CHAPTER 1

Basic Simulation Modeling

Recommended sections for a first reading: 1.1 through 1.4 (except 1.4.8), 1.7, 1.9

1.1 THE NATURE OF SIMULATION

This is a book about techniques for using computers to imitate, or *simulate*, the operations of various kinds of real-world facilities or processes. The facility or process of interest is usually called a *system*, and in order to study it scientifically we often have to make a set of assumptions about how it works. These assumptions, which usually take the form of mathematical or logical relationships, constitute a *model* that is used to try to gain some understanding of how the corresponding system behaves.

If the relationships that compose the model are simple enough, it may be possible to use mathematical methods (such as algebra, calculus, or probability theory) to obtain *exact* information on questions of interest; this is called an *analytic* solution. However, most real-world systems are too complex to allow realistic models to be evaluated analytically, and these models must be studied by means of simulation. In a *simulation* we use a computer to evaluate a model *numerically*, and data are gathered in order to *estimate* the desired true characteristics of the model.

As an example of the use of simulation, consider a manufacturing company that is contemplating building a large extension onto one of its plants but is not sure if the potential gain in productivity would justify the construction cost. It certainly would not be cost-effective to build the extension and then remove it later if it does not work out. However, a careful simulation study could shed some light on the question by simulating the operation of the plant as it currently exists and as it *would* be *if* the plant were expanded.

Application areas for simulation are numerous and diverse. Below is a list of some particular kinds of problems for which simulation has been found to be a useful and powerful tool:

- Designing and analyzing manufacturing systems
- Evaluating military weapons systems or their logistics requirements
- Determining hardware requirements or protocols for communications networks
- Determining hardware and software requirements for a computer system
- Designing and operating transportation systems such as airports, freeways, ports, and subways
- Evaluating designs for service organizations such as call centers, fast-food restaurants, hospitals, and post offices
- Reengineering of business processes
- Determining ordering policies for an inventory system
- Analyzing financial or economic systems

Simulation is one of the most widely used operations-research and management-science techniques, if not *the* most widely used. One indication of this is the Winter Simulation Conference, which attracts 600 to 700 people every year. In addition, there are several simulation vendor users' conferences with more than 100 participants per year.

There are also several surveys related to the use of operations-research techniques. For example, Lane, Mansour, and Harpell (1993) reported from a longitudinal study, spanning 1973 through 1988, that simulation was consistently ranked as one of the three most important "operations-research techniques." The other two were "math programming" (a catch-all term that includes many individual techniques such as linear programming, nonlinear programming, etc.) and "statistics" (which is not an operations-research technique per se). Gupta (1997) analyzed 1294 papers from the journal *Interfaces* (one of the leading journals dealing with applications of operations research) from 1970 through 1992, and found that simulation was second only to "math programming" among 13 techniques considered.

There have been, however, several impediments to even wider acceptance and usefulness of simulation. First, models used to study large-scale systems tend to be very complex, and writing computer programs to execute them can be an arduous task indeed. This task has been made much easier in recent years by the development of excellent software products that automatically provide many of the features needed to "program" a simulation model. A second problem with simulation of complex systems is that a large amount of computer time is sometimes required. However, this difficulty is becoming much less severe as computers become faster and cheaper. Finally, there appears to be an unfortunate impression that simulation is just an exercise in computer programming, albeit a complicated one. Consequently, many simulation "studies" have been composed of heuristic model building, coding, and a single run of the program to obtain "the answer." We fear that this attitude, which neglects the important issue of how a properly coded model should be used to make inferences about the system of interest, has doubtless led

to erroneous conclusions being drawn from many simulation studies. These questions of simulation *methodology*, which are largely independent of the software and hardware used, form an integral part of the latter chapters of this book.

In the remainder of this chapter (as well as in Chap. 2) we discuss systems and models in considerably greater detail and then show how to write computer programs in general-purpose languages to simulate systems of varying degrees of complexity. All of the computer code shown in this chapter can be downloaded from http://www.mhhe.com/lawkelton.

1.2 SYSTEMS, MODELS, AND SIMULATION

A *system* is defined to be a collection of entities, e.g., people or machines, that act and interact together toward the accomplishment of some logical end. [This definition was proposed by Schmidt and Taylor (1970).] In practice, what is meant by "the system" depends on the objectives of a particular study. The collection of entities that comprise a system for one study might be only a subset of the overall system for another. For example, if one wants to study a bank to determine the number of tellers needed to provide adequate service for customers who want just to cash a check or make a savings deposit, the system can be defined to be that portion of the bank consisting of the tellers and the customers waiting in line or being served. If, on the other hand, the loan officer and the safety deposit boxes are to be included, the definition of the system must be expanded in an obvious way. [See also Fishman (1978, p. 3).] We define the *state* of a system to be that collection of variables necessary to describe a system at a particular time, relative to the objectives of a study. In a study of a bank, examples of possible state variables are the number of busy tellers, the number of customers in the bank, and the time of arrival of each customer in the bank.

We categorize systems to be of two types, discrete and continuous. A *discrete* system is one for which the state variables change instantaneously at separated points in time. A bank is an example of a discrete system, since state variables—e.g., the number of customers in the bank—change only when a customer arrives or when a customer finishes being served and departs. A *continuous* system is one for which the state variables change continuously with respect to time. An airplane moving through the air is an example of a continuous system, since state variables such as position and velocity can change continuously with respect to time. Few systems in practice are wholly discrete or wholly continuous; but since one type of change predominates for most systems, it will usually be possible to classify a system as being either discrete or continuous.

At some point in the lives of most systems, there is a need to study them to try to gain some insight into the relationships among various components, or to predict performance under some new conditions being considered. Figure 1.1 maps out different ways in which a system might be studied.

