Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

A MODELING LANGUAGE GENERATOR FOR A DISCRETE EVENT SIMULATION LANGUAGE IN MATLAB

Guy L. Curry
Amarnath Banerjee

Department of Industrial and Systems Engineering
Texas A&M University
3131 TAMU
College Station, TX 77843-3131, USA

Hiram Moya

Department of Manufacturing and Industrial Engineering
The University of Texas Rio Grande Valley
1201 West University Drive
Edinburg, TX 78539-2999, USA

Harry L. Jones

Department of Civil Engineering
Texas A&M University
3136 TAMU
College Station, TX 77843-3136, USA

ABSTRACT

A discrete-event simulation language was implemented in MATLAB. The approach is similar to the process/command modeling paradigm utilized in GPSS and other languages that followed. The language is a MATLAB Script File (m-file) and can be part of a larger analysis package as a sub-function of an optimization/simulation system. The modeler builds the simulation through support functions provided in this system but must insert them in the proper locations of the MATLAB master function. To develop a proper model, it is necessary to understand the internal simulation structure using the switch/cases statement and where various aspects of the simulation structure are located. To simplify this process, a model generator has been developed which parses a model text file and produces the required MATLAB master simulation function. The model generator also reduces the magnitude of understanding of the implementation specifics of the MATLAB simulation language and makes proper model development easier.

1 INTRODUCTION

There are many discrete-event simulation languages which have full support packages such as GPSS (Gordon 1961, Schriber 1974), SIMAN (Pegden 1983, Pegden 1986), ARENA (Kelton, Sadowski, and Sadowski 1998), and Simio (Kelton, Smith, and Strurrock 2013), to list a few. These languages are full scale compiled languages with large scale modeling capabilities and functionality. They are in general stand-alone fixed systems and not part of a general modeling/programming system. A simulation language was implemented in MATLAB which has the ease of model building structures found in these discrete-event simulation languages but with the advantage of being part of a more general problem solving environment (Curry and Banerjee 2016). MATLAB is one of the most widely taught and used tools in

engineering and the applied mathematical sciences (Kuncicky 2004). For example, MATLAB is introduced to all college of engineering students at Texas A&M University in their common first year two-course sequence fundamentals of engineering as well as the first two engineering calculus courses. MATLAB itself has a support package Simulink (MathWorks 2005) for dynamic systems modeling but it is also categorized as "interactive tools with limited control over the model" (Sklenar 2013). MATLAB's Simulink is a block diagram environment for multi-domain simulation and model-based design. Simulink is at a similar level as most higher level discrete event simulation languages, and supports continuous and discrete dynamic blocks, algorithmic blocks, and structural blocks as well as embedding of MATLAB, C, Fortran and Ada code.

The approach taken for the discrete-event simulation language is to use the modeling paradigm of GPSS with structural commands available to the modeler to help build reasonable models in a quick and easy fashion but to still have the full capability of the MATLAB language available at all times for the modeler. Apparently, the only other open MATLAB discrete simulation tool set is that of Sklenar (2013) where he developed a set of simulation support functions for use with MATLAB. His approach was specifically not to use the standard modeling paradigm. Here the approach is exactly the opposite; the ease of model development with this approach makes it an effective modeling tool and with this implementation it is also a function that can be part of a larger analysis system.

Students in a discrete event simulation course are exposed to higher level building blocks to construct their models while using the simulation languages mentioned earlier. It is frequently observed that students routinely develop models without fully comprehending the inner workings of a discrete event simulation system. Even when students implement pre-developed models (Moya, Curry, and Phillips 2015), they typically blame the software for their inability to accurately model a given problem statement. The available choice of parameters in the software is sometimes overwhelming, which leads to students making incorrect choices. This MATLAB-based discrete-event simulation language aims at providing some insight to students about the inner workings of discrete event systems. This, coupled with one of the simulation languages, is expected to provide the fundamental understanding of discrete event simulation, which can assist in the process of proper usage of the full support packages and ability to develop accurate models. Moreover, this simulation language will serve as additional exposure to the use, practice and capabilities of MATLAB for students.

