Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

INTRODUCTION TO PROCESS-ORIENTED SIMULATION AND CSIM

Herbert D. Schwetman

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78759-6509

ABSTRACT

CSIM is a simulation package which lets C and C++ programmers write process-oriented simulation models. This tutorial begins by presenting the basic concepts of the process-oriented paradigm. It then proceeds to illustrate this simulation paradigm as implemented in CSIM. Issues of the size and costs of simulation models written using this paradigm are discussed. The tutorial closes with a number of examples drawn from diverse application areas. These examples illustrate the descriptive power of process-oriented simulation models.

1. PROCESS-ORIENTED SIMULATION

There are three different approaches to discrete event simulation (Nance 1981): event-oriented simulation, activity-oriented simulation, and process-oriented simulation (Franta 1977) (Pritsker and Pegden 1979). This tutorial will focus on one package, called CSIM, which embodies the process-oriented approach.

In the process-oriented approach, the simulation programmer composes a set of process descriptions. Each process description serves as a model of one kind of active entity in the simulated system. An active instance of a process description will be called a process. In a simulation system, there is a process management facility which allows processes to become active, to operate in the simulated environment, and to eventually terminate. The process management facility is capable of managing many active processes so that they each appear to be active at the same time. This "pseudo parallelism" for simultaneously active processes is a very important feature of process-oriented simulation.

A real system is modeled, using the process-oriented approach, as collection of processes, each competing for the resources of the system. For example, in a factory, an item to be assembled could be modeled by a process which arrives at the head of the assembly line and then visits each of the stations on the line. At each station, the process, mimicking the behavior of the item it models, waits in a line of waiting items, seizes the station for a service interval, releases the station and then travels to the next station in the assembly line. Some items may have different travel patterns, based on attributes of the item or on the outcomes of different steps in the assembly process, such as the outcome of a test. When assembly of the item is completed, it "leaves" the system (see Figure 1).

In Figure 1, an item to be assembled arrives and travels to station 0. After spending some time (the service interval for station 0) there, it travels to station 1, where it stays for another service interval. On leaving station 1, some of the items visit station 2 and then leave, while the other items travel to station3 and leave. The choice of station 2 or station 3 could be based on the outcome of a test at station 1. In simulation models, it is common to model this choice as a statistical choice, with some predefined percentage of the items traveling to station 2 and the rest of the items traveling to Station 3.

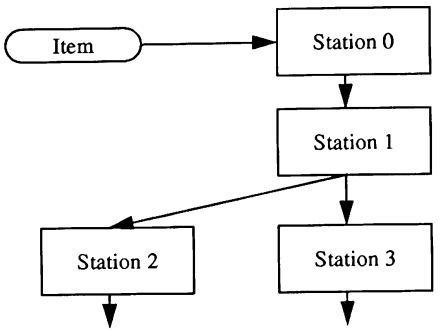

Figure 1. Example of Assembly Line

Notice that this model is specified by the rate that items arrive for assembly, by the times spent at each station and by the choice percentage.

In the preceding example, the processes of the model are the items being assembled, and the resources are the various stations in the assembly line. This approach to modeling, having processes model the behavior of active entities and simulated facilities model the resources of a system, has been shown to be a good approach to creating simulation models. Many kinds of systems have been modeled this way, systems such as computer systems, communications systems, manufacturing systems, transportation systems and service delivery systems.

When constructing a model such as the one in the example, there is always a choice to be made regarding what the processes represent and what the facilities represent. In the example, another model of the same system could be constructed in which the stations are represented by processes and the items are represented by tokens which are passed around among the stations. In some cases this alternative approach might be preferable (for example, when the model needs to focus on the processing being done at each station).

The purpose of constructing and executing such a simulation model is to gain insight into the dynamic properties of the modeled system. Estimates of the average time required to assemble an item (called system response time) and the rate at which items are assembled (called the system throughput rate) are examples of dynamic properties which are of interest to the management of the factory. In addition, information about the average number of waiting items at each station can help locate production bottlenecks. A major goal of developing a simulation model is to allow complex systems to be "tried out" before they are actually constructed. This allows suboptimal configurations of stations to be cast aside, so that the best possible configuration will the one that is constructed. In addition, such a model can be used to try out new schedules, to estimate system response to different arrival rates, and to experiment with alternative

travel paths. Models of existing systems can be used to help find arrangements which deliver better system performance.

2. CSIM

CSIM is a simulation package which allows programmers to write C (Kernighan and Ritchie 1978) and/or C++ (Stroustrup 1986) programs which are process-oriented simulation models. CSIM was developed at MCC, starting in 1984, and has been used at MCC to construct simulation models of computer systems, database machines, communications protocols, and system components (Boughter, et.al, 1987). In addition, CSIM has been distributed to a number of MCC shareholder companies and also to other institutions and companies which have requested a copy. Currently, about 63 different organizations have received one or more copies of some version of CSIM.