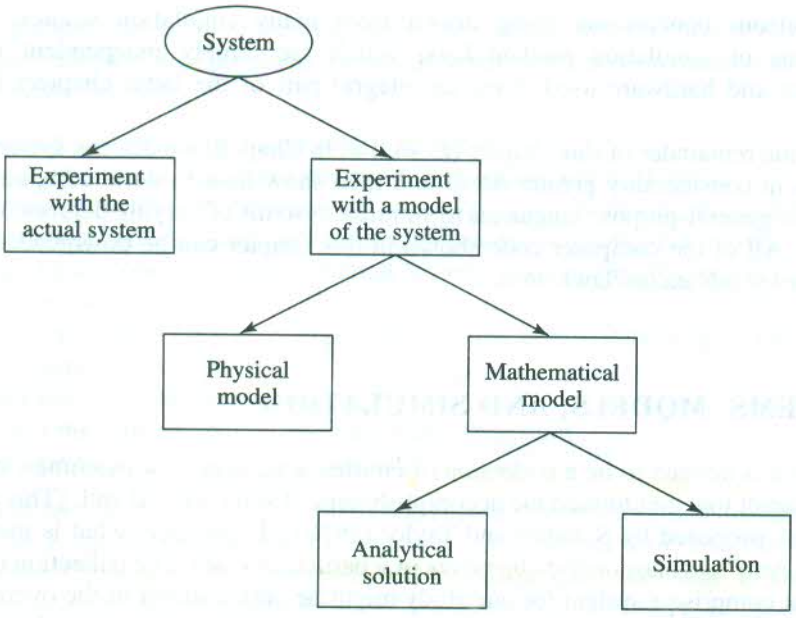

FIGURE 1.1
Ways to study a system.

- *Experiment with the Actual System vs. Experiment with a Model of the System.* If it is possible (and cost-effective) to alter the system physically and then let it operate under the new conditions, it is probably desirable to do so, for in this case there is no question about whether what we study is valid. However, it is rarely feasible to do this, because such an experiment would often be too costly or too disruptive to the system. For example, a bank may be contemplating reducing the number of tellers to decrease costs, but actually trying this could lead to long customer delays and alienation. More graphically, the "system" might not even exist, but we nevertheless want to study it in its various proposed alternative configurations to see how it should be built in the first place; examples of this situation might be a proposed communications network, or a strategic nuclear weapons system. For these reasons, it is usually necessary to build a *model* as a representation of the system and study it as a surrogate for the actual system. When using a model, there is always the question of whether it accurately reflects the system for the purposes of the decisions to be made; this question of model *validity* is taken up in detail in Chap. 5.
- *Physical Model vs. Mathematical Model.* To most people, the word "model" evokes images of clay cars in wind tunnels, cockpits disconnected from their airplanes to be used in pilot training, or miniature supertankers scurrying about in a swimming pool. These are examples of *physical* models (also called *iconic* models), and are not typical of the kinds of models that are usually of interest in operations research and systems analysis. Occasionally, however, it has been found useful to build physical models to study engineering or management

systems; examples include tabletop scale models of material-handling systems, and in at least one case a full-scale physical model of a fast-food restaurant inside a warehouse, complete with full-scale, real (and presumably hungry) humans [see Swart and Donno (1981)]. But the vast majority of models built for such purposes are *mathematical*, representing a system in terms of logical and quantitative relationships that are then manipulated and changed to see how the model reacts, and thus how the system *would* react—*if* the mathematical model is a valid one. Perhaps the simplest example of a mathematical model is the familiar relation $d = rt$, where r is the rate of travel, t is the time spent traveling, and d is the distance traveled. This might provide a valid model in one instance (e.g., a space probe to another planet after it has attained its flight velocity) but a very poor model for other purposes (e.g., rush-hour commuting on congested urban freeways).

- *Analytical Solution vs. Simulation.* Once we have built a mathematical model, it must then be examined to see how it can be used to answer the questions of interest about the system it is supposed to represent. If the model is simple enough, it may be possible to work with its relationships and quantities to get an exact, *analytical* solution. In the $d = rt$ example, if we know the distance to be traveled and the velocity, then we can work with the model to get $t = d/r$ as the time that will be required. This is a very simple, closed-form solution obtainable with just paper and pencil, but some analytical solutions can become extraordinarily complex, requiring vast computing resources; inverting a large nonsparse matrix is a well-known example of a situation in which there is an analytical formula known in principle, but obtaining it numerically in a given instance is far from trivial. If an analytical solution to a mathematical model is available and is computationally efficient, it is usually desirable to study the model in this way rather than via a simulation. However, many systems are highly complex, so that valid mathematical models of them are themselves complex, precluding any possibility of an analytical solution. In this case, the model must be studied by means of *simulation*, i.e., numerically exercising the model for the inputs in question to see how they affect the output measures of performance.

While there may be a small element of truth to pejorative old saws such as "method of last resort" sometimes used to describe simulation, the fact is that we are very quickly led to simulation in most situations, due to the sheer complexity of the systems of interest and of the models necessary to represent them in a valid way.

Given, then, that we have a mathematical model to be studied by means of simulation (henceforth referred to as a *simulation model*), we must then look for particular tools to do this. It is useful for this purpose to classify simulation models along three different dimensions:

- *Static vs. Dynamic Simulation Models.* A *static* simulation model is a representation of a system at a particular time, or one that may be used to represent a system in which time simply plays no role; examples of static simulations are Monte Carlo models, discussed in Sec. 1.8.3. On the other hand, a *dynamic* simulation model represents a system as it evolves over time, such as a conveyor system in a factory.

