Generic Discrete Event Simulation Environment

Nagender Parimi

Computer Science and Engineering,
Netaji Subhas Institute of Technology (formerly Delhi Institute of Technology),
Delhi *University*

ABSTRACT

We have designed and implemented a generic discrete event simulation environment in Java using Java libraries capable of simulating processes like job shops, production line environments, computer hardware/software, and a social system composed of interacting individuals.The generic nature of the environment suggested and developed facilitates its application in a variety of fields, from traditional traffic control systems to modeling mirrors for websites and ecommerce based systems.

Approach used :

The underlying approach used here is the discretisation of seemingly continuous events into a series of distinct and discrete ones.Any given process may be broken down into a series of discrete events.These events may then be simulated applying the suggested techniques,providing a convenient and accurate tool for the analysis of the required environment.

The Design allows for:

- A simulation environment to be modeled by specifying the various servers, the queue protocols feeding them, and the objects undergoing processing.
- Two or more objects to combine to form composite objects (modeling assembly line processes), and also for situations where objects hitch on to other objects for some duration of the processing (passengers alighting a bus and getting down later).
- Specifying the start and end time of a simulation and also the probability distribution function that controls the generation of the objects undergoing processing.
- Collection of statistics like queue lengths, delays at various servers, total processing time for objects, server idle and decide upon the field strategies.

We envisage that the statistics collected at the end of the simulation will help the administrator to study the interplay between different parameters and design the actual environment with maximal optimality.

A working prototype of the simulator is also presented herewith .

Keywords: Stochastic complexity, probabilistic models, Generic simulation, optimization , discretisation.

1. Introduction to the design

1.1 We build the entire simulator with **3 abstract classes** as the building blocks.

Define class ABSTRACT_SERVER{

Fields-----

1) Idle time of the server
2) Queue (to be served)
3) Queue length
4) Constructor
5) Serving time
6) Object (the current object it is servicing)

}

Define class ABSTRACT_OBJECT{

Fields------

1) Path
2) Time of entering the queue
3) Time at which the servicing starts
4) Time of creation

}

Define ABSTRACT_EVENT_MANAGER {

Fields------

1) Time (for the next event) *The policy followed here would be of Next Event Instead of Time Slicing*\
2) Server (this field will keep the server number which is going to complete the servicing at that particular time. If this number is –1 that will mean that it is an object generation event./so at that time a new object will be generated and initialized and sent to the queue of the 1st server specified in its path)

}

1.2 Inputs taken from the administrator for simulation

1) The number of servers
2) The number of paths required
3) The number of servers in each path.
4) The paths that the object is probable to follow (One of these paths would be allocated to every object when it is created)

5) The mean and the variance of the servicing time (This will help in calling the random (Guassian) function. In case of equal probability analysis, user inputs the minimum and maximum time limits for service time.
6) The Time gap after which new objects will be generated
7) The Starting and the Termination time of the simulation

1.3 Outputs at the end of simulation

- All the fields of every server (viz. idle time, servicing time and others as mentioned in the class specification)
- All the fields of each object that was serviced (viz. idle time, total servicing time and others as mentioned in the class specification)

The objective of the simulation will be to study the interplay between different parameters, which influence design of an environment, and to investigate how optimality can be achieved in them.

2 An Overview of Environment and Design Requirements

2.1 Types of Simulation

There are three major ways to approach discrete simulation. These are event scheduling, activity scanning, and process orientation. Each approach is adopting by some programming language, and, more importantly, offers a different way to look at a simulation problem. Each, in its own way, suggests mechanisms to model real situations. In our prototype we have used *"Event Scheduling*".

2.2 Comparing two systems

Suppose now we have two systems and we wish to determine which is better, based on some statistic. The simplest way to analyze these systems is to ensure (or force) the number of observations for the two systems to be the same. Suppose we get $x_1, x_2, ... x_n$ for system 1 and $y_1, y_2, ... y_n$ for system 2. Then to see if system 1 is better than system 2, we need only determine if **x-y > 0**. In other words, we can form $z_i = x_i - y_i$, calculate $\bar{Z}$ and S_Z^2 and form a confidence interval on **Z**. If the confidence interval in entirely among the positive numbers, then system 1 is better. If it is entirely among the negative numbers, system 2 is better. If it includes 0, then it is not possible to say.

This analysis holds as long as the z_i are independent; it is not necessary for x_i and y_i to be indepedent from one another. In fact, if x_i is positively correlated with y_i then the variance on **z** *decreases.* One standard simulation technique is to force positive correlation by using the same input random numbers in the run of the two systems. If a system has higher than normal wait time (due perhaps to a large number of arrivals in a short period) then an alternative will also likely be higher than normal.

