 TO PRODUCE SIMULATION MODELS AND ANIMATIONS OF COMPLEX COMPUTER SYSTEMS AND NETWORKS

Peter L. Haigh

High Performance Software, Inc.
4288 Upham Road
Dayton, OH 45429, U.S.A.

ABSTRACT

Simulation techniques are increasingly being applied to computer system and network design and analysis. The use of animation is also increasing as an integral part of the simulation environment. Animation greatly enhances model verification, system understanding, and results presentation. MOGUL™ is a powerful software package for simulating computer systems and communication networks. It supports graphical model building, interactive statistics perusal, and animation. MOGUL combines the power of Wolverine Software's GPSS/H™ and Proof Animation™ with its own repertoire of processor, peripheral device, and communication network models to provide an integrated environment for system and network modeling.

1 INTRODUCTION

MOGUL (MOdel Generator with User Lead-through) is a package for rapid generation of simulation models of computer systems and computer communication networks. It includes REGAL™ (REport Generator And Lister), a program for rapid examination and display of simulation results using a display screen, rather than a line printer. The reader should consult Haigh (1991) or High Performance Software for a more detailed description of the modeling capabilities of the MOGUL package.

This tutorial will begin with a review of the basic model building and analysis capabilities of MOGUL and REGAL. It will then highlight the use of animation in model building, model verification, and results analysis.

2 MOGUL OVERVIEW

MOGUL allows the modeler to take a high level view of the system to be modeled, build the model from the top down, and add detail as information becomes available. After the user selects the objects to be simulated and describes the activity flow in the model, MOGUL generates a complete simulation model in the GPSS/H simulation language.

2.1 Objects

The modeler creates the desired processors, communication links, and peripheral devices in his model by selecting from the repertoire of *object* types provided. The repertoire of *object* types is open ended, and may be expanded by the user. *Objects* are described by sets of parametric data in a definitions text file. Each type (or class, in OOP terminology) of processor, link, and peripheral that the modeler may select is listed in the definitions file. The attributes of each *object* type are defined using a pre-defined syntax. The GPSS code to simulate all *object* types is included with the MOGUL package.

Protocols. Simulation code to model the major link level protocols is also provided. These include CSMA/-CD, SDLC/HDLC/X.25 family, various modes of BiSync, Token Ring, token bus, FDDI, and various Async protocols. Using the *activity path language* and these protocols, any TCP/IP or other type network may be simulated. If new link level protocols are needed, new GPSS code may be added to simulate them.

2.2 Activity Paths

Simulation activity flow is specified using a high level language and an easy to use interactive editor, which comes with comprehensive on-line help. The modeler creates a series of *activity paths*, or process descriptions, using statements from the language. The actions are such things as: send an information frame across a communication link, use a processor for a specified time (to simulate program execution), read or write to a peripheral device, form a queue, delay, perform arithmetic and control operations, test conditions, stop the simulation, etc. *Activity paths* are referred to as *software execution graphs* in Smith (1990).

2.3 Messages

When the model is executed, entities called *messages* flow along the *activity paths* and cause the activity specified in each statement to occur. *Messages* may be

created automatically at the beginning of any path, using a Poisson arrival process and a user specified mean inter-arrival time. Alternatively, a *message* may be spawned by another *message*. A *message* may do such things as branch to another activity path, skip statements, call an activity path as a subroutine, spawn other *messages*, branch to user written code, and terminate itself.

3 ACTIVITY PATH LANGUAGE

All statement types in the *activity path language* are defined in a language definition text file. Each statement type is implemented as a GPSS routine. The label of the routine in the simulation code is the same as the statement keyword. To add a new statement type 'XXX' to the language, all that is necessary is to add a routine labeled 'XXX' and edit the language definition file to include the 'XXX' statement definition.

3.1 System activity statements.

The statement:

```
PRO 3
```

causes the *message* executing the statement to seize the CPU for 3 milliseconds to simulate program execution. There is the notion that a *message* is resident in some particular processor - the 'current' CPU. That is the CPU referred to by the PRO statement. MOGUL keeps track of a *message*'s whereabouts (in a CPU, on a communication link, outside the system hardware, etc.).

A single PRO statement, accompanied by the appropriate menu selections of processor type and *message* arrival rate, forms a complete single server queuing model. MOGUL automatically creates and manages a queue associated with each resource (in this case the processor referenced by the *message* executing the PRO statement). *Messages* of higher priority arriving at a PRO statement will preempt a *message* of lower priority currently using the referenced CPU.