- *Deterministic vs. Stochastic Simulation Models.* If a simulation model does not contain any probabilistic (i.e., random) components, it is called *deterministic*; a complicated (and analytically intractable) system of differential equations describing a chemical reaction might be such a model. In deterministic models, the output is "determined" once the set of input quantities and relationships in the model have been specified, even though it might take a lot of computer time to evaluate what it is. Many systems, however, must be modeled as having at least some random input components, and these give rise to *stochastic* simulation models. (For an example of the danger of ignoring randomness in modeling a system, see Sec. 4.7.) Most queueing and inventory systems are modeled stochastically. Stochastic simulation models produce output that is itself random, and must therefore be treated as only an estimate of the true characteristics of the model; this is one of the main disadvantages of simulation (see Sec. 1.9) and is dealt with in Chaps. 9 through 12 of this book.
- *Continuous vs. Discrete Simulation Models.* Loosely speaking, we define *discrete* and *continuous* simulation models analogously to the way discrete and continuous systems were defined above. More precise definitions of discrete (event) simulation and continuous simulation are given in Secs. 1.3 and 1.8, respectively. It should be mentioned that a discrete model is not always used to model a discrete system, and vice versa. The decision whether to use a discrete or a continuous model for a particular system depends on the specific objectives of the study. For example, a model of traffic flow on a freeway would be discrete if the characteristics and movement of individual cars are important. Alternatively, if the cars can be treated "in the aggregate," the flow of traffic can be described by differential equations in a continuous model. More discussion on this issue can be found in Sec. 5.2, and in particular in Example 5.2.

The simulation models we consider in the remainder of this book, except for those in Sec. 1.8, will be discrete, dynamic, and stochastic and will henceforth be called *discrete-event simulation models*. (Since deterministic models are a special case of stochastic models, the restriction to stochastic models involves no loss of generality.)

1.3 DISCRETE-EVENT SIMULATION

Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. (In more mathematical terms, we might say that the system can change at only a *countable* number of points in time.) These points in time are the ones at which an event occurs, where an *event* is defined as an instantaneous occurrence that may change the state of the system. Although discrete-event simulation could conceptually be done by hand calculations, the amount of data that must be stored and manipulated for most real-world systems dictates that discrete-event simulations be done on a digital computer. (In Sec. 1.4.2 we carry out a small hand simulation, merely to illustrate the logic involved.)

EXAMPLE 1.1. Consider a service facility with a single server—e.g., a one-operator barbershop or an information desk at an airport—for which we would like to estimate the (expected) average delay in queue (line) of arriving customers, where the delay in queue of a customer is the length of the time interval from the instant of his arrival at the facility to the instant he begins being served. For the objective of estimating the average delay of a customer, the state variables for a discrete-event simulation model of the facility would be the status of the server, i.e., either idle or busy, the number of customers waiting in queue to be served (if any), and the time of arrival of each person waiting in queue. The status of the server is needed to determine, upon a customer's arrival, whether the customer can be served immediately or must join the end of the queue. When the server completes serving a customer, the number of customers in the queue is used to determine whether the server will become idle or begin serving the first customer in the queue. The time of arrival of a customer is needed to compute his delay in queue, which is the time he begins being served (which will be known) minus his time of arrival. There are two types of events for this system: the arrival of a customer and the completion of service for a customer, which results in the customer's departure. An arrival is an event since it causes the (state variable) server status to change from idle to busy or the (state variable) number of customers in the queue to increase by 1. Correspondingly, a departure is an event because it causes the server status to change from busy to idle or the number of customers in the queue to decrease by 1. We show in detail how to build a discrete-event simulation model of this single-server queueing system in Sec. 1.4.

In the above example both types of events actually changed the state of the system, but in some discrete-event simulation models events are used for purposes that do not actually effect such a change. For example, an event might be used to schedule the end of a simulation run at a particular time (see Sec. 1.4.7) or to schedule a decision about a system's operation at a particular time (see Sec. 1.5) and might not actually result in a change in the state of the system. This is why we originally said that an event *may* change the state of a system.

1.3.1 Time-Advance Mechanisms

Because of the dynamic nature of discrete-event simulation models, we must keep track of the current value of simulated time as the simulation proceeds, and we also need a mechanism to advance simulated time from one value to another. We call the variable in a simulation model that gives the current value of simulated time the *simulation clock*. The unit of time for the simulation clock is never stated explicitly when a model is written in a general-purpose language such as FORTRAN or C, and it is assumed to be in the same units as the input parameters. Also, there is generally no relationship between simulated time and the time needed to run a simulation on the computer.

Historically, two principal approaches have been suggested for advancing the simulation clock: *next-event time advance* and *fixed-increment time advance*. Since the first approach is used by all major simulation software and by most people coding their model in a general-purpose language, and since the second is a special case of the first, we shall use the next-event time-advance approach for all discrete-event simulation models discussed in this book. A brief discussion of fixed-increment time advance is given in App. 1A (at the end of this chapter).

With the next-event time-advance approach, the simulation clock is initialized to zero and the times of occurrence of future events are determined. The simulation clock is then advanced to the time of occurrence of the *most imminent* (first) of these future events, at which point the state of the system is updated to account for the fact that an event has occurred, and our knowledge of the times of occurrence of future events is also updated. Then the simulation clock is advanced to the time of the (new) most imminent event, the state of the system is updated, and future event times are determined, etc. This process of advancing the simulation clock from one event time to another is continued until eventually some prespecified stopping condition is satisfied. Since all state changes occur only at event times for a discrete-event simulation model, periods of inactivity are skipped over by jumping the clock from event time to event time. (Fixed-increment time advance does not skip over these inactive periods, which can eat up a lot of computer time; see App. 1A.) It should be noted that the successive jumps of the simulation clock are generally variable (or unequal) in size.