The modeling terminology and syntax chosen is based on the language MOR/DS (Curry, Deuermeyer, and Feldman 1989). The language structure is based on the command set: Arrive, Schedule, Seize, Wait, Release, Depart, GoTo, SetGate, TestGate, Copy, Combine, Link, Unlink, Observation, TimeObs; a set of data structures: Resource, Gate, Chain, Histogram, TimeHistogram; and a set of attribute commands: Mark, Attr, AttrValue, AttrInc. The modeler can define and update their own variables and collect data about these variables via the Histogram (for value data such as cycle times) or TimeHistogram (for time persistent data such as work-in-process) structures. These can be plotted as appropriate using the MATLAB graphics functions. Since all functions are available to the modeler, enhancements, corrections and additional functions can be developed by the user.

2 SIMULATION CONCEPT

The MATLAB discrete-simulation system resides in an MATLAB Script File (m-file) where the model is developed and executed in the main function (here in called 'Simulate'). This function defines all of the internal data structures used in the simulation as well as those separately defined by the modeler. A group of support functions such as all of the simulation commands reside in this Script File. The Script File (the base model m-file is Simulate.m) is loaded and used as a basic structure for developing additional models. There are several functions in the simulation m-file but only the first function (Simulate) needs to be changed by the modeler. All other functions in the m-file are support functions for the main simulation model function – Simulate. A simulation model has a specific structure and order that simulation commands (function calls) must be executed to accomplish the tasks desired. The model logic execution

structure is a MATLAB Switch/Cases statement. The model is developed by placing the executable commands in the order needed. This is the typical process for developing discrete-event simulation models in almost any language. However, here these statements/commands/functions are placed in order and labeled via the Case statements within the Switch/Cases structure. The functions which are generally considered as simulation commands are setup to move the entity flow through the case statements as appropriate. In general, these commands return the system entity Switch/Case locator variable (SVloc) to point to the next case in the model sequence. If flow is to be in any order other than sequential, then the GoTo function is used to branch the entity to the appropriate case. When the active entity loses control, such as being blocked by a resource limitation in a Seize function or it is required to wait a specified time such as in a Wait function or even deleted from the system via the Depart function, the system variable SVloc is set to zero and, when all of the statements in the particular Switch/Case have been executed, processing for that entity is closed and another entity is selected to begin/continue its progression through

```
while (ClockTime < StopTime) & etc.
       [ClockTime, SVactEnt, SVloc] = get next entity from Future Event List
       while SVloc > 0
          switch SVloc
             case 1
             ⋮
             case n
          end % switch
       end % while SVloc > 0
end % while (ClockTime < StopTime)
```

Figure 1: Simulation Control Logic.

the model statements. The simulation control logic is displayed in Figure 1.

Most discrete-event simulation systems (such as GPSS, SIMAN, ARENA, Simio, MOR/DS) take an entity and move it through the simulation model structure as far as it can be moved until a block or time delay occurs. Then another entity is selected and the process is repeated. To control the timing of events, in this language a Future Event List is used and entities are placed on this list in the time sequence of their next time event. So when an entity reaches a Wait command, for example, and must be delayed until the clock reaches a specified future time, the entity is placed on the Future Event List with that time as its activation time and control is passed to another entity. The next entity selected is the first entity on the time sequenced Future Event List and the current clock time is advanced to the activation time of the newly activated entity. Note this operation is always a non-decreasing time sequence. The current clock time is the system variable ClockTime and this value is available to the modeler. When the system time exceeds the modeler provided simulation StopTime, then the simulation is terminated and the statistics functions (ResourceUtil and Statistics) are executed for model output results. The modeler can also place any desired results output including graphics in the same functional area of the model. Note that if replications are used, the modeler may not want to see each replication result and only view the summary statistics result. So the variable PrintRepls is available to suppress the individual replication output.

3 LANGUAGE CONSTRUCTS AVAILABLE

Every simulation language must have entity create and destroy capabilities; these are the functions Arrive (or Schedule) and Depart, respectively. For this language implementation, we have also added a SeedArrive function that is needed to establish the initial entity for each Arrive command in the model. Each Arrive statement is an entry point into the simulation model domain from the un-modeled aspects of the remainder of the universe or real world. Of course the Depart function terminates the specific entity's residence in the model and can be thought of as returning to the un-modeled domain. There are three modeling constructs available: Resources, Gates and Chains. Resources represent limited capacity situations such as

machines, tools, people, etc. that need to be obtained before performing some operation. Entities are queued if a unit of the specified Resource is not available. Gates are non-capacitated structures where entities are allowed to pass if the Gate is open and queued (blocked) if it is closed. Once the Gate is re-opened, all blocked entities are allowed to pass through. Finally Chains are queueing structures where the modeler handles the placing and releasing of entities. Each of these structures has associated with them two functions used to interact with the base structure. For Resources, the request for a unit of resource is the Seize command and the unit is returned via the Release command. For Gates, TestGate checks if it is open to pass through and if it closed then the entity is queued. The SetGate command can open or close the Gate depending of the command argument. The Link and Unlink functions control placing and removing entities from user defined queues/lists (Chains).

