DISCRETE-EVENT SIMULATION: GENERAL PRINCIPLES AND COMPUTER SIMULATION LANGUAGES

3

This chapter develops a common framework for the modeling of complex systems, and introduces some of the major discrete-event simulation languages currently in use. The modeling approach is called discrete-event simulation. As briefly discussed in Chapter 1, a system is modeled in terms of its state at each point in time, and various events whose occurrence causes a change in state. Discrete-event modeling is appropriate for those systems for which significant changes in system state occur at discrete points in time.

Every discrete-event simulation language has its own world view, or way of looking at the system being modeled. The languages described in this chapter can be classified as taking the event-scheduling approach or the process-interaction approach to discrete-event modeling. The event-scheduling approach requires that the analyst concentrate on the events and how they affect system state. The process-interaction approach allows the analyst to concentrate on a single entity (such as a customer) and the sequence of events and activities it undergoes as it "passes through" the system. When using a general-purpose language, such as FORTRAN, ALGOL, BASIC, or Pascal, a simulator would most likely adopt the event-scheduling approach. Languages such as GASP facilitate the use of the event-scheduling approach, while GPSS provides one implementation of the process-interaction approach. Some more modern languages, such as SIMSCRIPT and SLAM, allow the simulator to use either approach or a mixture of the two, whichever is more appropriate for the problem at hand.

Section 3.1 discusses the general principles of the event-scheduling and process-interaction approaches, and gives several examples by means of hand simulations. Section 3.2 gives examples of modeling a simple system using FORTRAN, SIMSCRIPT, GPSS, and SLAM, as well as briefly discussing GASP.

3.1. Concepts in Discrete-Event Simulation

The concept of a system and a model of a system were discussed briefly in Chapter 1. This chapter deals exclusively with dynamic, stochastic systems (i.e., involving time and containing random elements) which change in a discrete manner. This section expands on these concepts and develops a framework for the development of a discrete-event model of a system. The major concepts are briefly defined and then illustrated by examples:

System	A collection of entities (e.g., people and machines) that interact together over time to accomplish one or more goals
Model	An abstract representation of a system, usually containing logical and/or mathematical relationships which describe a system in terms of state, entities and their attributes, sets, events, activities, and delays

System state	A collection of variables that contain all the information necessary to describe the system at any time
Entity	Any object or component in the system which requires explicit representation in the model (e.g., a server, a customer, a machine)
Attributes	The properties of a given entity (e.g., the priority of a waiting customer, the routing of a job through a job shop)
Set	A collection of (permanently or temporarily) associated entities, ordered in some logical fashion (such as all customers currently in a waiting line, ordered by first come, first served, or by priority)
Event	An instantaneous occurrence that changes the state of a system (such as an arrival of a new customer)
Activity	A duration of time of specified length (e.g., a service time or interarrival time), which length is known when it begins (although it may be defined in terms of a statistical distribution)
Delay	A duration of time of unspecified length, which length is not known until it ends (e.g., a customer's delay in a last in, first out waiting line, which when it begins, depends on future arrivals)

Sets are sometimes called lists, queues, or chains. An activity can be deterministic (e.g., a service time that always takes 5 minutes), or probabilistic (e.g., 5 ± 3 minutes, uniformly distributed), or, in general, any type of mathematical function. However it is characterized, the duration of an activity is computable in the model at the instant it begins. By contrast, a delay typically ends when some logical condition becomes true; this logical condition is usually a result of the interaction of many events. A customer's time spent in a waiting line is a typical example of delay. A delay is sometimes called a conditional wait, in contrast to an activity, which is called an unconditional wait. Note that the ending of an activity is an event, often termed a primary event. The beginning and ending of a delay is called a conditional event. (The beginning of an activity may be a primary or a conditional event.) The term "event" in this text refers to a primary event.

The systems considered here are dynamic, that is, changing over time. Therefore, system state, entity attributes and the number of active entities, the contents of sets, and the activities and delays currently in progress are all functions of time and are constantly changing over time. Time itself is represented by a variable called CLOCK.