EXAMPLE 1.2. We now illustrate in detail the next-event time-advance approach for the single-server queueing system of Example 1.1. We need the following notation:

t_i = time of arrival of the ith customer ($t_0 = 0$)
$A_i = t_i - t_{i-1}$ = interarrival time between $(i-1)$st and ith arrivals of customers
S_i = time that server actually spends serving ith customer (exclusive of customer's delay in queue)
D_i = delay in queue of ith customer
$c_i = t_i + D_i + S_i$ = time that ith customer completes service and departs
e_i = time of occurrence of ith event of any type (ith value the simulation clock takes on, excluding the value $e_0 = 0$)

Each of these defined quantities will generally be a random variable. Assume that the probability distributions of the interarrival times $A_1, A_2, \ldots$ and the service times $S_1, S_2, \ldots$ are known and have cumulative distribution functions (see Sec. 4.2) denoted by F_A and F_S, respectively. (In general, F_A and F_S would be determined by collecting data from the system of interest and then specifying distributions consistent with these data using the techniques of Chap. 6.) At time $e_0 = 0$ the status of the server is idle, and the time t_1 of the first arrival is determined by generating A_1 from F_A (techniques for generating random observations from a specified distribution are discussed in Chap. 8) and adding it to 0. The simulation clock is then advanced from e_0 to the time of the next (first) event, $e_1 = t_1$. (See Fig. 1.2, where the curved arrows represent advancing the simulation clock.) Since the customer arriving at time t_1 finds the server idle, she immediately enters service and has a delay in queue of $D_1 = 0$ and the status of the server is changed from idle to busy. The time, c_1, when the arriving customer will complete service is computed by generating S_1 from F_S and adding it to t_1. Finally, the time of the second arrival, t_2, is computed as $t_2 = t_1 + A_2$, where A_2 is generated from F_A. If $t_2 < c_1$, as depicted in Fig. 1.2, the simulation clock is advanced from e_1 to the time of the next event, $e_2 = t_2$. (If c_1 were less than t_2, the clock would be advanced from e_1 to c_1.) Since the customer arriving at time t_2 finds the server already busy, the number of customers in the queue is increased from 0 to 1 and the time of arrival of this customer is recorded; however, his service time S_2 is not generated at this time. Also, the time of the third arrival, t_3, is computed as $t_3 = t_2 + A_3$. If $c_1 < t_3$, as depicted in the figure, the simulation clock is advanced from e_2 to the time of the next event, $e_3 = c_1$, where the customer

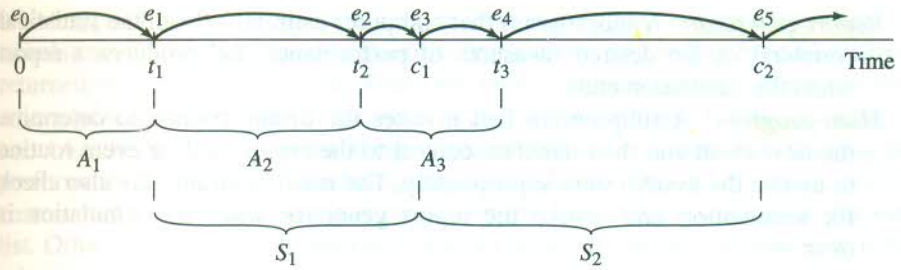

FIGURE 1.2
The next-event time-advance approach illustrated for the single-server queueing system.

completing service departs, the customer in the queue (i.e., the one who arrived at time t_2) begins service and his delay in queue and service-completion time are computed as $D_2 = c_1 - t_2$ and $c_2 = c_1 + S_2$ (S_2 is now generated from F_S), and the number of customers in the queue is decreased from 1 to 0. If $t_3 < c_2$, the simulation clock is advanced from e_3 to the time of the next event, $e_4 = t_3$, etc. The simulation might eventually be terminated when, say, the number of customers whose delays have been observed reaches some specified value.

1.3.2 Components and Organization of a Discrete-Event Simulation Model

Although simulation has been applied to a great diversity of real-world systems, discrete-event simulation models all share a number of common components and there is a logical organization for these components that promotes the programming, debugging, and future changing of a simulation model's computer program. In particular, the following components will be found in most discrete-event simulation models using the next-event time-advance approach programmed in a general-purpose language:

System state: The collection of state variables necessary to describe the system at a particular time

Simulation clock: A variable giving the current value of simulated time

Event list: A list containing the next time when each type of event will occur

Statistical counters: Variables used for storing statistical information about system performance

Initialization routine: A subprogram to initialize the simulation model at time 0

Timing routine: A subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur

Event routine: A subprogram that updates the system state when a particular type of event occurs (there is one event routine for each event type)

Library routines: A set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model

Report generator: A subprogram that computes estimates (from the statistical counters) of the desired measures of performance and produces a report when the simulation ends

Main program: A subprogram that invokes the timing routine to determine the next event and then transfers control to the corresponding event routine to update the system state appropriately. The main program may also check for termination and invoke the report generator when the simulation is over.

