General Simulation Principles

Week 6
Chuljin Park
(Attributed by Dr. Dave Goldman

Steps in a Simulation Study

1. Problem Formulation — statement of problem
2. Objectives and Planning — what questions should be answered?
3. Model Building
4. Data Collection — what kinds of data, how much?
5. Coding
6. Verification — is the code OK?
7. Validation — is the model OK?
8. Experimental Design — what experiments need to be run to answer our questions
9. Run Experiments — production runs
10. Output Analysis — statistical analysis
11. Make Reports
12. Implement

General Principles for Computer Simulation Languages

- A *system* is a collection of entities (e.g., people, machines, etc.) that interact together to accomplish a goal.
- A *model* is an abstract representation of a system, usually containing math/logical relationships describing the system in terms of states, entities, sets, events, etc.

- *System state:* A set of variables that contains enough information to describe the system. For example, in a single-server queue, all you might need to describe the state are $L_Q(t)$ = number of people in queue at time t, and $B(t) = 1$ if server is busy at time t and $= 0$ if server is idle at time t.

- An *entity* is an object or component explicitly represented in the model. Can be permanent (e.g., machine) or temporary (e.g., customers).

- *Attributes* are properties or characteristics of entities (e.g., priority of a customer or average speed of a server).

- A *set* (or *list* or *queue*) is an ordered list of associated entities (e.g., a line).

- An *event* is a point in time at which the system state changes (and which can't be predicted with certainty beforehand). E.g., an arrival event which finds the server busy (so that the queue length increases by 1). Event technically means the time that the thing happens, but loosely refers to "what" happens (e.g., an arrival).

- An *activity* is a duration of time of specified length (an unconditional wait). E.g., a customer interarrival time or a service time (since we explicitly generate those).

- An *delay* is a duration of time of specified length (a conditional wait). E.g., a customer waiting time — don't know that directly.

- The *simulation clock* is a variable whose value represents simulated time (which doesn't equal real time).

Two Modeling Approaches

- Event-Scheduling Approach. Concentrate on the events and how they affect the system state. We have to help the simulation evolve over time by keeping track of every event in increasing order of time of occurrence. This gets to be a bookkeeping hassle. You might have to use this approach if you were using C++ or FORTRAN.

- Process-Interaction Approach. We use this approach in class. Concentrate on a generic customer (entity) and the sequence of events and activities it undergoes as it progresses through the system. At any time, the system may have many customers interacting with each other as they compete for resources. You do the generic customer modeling in this approach, but you don't have to bother with the event bookkeeping — the computer simulation language handles the event-scheduling deep inside. Saves lots of programming effort. E.g., Arena or any other good simulation language does this.

General Simulation Principles, Part II

Week 6
Chuljin Park
(Attributed by Dr. Dave Goldman

Two Modeling Approaches

- Event-Scheduling Approach. Concentrate on the events and how they affect the system state. We have to help the simulation evolve over time by keeping track of every event in increasing order of time of occurrence. This gets to be a bookkeeping hassle. You might have to use this approach if you were using C++ or FORTRAN.
- Process-Interaction Approach. We use this approach in class. Concentrate on a generic customer (entity) and the sequence of events and activities it undergoes as it progresses through the system. At any time, the system may have many customers interacting with each other as they compete for resources. You do the generic customer modeling in this approach, but you don't have to bother with the event bookkeeping — the computer simulation language handles the event-scheduling deep inside. Saves lots of programming effort. E.g., Arena or any other good simulation language does this.

Note: Much of the internal logic (system states, time advance mechanisms, etc.) within the simulation code (which you don't have to worry about) is the same, whether or not you use the E-S or P-I.

Every simulation language maintains a *future event list* (FEL). Thisis the list of all activities' scheduled times of completion — the list of all the events that we know about. The FEL is a set ordered by completion times, $t_1 < \cdots < t_n$.

Need to be careful about events that happen to take place at the same time, so be careful about ties. Simply establish ground rules for how to deal with ties.

Example (from Banks, Carson, Nelson, and Nicol text): Consider the usual single-server FIFO (first-in, first-out) queue that will process exactly 10 customers. The arrival times and service times of the 10 customers are:

customer	1	2	3	4	5	6	7	8	9	10
arrival time	0	4	8	10	17	18	19	20	27	29
service time	5	5	1	3	2	1	4	7	3	1

Let's construct a table containing entries for event time, system state, queue, future events list, and cumulative statistics (cumulative server busy time and customer waiting time). Let A denote an arrival event and C a completion. Suppose that services have priorities over arrivals in terms of updating the FEL; so we'll break ties by handling a service completion first — get those guys out of the system!

Clock	System State		Queue	FEL	Cumulative Stats	
t	$L_Q(t)$	$B(t)$	(cust, arrl time)	(event time,type,arrl time)	busy time	time in sys
0	0	0	$\emptyset$	(0,A,0)	0	0
0	0	1	$\emptyset$	(4,A,4), (5,C,0)	0	0
4	1	1	(2,4)	(5,C,0), (8,A,8)	4	4
5	0	1	$\emptyset$	(8,A,8), (10,C,4)	5	6
8	1	1	(3,8)	(10,C,4), (10,A,10)	8	9
10	0	1	$\emptyset$	(10,A,10), (11,C,8)	10	13
$\vdots$						

Computer handles all of this stuff in a flash.

The simulation clock progresses from *imminent (next) event* to imminent event in time order. In discrete-event simulation, the system state can only change at event times. The simulation progresses by sequentially executing and updating events (e.g., inserting new events, delete events, move them around, do nothing) on the FEL in chronological order. Nothing really happens between events.

Generic Flowchart of Simulation Program (from Law and Kelton). (See the lecture itself for pretty pictures.)

Main Program

0. Initialization Routine
   - Set clock to 0

- Initialize system state and statistical counters
- Initialize FEL.

1. Invoke Timing Routine — when does the next event occur, and what is it?
   - Determine next event type — arrival, departure, end simulation event
   - Advance clock to next time
2. Invoke Event Routine — for event type i
   - Update system state
   - Update statistical counters
   - Make any necessary changes to FEL. For example, if you have an arrival, schedule the next arrival or maybe schedule end of simulation event. If simulation is now over, write report and stop. If simulation isn't over, go back to the Timing Routine.
3. Go back to 1.

Details on Arrival Event

- Schedule next arrival
- Is server busy?
  - If server not busy,
    - Set delay (wait) for that customer = 0
    - Gather appropriate statistics (e.g., in order to ultimately calculate average customer waiting times)
    - Add one to the number of customers processed
    - Make server busy
    - Return to main program
  - If server busy,
    - Add one to number in queue.
    - If queue is full, panic (or do something about it)
    - If queue isn't full, store customer arrival time
    - Return to main program

Details on Depature Event

- Is the queue empty?
  - If queue is empty,
    - Make server idle
    - Eliminate departure event from consideration of being the next event
    - Return to main program
  - If queue isn't empty,
    - Subtract one from number in queue
    - Compute delay of customer entering service and gather statistics
    - Add one to the number of customers delayed (processed)
    - Schedule departure for this customer
    - Move each remaining customer up in the queue
    - Return to main program

Putting together and coding the previous three charts (and how they interact with each other) gives us a nice but tedious event-scheduling solution. But this is really quite a mess!

Luckily, to do this in Arena, which is a process-interaction language, all you have to do is:

- Create customers every once in a while
- Process (serve) them, maybe after waiting in line
- Dispose of the customers after they're done being processed

This is really easy to do.... Next time!