Statistical support is provided in two ways. First, multiple runs of the simulation and computing the 95% confidence limits for composite statistics is provided by specifying the number of replications desired (NumRepl). Two types of user determined statistics can be obtained via the Histogram structure (value observations such as system cycle time and machine processing time) and the TimeHistogram structure (time persistent data such as time average work-in-process). Observation or tally type of data is recorded using the Observation function and time average data using the TimeObs function.

Time delays generally occur in two situations. The first is the delay time between arrivals to the system and the second type is a processing type delay when in control of a resource. The Arrive command has an inter-arrival delay as one of the arguments of the function call. Delays such as processing delays are instituted by the Wait function which has as its argument the type and parameters for the associated delay distribution. Four distribution classes are installed at this time for user convenience: type 1 is for constant values, type 2 is for the exponential family including generalized Erlang distributions, type 3 is for a continuous uniform and type 4 is for the truncated normal. The modeler can also utilize any MATLAB implemented distribution or even create their own. By generating a random variate value and assigning it to a variable say "xtime", the constant type 1 can be called using "xtime" as the associated value.

Groups of entities can be combined together using the Combine function and groups created with the Copy function. For branching entities to other model cases the GoTo function is provided and was discussed in Section 2.

Standard simulation modeling controls are provided such as ending the simulation after a specified simulated time StopTime, or stopping based on departure counts using StopCount. Most simulation models have a startup period when results statistics are biased due to the system starting empty and thus a ResetTime control is also provided.

4 THE NEED FOR A MODEL GENERATOR

Developing a model in any simulation language requires that the modeler obviously know the syntax of the modeling language. However, when the modeler is actually coding in the language within which the system was developed they must know that language as well as understand how the simulation language particulars were developed and implemented. In addition to knowing the MATLAB language, the modeler in this case needs to understand the structure of the simulation language. It is necessary to understand how the language was implemented in MATLAB and some of the more specific details of the language design itself. For example, the MATLAB discrete-event simulation language uses the switch/cases structure for implementing the steps of the simulation model such as the Seize-Wait-Release modeling sequence. If an entity requests a unit of a resource (via the Seize function), this unit may be immediately available and hence the entity proceeds to the next function call or it may be that the entity must be blocked until its turn for a resource unit comes up. This simulation language implementation uses an internal location indicator to establish, when an entity is activated, where it resides in the model switch/cases sequence. Certain simulation commands were designed to impact this location indicator and other commands to not impact the switch/case location. Understanding and properly using the simulation function calls within the model logic case statements adds an additional burden to the modeler. Another example is that for time persistent

data collection, via the TimeHistogram structure, the modeler must place the TimeObs commands to collect this data within the simulation controller loop but outside of the model switch/cases structure. The modeler thus needs to understand somewhat how the simulation language is structured to understand the need and proper location of these data collection commands. In addition, if the modeler wants to graph a time-persistent variable, data collection occurs at this same location within the controller loop. Note that this data probably needs to be sampled because of the high frequency of this loop being executed and these commands also need to occur at this same location.

For the above reasons and others to properly model with this MATLAB based discrete-event simulation system, the modeler needs to understand how the system was designed and implemented. This knowledge is beyond that necessary to merely develop models in the language. To allow the modeler to concentrate on developing the simulation model and not be too concerned about the structure of its implementation, we devised a model generator which will take a text version of the simulation model and insert these statements into the MATLAB master simulation function in their proper locations and when possible insert extra statements to make the model more usable and less directly connected to the implementation approach. For example in the previously mentioned collection of time-persistent data, the modeler needs to define a system collection device (TimeHistogram) for each variable to be observed (such as WIP in each workstation). The model generator will then create a statistical collection structure for that variable and also create a TimeObs statement for the same variable to be inserted (automatically) in its proper location when the generator writes the statements for the simulation controller loop. If graphics data for plotting this variable is also to be collected, then by placing this coding in a special segment definition area for time persistent data, the model generator will place these MATLAB statements in the controller loop with the TimeObs statements.