The logical relationships (flow of control) among these components are shown in Fig. 1.3. The simulation begins at time 0 with the main program invoking the initialization routine, where the simulation clock is set to zero, the system state and the statistical counters are initialized, and the event list is initialized. After control has been returned to the main program, it invokes the timing routine to determine

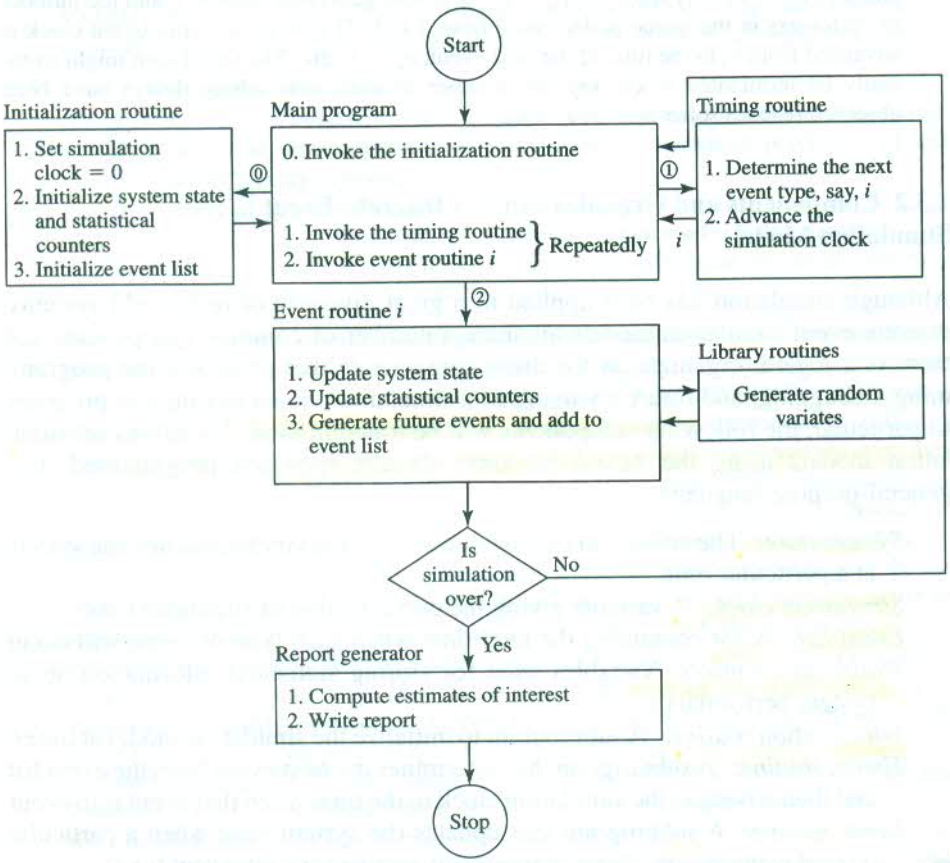

FIGURE 1.3
Flow of control for the next-event time-advance approach.

which type of event is most imminent. If an event of type i is the next to occur, the simulation clock is advanced to the time that event type i will occur and control is returned to the main program. Then the main program invokes event routine i, where typically three types of activities occur: (1) The system state is updated to account for the fact that an event of type i has occurred; (2) information about system performance is gathered by updating the statistical counters; and (3) the times of occurrence of future events are generated, and this information is added to the event list. Often it is necessary to generate random observations from probability distributions in order to determine these future event times; we will refer to such a generated observation as a *random variate*. After all processing has been completed, either in event routine i or in the main program, a check is typically made to determine (relative to some stopping condition) if the simulation should now be terminated. If it is time to terminate the simulation, the report generator is invoked from the main program to compute estimates (from the statistical counters) of the desired measures of performance and to produce a report. If it is not time for termination, control is passed back to the main program and the main program–timing routine–main program–event routine–termination check cycle is repeated until the stopping condition is eventually satisfied.

Before concluding this section, a few additional words about the system state may be in order. As mentioned in Sec. 1.2, a system is a well-defined collection of *entities*. Entities are characterized by data values called *attributes*, and these attributes are part of the system state for a discrete-event simulation model. Furthermore, entities with some common property are often grouped together in *lists* (or *files* or *sets*). For each entity there is a *record* in the list consisting of the entity's attributes, and the order in which the records are placed in the list depends on some specified rule. (See Chap. 2 for a discussion of efficient approaches for storing lists of records.) For the single-server queueing facility of Examples 1.1 and 1.2, the entities are the server and the customers in the facility. The server has the attribute "server status" (busy or idle), and the customers waiting in queue have the attribute "time of arrival." (The number of customers in the queue might also be considered an attribute of the server.) Furthermore, as we shall see in Sec. 1.4, these customers in queue will be grouped together in a list.

The organization and action of a discrete-event simulation program using the next-event time-advance mechanism as depicted above are fairly typical when coding such simulations in a general-purpose programming language such as FORTRAN or C; it is called the *event-scheduling approach* to simulation modeling, since the times of future events are explicitly coded into the model and are scheduled to occur in the simulated future. It should be mentioned here that there is an alternative approach to simulation modeling, called the *process approach*, that instead views the simulation in terms of the individual entities involved, and the code written describes the "experience" of a "typical" entity as it "flows" through the system; coding simulations modeled from the process point of view usually requires the use of special-purpose simulation software, as discussed in Chap. 3. Even when taking the process approach, however, the simulation is actually executed behind the scenes in the event-scheduling logic as described above.

1.4 SIMULATION OF A SINGLE-SERVER QUEUEING SYSTEM

This section shows in detail how to simulate a single-server queueing system such as a one-operator barbershop. Although this system seems very simple compared with those usually of real interest, how it is simulated is actually quite representative of the operation of simulations of great complexity.