There are standard statistical methods in the case that the number of observations are not the same (and assuming you are not willing to throw away data). Since there is nothing special about simulation here, I will simply refer you back to your statistics notes.

2.3 The Role Of Random Numbers

One aspect of simulation that is often confusing is the role of random numbers. How can the computer generate randomness? And how can ``random" simulations be repeated over and over again. In some sense, the confusion is justified, for a computer cannot generate true randomness. It can only generate *pseudo--randomness*.

Pseudo--random numbers can be generated many ways. We approach the problem in two ways. We suggest 2 naïve models for generating the random numbers.

2.3.1) Equal Probability

The most common is by a *linear congruential method*, a complicated word for a simple concept. Let's suppose I want to generate random numbers betwee 0 and 15 (integers only). We will need to begin with a single number, perhaps created by rolling a 16 sided die.

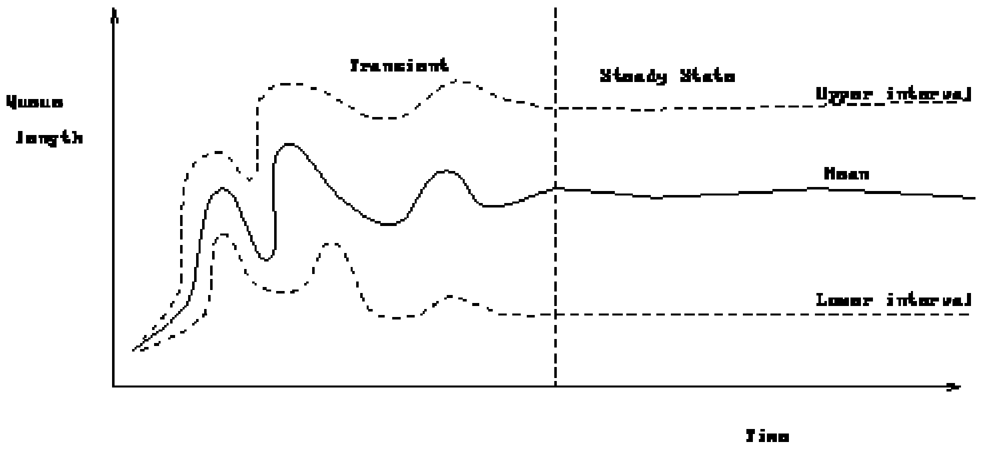
2.3.2) Gaussian function

The formula for generation of random numbers in this scheme is:-

F(Y)= 1/(1.414*3.1416*V) * exp[-(Y-M)*(Y-M)/2*V*V]

Where
M=mean of given input,
V=variance of given input
Y=input(generated random number)
F(Y)=probability density function

A good method for calculating the mean and variance for the analyses is falling back to the fundamentals of statistics.



The first goal is to identify the transient period so we can throw it away. There is no good statistical test to positively identify transient periods, so this act is really a judgement call. Graphing the data is one good way to see the transient period, but people often mistake extreme, but valid, values for the transient period. The steady state period need not be one where the system has ``settled down''. In fact, typically the variance of the readings is higher in steady state than in the trasient period! Steady state is defined by having the mean value be independent of time. If we draw some lines to denote the variance, we might see something like that in figure. Once we have thrown out the transient period, we are left with trying to analyze the steady state.

3. ALGORITHM

What follows in this sub-section is the algorithm in its entirety, which goes into the design and simulation of a discrete event environment.

//Read input from user

Define class ABSTRACT_SERVER{

Fields-----

1) Idle time of the server
2) Queue (to be served)
3) Queue length
4) Constructor
5) Serving time
6) Object (the current object it is servicing)

}

Define class ABSTRACT_OBJECT{

Fields------

1) Path
2) Time of entering the queue
3) Time at which the servicing starts
4) Time of creation

}

Define ABSTRACT_EVENT_MANAGER {

Fields------

1) Time (for the next event) *The policy followed here would be of Next Event Instead of Time Slicing*\
2) Server (this field will keep the server number which is going to complete the servicing at that particular time. If this number is –1 that will mean that it is an object generation event./so at that time a new object will be generated and initialized and sent to the queue of the 1st server specified in its path)