EXAMPLE 3.1 (ABLE AND BAKER, REVISITED)

Consider the Able–Baker carhop system of Example 2.2. A discrete-event model has the following components:

System state	$L_Q(t)$, the number of cars waiting to be served at time t $L_A(t)$, 0 or 1 to indicate Able being idle or busy at time t $L_B(t)$, 0 or 1 to indicate Baker being idle or busy at time t
Entities	Neither the customers (i.e., cars) nor the servers need to be explicitly represented, except in terms of the state variables, unless certain customer averages are desired (compare Examples 3.2 and 3.3)
Events	Arrival event Service completion by Able Service completion by Baker
Activities	Interarrival time, defined in Table 2.11 Service time by Able, defined in Table 2.12 Service time by Baker, defined in Table 2.13
Delay	The wait in queue until Able or Baker becomes free

The definition of the model components provides a static description of the model. In addition, a description of the dynamic relationships and interactions between the components is also needed. Some questions that need answers include:

1. How does each event affect system state, entity attributes, and set contents?
2. How are activities defined (i.e., deterministic, probabilistic, or some other mathematical equation)? What event marks the beginning or end of each activity? Can the activity begin regardless of system state, or is its beginning conditioned on the system being in a certain state? (For example, a machining "activity" cannot begin unless the machine is idle, not broken, and not in maintenance.)
3. Which events trigger the beginning (and end) of each type of delay? Under what conditions does a delay begin, or end?
4. What is the system state at time 0? What events should be generated at time 0 to "prime" the model—that is, to get the simulation started?

A discrete-event simulation is the modeling over time of a system all of whose state changes occur at discrete points in time—those points when an event occurs. A discrete-event simulation (hereafter called a simulation) proceeds by producing a sequence of system snapshots (or system images) which represent the evolution of the system through time. A given snapshot at a given time (CLOCK $= t$) includes not only the system state at time t, but also a list (called the *future event list*) of all activities currently in progress and when each such activity will end, the status of all entities and current membership of all sets, plus the current values of cumulative statistics and counters that will be used to calculate summary statistics at the end of the simulation. A prototype system snapshot is shown in Figure 3.1. (Not all models will contain every element exhibited in Figure 3.1. Further illustrations are provided in the examples in this chapter.)

The mechanism for advancing simulation time and guaranteeing that all events occur in correct chronological order is based on the future event list. This list is a special set which contains all events that have been scheduled to occur at a future time. Scheduling a future event means that at the instant an activity begins, its duration is computed (perhaps it is generated in "random" fashion) and the end-activity event, together with its event time, is placed on the future event list. In the real world, most

Clock	System state	Entities and attributes	Set 1	Set 2	. . .	Future event list, FEL	Cumulative statistics and counters
t	$(x, y, z, \ldots)$					$(3, t_1)$ – Type 3 event to occur at time t_1 $(1, t_2)$ – Type 1 event to occur at time t_2	

Figure 3.1. Prototype system snapshot at simulation time t.

future events are not scheduled but merely happen—such as random breakdowns or random arrivals. In the model, such random events are represented by the end of some activity, which in turn is represented by a statistical distribution.

At any given time t, the future event list (FEL) contains all previously scheduled future events and their associated event times (called $t_1, t_2, \ldots$ in Figure 3.1). The FEL is ordered by event time, meaning that the events are arranged chronologically; that is, the event times satisfy

$$t < t_1 \leq t_2 \leq t_3 \leq \cdots \leq t_n$$

Time t is the value of CLOCK, the current value of simulation time. The event associated with time t_1 is called the imminent event; that is, it is the next event that will occur. After the system snapshot at simulation time CLOCK $= t$ is complete, the CLOCK is advanced to simulation time CLOCK $= t_1$, and the imminent event is removed from the FEL and executed. Execution of the imminent event means that a new system snapshot for time t_1 is created based on the old snapshot at time t and the nature of the imminent event. At time t_1, new future events may or may not be generated, but if any are, they are scheduled by putting them in their proper position on the FEL. After the new system snapshot for time t_1 is completed, the clock is advanced to the time of the new imminent event and that event is executed. This process repeats until the simulation is over. The sequence of actions which a simulator (or simulation language) must perform to advance the clock and build a new system snapshot is called the *event-scheduling/time-advance algorithm*, whose steps are listed in Figure 3.2 (and explained below).