To perform I/O to a peripheral device, the statement:

```
IOP 3 RND 512
```

simulates a random access of device number 3 and a transfer of 512 bytes to or from main memory. Again, the CPU in which the *message* is resident is relevant.

To send a communication message across a link:

```
XMT 7 2
```

transmits a communication message from the current CPU across link number 7 to CPU number 2.

The above examples illustrate a few of the system activity statement types.

3.2 Flow control statements.

The following are examples of flow control statements:

```
BRNCH 3  750  Branch to path 3 75% of the time
SPN   7  1000 Spawn a msg on path 7 100% of time
SUBR  5       Call path 5 as a subroutine
TRM           Terminate this msg
SEND  1  512  Split msg into 512 byte packets,
              key=1
RCV   1       Receive & assemble packets with
              key=1
```

The SEND and RCV statements allow the modeler to simulate splitting a communication message into packets which may follow different routes to their common destination. The RCV statement automatically assembles the packets and rebuilds the original message at the destination, noting any packets that are late or fail to arrive.

3.3 Arithmetic statements.

There are 100 general purpose registers available for storing and manipulating variables. The arithmetic statements allow the modeler to load, operate on, and test the contents of these registers. For example:

```
ADD 1 3 SKG   Add register 1 to register 3,
              skip if result > 0
DEC 5 1 SKE   Decrement register 5 by 1,
              skip if result = 0
```

3.4 Timing statements.

The statements:

```
      TIME0   (Time zero)
and   TMTBL   (Time table)
```

are used to measure the transit time of a *message* from one point in the model to another. TIME0 sets the elapsed time of the *message* to 0. TMTBL places the elapsed time of the *message* into a table associated with the *activity path*.

3.5 Threads

A *thread* is a portion of an *activity path* which represents a resource. For example, the statements

```
      BTHRD  5
and   ETHRD  5
```

mark the beginning and end of *thread* number 5. The *thread* may represent a process, a computer program, or, in general, the use of a resource by the *message*. The user may specify how many *messages* may enter the *thread*. MOGUL automatically maintains a queue (a GPSS user chain) for *messages* attempting to enter a full *thread*.

3.6 Other statement types.

These have been only a few examples of statements from the *activity path language*. There are currently 41 statement types available. Users, of course, may add any number of statement types for their own special pur-

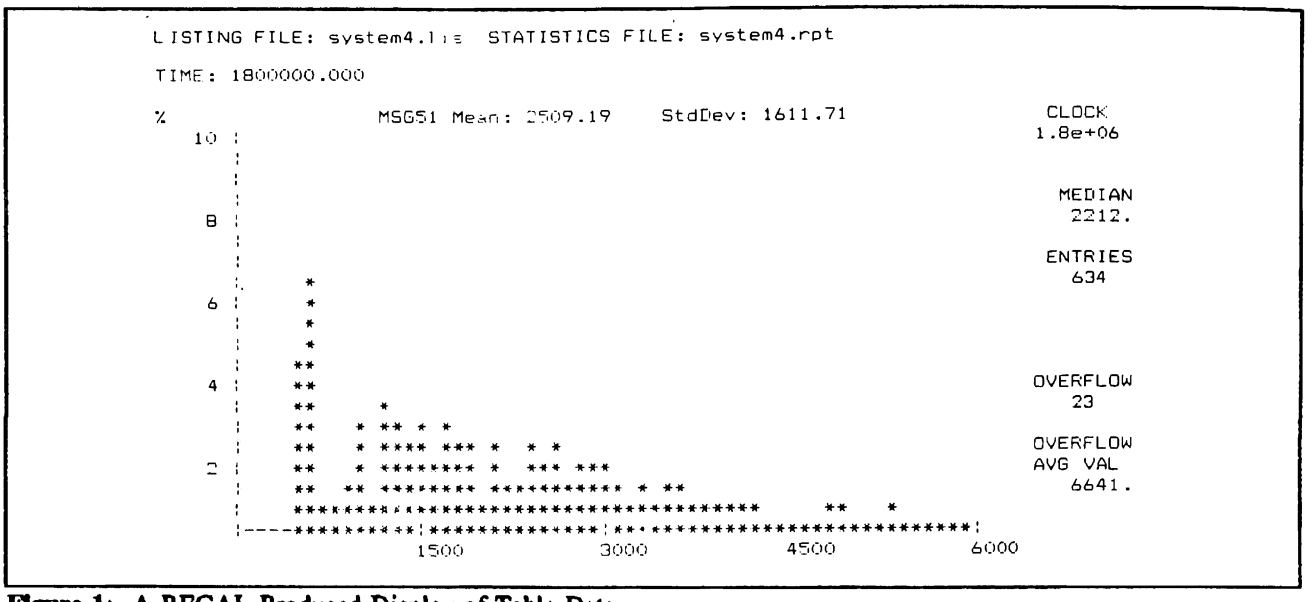

Figure 1: A REGAL Produced Display of Table Data

poses.