A CSIM model is a program (assume it is written in the C programming language) in which some of its internal procedures can become CSIM processes. These processes operate just like the processes described in the preceding section. CSIM processes operate in a pseudo-parallel fashion; they can use facilities, wait for messages to arrive in a mailbox, wait for events to happen, as well as send messages to mailboxes and cause events to happen. In addition, executing a *hold()* statement causes simulated time to pass.

The result is a C program which produces a report on the different aspects of the behavior of this model. This C program is compiled by the C compiler and linked with the CSIM library, to acquire all of the functionality of the CSIM package. Because it is compiled, a CSIM program tends to execute very efficiently. Because it is a C program, a CSIM program has access to all of the tools and features available to any C program executing on the host system.

To illustrate this approach, a CSIM program which implements the model shown in Figure 1 is given in Figure 2.

The example in Figure 2 may require further explanation. The stations of the assembly line are represented by an indexed array of FACILITY'S. The TABLE, resp_tm, is used to collect system response time statistics. The procedure named "sim" is the first procedure of the model to be executed. In CSIM, a process is a C procedure which executes a *create ()* statement. This *create ()* statement is what causes the facility management system to create a new process (using the procedure as the process description) and to activate that process as soon as it can. The *create ()* statement also returns control to the calling process (except in the case of the *sim()* process).

In the example, the *sim()* process initializes the array of stations and the table and then generates 100 arrivals of the "item" process (in the *for* loop). There is a random interval of simulated time between each arrival of *item ()* (caused by the *hold ()* statement). As each "item" arrives, it executes a *create()* statement which creates a new instance of the process *item* and returns control to *sim*. *Sim* executes the next *hold()* statement, at which point the new copy of *item* can begin execution. *"Item"* saves its arrival time, and then "uses" station 0 for a constant interval of time (3.0 time units) and station 1 for 2.5 time units. The *use()* statement checks the status of the station; if it is "busy" (already in use), the process is put in a queue of waiting processes; if the station is "free" (not in use), the process occupies the station and "holds" for the specified interval of time. Whenever a process finishes using a station, the next process in the queue is activated.

After the second *use()* statement, the process draws a random number (uniformly distributed between 0.0 and 1.0. Approximately half of these drawn numbers will be less than 0.5, so station 2 will be visited; the other drawn numbers will be greater than 0.5 and station 3 will be visited. As the process finishes, it "records" the process

```
/* CSIM model of assembly line */
#include "csim.h"
FACILITY station[4];
TABLE resp_tm;

sim()
{
    int i;

    create("sim");
    facility_set(station, "station", 4);
    resp_tm = table("response time");
    for(i = 0; i < 100; i++) {
        item();
        hold(expntl(10.0));
        }
    wait(event_list_empty);
    report();
}

item()
{
    TIME t;

    create("item");
    t = clock;
    use(station[0], 3.0);
    use(station[1], 2.5);
    if(prob() < 0.5)
        use(station[2], 4.0);
    else
        use(station[3], 4.5);
    record(clock - t, resp_tm);
}
```

Figure 2. CSIM Example

response time (time of arrival to time of departure) in the table. When a process "exits", it is automatically terminated.

The first process (sim) "waits" for all of the "items" to finish. It then prints a report and exits. When *sim()* exits, execution of the model has completed. The output from a run of this example appears in Appendix A.

CSIM has a structure called a mailbox; processes can receive messages from a mailbox and send messages to a mailbox. Mailboxes are used in the program shown in Figure 3 to help construct another model of the system shown in Figure 1. This second model implements the alternative approach mentioned in Section 1, in which the stations are represented by processes.

In this alternative approach, each station is modeled by a process and the items to be assembled are modeled by "messages" sent from one station process to another. Each different station "receives" items in its own mailbox, "holds" for the correct service interval and then sends a message (the item) to the next station based on which station is processing the item. If response times are needed, this example would have to be modified, so that the arrival time for each item is part of the item message.

3. USING CSIM

As can be seen in the previous section, CSIM is used by writing a C program and including calls to the procedures and functions in the CSIM library or invoking the CSIM macros. A great effort has been made toward simplifying the writing of CSIM programs. For

```
/* alternative CSIM model of assembly line */
#include "csim.h"
MBOX station_mb[4];
float serv_tm[4] = {3.0, 2.5, 4.0, 4.5};

sim()
{
    int i;

    create("sim");
    for(i = 0; i < 4; i++) {
        station_mb[i] = mailbox("station");
        station(i);
        }
    for(i = 0; i < 100; i++) {
        send (station[0], 0);
        hold(expntl(1.0));
        }
    wait(event_list_empty);
    report();
}

station(i)
int i;
{
    int x;

    create("station");
    while(1) {
        receive(station_mb[i], &x);
        hold(serv_tm[i]);
        if(i == 1) {
            if (prob() <0.5)
                send(station_mb[2], 2);
            else
                send(station_mb[3], 3;
            }
        if(i == 0)
            send(station_mb[1], 1);
    }
}
```

Figure 3. Alternative CSIM Model

example, all variables "local" to each process are automatically maintained by the process management facility. The variable "t" in the *item* process in the program in Figure 2 is an example of such a local variable. Each instance of *item* must have its own version of "t". As another example, statistics on the use of each facility in a model are automatically collected; these statistics are summarized to the standard output file when the *report()* statement is executed.