The length and contents of the FEL are constantly changing as the simulation progresses, and thus its efficient management in a computerized simulation will have a major impact on the efficiency of the computer program representing the model. The management of a list is called list processing. The major list processing operations performed on a FEL are removal of the imminent event, addition of a new event to the list, and occasionally removal of some event (called cancellation of an event). As the imminent event is usually at the top of the list, its removal is as efficient as possible. Addition of a new event (and cancellation of an old event) requires a search of the list. The efficiency of this search depends on the logical organization of the list and on how the search is conducted. In addition to the FEL, all the sets in a model are

Old system snapshot at time t

CLOCK	System state	. . .	Future event list	. . .
t	(5, 1, 6)		$(3, t_1)$ – Type 3 event to occur at time t_1 $(1, t_2)$ – Type 1 event to occur at time t_2 $(1, t_3)$ – Type 1 event to occur at time t_3 . . . $(2, t_n)$ – Type 2 event to occur at time t_n	

Event-scheduling/time-advance algorithm

Step 1. Remove imminent event (event 3, time t_1) from FEL.
Step 2. Advance CLOCK to imminent event time (i.e., advance CLOCK from t to t_1).
Step 3. Execute imminent event: update system state, change entity attributes, and set membership as needed.
Step 4. Generate future events (if necessary) and place on FEL in correct position. (Example: Event 4 to occur at time t^*, where $t_2 < t^* < t_3$.)
Step 5. Update cumulative statistics and counters.

New system snapshot at time t_1

CLOCK	System state	. . .	Future event list	. . .
t_1	(5, 1, 5)		$(1, t_2)$ – Type 1 event to occur at time t_2 $(4, t^*)$ – Type 4 event to occur at time t^* $(1, t_3)$ – Type 1 event to occur at time t_3 . . . $(2, t_n)$ – Type 2 event to occur at time t_n	

Figure 3.2. Advancing simulation time and updating system image.

maintained in some logical order, and the operations of addition and removal of entities from the set also require efficient list-processing techniques. A brief introduction to list processing in simulation is given by Law and Kelton [1982, Chap. 2].

The removal and addition of events from the FEL is illustrated in Figure 3.2. Event 3 with event time t_1 represents, say, a service completion event at server 3. Since it is the imminent event at time t, it is removed from the FEL in step 1 (Figure 3.2) of the event-scheduling/time-advance algorithm. When event 4 (say, an arrival event) with event time t^* is generated at step 4, one possible way to determine its correct position on the FEL is to conduct a top-down search:

If $t^* < t_2$, place event 4 at the top of the FEL.
If $t_2 \leq t^* < t_3$, place event 4 second on the list.
If $t_3 \leq t^* < t_4$, place event 4 third on the list.

.
.
.

If $t_n \leq t^*$, place event 4 last on the list.

(In Figure 3.2, it was assumed that t^* was between t_2 and t_3.) Another way is to conduct a bottom-up search. The least efficient way to maintain the FEL is to leave it as an unordered list (additions placed arbitrarily at the top or bottom), which would require at step 1 of Figure 3.2 a complete search of the list for the imminent event before each clock advance. (The imminent event is the event on the FEL with the lowest event time.)

The system snapshot at time 0 is defined by the initial conditions and the generation of the so-called exogenous events. The specified initial conditions define the system state at time 0. For example, in Figure 3.2, if $t = 0$, then the state (5, 1, 6) might represent the initial number of customers at three different points in the system. An exogenous event is a happening "outside the system" which impinges on the system. An important example is an arrival to a queueing system. At time 0, the first arrival event is generated, and scheduled on the FEL. The interarrival time is an example of an activity. When the clock eventually is advanced to the time of this first arrival, a second arrival event is generated. First, an interarrival time is generated, say a^*; it is added to the current time, say CLOCK $= t$; the resulting (future) event time, $t + a^* = t^*$, is used to position the new arrival event on the FEL. This method of generating an external arrival stream is called bootstrapping, and provides one example of how future events are generated in step 4 of the event-scheduling/time-advance algorithm. Bootstrapping is illustrated in Figure 3.3. The first three interarrival times generated are 3.7, 0.4, and 3.3 time units. The beginning and end of an interarrival interval are examples of primary events.