In Sec. 1.4.1 we describe the system of interest and state our objectives more precisely. We explain intuitively how to simulate this system in Sec. 1.4.2 by showing a "snapshot" of the simulated system just after each event occurs. Section 1.4.3 describes the language-independent organization and logic of the FORTRAN and C codes given in Secs. 1.4.4 and 1.4.5. The simulation's results are discussed in Sec. 1.4.6, and Sec. 1.4.7 alters the stopping rule to another common way to end simulations. Finally, Sec. 1.4.8 briefly describes a technique for identifying and simplifying the event and variable structure of a simulation.

1.4.1 Problem Statement

Consider a single-server queueing system (see Fig. 1.4) for which the interarrival times $A_1, A_2, \ldots$ are *independent and identically distributed* (IID) random variables.

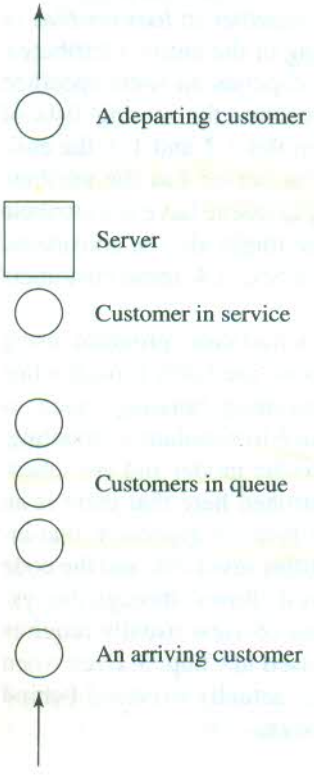

FIGURE 1.4
A single-server queueing system.

("Identically distributed" means that the interarrival times have the same probability distribution.) A customer who arrives and finds the server idle enters service immediately, and the service times $S_1, S_2, \ldots$ of the successive customers are IID random variables that are independent of the interarrival times. A customer who arrives and finds the server busy joins the end of a single queue. Upon completing service for a customer, the server chooses a customer from the queue (if any) in a first-in, first-out (FIFO) manner. (For a discussion of other queue disciplines and queueing systems in general, see App. 1B.)

The simulation will begin in the "empty-and-idle" state; i.e., no customers are present and the server is idle. At time 0, we will begin waiting for the arrival of the first customer, which will occur after the first interarrival time, A_1, rather than at time 0 (which would be a possibly valid, but different, modeling assumption). We wish to simulate this system until a fixed number (n) of customers have completed their delays in queue; i.e., the simulation will stop when the nth customer enters service. Note that the *time* the simulation ends is thus a random variable, depending on the observed values for the interarrival and service-time random variables.

To measure the performance of this system, we will look at estimates of three quantities. First, we will estimate the expected average delay in queue of the n customers completing their delays during the simulation; we denote this quantity by $d(n)$. The word "expected" in the definition of $d(n)$ means this: On a given run of the simulation (or, for that matter, on a given run of the actual system the simulation model represents), the actual average delay observed of the n customers depends on the interarrival and service-time random variable observations that happen to have been obtained. On another run of the simulation (or on a different day for the real system) there would probably be arrivals at different times, and the service times required would also be different; this would give rise to a different value for the average of the n delays. Thus, the average delay on a given run of the simulation is properly regarded as a random variable itself. What we want to estimate, $d(n)$, is the *expected value* of this random variable. One interpretation of this is that $d(n)$ is the average of a large (actually, infinite) number of n-customer average delays. From a single run of the simulation resulting in customer delays $D_1, D_2, \ldots, D_n$, an obvious estimator of $d(n)$ is

$$\hat{d}(n) = \frac{\sum_{i=1}^{n} D_i}{n}$$

which is just the average of the n D_i's that were observed in the simulation [so that $\hat{d}(n)$ could also be denoted by $\overline{D}(n)$]. [Throughout this book, a hat ($\hat{}$) above a symbol denotes an estimator.] It is important to note that by "delay" we do not exclude the possibility that a customer could have a delay of zero in the case of an arrival finding the system empty and idle (with this model, we know for sure that $D_1 = 0$); delays with a value of 0 *are* counted in the average, since if many delays were zero this would represent a system providing very good service, and our output measure should reflect this. One reason for taking the average of the D_i's, as opposed to just looking at them individually, is that they will not have the same distribution (e.g., $D_1 = 0$, but D_2 could be positive), and the average gives us a single composite

measure of all the customers' delays; in this sense, this is not the usual "average" taken in basic statistics, as the individual terms are not independent random observations from the same distribution. Note also that by itself, $\hat{d}(n)$ is an estimator based on a sample of size *1*, since we are making only one complete simulation run. From elementary statistics, we know that a sample of size 1 is not worth much; we return to this issue in Chaps. 9 through 12.

While an estimate of $d(n)$ gives information about system performance from the customers' point of view, the management of such a system may want different information; indeed, since most real simulations are quite complex and may be time-consuming to run, we usually collect many output measures of performance, describing different aspects of system behavior. One such measure for our simple model here is the expected average number of customers in the queue (but not being served), denoted by $q(n)$, where the n is necessary in the notation to indicate that this average is taken over the time period needed to observe the n delays defining our stopping rule. This is a different kind of "average" than the average delay in queue, because it is taken over (continuous) time, rather than over customers (being discrete). Thus, we need to define what is meant by this *time*-average number of customers in queue. To do this, let $Q(t)$ denote the number of customers in queue at time t, for any real number $t \geq 0$, and let $T(n)$ be the time required to observe our n delays in queue. Then for any time t between 0 and $T(n)$, $Q(t)$ is a nonnegative integer. Further, if we let p_i be the expected *proportion* (which will be between 0 and 1) of the time that $Q(t)$ is equal to i, then a reasonable definition of $q(n)$ would be

