

## CSIM19: A POWERFUL TOOL FOR BUILDING SYSTEM MODELS

Herb Schwetman

Mesquite Software, Inc.  
PO Box 26306  
Austin, TX 78755-0306, USA

### ABSTRACT

CSIM19 is the latest version of the system modeling toolkit from Mesquite Software. CSIM19 offers many features that enable a modeler to develop robust and realistic models of complex systems. These models represent the system as a collection of processes and resources; in most cases the processes mimic the behavior of the entities of the system as they compete for use of the system's resources.

This paper presents CSIM19 and the new features in CSIM19. These features include new capabilities for managing processes at facilities, enhanced handling of messages at mailboxes, and performance improvements. Using the OptQuest optimization package with CSIM19 is also discussed. The talk concludes with an extended example.

### 1 INTRODUCTION

Many systems analysts find that they need to construct and then use simulation models of the systems being analyzed. The primary reason is that they need to understand and predict the behavior of this system. Often, these systems are very complex, both in terms of the number of components in the system as well as in terms of the types of behavior of the active entities in the system.

CSIM19 is both a simulation engine and a set of classes and methods, together with a process structure and support functions and procedures. Programmers can write C++ (or C) programs that use the underlying library and header files to create realistic simulation models of complex systems. Based on the prior version, CSIM18, CSIM19 provides extensive capabilities for modeling sets of resources with facilities being scheduled as a unit; an example is a set of facilities in which customers are moved between waiting lines so as to move to the facility with the shortest queue, in a dynamic fashion.

This paper gives the CSIM approach to implementing models of systems. It briefly describes the OptQuest opti-

mization package, which is an "add on" for CSIM19. It concludes with an example that illustrates some of the advanced features found in CSIM19.

### 2 CSIM19 MODELS

In many cases, a simulation model mimics the behavior of a real system. The usual technique is to represent the active elements of the system with entities and the passive elements of the system as resources. The resources are often implemented in the model as queues (consisting of one or more servers and an internal queue for waiting entities). CSIM19 is a toolkit for constructing simulation models of complex systems. In a CSIM19 model, the active elements of the system are represented by CSIM processes, and the passive elements by resources such as facilities and storages. A CSIM19 model of a complex system is a C or C++ program in which CSIM19 processes mimic the behavior of the active entities of the system as they compete for use of resources of the system. The statistics collection features of CSIM19 give insight into the performance of the system as reflected in the response times and flow rates of the processes and the utilization of and contention for the resources.

Creating a model of a system requires specifying the set of processes and resources which will be used to model the important components of the system. An example will assist in illustrating these concepts. There is a (fictitious) new airport for a small (fictitious) city. One important issue is determining the number of gates required for this airport. As a first cut, a simple CSIM19 model is designed to give a quick answer. In this model, each arriving airplane is represented by a CSIM process. The collection of gates is represented by a single facility with multiple servers; each server represents a gate. An arrival generator process generates arriving airplanes at the pre-specified rate. The complete program is shown in Appendix A.

In this model, the facility is named *gates*. The two key processes for this model are as shown in Figure 1.

```

facility_ms *gates;
. . .
void airplaneGenerator()
{
    create("gen");
    while(TRUE) {
        airplane();
        hold(exponential(airplaneInterarrivalTm));
    }
}

void airplane()
{
    create("airplane");
    gates->use(uniform(minGateTime,          max-
GateTime));
}

```

Figure 1: Listings of Generator and Airplane Processes

In the generator process, the *while* loop invokes one instance of the airplane process and then delays for a computed amount of simulated time to pass (the *hold* statement); this sequence of airplane arrivals and delays simulates arrivals of airplanes at the airport. The computed interarrival intervals are each derived from a negative exponential probability distribution with mean as specified.

Each instance of the airplane process “uses” a gate; the gate holding times (also called service times) are each derived from a uniform probability distribution with the range of values as specified. The *use* method for the multi-server facility causes the process (using the facility) to wait until one of the eleven gates (servers) is free; it then assigns a free server to the process, causes the processor to delay for a computed service interval and then releases the server (makes it free again) and allows the process to continue. The output for this model is shown in Figure 2.

This output shows that, on average, an arriving airplane waits for about 12 minutes (the average response time minus the average service time) and there are an average of about 1.8 airplanes waiting to acquire a gate (the average queue length minus the utilization). Clearly, the airport will need more than eleven gates.