A second example of how future events are generated (step 4 of Figure 3.2) is provided by a service completion event in a queueing simulation. When one customer completes service, say at current time CLOCK $= t$, if the next customer is present, then

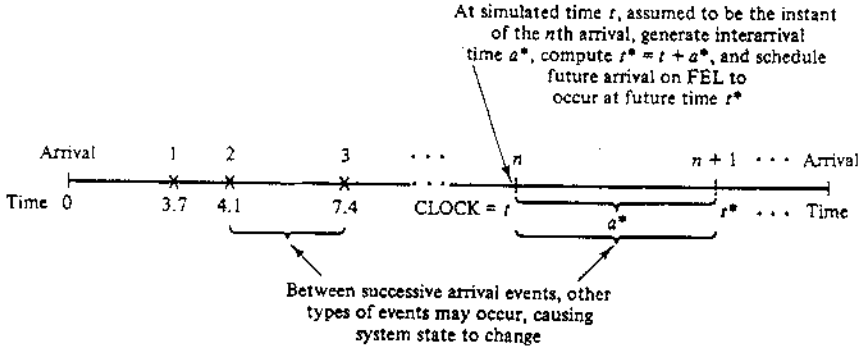

Figure 3.3. Generation of an external arrival stream by bootstrapping.

a new service time, say s^*, will be generated for the next customer. The next service completion event will be scheduled to occur at future time $t^* = t + s^*$, by placing onto the FEL a new service completion event with event time t^*. In addition, a service completion event will be generated and scheduled at the time of an arrival event provided that, upon arrival, there is at least one idle server in the server group. A service time is an example of an activity. Beginning service is a conditional event, because its occurrence is triggered only on the condition that a customer is present and a server is free. Service completion is an example of a primary event. Note that a conditional event, such as beginning service, is triggered by a primary event occurring and certain conditions prevailing in the system.

A third important example is the alternate generation of runtimes and downtimes for a machine subject to breakdowns. At time 0, the first runtime will be generated and an end-of-runtime event scheduled. Whenever an end-of-runtime event occurs, a downtime will be generated and an end-of-downtime event scheduled on the FEL. When the CLOCK is eventually advanced to the time of this end-of-downtime event, a runtime is generated and an end-of-runtime event scheduled on the FEL. In this way, runtimes and downtimes continually alternate throughout the simulation. A runtime and a downtime are examples of activities, and end of runtime and end of downtime are primary events.

Every simulation must have a stopping event, here called E, which defines how long the simulation will run. There are generally two ways to stop a simulation:

1. At time 0, schedule a stop simulation event at a future time T_E. Thus, before simulating it is known that the simulation will run over the time interval $[0, T_E]$. Example: Simulate a job shop for $T_E = 40$ hours.
2. Run length T_E is determined by the simulation itself. Generally, T_E is the time of occurrence of some specified event E. Examples: T_E is the time of the 100th service completion at a certain service center. T_E is the time of breakdown of a complex system. T_E is the time of disengagement or total kill (whichever occurs first) in a combat simulation.

In case 2, T_E is not known ahead of time. Indeed, it may be one of the statistics of primary interest to be produced by the simulation.

A systematic approach to simulation which concentrates on events and their effects on system state is called an *event-scheduling* approach to discrete-event simulation. This approach will be illustrated by the manual simulations of Section 3.1.1 and the FORTRAN and SIMSCRIPT simulations of Section 3.2. A different outlook is provided by the *process-interaction* approach. A process is a time-ordered collection of events, activities, and delays which are somehow related, say to some entity. An example of a "customer process" is shown in Figure 3.4. Many processes are usually simultaneously active in a model, and the interaction among processes may be quite complex. Figure 3.4 also illustrates the interaction between two successive customer processes in a first come, first served single-server queue. Languages based on the process-interaction approach include GPSS and SLAM; also, SIMSCRIPT II.5 (Release 8) optionally allows the use of the process-interaction approach. Illustrations of these process-based languages are given in Section 3.2.

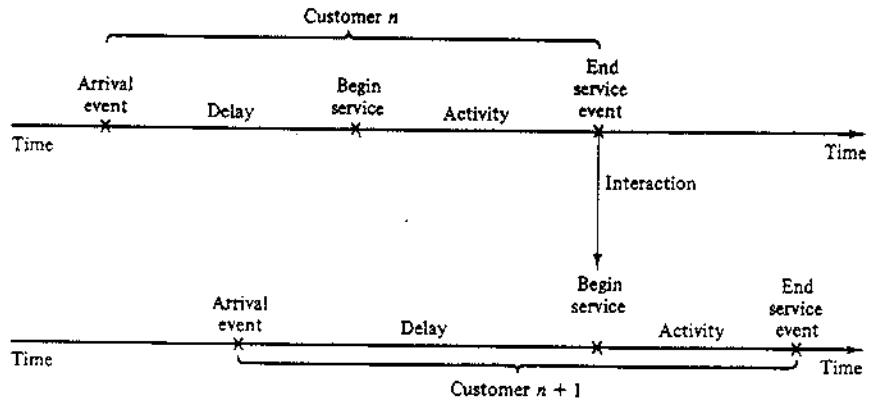

Figure 3.4. Two interacting customer processes in a single-server queue.