$$q(n) = \sum_{i=0}^{\infty} ip_i$$

Thus, $q(n)$ is a weighted average of the possible values i for the queue length $Q(t)$, with the weights being the expected proportion of time the queue spends at each of its possible lengths. To estimate $q(n)$ from a simulation, we simply replace the p_i's with estimates of them, and get

$$\hat{q}(n) = \sum_{i=0}^{\infty} i\hat{p}_i \tag{1.1}$$

where $\hat{p}_i$ is the *observed* (rather than expected) proportion of the time *during the simulation* that there were i customers in the queue. Computationally, however, it is easier to rewrite $\hat{q}(n)$ using some geometric considerations. If we let T_i be the *total* time during the simulation that the queue is of length i, then $T(n) = T_0 + T_1 + T_2 + \cdots$ and $\hat{p}_i = T_i/T(n)$, so that we can rewrite Eq. (1.1) above as

$$\hat{q}(n) = \frac{\sum_{i=0}^{\infty} iT_i}{T(n)} \tag{1.2}$$

Figure 1.5 illustrates a possible time path, or *realization*, of $Q(t)$ for this system in the case of $n = 6$; ignore the shading for now. Arrivals occur at times 0.4, 1.6, 2.1, 3.8, 4.0, 5.6, 5.8, and 7.2 Departures (service completions) occur at times 2.4, 3.1, 3.3, 4.9, and 8.6, and the simulation ends at time $T(6) = 8.6$. Remember in looking

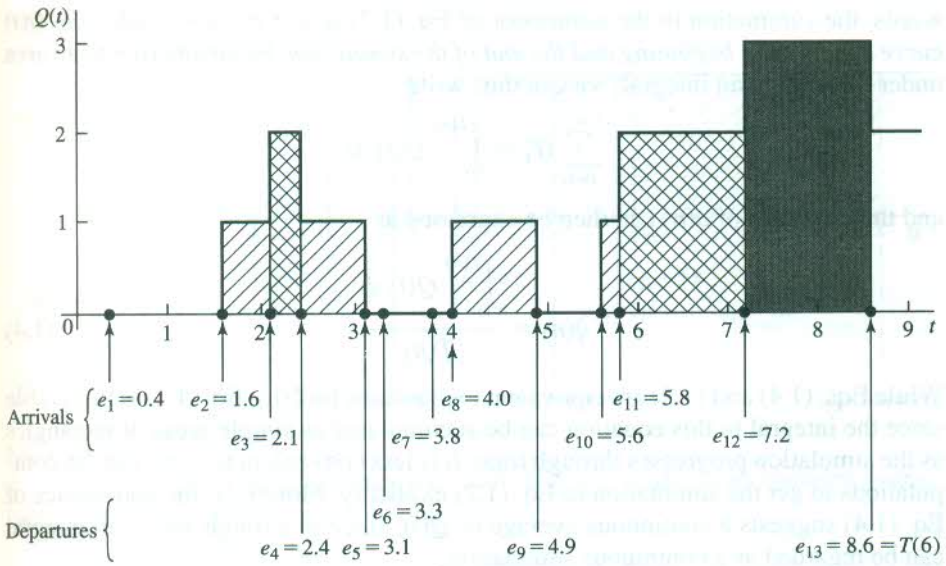

FIGURE 1.5
$Q(t)$, arrival times, and departure times for a realization of a single-server queueing system.

at Fig. 1.5 that $Q(t)$ does not count the customer in service (if any), so between times 0.4 and 1.6 there is one customer in the system being served, even though the queue is empty [$Q(t) = 0$]; the same is true between times 3.1 and 3.3, between times 3.8 and 4.0, and between times 4.9 and 5.6. Between times 3.3 and 3.8, however, the system is empty of customers and the server is idle, as is obviously the case between times 0 and 0.4. To compute $\hat{q}(n)$, we must first compute the T_i's, which can be read off Fig. 1.5 as the (sometimes separated) intervals over which $Q(t)$ is equal to 0, 1, 2, and so on:

$$T_0 = (1.6 - 0.0) + (4.0 - 3.1) + (5.6 - 4.9) = 3.2$$
$$T_1 = (2.1 - 1.6) + (3.1 - 2.4) + (4.9 - 4.0) + (5.8 - 5.6) = 2.3$$
$$T_2 = (2.4 - 2.1) + (7.2 - 5.8) = 1.7$$
$$T_3 = (8.6 - 7.2) = 1.4$$

($T_i = 0$ for $i \geq 4$, since the queue never grew to those lengths in this realization.) The numerator in Eq. (1.2) is thus

$$\sum_{i=0}^{\infty} iT_i = (0 \times 3.2) + (1 \times 2.3) + (2 \times 1.7) + (3 \times 1.4) = 9.9 \qquad (1.3)$$

and so our estimate of the time-average number in queue from this particular simulation run is $\hat{q}(6) = 9.9/8.6 = 1.15$. Now, note that each of the nonzero terms on the right-hand side of Eq. (1.3) corresponds to one of the shaded areas in Fig. 1.5: 1×2.3 is the diagonally shaded area (in four pieces), 2×1.7 is the cross-hatched area (in two pieces), and 3×1.4 is the screened area (in a single piece). In other

words, the summation in the numerator of Eq. (1.2) is just the *area under the $Q(t)$ curve between the beginning and the end of the simulation*. Remembering that "area under a curve" is an integral, we can thus write

$$\sum_{i=0}^{\infty} iT_i = \int_0^{T(n)} Q(t)\, dt$$

and the estimator of $q(n)$ can then be expressed as

$$\hat{q}(n) = \frac{\int_0^{T(n)} Q(t)\, dt}{T(n)} \tag{1.4}$$

While Eqs. (1.4) and (1.2) are equivalent expressions for $\hat{q}(n)$, Eq. (1.4) is preferable since the integral in this equation can be accumulated as simple areas of rectangles as the simulation progresses through time. It is less convenient to carry out the computations to get the summation in Eq. (1.2) explicitly. Moreover, the appearance of Eq. (1.4) suggests a continuous average of $Q(t)$, since in a rough sense, an integral can be regarded as a continuous summation.