The major features of a CSIM19 model as shown in the example are as follows:

- a CSIM *process* is a procedure which executes the *create* statement;
- CSIM processes operate in an asynchronous parallel manner, mimicking the behavior of multiple entities which are active at the same time
- CSIM has a number of functions which use a random number generator to return values which appear to be drawn from a specified probability distribution; in the example, the exponential and the uniform distributions are used to model the distributions of airplane interarrival intervals and gate holding times respectively
- a multi-server *facility* has one or more servers and a single queue for processes waiting to gain access to one of the servers
- statistics on the usage of a facility are automatically collected; these are presented as part of a report
- the *hold(t)* statement causes simulated time to pass in the life of the model; the amount of time is specified by the value of *t*; *hold(t)* statements are executed by processes

Sat Jul 28 12:34:25 2001

```

Ending simulation time:      1080.000
Elapsed simulation time:    1080.000
CPU time used (seconds):    0.001

```

## FACILITY SUMMARY

facility name	service disc	service time	util.	through-put	queue length	response time	compl count
gates	fcfs	58.73686	8.974	0.15278	10.80909	70.75044	165
> server 0		53.04960	0.884	0.01667			18
> server 1		62.74292	0.871	0.01389			15
> server 2		62.90573	0.932	0.01481			16
> server 3		58.01635	0.913	0.01574			17
> server 4		59.24680	0.823	0.01389			15
> server 5		58.47125	0.812	0.01389			15
> server 6		56.29535	0.834	0.01481			16
> server 7		56.86826	0.790	0.01389			15
> server 8		62.10498	0.690	0.01111			12
> server 9		58.56655	0.759	0.01296			14
> server 10		59.83868	0.665	0.01111			12

Figure2: Output from Example

### 3 FEATURES OF CSIM19

CSIM19 has several other features which are used in other models, as required by the structure and operation of the model; so these other features include the following:

- *storage*: a pool of tokens and a queue for processes waiting to *allocate* some of the tokens
- *event*: a two-state variable and a queue for processes waiting for the event to “occur”
- *mailbox*: a list of unreceived messages and a queue for processes waiting to *receive* a message; other processes can *send* a message to the mailbox at any point in time
- *table*: a structure used to hold data which will be presented as part of a *report*; the data in a table is *recorded*
- a table can also be used in conjunction with the automatic *run length control* feature [Schwetman, 1997] to guarantee that a model is presenting statistically valid results

All of the features, functions, procedures, constants and objects in CSIM19 are described in the CSIM19 User’s Guide [Mesquite, 2001]

So far, all of the features described were part of CSIM18, the earlier version of this simulation package. Some of the features which are new in CSIM19 are as follows:

1. Facilities, storages, events and mailboxes each have a queue for waiting processes. In some cases, it is desirable for a supervisory process to be able to manipulate the processes waiting in a queue. Examples include jockeying (so a waiting process in one queue can be moved to another queue) and queue reordering (to age or adjust the priorities of processes in the queue). CSIM19 has a set of functions (or methods) which can allow these kinds of behavior to be implemented in a model
2. A mailbox has, in addition to a queue for waiting processes, a queue for messages waiting to be received. On occasions, it is desirable for a process to be able to manipulate the messages in the message queue. CSIM19 has functions and methods to let programs do this.
3. In many models, it is convenient to have an array or set of simulation objects. In CSIM18, there is an *event\_set* structure. In addition to being an array of individual events, a process can execute the *wait\_any* statement. With this statement, a waiting process is resumed when one of the events in the set is set by another process; furthermore, the process receives the index of the event in the array which was set. This concept of having a set of objects and operations which deal with the entire set has proven to be useful in many kinds of mod-

els. The following kinds of aggregated objects are part of CSIM19:

- a. *facility\_set* – a facility set can have one of a number of scheduling strategies, including *choose-the-facility-with-the-shortest-queue* (with or without queue-jockeying), *choose-the-facility-with-the-lowest-utilization*, and *get-a-server-at-all-of-the-facilities-at-one-time*
  - b. *storage\_set* – a storage set can have one of a number of scheduling strategies including *choose-the-storage-with-the-shortest-queue* (with or without queue jockeying) and *choose-the-storage-with-the-lowest-utilization*.
  - c. *event\_set* – a process can wait for any of the events in the set to be set and a process can wait for all of the events in a set to be set.
  - d. *mailbox\_set* – a process can wait to receive a message which is sent to any mailbox in the set of mailboxes
4. In some cases, it is useful to collect queueing statistics at events and mailboxes; in CSIM19, collecting queueing statistics can be specified for any event or mailbox; in addition, collecting statistics on delays experienced by messages in a mailbox can also be specified

As can be seen, CSIM19 is a comprehensive modeling tool, with a full range of capabilities for addressing a multitude of analysis scenarios. In addition, the underlying simulation engine [Schwetman 1996] is very efficient, allowing models to execute at rapid rates.