3.1.1. Manual Simulation Using Event Scheduling

In conducting an event-scheduling simulation, a simulation table is used to record the successive system snapshots as time advances.

Example 3.2 (Single-Channel Queue)

Reconsider the grocery store with one checkout counter which was simulated in Example 2.1 by an ad hoc method. The system consists of those customers in the waiting line plus the one (if any) checking out. The model has the following components:

System state	(LQ(t), LS(t)), where LQ(t) is the number of customers in the waiting line, and LS(t) is the number being served (0 or 1), at time t
Entities	The server and customers are not explicitly modeled, except in terms of the state variables above
Events	Arrival (A) Departure (D) Stopping event (E), scheduled to occur at time 60
Activities	Interarrival time, defined in Table 2.6 Service time, defined in Table 2.7
Delay	Customer time spent in waiting line

The events on the FEL are written as (event type, event time). In this model, the FEL will always contain either two or three events. The effect of the arrival and departure events was first shown in Figures 2.2 and 2.3, and is shown in more detail in Figures 3.5 and 3.6.

The simulation table for the checkout counter is given in Table 3.1. The reader should cover all system snapshots except one, starting with the first, and attempt to construct the next snapshot from the previous one and the event logic in Figures 3.5 and 3.6. The interarrival times and service times will be identical to those used in Table 2.10, namely:

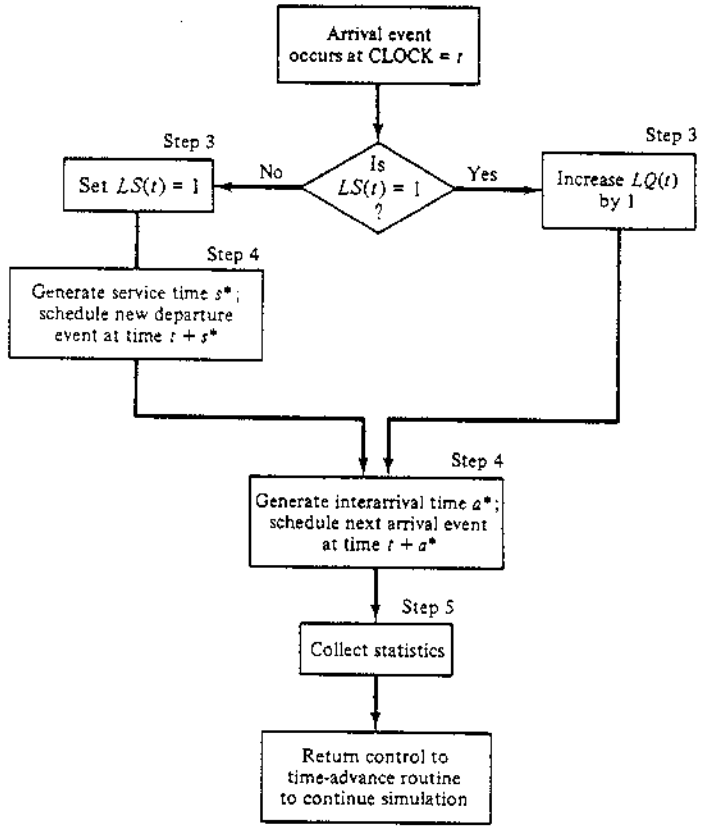

Figure 3.5. Execution of the arrival event.

Interarrival Times	8	6	1	8	3	8	...
Service Times	4	1	4	3	2	4	...

Initial conditions are that the first customer arrives at time 0 and begins service. This is reflected in Table 3.1 by the system snapshot at time zero (CLOCK = 0), with LQ(0) = 0, LS(0) = 1, and both a departure event and arrival event on the FEL. Also, the simulation is scheduled to stop at time 60. Only two statistics, server utilization and maximum queue length, will be collected. Server utilization is defined by total server busy time (B) divided by total time (T_E). Total busy time, B, and maximum queue length, MQ, will be cumulated as the simulation progresses. A column headed "comments" is included to aid the reader. (a^* and s^* are the generated interarrival and service times, respectively.)