4 RUNNING & EVALUATING THE MODEL

After the modeler has defined the objects in the model and built the *activity paths* describing the dynamic activity, MOGUL will generate the source code for a complete GPSS simulation model. The GPSS/H compiler is then used to run the model.

4.1 Checking Statistics with REGAL

REGAL can be used to quickly inspect the statistics from a simulation run. REGAL parses the GPSS listing file and builds a set of structures containing the pertinent statistics. These may then be viewed using menu selection. The modeler may see charts of resource utilizations, queue statistics, and various other model variables (v.i.z. GPSS savevalues). Table data may be seen in graph form, which may be displayed on any 80 by 25 character oriented CRT, as shown in Figure 1.

Snapshots. REGAL can also display snapshot output. A snapshot is a statistical report printed to the listing file before the end of the simulation. The user may select the specific report to view, or may produce a graph of a specific model variable taken from a series of snapshots.

Multiple Runs. If the modeler wishes to view the same model variable across a series of simulation runs, REGAL can pick a statistical value from a series of output files and produce a graph of that variable as a function of the simulation run number.

5 ANIMATION

Recently there has been a proliferation of commercially available animation software. MOGUL uses an inexpensive, but extremely powerful, animator called Proof Animation, available from Wolverine Software Corporation. An advantage of this approach is that the animator may be used with applications other than MOGUL and GPSS/H. Furthermore, if the user does not wish to use animation, the animator need not be purchased.

5.1 Why Use Animation?

It can be said that animation has at least the following purposes when used in conjunction with simulation modeling:

1. Presenting results to decision makers
2. Verifying & debugging the model
3. Understanding the modeled system

The relative importance of these uses is usually presented as the order listed above. Those who use animation, however, (the author among them) will usually rate these uses in the reverse order. *Understanding the modeled system* is by far the most important benefit. The modeler, as well as anyone needing to grasp the workings of the subject system, benefits beyond measure by viewing an animation of the simulated system.

Seeing Is Believing. Looking at a print-out of statistics from a simulation run, one does not see *when* a queue reached its maximum length or *when* the longest response times occurred. The *time relationship* of various statistics and variable values can not be ascer-

tained from a flat statistical summary. Watching an animation, however, dynamic relationships among the resources and system activity become immediately apparent and lead the modeler to experiment with the system design to improve the system's performance.

5.2 Graphical Model Building

The animation layout for a MOGUL simulation model may serve as input to the model creation process. This allows the modeler to conceive of the picture of the system he or she wishes to observe, create it graphically using Proof Animation, then read it into MOGUL. MOGUL is then used to complete the model definition (assign CPU types, create *activity paths*, etc.). When the simulation is run, a trace file is created which drives the graphical layout created by the modeler. Figures 2 and 3 show examples of MOGUL driven animations.

Creating System *Objects*. The MOGUL package provides a basic set of classes from which to select *objects*. There are graphic shapes to represent CPUs, I/O devices, and various types of communication links. Users can create additional shapes, if desired, to better represent specific network geometries, peculiar devices, etc. A simple naming convention lets MOGUL know what type each *object* is. For example, peripheral devices must be named DEV1, DEV5, etc.

When the simulation is run, MOGUL outputs the animation commands to set the color of an *object* GREEN, if it is idle, and RED, if it is in use. Also, any disk *objects* will rotate at their proper rotational speed.

Queues. Every processor, peripheral device, communication link, and *thread* has a queue associated with it. To display a queue, the modeler creates a queue object, places it where desired in the diagram, and gives it a name. As with system *objects*, the name given to it tells MOGUL which queue should be displayed. The name TQ3, for example, specifies the queue (user chain) for *thread* number 3.

To implement a queue display, MOGUL creates ten *queue entry* elements, which are stacked one above the other in the layout, with some space between them. When the simulation is run, the *queue entry* elements are changed from background color (invisible) to YELLOW as *messages* join the queue. If the queue size gets greater than ten, the color RED is used, with each element representing ten queue entries. This allows a viewer of the animation to tell visually how many objects are in the queue (up to a queue size of 100).