CSIM programs are efficient in their use of memory. All structures required to support the processes, the facilities, etc. are dynamically allocated. This means that "small models" use small amounts of memory, while "large models" can be executed without modifying the CSIM library. (Of course, the amount of memory available on the host system limits the size of any program executing on that system.)

Finally, CSIM programs make efficient use of computing time. Most of the interval queue-handling algorithms have been optimized in order to execute models with many active processes efficiently. There is some overhead associated with the process management facility, but, in most cases, this is held to acceptable levels.

As stated previously, CSIM has been used to model many kinds of real systems. Some additional examples, showing CSIM modeling different kinds of systems appear in (Schwetman 1986) and (Schwetman 1988). A CSIM Reference Manual is available as an MCC Technical Report (Schwetman 1990).

4. C++/CSIM

C++ (Stroustrup 1986) is an object-oriented programming language which is based on C. A major feature of C++ is capability for programmers to define *classes*. A class consists of data structure and *methods* (procedures and functions) which operate on these data structures. An instance of a class called an object. One class can *inherit* another class. Thus, one class (the base class) can be used to construct another class (the derived class). These and other features make C++ a good language for writing many kinds of programs, including simulation programs.

The CSIM library has recently been extended so that it can be used with C++ programs (Schwetman 1990). This means that C++ programmers now have access to the simulation facilities provided by CSIM. One benefit of using C++/CSIM is that the C++ compiler enforces rigid type checking (most C compilers do not do this). Thus, many common programming errors (such as "holding" for an integer time interval) can be detected by the C++ compiler. A benefit to CSIM programmers comes from the inheritance mechanisms in C++. A C++/CSIM class, such as a facility, a mailbox or an event, can be inherited by a programmer-defined class and given new properties. A trivial example would be to define a new kind of facility which would gather additional usage statistics every time it was "used".

C++/CSIM has been used at MCC to develop a few simulation models. The experience, so far, has been positive. The enhanced type checking alone has caught many errors and, consequently, has reduced the time required to implement and debug the simulation models.

5. SUMMARY

CSIM is a library of procedures, functions and macros which give C (and C++) programmers a powerful tool for developing process-oriented simulation models. Since CSIM models are, in reality, C programs, there are almost no limits as to the size and complexity of these models. The model implementer can write (or import) any algorithms which may be needed to develop an accurate model.

This tutorial has introduced CSIM to the reader. There are many features and facilities which have been omitted, in order to simplify this introduction. All of these are described in the Reference Manual.

APPENDIX

```
Tue Jul 17 10:14:26 1990                CSIM Simulation Report Version 141

Model: CSIM
                              Time:          964.207
                              Interval:      964.207
                              CPU Time:        0.417 (seconds)


                          Facility Usage Statistics

+----------------------+--------------means---------------+---counts----+
facility      srv    disp  serv_tm   util    tput   qlen    resp    cmp     pre

station[0]                   3.000  0.311     0.1  0.369   3.557    100       0
station[1]                   2.500  0.259     0.1  0.259   2.500    100       0
station[2]                   4.000  0.216     0.1  0.226   4.192     52       0
station[3]                   4.500  0.224     0.0  0.248   4.990     48       0


                                Table 1

                          Table Name: response ti

          mean                 10.632                 min          9.500
          variance              2.461                 max         17.863

                          Number of entries 100
```

REFERENCES

Boughter, E., W. Alexander, and T. Keller (1987), "A Tool for Performance Driven Design of Parallel Systems," Technical Report ACA-ST-312-87, Microelectronics and Computer Technology Corporation, Austin, TX.

Franta, W.R. (1977), *The Process View of Simulation*, North-Holland, Amsterdam.

Kernighan, B.W. and D.M. Ritchie (1978), *The C Programming Language,* Prentice-Hall, Englewood Cliffs, NJ.

Nance, R.E. (1981), "The Time and State Relationships in Simulation Modeling," *Communications of the ACM 24*, 173-179.

Pritsker, A.A.B. and C.D. Pegden (1979), *Introduction to Simulation and SLAM*, Halstead Press, New York.

Schwetman, H.D. (1986), "CSIM: A C-Based Process-Oriented Simulation Language," In *Proceedings of the 1986 Winter Simulation Conference*, J.R. Wilson, J.O. Henriksen, and S.D. Roberts, Eds. IEEE, Piscataway, NJ, 387-396.

Schwetman, H.D. (1988), "Using CSIM to Model Complex Systems," In *Proceedings of the 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 246-253.

Schwetman, H.D. (1990), "CSIM Reference Manual (Revision 14)," Technical Report ACA-ST-257-87, Microelectronics and Computer Technology Corporation, Austin, TX.

Stroustrup, B. (1986), *The C++ Programming Language*, Addison-Wesley, Reading, MA.