}

//Initialize AEM Vector

```
for (I ß 1 to (number of objects to be created) ){
AEM[I].server  ß  -1;
```

```
AEM[I].time ß T1+it;
}

While (present_time< (Simulation end time) ){
        aem ß Next AEM in AEM Vector;
        //aem keeps track of present AEM



//START SIMULATION

If (aem.server = -1) {
//Initialize new object
obj.path  ß  Randompath;
obj.tcreated  ß  Present time;
ser  ß  obj.path[0];
Insert object in Q of server;

//Create new aem
new_aem.server ß ser;
new_aem.time ß present_time;
Insert new_aem in AEM[] such that
Next_AEM[] ß new_aem;       }//end if

Else
{
        //if aem.server != -1
        sever_time  ß  random();

        //server changes
        aem.server.stime += service_time;
        present_object  ß  aem.server.Q[front];

        //Delete Q[front]
        front ß front + 1;

        //object changes
        present_object.stime += service_time;
        new_server  ß  present_object.path[index];
        index ++;
        new_server.Q[rear +1] ß present_object;

        If (rear = front)          //new server.Q was empty
        {
                //Create new_aem
                new_aem.server  ß  new_server;
                new_aem.time ß present_time + service_time + 1;
                Insert new_aem in AEM[];
        }

        If present_object.path[index] = 0       //next server in object path is zero
                {
                        output(present_object);
                }

      }

        //Create new_aem
        new_aem.server ß present_server;
        new_aem.time ß present_time + service_time + 1;
        Insert new_aem in AEM[];
        Remove aem from AEM[];
}       //end while

//Output Object Details
for (I ß 1 to (number of objects) ){
print (object ß number);
print (object ß stime);
print (object ß path);
print (object ß t_created);
}

//Output Server Details
for (I ß 1 to (number of servers) ){
print (server ß number);
print (server ß stime);
print (server ß Q);
print (server ß t_created);
}

//END OF SIMULATION
```

4. Applications

The scope of the presented simulation environment is unlimited. It can be used to model any environment that can be discretized into a series of events. The following few high level applications should be able to demonstrate the usefulness of the Generic Discrete Event Simulator.

4.1 Modeling of Traditional Traffic Control Systems

Eg. Control of trains arriving at the New Delhi junction. The server class is now inherited by various platforms. The queuing objects in the plan would be the trains arriving at the platforms.

4.2 Modeling mirrors for websites, for better NET-traffic control

Mechanized modeling (division) of Internet traffic over a group of mirrors for the same site can be optimized for better results in terms of managing net traffic, reducing idle time (and overload) of any particular mirror server etc. Further, the simulator can be used to decide the world wide location of the mirrors for minimum Bandwidth usage. In this case, the mirrors would be instances of the server class and the individual HTTP requests would be the instances of the Abstract object.

4.3 E-commerce based applications

The simulator can be used in determining the strategic locations for distribution outlets of B2B/B2C web sites. Here, the outlets would be the instances of the Abstract Server and the individual online orders would be the instances of the abstract object.

5. Some Working Examples

We present the implementation of two real-life problems that were simulated on the simulator. Optimality is evidently achievable through such simulations.

5.1 A Bank Teller Machine

A bank is planning on installing an automated teller machine and must choose between buying one Zippy machine or two Klunky machines. A Zippy costs exactly twice one Klunky to buy and operate, so the goal of the bank is simply to provide the best service.

From the data available, it appears that customers arrive according to a Poisson process at the rate of 1 per minute. Zippy provides service that is exponential with mean .9 minutes. Each Klunky provides service that is exponential with mean 1.8 minutes. We will assume that customers lined up for the the two Klunkies will form a single queue. The performance measure we will use is the average time waiting in the queue for the first 100 customers (the bank has decided it is most irritating to wait and customers are pacified if they are being served). Should the bank buy one Zippy or two Klunkies?

One method would be to install one Zippy for a few weeks, measure the average wait, and then rip it out and install two Klunkies and measure their wait. If necessary, then, the Klunkies could be ripped out, and the Zippy reinstalled.

Simulation, of course, gives a much more appealing solution. We can simply create a computer simulation of the above experiment. To do this by hand, we would generate (perhaps by using a table of random numbers) a series of arrival times and service times and determine how long the wait was. For instance, we might end up with arrival times of .2, .7, 1.6, 2.3, 3.4, 5.6, and so on and service times for Zippy of .9, .7, 2.3, 1.6, .1, .6, and so on (and double that amount for Klunkies). The simulation for one Zippy would then have a customer arrive at time .2 and go right to the machine. At time .7 a customer arrives and waits in line. At time 1.1, customer 1 leaves, and customer 2 uses the machine (having waited .4). Customer 3 arrives at 1.6, customer 2 leaves at 1.8, allowing customer 3 (after having waited .2) to use the machine (total wait so far is .6). And so on. Similar analysis could be done with the two Klunky system. Fortunately, we can have this simulator do all of the work for us.