5 MODEL GENERATOR

The model definition text file is segmented into eight natural groupings of data types. These are listed here in order that they should appear in the model text file (some categories are optional and are indicated as such): Program, Control, System Definitions, Variable Definitions, Replicate Outputs (optional), Final Outputs (optional), Time Persistent Logic (optional), Reset Logic (optional), and Model Logic. The first word of each category is the keyword for that category. The remainder of the line is ignored and only used to help connote to the modeler what the category entails. The Replicate, Final, Reset and Time categories are optional and can be listed in any order but must lie between the Variable and Model categories. A discussion of each of these categories follows:

a) Program: this category is merely to start the model text file (it must be the first line of the file) and consists of comments statements that the modeler may choose to use to document what problem is being modeled. Note that MATLAB comment statements start with %.
b) Control: the control category consists of all of the variables used to specify how long the model simulation is to last in simulated time units, the time to reset statistics if desired, the number of replicates to be run, if replicate results are to be printed, and the number of attributes that each entity possesses.
c) System Definitions: this category is where the modeler defines simulation structures needed in the model; how many, their names if appropriate, and capacities, etc. These include: Resource, Histogram, TimeHistogram, Gate and Chain. An additional feature of this category is that these commands are placed in the MATLAB master simulation function above the replicates loop. So user variables defined here will be available as with all user defined variables, but will not be reset to their initial values at the start of a replication. Thus, if the modeler wants to run some total across all replications, this total variable would be defined in this category and used normally, but it will not be reset to zero by the system at the beginning of each replication.

d) Variable Definitions: these are user variables such as counters, arrays, model data such as processing times, etc. These variables are reset to their initial values at the beginning of each replication.
e) Replicate Outputs (optional): these statements are placed at the end of each replication and the purpose is to allow the modeler to specify their own output results along with the normal replicate results such as resource utilizations, histograms, etc. This is where one would place plotting commands for data collected during the replicate simulation run.
f) Final Outputs (optional): these statements are placed at the end of the complete simulation at the point where system resources data is averaged over all replicates and displayed. The modeler can also specify herein output results for data that they developed. This is also where the modeler would place plotting commands for data collected during the entire simulation run or if there are no replications (just a single run of the model).
g) Reset Logic (optional): these statements are placed at the location where the system resets all of the data collection devices to help mitigate the initialization aspects of the data collection. This allows the modeler to reset any variables that they are using to collect their own statistics/information. This is executed once per replication run if the ResetTime option is selected.
h) Time Persistent Logic: this category allows the modeler to incorporate MATLAB statements within the controller loop logic at the points in time when the simulation clock is advanced (more specifically, when a new active entity is selected). This allows for collecting time persistent data for graphing and even random sampling of these observations due to the high frequency of data observations within the controller loop.
i) Model Logic: this is the main aspect of the simulation program with the simulation commands linked in a manner so as to model the situation being analyzed.

The list of model generator categories (in their required sequence) are displayed in Table 1.

Table 1: Model generator categories and required sequence.

Sequence	Model Categories
1	Program
2	Control
3	System Definitions
4	Variable Definitions
5	Replicate Outputs (optional and variable placement)
6	Final Outputs (optional and variable placement)
7	Time Persistent Logic (optional and variable placement)
8	Reset Logic (optional and variable placement)
9	Model Logic

The model generator parses this text file and writes the simulation master function which is then executed by the modeler. The model generator parses the input lines and separates these commands and MATLAB statements into the categories within which they are defined and places them into the proper locations within the master simulation program. The model generator has additional functionality such as creating code for observing time persistent variables by automatically inserting these commands into the controller loop and establishing variables for structure names and assigning their proper structure numbers. This allows the modeler to refer to these structures by name for better readability and ease of model development. However, the model generator's main purpose is to partition the Model Logic section into groups of commands which perform the desired simulation model while fitting into the MATLAB switch/cases implementation and controlling flow between cases appropriately. Thus, the model generator

translates the modeler's intended model into the structure imposed by the basic language capabilities and simulation system implementation within this language. Different implementations can easily be envisioned based on the modeling paradigm used and the system developer's approach. So the model generator separates these details from the user and allows them to develop proper models without as much concern about the implementation.