Statistics. Various statistics can be displayed. The average utilizations of processors, peripheral devices, communication links, and *threads*, and the average values of tables may be displayed. To display the average utilization of peripheral device 3, for example, the modeler creates a message named DVUTL3. Then, during the simulation, MOGUL periodically writes the average utilization of device 3 to the spot where the message is positioned.

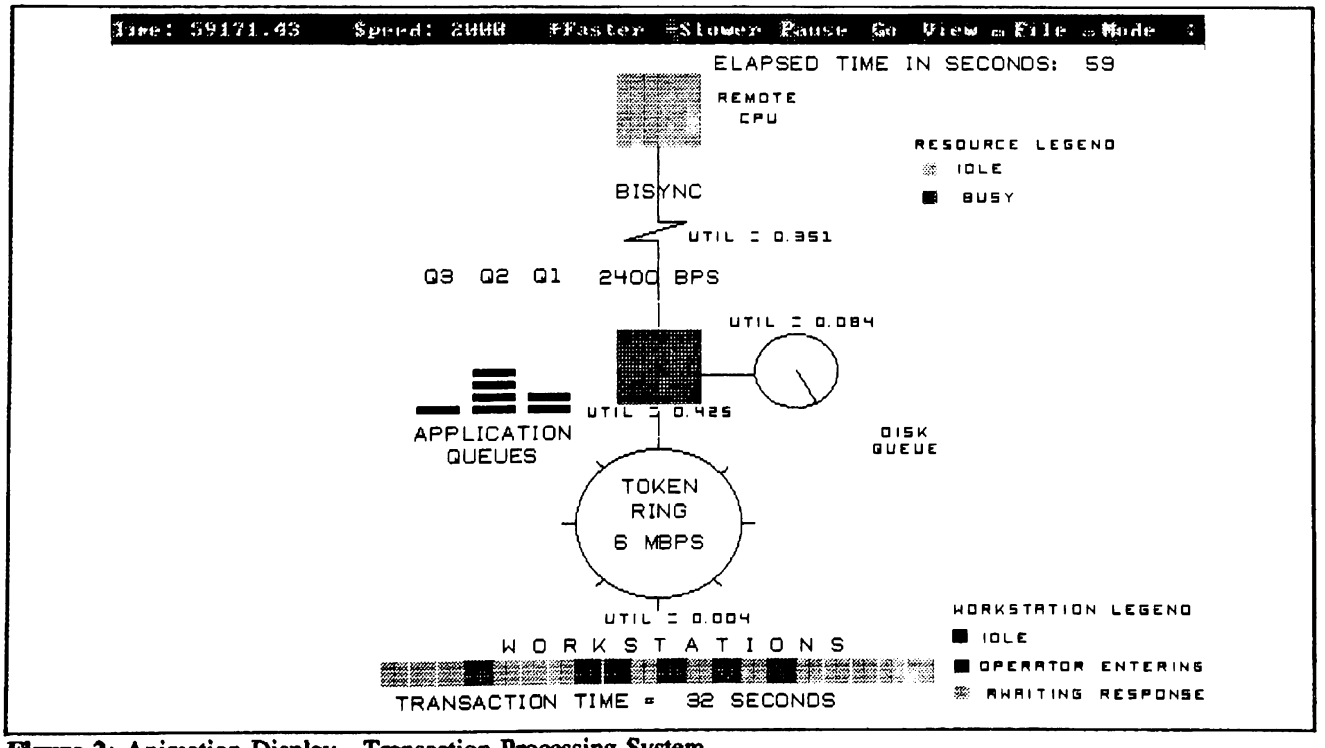

Figure 2: Animation Display - Transaction Processing System

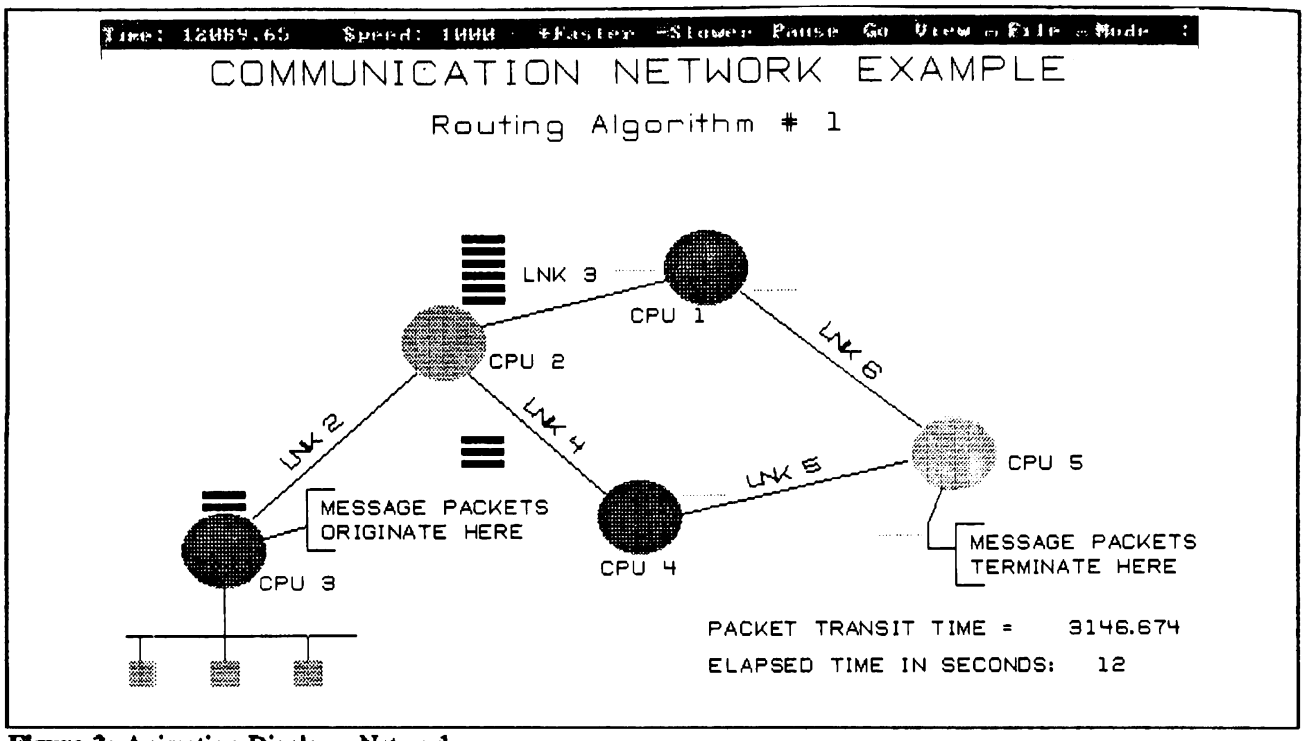

Figure 3: Animation Display - Network

5.3 Verifying & Debugging the Model

Using Proof Animation, the modeler can observe each state change in the system. In the DEBUG mode, the animation may be single stepped by event, to observe each event, or by time, to observe all changes at each clock update. This procedure is much easier than the process of setting breakpoints, checking variable values, and single stepping used with a traditional debugger.

6 CONCLUSION

The MOGUL package, combined with GPSS/H and Proof Animation, provides a powerful environment for simulation modeling of computer systems and networks. The environment supports graphical model building, high level model building with any desired level of added detail, and animation.