As soon as the system snapshot at time CLOCK = 0 is complete, the simulation begins. At time 0, the imminent event is (D, 4). The CLOCK is advanced to time 4, and (D, 4) is removed from the FEL. Since LS(t) = 1 for $0 \le t \le 4$ (i.e., the server was busy for 4 minutes), the cumulative busy time is increased from B = 0 to B = 4. By the event logic in Figure 3.6, set LS(4) = 0 (the server becomes idle). The FEL is

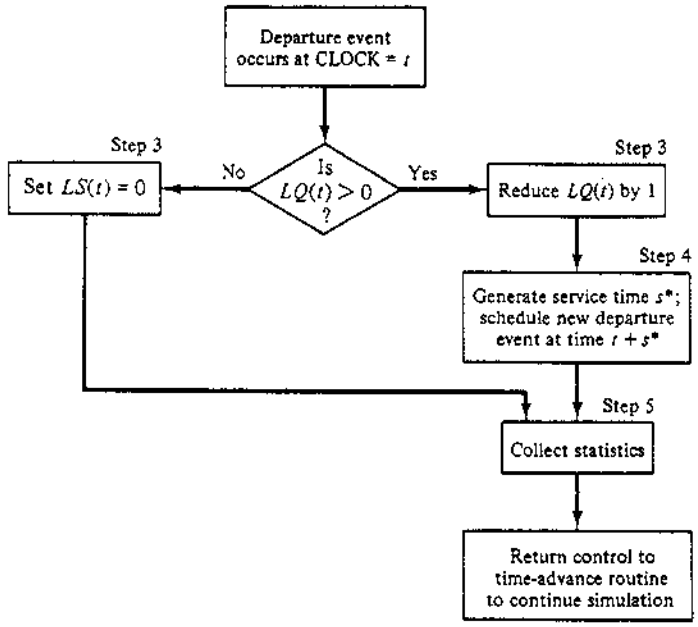

Figure 3.6. Execution of the departure event.

left with only two future events, (A, 8) and (E, 60). The simulation CLOCK is next advanced to time 8 and an arrival event executed. The interpretation of the remainder of Table 3.1 is left to the reader.

The simulation in Table 3.1 covers the time interval [0, 21]. At simulated time 21, the system is empty, but the next arrival will occur at future time 23. The server was busy for 12 of the 21 time units simulated, and the maximum queue length was one. This simulation is, of course, too short to draw any reliable conclusions. Exercise 1 asks the reader to continue the simulation and to compare the results to those in Example 2.1. Note that the simulation table gives the system state at all times, not just the listed times. For example, from time 15 to time 18, there is one customer in service and one in the waiting line.

When an event scheduling algorithm is computerized, only one snapshot (the current one or partially updated one) is kept in computer memory. With the idea of implementing event scheduling in FORTRAN or some other general-purpose language, the following rule should be followed. A new snapshot can be derived only from the previous snapshot, newly generated random variables, and the event logic (Figures 3.5 and 3.6). Past snapshots should be ignored when advancing the clock. The current snapshot should contain all information necessary to continue the simulation.

EXAMPLE 3.3 (THE CHECKOUT COUNTER SIMULATION, CONTINUED)

Suppose that in the simulation of the checkout counter in Example 3.2 the simulator desires to estimate mean response time and mean proportion of customers who spend 4 or more minutes in the system. A response time is the length of time a customer

Table 3.1. SIMULATION TABLE FOR CHECKOUT COUNTER (EXAMPLE 3.2)