6 EXAMPLE MODEL TEXT-FILE

Consider a simple system that has one machine and jobs arrive individually with an exponentially distributed time between arrivals of 2 hours. Jobs are processed in the order that they are received and the mean processing time is also exponentially distributed with a mean time of 1.5 hours. We want to collect machine utilization, number of processed jobs in 1000 hours and the total across all 5 replication runs, and the average time in the system for each job. In addition, we would like to see how the system WIP varies over time (we will capture this data from the last replication). We need to define: Resource 1 with a capacity of 1 via Resource(1,1,0, 'mach'), one statistical collector Histogram(1, 'CT'), one time persistent data collector TimeHistogram(1, 'WIP') and one entity attribute via nEntAttrs = 1 to keep the job (entity) entry time into the system so that cycle-time CT can be computed. Once the model text-file has been developed (here labeled Model1.txt, illustrated in Figure 2), then from the MATLAB command line the user types:

```
SimModelParser('Model1.txt', 'SimModel1.m')
```

The result is the simulation master m-file (SimModel1.m) that can be executed from either the command line or within the MATLAB editor by selecting the RUN arrow. The output results appear in the command window and can be captured and stored using Windows Notepad. Notepad is recommended because of the line spacing that MATLAB includes in the command window output and Notepad condenses this and omits the inserted blank lines. Graphics results appear in the MATLAB graphics windows which can be easily viewed and if desired captured and/or printed separately. The full listing of this MATLAB function to setup and run the simulation which would results from the model generator for the example model is too extensive to list here. However, the section for the MATLAB switch/cases generated code is illustrated in Figure 3. The results include calculated average and 95% confidence limits. Using the features in MATLAB, the model can output results for WIP and cycle time as a function of time. Figure 4 shows the plots generated for cycle time and WIP for the example. Table 2 shows the example model summary statistics from the simulation run with 5 replications.

Table 2: Summary statistics from simulation run.

Reached End of Simulate: System Measures					
Histogram Average and 95% Confidence Limits					
Hist	Var.	repls.	avg.	LB95%	UB95%
1	CT	5	5.2700	3.5614	6.9785
Average Resource Utilization and 95% Confidence Limits					
Res.	Name	repls.	Util.	LB95%	UB95%
1	mach	5	0.7462	0.68513	0.8073
Average Time Persistent Variables and 95% Confidence Limits					
Var.	Name	repls.	Avg.	LB95%	UB95%
1	WIP	5	2.6442	1.7178	3.5706
Total Th: 2478					

```
    Program Start
    % Single Workstation Model with One Machine
    Control
    NumRepl = 5;
    StopTime = 1000;
    nEntAttrs = 1;
    PrintRepls = 1;

    System Definitions
    Resource(1,1,0,'mach');
    Histogram(1,'CT');
    TimeHistogram(1,'WIP');
    % user variables not reset each replication
    totTh = 0;

    Variable Definitions
    WIP = 0;
    Th = 0;
    PltWIP = [];
    PltCT = [];

    Replicate Outputs
    fprintf(' Repl Th: %3i \n',Th);
    totTh = totTh + Th;

    Final Outputs
    figure(1)
    plot(PltCT(:,1),PltCT(:,2));
    title('CT by Completed Job');
    xlabel('Completed Job');
    ylabel('CT');
    fprintf(' Total Th: %3i \n',totTh);
    figure(2)
    plot(PltWIP(:,1),PltWIP(:,2));
    title('System WIP over Time');
    xlabel('Time');
    ylabel('WIP');

    Time Persistent Logic
      if rand < 1/3
        PltWIP = [PltWIP; [ClockTime,WIP]];
      end

    Model Logic
    C1: Arrive(2,3/6,SVmany,C1);
        Mark(1);
        WIP = WIP + 1;
        Seize(mach,1);
        Wait(2,4/6);
        Release(mach,1);
        Th = Th + 1;
        CT = ClockTime - AttrValue(1);
        Observation(1,CT);
        PltCT = [PltCT; [Th,CT]];
        WIP = WIP - 1;
        Depart(0);

% END
```

Figure 2: Model Text File.

```
while SVloc > 0
  switch SVloc
     % user simulation commands follow in sequence as cases
     case 1
        SVloc = Arrive(2,3/6,SVmany,1);
        Mark(1);
        WIP = WIP + 1;

     case 2
        SVloc = Seize(mach,1);

     case 3
        SVloc = Wait(2,4/6);

     case 4
        SVloc = Release(mach,1);
        Th = Th + 1;
        CT = ClockTime - AttrValue(1);
        Observation(1,CT);
        PltCT = [PltCT;[Th,CT]];
        WIP = WIP - 1;

     case 5
         SVloc = Depart(0);

     otherwise
         disp('Error: no case match for SVloc value')
         disp(SVloc)
         disp('Simulation Aborted')
         return

  end % end of switch/cases
end   % end of while SVloc > 0
```

Figure 3: Section of the MATLAB switch/cases generated code.