### 4 CSIM19 AND OPTQUEST

CSIM19/OptQuest combines the powerful simulation package, CSIM19, with a state of the art optimization package, OptQuest [Glover, 1999], to allow users to build models which can automatically find the best system configuration [Schwetman, 2000]. Briefly, a CSIM19 model can be designed so that critical aspects of the configuration of the model are provided as inputs to the model. In addition, the model yields results as outputs from the model. Such a model can be setup with components of the OptQuest package so that the OptQuest run supervisor can call the model with a specific configuration and then assimilate the results of a run to evaluate the configuration and try out additional configurations, searching for the best configuration.

### 5 AN EXAMPLE

The example in an earlier section was extended to become a more realistic model of an airport in operation. The two changes are as follows:

- The arrival rate for airplanes was changed from a single overall rate to a set of hourly arrival rates, one per hour, to more accurately model the fluctuations in arrival rates over the day, and
- The single airport facility with multiple indistinguishable gates was enlarged to become a set of facilities: gates, tugs and air conditioning units; an arriving airplane needs one gate, one air conditioning unit, both for the entire gate time, and one tug for 10 minutes after arrival and one tug for 10 minutes prior to departure (to tow the airplane to the gate and to push the airplane back from the gate).

The non-constant mean arrival rates were implemented using a vector of mean arrival rates and by modifying the `airplaneGenerator` process, as shown in Figure 3

```
const long arrivalsByHour[hoursPerDay] =
    { 2, 6, 10, 12, 8, 7, 10, 11, 6, 7, 9, 20,
      21, 12, 10, 12, 7, 3};

void airplaneGenerator()
{
    long hour;
    double iarTime;

    create("gen");
    while(clock < dayTime) {
        numAct++;
        airplane();
        hour = (long)(clock/60.0);
        iarTime = 60.0/arrivalsByHour[hour];
        hold(exponential(iarTime));
    }
    if(numAct == 0)
        done->set();
}
```

Figure 3: Modeling Varying Arrival Rates

The extended set of facilities was modeled using a *facility\_setx*. Each of the components of the airport (gates, tugs, and airconditioners) are first modeled as a multi-server *facility*. Then, each of these are “added” to the *facility\_setx*. When an airplane arrives, it executes the *reserve\_all* method; this method delays the airplane process until it can reserve one of each of the component facilities. The process then does a *hold* for an interval corresponding for the tug service time (10 minutes). It then releases the tug facility and does a hold for the remainder of the gate time (minus the push-back time). Prior to departure, the process acquires another tug, delays for the tug service time (10 minutes) and then executes the *release\_all* method, freeing the gate, airconditioner and tug it had acquired. The listings of these modified routines are shown in Figure 4.

```
void initDay()
{
    facility_ms *f;

    airport = new facility_setx("airport", 3);
```

```
    f = new facility_ms("gates", numGates);
    airport->add_facility(f);
    f = new facility_ms("tugs", numTugs);
    airport->add_facility(f);
    f = new facility_ms("a/c", numGates);
    airport->add_facility(f);
    done = new event("done");
    numAct = 0;
}

void airplane()
{
    double gt;

    create("airplane");
    airport->reserve_all();
    hold(tugTime);
    airport->release(1);
    gt = uniform(minGateTime, maxGateTime) -
        2*tugTime;
    gt = (gt < 0.0) ? 0.0 : gt;
    hold(gt);
    airport->reserve(1);
    hold(tugTime);
    airport->release_all();
    numAct--;
    if(clock >= dayTime && numAct == 0)
        done->set();
}
```

Figure 4: Listings Showing Extended Facility Set

In addition to the enhancements listed above, the model was altered, so as to model many days of operation, as opposed to one day. The output was modified so that the number of arrivals for each day, the average airport utilization (equivalent to the average number of gates in use), the average airport service time per airplane, the average response time per airplane and the average number of planes at the airport, all for each day, are collected and summarized in a set of *permanent\_table* structures. Each day of operation is a separate instance of the model; the *re-run()* statement allows successive instances of the model to be constructed and executed. The output for this model with 15 gates, modeling 365 days of operation appears in the Figure 5.

```
Airport Model;
Number of gates:      15
Number of airconditioners:  15
Number of tugs:      15
Number of days:      365
Hours per day:       18

Average arrivals per day: 171.595
Average service time: 60.125
Average gate utilization: 9.124
Average response time: 69.646
Average number at airport: 10.615
```

Figure 5: Output from Enhanced Model of the Airport

This model could be extended to increase the degree of realism and to obtain additional outputs, such as the distribution of the number of airplanes on the ground at the airport. An obvious extension is to use OptQuest to find

the best combination of gates, air conditioners and tugs so as to “optimize” operations at this airport. These and other enhancements would enable the managers and designers of the airport to develop a configuration of the airport, with its many components, that would meet the operational and performance goals, and to predict the behavior of the airport over time as the operational characteristics change.