	System State					*Cumulative Statistics*	
Clock	$LQ(t)$	$LS(t)$	*Future Event List*	*Comment*	B	MQ
0	0	1	(D, 4) (A, 8) (E, 60)	First A occurs	0	0
				($a^* = 8$) Schedule next A		
				($s^* = 4$) Schedule first D		
4	0	0	(A, 8) (E, 60)	First D occurs: (D, 4)	4	0
8	0	1	(D, 9) (A, 14) (E, 60)	Second A occurs: (A, 8)	4	0
				($a^* = 6$) Schedule next A		
				($s^* = 1$) Schedule next D		
9	0	0	(A, 14) (E, 60)	Second D occurs: (D, 9)	5	0
14	0	1	(A, 15) (D, 18) (E, 60)	Third A occurs: (A, 14)	5	0
				($s^* = 4$) Schedule next D		
15	1	1	(D, 18) (A, 23) (E, 60)	Fourth A occurs: (A, 15)	6	1
				(Customer delayed)		
18	0	1	(D, 21) (A, 23) (E, 60)	Third D occurs: (D, 18)	9	1
				($s^* = 3$) Schedule next D		
21	0	0	(A, 23) (E, 60)	Fourth D occurs: (D, 21)	12	1

spends in the system. In order to estimate these customer averages, it is necessary to expand the model in Example 3.2 to explicitly represent the individual customers. In addition, to be able to compute an individual customer's response time when that customer departs, it will be necessary to know that customer's arrival time. Therefore, a customer entity with arrival time as an attribute will be added to the list of model components in Example 3.2. These customer entities will be stored in a set to be called "CHECKOUT LINE"; they will be called C1, C2, C3, ... Finally, the notation for events on the FEL will be expanded to indicate which customer is affected. For example, (D, 4, C1) means that customer C1 will depart at time 4. The additional model components are listed below:

Entities	(Ci, t), representing customer Ci who arrived at time t
Events	(A, t, Ci), the arrival of customer Ci at time t
	(D, t, Cj), the departure of customer Cj at time t
Set	"CHECKOUT LINE," the set of all customers currently at the checkout counter (being served or waiting to be served), ordered by time of arrival

Three new cumulative statistics will be collected: S, the sum of customer response times for all customers who have departed by the current time; F, the total number of customers who spend 4 or more minutes at the checkout counter; and N_D, the total number of departures up to the current simulation time. These three cumulative statistics will be updated whenever the departure event occurs; the logic for collecting these statistics would be incorporated into step 5 of the departure event in Figure 3.6.

The simulation table for Example 3.3 is shown in Table 3.2. The same data for interarrival and service times will be used again, so that Table 3.2 essentially repeats Table 3.1, except that the new components are included (and the comment column has been deleted). These new components are needed for the computation of S, F, and N_D. For example, at time 4 a departure event occurs for customer C1. The customer entity C1 is removed from the set "CHECKOUT LINE"; the attribute "time of arrival" is noted to be 0, so the response time for this customer was 4 minutes. Hence, S is incremented by 4 minutes, and F and N_D are incremented by one customer. Similarly, at time 21 when the departure event (D, 21, C4) is being executed, the response time for customer C4 is computed by

$$\begin{aligned}\text{Response time} &= \text{CLOCK TIME} - \text{attribute "time of arrival"} \\ &= \quad 21 \quad - \quad 15 \\ &= 6 \text{ minutes}\end{aligned}$$

Then S is incremented by 6 minutes, and F and N_D by one customer.

For a simulation run length of 21 minutes, the average response time was $S/N_D = 15/4 = 3.75$ minutes, and the observed proportion of customers who spent 4 or more minutes in the system was $F/N_D = 0.75$. Again, this simulation was far too short to regard these estimates with any degree of accuracy. The purpose of Example 3.3, however, was to illustrate the notion that in many simulation models the information desired from the simulation (such as the statistics S/N_D and F/N_D) determine to some extent the structure of the model.

Table 3.2. SIMULATION TABLE FOR EXAMPLE 3.3

System State			*Set*	*Future Event List*	*Cumulative Statistics*		
Clock	LQ(t)	LS(t)	*"CHECKOUT LINE"*		S	N_D	F
0	0	1	(C1, 0)	(D, 4, C1) (A, 8, C2) (E, 60)	0	0	0
4	0	0		(A, 8, C2) (E, 60)	4	1	1
8	0	1	(C2, 8)	(D, 9, C2) (A, 14, C3) (E, 60)	4	1	1
9	0	0		(A, 14, C3) (E, 60)	5	2	1
14	0	1	(C3, 14)	(A, 15, C4) (D, 18, C3) (E, 60)	5	2	1
15	1	1	(C3, 14) (C4, 15)	(D, 18, C3) (A, 23, C5) (E, 60)	5	2	1
18	0	1	(C4, 15)	(D, 21, C4) (A, 23, C5) (E, 60)	9	3	2
21	0	0		(A, 23, C5) (E, 60)	15	4	3