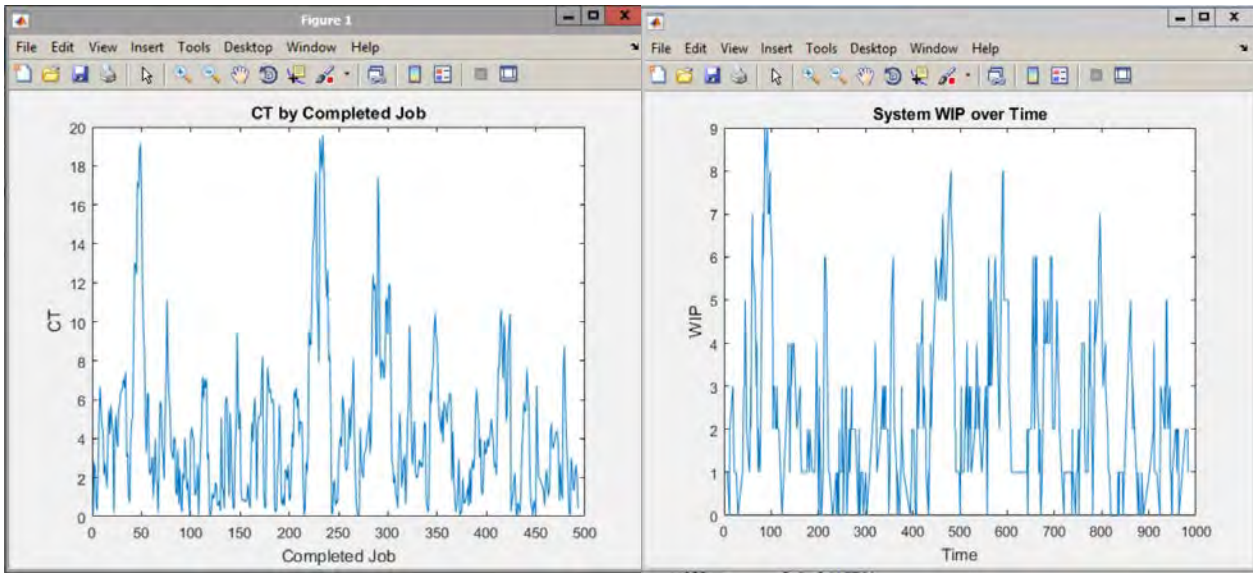

Figure 4: Cycle time and WIP plots from simulation run.

7 CONCLUSIONS

The MATLAB system is an excellent support system for the development of a discrete-event simulation modeling language. Most mathematical science orientated students have a basic knowledge of MATLAB or a similar matrix orientated language such as Mathematica or Maple. MATLAB affords the modeler a variety of language structures (for, if, while, etc.), standard data structures (matrices and cell arrays), graphics support, and printing and storage capabilities among other support functions.

As this language is an initial implementation, no attempt has been made at optimizing the code. This language is not fully developed in the sense of the commercial languages such as GPSS, ARENA, and Simio. This is a base structure that can be enhanced by the user but has the standard fundamental features. Advanced features that would be nice to add include bundle/unbundle (temporary combine) and material handling structures. The language is merely a MATLAB function and as such can be subjugated to a high level function such as an optimization schema or as discussed here a parser function to read a model file and populate the simulation main function program. The simulation system script file (m-file) is open so that modelers interested in specializing or even generalizing the system have full access to all of its functions and features. The system script file can be used in "closed mode" by students starting to learn about discrete event simulation, and are in the process of building, testing and running their first set of simple models. In the "open mode", advanced students and other modelers can add custom features that are required for developing specialized models. They can also use the system as part of another simulation language by writing custom scripts.