## 6 SUMMARY

CSIM19 combines an efficient simulation engine and a comprehensive set of tools that allow modelers to develop realistic models of complex systems. The variety of objects and services provided means that these models can be designed, implemented and tested in an economical manner. Furthermore, the efficient underlying simulation engine means that these models execute quickly. The addition of the OptQuest package extends the usefulness of the package, by allowing the model to automatically search for the best possible system configuration.

The new features in CSIM19 extend the kinds of systems which can be readily modeled. In many cases, users of CSIM18 could have achieved similar results, but now, the ease-of-use provided by the enriched feature set reduces the costs of implementing these models and improves their reliability. CSIM19 will prove to be a valuable tool in the pursuit of better models..

## ACKNOWLEDGEMENTS

CSIM is copyrighted by Microelectronics and Computer Technology Corporation (MCC). CSIM19 is supported and marketed by Mesquite Software, Inc. under license from MCC. Dr. Jeff Brumfield developed the data collection and presentations functions including the run-length control algorithm in CSIM19. OptQuest is a product of OptTek Systems, Inc. It is available for use with CSIM19 from Mesquite Software, Inc.

## APPENDIX: LISTING OF THE MODEL OF AN AIRPORT WITH MULTIPLE GATES

```
// model of airplanes arriving to use a gate
and an airport

#include "cpp.h"

const double simTime = 18*60.0;
const long numGates = 11;
const double interarrivalTime = 6.0;
const double minGateTime = 30.0;
const double maxGateTime = 90.0;

facility_ms *gates;

void init();
void airplaneGenerator();
void airplane();
```

```
extern "C" void sim()
{
    create("sim");
    init();
    airplaneGenerator();
    hold(simTime);
    report();
}

void airplaneGenerator()
{
    create("gen");
    while(clock < simTime) {
        airplane();
        hold(exponential(interarrivalTime));
    }
}

void airplane()
{
    create("airplane");
    gates->use(uniform(minGateTime, max-
GateTime));
}

void init()
{
    gates = new facility_ms("gates",
numGates);
}
```

## LIST OF REFERENCES

- Glover, F, Kelly, J. and M. Laguna, 1999, New Advances for Wedding Optimization and Simulation, *Proceedings of the 1999 Winter Simulation Conference*, Ed. P Farrington, H. Nembhard, D. Surrock and G. Evans, 255 – 259.
- Mesquite Software, Inc. 2001, *CSIM19 User's Guide*, Austin, TX.
- Schwetman, H., 1996, CSIM18 – The Simulation Engine, *Proceedings of the 1996 Winter Simulation Conference*, Ed. J. Charnes, D. Morrice, D. Brunner, and J. Swain, 515 – 521, San Diego.
- Schwetman, H. and J. Brumfield, 1997. Data Analysis and Automatic Run-Length Control in CSIM18. *Proceedings of the 1997 Winter Simulation Conference*. Ed. S Andradottir, K. Healy, D. Withers, and B. Nelson, 687 - 692. Atlanta, GA.
- Schwetman, H., 1999, “Model, Then Build”: A Modern Approach to Systems Development Using CSIM18, *Proceedings of the 1999 Winter Simulation Conference*, Ed. P Farrington, H. Nembhard, D. Surrock and G. Evans, 249-254, Phoenix.
- Schwetman, H., 2000, Optimizing Simulations with CSIM18/OptQuest: Finding the Best Solution, In *Proceedings of the 2000 Winter Simulation Conference* Ed. J. Joines, R. Barton, P. Fishwick, and K. Kang, 268 – 273, Orlando.

**AUTHOR BIOGRAPHY**

**HERB SCHWETMAN** is founder and Chairman of the Board of Mesquite Software, Inc. Prior to founding Mesquite Software in 1994, he was a Senior Member of the Technical Staff at MCC from 1984 until 1994. From 1972 until 1984, he was a Professor of Computer Sciences at Purdue University. He received his Ph.D. in Computer Science from The University of Texas at Austin in 1970. He has been involved in research into system modeling and simulation as applied to computer systems since 1968.