6. Designing simulation experiments

The final stage in analyzing a system with simulation is to optimize over the system parameters in such a way as to optimize performance. As stated earlier, simulations is fundamentally a descriptive tool, not a prescriptive tool. With care and a lot of computer time, it is possible to optimize with simulation. In many ways, optimizing with simulation is very similar to nonlinear optimization.

First, let's suppose we have a single quantity to optimize over. This might be the number of machines to place in a plant, the number of units at which to reorder in an inventory system, or the number of dollars to put into a particular portfolio. Let this quantity be denoted **x**. Suppose you have a simulation that will determine the performance given **x**, denoted (f(x1) is better than f(x2)) are made correctly. Because these tests must be made repeatedly, either the allowance for error must be very, very small, or some allowance for making an error must devised.

This is particularly true when working with more than one parameter. In this case, it is not even clear how to use nonlinear algorithms. After all, such things as derivatives are not available. In this case, you begin at a particular point. For each parameter, you look at increasing it a bit and decreasing it a bit. If there are **n** parameters, there are 2^n possibilities so you get 2^n. You then choose the best and that is the next point you examine. This is continued until you reach a point where no increase or decrease improves your solution. This can be sped up somewhat by using factor analysis, and factorial design, which reduces the 2^n possibilities. to a more manageable number.

Consider the following data points, giving the simulation results of two different systems (each data point is the aggregation of the results of a single run):

X: 10 16 22 2 46 23 14 89 32 12

Y: 11 33 83 12 21 41 3 19 15 8

Analyze the above data to determine if the expected value of X is greater than the expected value of Y by subtracting each X value from the corresponding Y value. Can you distinguish the systems?

Do the same for the following sets of data. Now can you distinguish the data? Note that the numbers are the same: why does this lead to a different conclusion?

X: 10 16 22 2 46 23 14 89 32 12

Y: 8 15 19 3 41 21 12 83 33 11

At the beginning of each week, a machine is either running or broken down. If a machine is running throughout a week, it earns revenue of $100. If it breaks at any time in a week, the revenue for that week is $0. At the beginning of any week that the machine is running, it is possible to perform preventive maintenance. If such maintenance is performed, the probability of the machine failing is .4; otherwise, the probability of failing is .7. Maintenance costs $20. If the machine is broken at the beginning of a week, it must be fixed at a cost of $40. Such a machine will then work for that entire week with probability .6 and will fail with probability .4.

Assuming the machine is working at the beginning of week 1, determine an optimal maintenance policy over a four--week period. What is the expected profit over that period?

A company has $1 million on hand. In four weeks, a note is due for $5 million. The firm wishes to maximize its probability of having the money on hand. Each week, the company can place any amount of money it wishes into a market. If it places money in the market, it loses the money with probability .4 and doubles it with probability .3 and gets its money back with probability .3. The firm can only enter the market with the money it has on hand. Assuming it is only possible to enter the market in ``millions'' (i.e. 1.23 million is not possible), and the firm has the choice not to enter the market during a week, how can the firm maximize its probability of having $5 million at the end of four weeks.

8. Conclusion

Probabilistic models of the real world can seldom be solved and optimized using a pure mathematical approach. We have suggested a structure of a generic discrete event simulator that can be used very effectively to model and optimize environments that can be discretized into events. We have discussed some selected scenarios that can be modeled using our simulator. The objective of the simulation will be to study the interplay between different parameters, which influence design of an environment, and to investigate how optimality can be achieved in them.

The simulation results can be used to achieve optimality in a scientific manner using stochastic models of the real world environments.

9. Acknowledgements

We are grateful to Prof. SN Maheshwari, Dept of Computer Science and Engineering, IIT Delhi for his guidance and support during the design and implementation of the simulator.

10. Bibliography and References

a) P. Bratley, B. L. Fox and L. E. Schrage, A Guide to Simulation.
b) A. Law and D. Kelton, Simulation Modeling and Analysis.
c) P. Fishwick, Simulation Model Design and Execution.
d) B. Khoshnevis, Discrete Systems Simulation.
e) B. Ziegler, Theory of Modeling and Simulation.
f) L. Devroye, Nonuniform Random Variate Generation.