The simulation model generator is a very useful tool in that it allows the modeler to concentrate on the model building aspects of the simulation and not to be too concerned with the implementation aspects of the modeling system. There are model segments within the model generator system to allow the user to accomplish most modeling aspects without resorting to directly coding in the master simulation function. However, this also direct coding is available to the modeler if the need arises. The example problem illustrates a couple of these special features such as capturing data across replications (totTh) and graphing time-persistent data and using only about 1/3 of the many observations (PltWIP). This whole system is developed within MATLAB including the model generator. However the user can create model files using Notepad or a similar text editing software and then open them within the MATLAB editor to incorporate changes and corrections in an interactive environment.

REFERENCES

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2000. *Discrete-Event System Simulation.* 3rd ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Curry, G. L., B. L. Deuermeyer, and R. M. Feldman. 1989. *Discrete Simulation: Fundamentals and Microcomputer Support*, Holden-Day Inc.

Curry, G. L., and A. Banerjee. 2016. "A Discrete Event Simulation Language in MATLAB." In *Proceedings of the 2016 Industrial and Systems Engineering Research Conference*, edited by H. Yang, Z. Kong, and MD Sarder, Anaheim, CA, May 21-24, 2016. Available on line at http://dx.doi.org/10.13140/RG.2.1.2380.7601.

Gordon, G. 1961. *A General Purpose Systems Simulation Program.* Proc. EJCC, Washington, D. C., New, York: Macmillan Publishing Co., Inc.

Kelton, W. D., R. P. Sadowski, and D. A. Sadowski. 1998. *Simulation with Arena.* WCB-McGraw-Hill.

Kelton, W. D., J. S. Smith, and D. T. Strurrock. 2013. *Simio and Simulation: Modeling, Analysis, Applications.* 3rd ed., Simio LLC. (www.simio.com)

Kuncicky, D. C. 2004. *MATLAB Programming*. Prentice Hall Engineering Source (E-Source). (www.mathworks.com)

Moya, H., G. L. Curry, and D. T. Phillips. 2015. "U-line Assessment Heuristic & Evaluation Model." In *Proceedings of the 2014 Industrial and Systems Engineering Research Conference*, edited by S. Cetinkaya, and J. K. Ryan, Nashville, TN, May 30 – June 2, 2015.

Pegden, C. D. 1983. Introduction to SIMAN, Proceedings of the 1983 Winter Simulation Conference, Arlington, VA.

Pegden, C. D. 1986. *Introduction to SIMAN with Version 3.0 Enhancements*, Systems Modeling Corporation.

Sklenar, J. 2013. *Tool for Discrete Event Simulation in MATLAB.* Proceedings of the 27th European Conference on Modeling and Simulation, Editors: Webjorm Rekdalsbakken, Robin Bye, and Houxiang Zhang, Alesund, Norway.

Schriber, T. J. 1974. *Simulation Using GPSS*, John Wiley & Sons, Inc., New York.

The MathWorks Inc. 2005. *Simulink: A Program for Simulating Dynamic Systems, Users Guide.* (www.mathworks.com)

AUTHOR BIOGRAPHIES

GUY L. CURRY is a Senior Professor of Industrial and Systems Engineering at Texas A&M University. He holds a Ph.D. in Industrial Engineering from University of Arkansas. His current research interests include applying operations research methods to production systems, and the development of modeling systems for optimization and simulation. His email address is g-curry@tamu.edu.

AMARNATH BANERJEE is an Associate Professor and Corrie and Jim Furber '64 Faculty Fellow of Industrial and Systems Engineering at Texas A&M University. He received his Ph.D. in Industrial Engineering and Operations Research from the University of Illinois at Chicago. His research interests are in modeling, simulation and visualization, with applications in manufacturing, health care, and information systems. His email address is banerjee@tamu.edu.

HIRAM MOYA received his Ph.D. from Texas A&M University, and currently is an Assistant Professor in the Manufacturing and Industrial Engineering Department at the University of Texas Rio Grande Valley. His research interests include queueing theory, optimization, simulation, operations research, supply chain management and applied probability in the areas of homeland security, healthcare, green energy and engineering education. His email address is hiram.moya@utrgv.edu.

HARRY JONES is Senior Professor in the Zachry Department of Civil Engineering at Texas A&M University. He holds a Ph.D. from the University of Illinois. His research interests include the simulation of complex structural systems subjected to rare environmental forces. His email address is h-jones@tamu.edu.