There is no theoretical limit to the size of a MOGUL generated model or its animation. The only practical limit is the amount of memory available in the hosting computer. Both Unix and DOS environments are supported. Model building, simulation, and animation may be performed on different computer systems, if necessary. This flexibility affords a solution for almost any computing situation.

REFERENCES

Brunner, D. T, N. J. Earle, J. O. Henriksen. 1991. Proof Animation: the general purpose animator. In *Proceedings of the 1991 Winter Simulation Conference*, ed. B. L. Nelson, W. D. Kelton, and G. M. Clark, 90-94. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Haigh, P. L. 1991. Simulation of computer systems and networks with MOGUL and REGAL. In *Proceedings of the 1991 Winter Simulation Conference*, ed. B. L. Nelson, W. D. Kelton, and G. M. Clark, 86-89. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Henriksen, J. O. and R. C. Crain. 1989. *GPSS/H Reference Manual*, Third Edition. Wolverine Software Corporation, Annandale, Virginia.

Law, A. M. and W. D. Kelton. 1991. *Simulation Modeling & Analysis*. 2nd ed. New York: McGraw-Hill.

Smith, C. U. 1990. *Performance Engineering of Software Systems*. Reading, Massachusetts: Addison-Wesley.

AUTHOR BIOGRAPHY

PETER L. HAIGH is founder and president of High Performance Software, Inc. He has authored papers on simulation and computer system performance topics. He is a member of ACM, IEEE, and SCS; has chaired the Greater Dayton ACM chapter; and was general chairman of the 1988 Winter Simulation Conference.