The third and final output measure of performance for this system is a measure of how busy the server is. The expected *utilization* of the server is the expected proportion of time during the simulation [from time 0 to time $T(n)$] that the server is busy (i.e., not idle), and is thus a number between 0 and 1; denote it by $u(n)$. From a single simulation, then, our estimate of $u(n)$ is $\hat{u}(n)$ = the *observed* proportion of time during the simulation that the server is busy. Now $\hat{u}(n)$ could be computed directly from the simulation by noting the times at which the server changes status (idle to busy or vice versa) and then doing the appropriate subtractions and division. However, it is easier to look at this quantity as a continuous-time average, similar to the average queue length, by defining the "busy function"

$$B(t) = \begin{cases} 1 & \text{if the server is busy at time } t \\ 0 & \text{if the server is idle at time } t \end{cases}$$

and so $\hat{u}(n)$ could be expressed as the proportion of time that $B(t)$ is equal to 1. Figure 1.6 plots $B(t)$ for the same simulation realization as used in Fig. 1.5 for $Q(t)$. In this case, we get

$$\hat{u}(n) = \frac{(3.3 - 0.4) + (8.6 - 3.8)}{8.6} = \frac{7.7}{8.6} = 0.90 \tag{1.5}$$

indicating that the server was busy about 90 percent of the time during this simulation. Again, however, the numerator in Eq. (1.5) can be viewed as the area under the $B(t)$ function over the course of the simulation, since the height of $B(t)$ is always either 0 or 1. Thus,

$$\hat{u}(n) = \frac{\int_0^{T(n)} B(t)\, dt}{T(n)} \tag{1.6}$$

and we see again that $\hat{u}(n)$ is the continuous average of the $B(t)$ function, corresponding to our notion of utilization. As was the case for $\hat{q}(n)$, the reason for writing

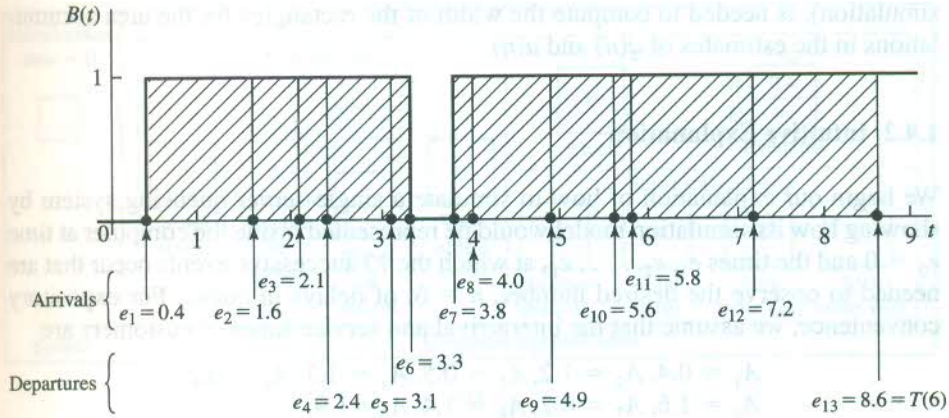

FIGURE 1.6
$B(t)$, arrival times, and departure times for a realization of a single-server queueing system (same realization as in Fig. 1.5).

$\hat{u}(n)$ in the integral form of Eq. (1.6) is that computationally, as the simulation progresses, the integral of $B(t)$ can easily be accumulated by adding up areas of rectangles. For many simulations involving "servers" of some sort, utilization statistics are quite informative in identifying bottlenecks (utilizations near 100 percent, coupled with heavy congestion measures for the queue leading in) or excess capacity (low utilizations); this is particularly true if the "servers" are expensive items such as robots in a manufacturing system or large mainframe computers in a data-processing operation.

To recap, the three measures of performance are the average delay in queue $\hat{d}(n)$, the time-average number of customers in queue $\hat{q}(n)$, and the proportion of time the server is busy $\hat{u}(n)$. The average delay in queue is an example of a *discrete-time statistic*, since it is defined relative to the collection of random variables $\{D_i\}$ that have a discrete "time" index, $i = 1, 2, \ldots$. The time-average number in queue and the proportion of time the server is busy are examples of *continuous-time statistics*, since they are defined on the collection of random variables $\{Q(t)\}$ and $\{B(t)\}$, respectively, each of which is indexed on the continuous time parameter $t \in [0, \infty)$. (The symbol $\in$ means "contained in." Thus, in this case, t can be any nonnegative real number.) Both discrete-time and continuous-time statistics are common in simulation, and they furthermore can be other than averages. For example, we might be interested in the *maximum* of all the delays in queue observed (a discrete-time statistic), or the *proportion* of time during the simulation that the queue contained at least five customers (a continuous-time statistic).

The events for this system are the arrival of a customer and the departure of a customer (after a service completion); the state variables necessary to estimate $d(n)$, $q(n)$, and $u(n)$ are the status of the server (0 for idle and 1 for busy), the number of customers in the queue, the time of arrival of each customer currently in the queue (represented as a list), and the time of the last (i.e., most recent) event. The time of the last event, defined to be e_{i-1} if $e_{i-1} \leq t < e_i$ (where t is the current time in the